Interactive Diagrams for Design Genetics - Implementing Intelligent Processors

Abstract: The two basic components (the algorithmic and the nonalgorithmic) of the multifunctional intelligent design (MIND**)** machine, whose implementation forms the object of this paper, correspond to our conceptual model of design and to the epistemological decision we adopted for its implementation. We present here an operational mechanism that delegates parametric variation of a large number of elements in ways similar to random mutation and cumulative change through the propagation of coded genetic design information. Thus, the evolving of design and its improvement are accomplished. In order to implement this model, we decided to adopt a neural network mechanism. Implementation consists of networks of intelligent processors acting on diagrams, which control design genetic elements. The machine is designed to operate in three basic modes: direct, delegation, and evolution.

Keywords: Concurrency, Design Evolution, Diagrams, Design Genetics, System of Representation, System of Relations, Design Gene, Design, Chromosome, Design Intelligence, Parametric Variation, Neural Networks.

1. Introduction

In studying the problems involved in implementing Intelligent Processors (IPs), we recognized that the metaphor of the design team advanced in the conceptual phase of our work [7] represented only a methodological framework. In order to achieve effective implementation, an operational mechanism had to be conceived. Such a mechanism had to allow for operations on design entities as they resulted in the processes of design. More specifically, the necessary mechanism needed to be appropriate to intelligent processes applied to:
1.all categories of design
2. the wide range of designed artifacts
3. the flow state (design as a fluid process)
as defined within the conceptual model. Our implementation work led to the discovery that a combination of a genetic mechanism and neural network implementation are well suited to the problem. The most complex objects we know--living forms--have evolved through a particular kind of interaction of patterns, i.e., random mutation and cumulative change through the propagation of the *coded* genetic information of surviving offspring [4]. Design artifacts, as products of complex living entities, can be seen as the results of processes analogous to the genetic.

The need to delegate the task of the parametric variation of a large number of elements to the system, with little or no direct supervision, prompted us to develop a genetically inspired mechanism that allows design alternatives to ‘evolve,’ as well as to develop through direct designer-machine interaction. The high degree of interconnection and the multiple interrelations of design parameters suggested a neural network implementation, within the overall structure of a hybrid machine. By no accident, neural networks implementations prove to be best suited for the mechanism adopted.

Our implementation framework, in outline form, consists of a network of IPs that acts on diagrams. The diagrams control ‘design gene’ and ‘design chromosome’ coding strips within a genetically inspired mechanism. The ‘design genes’ provide weighted inputs to a neural net; and the neural net determines the parameters that form particular instances of pre-existing archetypes. Each of these aspects of the Design Machine, their interrelations and functions, and the modes of operation they allow will be presented in detail below.

2. Overview

2.1. A working definition of intelligence (conceptual model revisited)

Our definition of intelligence [7] is recognition, manipulation, and synthesis of patterns (and patterns of patterns). Our hypothesis is that these operations happen at many levels. At very low levels, they are

evidenced in the ability of intelligent entities to observe similarity, repetition, and affinity, while at higher levels, they result in the recognition, manipulation, and synthesis of high-level types (commonly known as typologies).

2.2. Search vs. generation

Design has often been described as a search, and various heuristic methods for dealing with what seems to be a very large search space have been proposed. Yet, it often seems that designers do not search at all, but arrive at a solution by introducing a set of relationships that the design is to have. This normative aspect of design is analogous to the approach one takes in solving differential equations. While during the design of a complex object there are many instances that require efficient search techniques, the most complex and interesting designs result from the complex interactions of predetermined patterns. In the machine we propose, this is accommodated by offering the designer the capability of setting generative systems to guide the parametric variation of parts, or the capability to set the direction, vector, or gradient of the mutation of 'design genes'. However, as it will become clear in the next section, we do not limit ourselves to the pattern imposition model.

2.3. The design machine as a network

MIND is a Design Machine constituted as a network of intelligent processors:

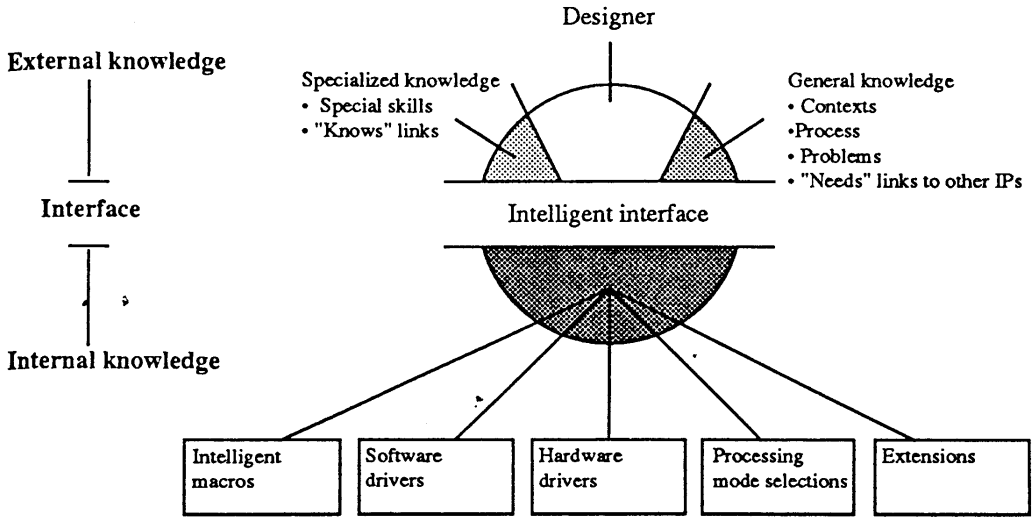


Fig. 1. MIND: a design machine as a network

The context for its functioning is the culture at large. Within culture, the categories of designed objects, events, systems of relations, and systems of references are acknowledged and constitute a referential framework for future design work. Newly designed items are compared to those available and accepted or refused not only in view of their intrinsic characteristics, but also in respect to the "design language" of any given time and environment. Within the network, no hierarchy is established among the various processors. This network constitutes and embodies the metaphor of the *Design Team.* The "knows" and "needs" links of the IPs interact and "fire" multiple processors in order to distribute, collect, and integrate information.

The intelligent design process is one of design evolution and consists of contributions to the final design by each member of the team. The branching out (hermeneutics) and the coagulation of the design solution (heuristic) are the result of an implementation of intelligent parallelism as a dynamic

interconnection of neural cells. The design team metaphor makes parallelism necessary; parallelism influences implementation decisions.

3. Designata and Cognitive Characteristics of Design

The development of Intelligent Computer-Aided Design (ICAD) systems requires a three-part definition of the problem, derived from the answers tot he following questions: How do people design? What do people design? What are the characteristics of the design process?

3.1. The 'how' of design

There is a limited number of ways in which people can approach a design task, i.e., design as re-design, as problem-solving, as evolution, as invention, as pattern imposition, and as selection from existing parts/ types and recombination. Each is computationally different, but an ICAD system should accommodate all of them. Our previous insistence on the pattern imposition path reflects an interest in an area of design considered until now too hard to be dealt with [3].

3.2. The 'what' of design

The range of designed artifacts, which we call *designata,* can be specified as objects, events, systems of relations, and systems of reference. The first group mentioned is made up of design artifacts commonly identifiable as *objects:* graphic design in 2-D, product design in 3-D, architectural spaces as 3-D voids. The second category, identified as *events,* comprises sequential ceremonies or sequences of actions pertinent to a task, configurational designs, such as temporal reasoning, plans of action. The third, *systems of relation,* covers games, simulations, organizational structures, etc. And the fourth category, *systems of references,* refers to the artificial, notational, textual, etc. This method of categorizing is meant cover the entire gamut of entities to which the qualifier *designed* applies within culture, and which a design machine should aim to accommodate.

3.3. The process of design

The support of a fluid state of thinking, for which we have appropriated the term 'flow state,' is the third layer of concern. In order to maintain the appropriate cognitive characteristics during the activity of designing, we propose a range of interactive possibilities form direct manipulation of design elements to automatic design through the evolutionary model. A time-memory window (cf. 4.2.2) will be specifically implemented, given the implications of the non-serial model of design pursued.

We want to stress here that all the layers described are to be supported by the ICAD system. This certainly constrains the implementation of IPS, but will result in a system that is not only a collection of tools controlled in an intelligent way, but also an environment for intelligent design processes.

4. Implementation Aspects

The implementational framework of the Design Machine is constituted by the architecture through which the IPs act. The layers of the architecture are diagrams, design genetics, and parametrics. We join an algorithmic approach to a non-algorithmic (neural network) one. On the algorithmic side, the structure of tasks and algorithms require that we maintain an inference procedure and a representation function. The inference procedure implemented in the activity of the IPs is abduction. In the process of design, each new potential design is an implicit explanation. These explanations are given in the form of diagrams and are applied as input on IPs, implemented as neural networks. On the non-algorithmic side the need for parallelism, robustness, and plasticity requires that we ensure relaxation and learning (basically through back-propagation). The so-called structured neural networks [5] seem best suited for IPs implementation. The interaction between the two approaches definitely reflects the complexity of the design tasks considered in our model. An issue of special concern is the interaction between the network and routines necessary to support design work (such as CAD tools).

4.1. Diagrams

The main element of interaction that we propose, and which is central to our data structures, is the diagram. Diagrams are structured templates for the interactive control of coding strips. At the macro-level, the diagram functions as an organizing device for different kinds of information. It contains scripts of expected actions, frames of expected values with slots containing default values, and special categories called roles and performances. Roles contain information that modifies the information in the diagram to correspond to different viewpoints (e.g. designers vs. client vs. user). Performances contain information about how things should act, that is, information that can guide the system in simulations of reality.

Diagrams, by definition, explain rather than represent--showing components, arrangements, and relations--and in that sense embody a viewpoint. Different viewpoints, identifying different contexts, can thus be seen as different diagrams (i.e., showing different characteristics by explaining the object according to different “filters”). Diagrams are descriptions structured according to a determined perspective. They act as implicit context identifiers and filters. They are both visual and verbal. Since diagrams are viewpoints, multiple diagrams can be generated from the same object description by filtering out dimensions of the problem that are secondary to the task at hand. Thus, diagrams are also projections.

Diagrams are the main token of exchange between IPs and the designer. They consist of parts, attributes of parts, hierarchies of parts, degrees of similarity between parts (affinity of extensional attribute lists within a particular view) and are nested, layered, and interactive. They present relations among parts, (again attributes, hierarchies, degrees of similarity). Parametric relations [6] can be described as how a change of parameter allows for the generation of new diagrams showing the influence of the chosen parameter. These relations are implemented in the algorithmic path of the machine as associations between parameters specified through functions (a generative system may be thought of as a collection of such functions).

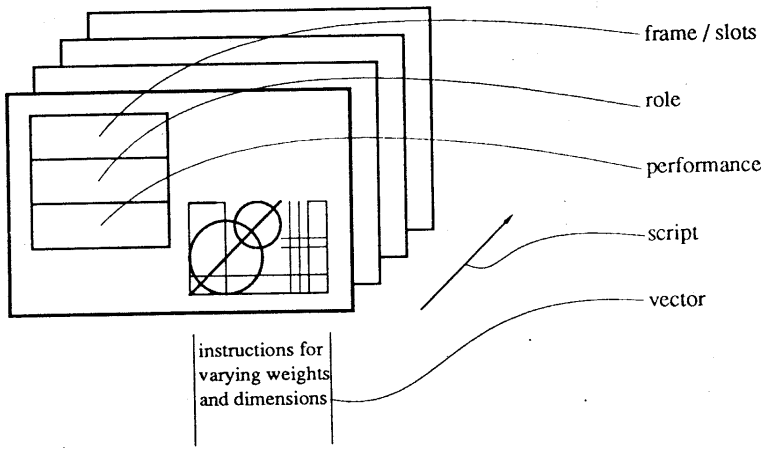


Fig. 2.

Actions on the components (parts, relations, attributes) are supported by using operations on diagrams (insertions, omissions, substitutions as well as higher level operations). In view of their condition as explanatory visual descriptions, diagrams can be processes as abductions. In this case, inputting diagrams into IPs by acting on components leads to processes of abduction, i.e., design inferences.

Diagrams allow for the implementation of tree specific aspects of design. First, diagrams act as filters, allowing the designer to impose an abstract structure on the problem by concentrating on particular information, while ignoring aspects that are not directly relevant. Second, diagrams embody viewpoints.

The problem at hand is different according to the viewpoints we choose to emphasize--that of the designer, the client, the user--and is represented by different diagrams. Third, diagrams imply contexts. A diagram representing the design of a bridge in the context of the historical development of bridges differs from a diagram representing the design in the context of regional differences in vernacular form. It is different again from one showing the design in the context of technological development.

Obviously, the main data component in a diagram is visual. Parts and relations of parts are represented in a network. Each node of the diagram can be another diagram. Geometric relations in the diagram, such as size, proximity, overlap, alignment, and containment, carry meaning; modification of the diagram in such ways directly affects the internal data structures and the design itself. Although the diagram is two-dimensional, it can be interpreted as representing more dimensions. In fact, the problem of 'emergent form' [6] pertaining to diagrams is one that we attempt to resolve by establishing 'implied' nodes that the designer has access to. Multiple interpretations of diagrams can coexist as modified instances of the initial diagram.

For reasons of terminological clarity, we want to specify that the diagram is not an association net. This "negative" definition (what it is not) allows us to define some important characteristics:
1. It is a diagram of diagrams, i.e. it is nested.
2. It is more than a topological graph; it is also a geometric entity. In diagrams, proximity and size, for example, are recognized.
3. Diagrams can recognize union-intersection-difference relations, etc., i.e., display properties characteristic of a set-theoretic entity.
4. Diagrams are used as an intelligent interface, allowing direct action on relations and parts.
5. They provide assistance in the 'emergent form' problem by constructing node and relation variables automatically at the intersections of pre-existing connections.
6.Like tracing paper, diagrams are layered, so that alternative diagrams can be superimposed and 'traced' to produce new variations.

4.1.1. Stacks of diagrams.

As far as the designer is concerned, the final design is represented as a stack of diagrams. This implies that the system has an underlying layer of 'translators' which take the information in the diagrams as input and, through interaction with the design genes, or with the design chromosomes, drive the appropriate parametric modeling software Alternative stacks of diagrams imply alternative design solutions still under consideration.

Diagrams have one additional role in the Design Machine. A stack of diagrams showing the present state of the design process and the history of the project to date is generated by the system and is accessible to the designer. Thus the 'flow state' is supported in the manner of nonlinear possible associations characteristic of human process of design. An economic current implementation of this can be seen in HyperCard (a trademark of Apple Computer, Inc.), which is based on the non-linear relational model of hypertext [2]. Other such implementations are either already available or soon to be released. In the process of implementing intelligent processors we took advantage of HyperCard in order to test the establishment of links among diagrams, and the intelligent supervision of such links (delete what is not relevant, establish new links, create meta-levels of links, etc.).

4.1.2. Diagrams and intelligence.

In order to embody intelligence, diagrams must accommodate analysis and synthesis, and have the ability to withstand incompleteness and conflict. This is accomplished by linking the diagrams to the neural network [1]. Hence, the DM can indeed learn from the design experience as it results from its functioning. Neural networks have associated memory, adaptive learning from examples, and combinatorial optimization characteristics. Such characteristics are essential for the design process as we defined it.

4.1.3. Four categories of design.

Our initial premise was that design is essentially a visual activity occurring at a high cognitive level. We looked at what is common to all three types of design [3] and discovered that diagrams, as defined above, prove to be most suitable for the operational mechanism that we needed. This discovery is supported by Peirce's work on thinking [8] and the relation between diagrams and abduction. In applying diagrams in

our initial concept, we discovered that the three commonly acknowledged categories of design could easily be described as particular operations on diagrams.

Category Three: Modify “geometry” of existing diagram.
Category Two: Modify “topology” of existing diagram or combine diagrams.
Category One: Invent diagram.

Furthermore, a fourth category could be added.
Category Zero: Invent archetype.

4.1.4. Vector set.

A data structure, which we call a *vector-set,* is associated with each diagram or stack of diagrams. This allows the designer to delegate tasks to IPs by specifying the directions in which various variables are to be modified, either systematically, using a generative system, or randomly, in ‘evolution’ mode, along the genetic mechanism of design we adopted.

Behind the diagram is a second data structure which is usually hidden but which can be modified directly, either by the designer or by the IPS. This data structure is a ‘list of lists,’ a genetic strip containing the information of the parts that make up an object. Each sub-part has a corresponding strip. Changes to the diagram are directly translated into changes in the strip. Conversely, changes in the strip directly affect the diagram.

4.2. Design genetics

The genetic mechanism employed involves an analogy that is particularly powerful in assisting the design of complex artifacts. In biology, genes control specific characteristics of a life form, such as eye color or overall height, for instance, While genes remain the same, particular ‘values’ in the gene can differ, producing variations of that characteristic (changing the values of a gene will change blue eyes to brown eyes). An entire life form, on the other hand, is represented by a collection of genes. Variations of this collection of genes or ‘chromosomes,’ even by simply changing the order of identical genes, produces different species.

Besides the evident analogies to the “evolution” of design, we have chosen to appropriate this system because several difficult design operations can be expressed directly in genetic terms, which in turn allow for computational descriptions and operations upon them. Genes can be seen as parametric controls of pre-existing ‘types’, directly related to the idea of design as re-design. ‘Gene splicing’ expresses the invention of new design ‘species’ through the novel combination of pre-existing types into new ‘chromosomes’. ‘Genetic engineering’ allows the creation of entirely new genes. ‘Evolution’ and ‘breeding’ correspond to random and non-random mutation with cumulative selection and change as practiced by designers.

To avoid confusion between the literal terms in their proper biological use and in the computational use that we are implementing, we have adopted the terms ‘design gene’ for the computational equivalent of a gene and ‘design chromosome’ for that of a chromosome. Each design gene can be represented by an extensional ‘strip’ of control and parameter information. Hence, a design chromosome is a strip of strips that specifies a designed artifact uniquely as a particular collection of instances of the pre existing types. Of course, new types can be created when needed, as a separate operation.

4.2.1. Modes of operation.

Disposing of such strips of different length and complexity allows the system to operate in several distinct modes:

1. Direct interaction/Direct Mode. The designer (Natural Intelligent Processor, NIP) manipulates the diagrams directly, altering the underlying design gene structure. Changes in position of size modify the particular parameters in the strip only (i.e. the geometry only), while changes in number, kind, or correspondence modify the strip itself (i.e. the topology).
2. Indirect interaction/Delegation Mode. The designer specifies the direction in which specific parameters are to be modified through the use of the vector-set, and delegates the actual modification to IPs, which carry out the task in the background. The designer is only interrupted when several conditions

have been met within specified ‘time-memory window’.

3. Random mutation - Cumulative change/Evolution Mode. Once the strip/diagram tree for an object has been created by decomposing the object into its underlying types, and while the designer is occupied with other tasks (or away from the DM), the system continues the search in ‘evolution’ mode. Small random changes are made to the values in the design gene strip, producing alternatives. Alternatives that do not conflict with requirements survive and carry forward their design genes for further mutation, while the others become extinct. Unlike hill climbing, this heuristic recognizes that progress can be made occasionally by “climbing” down the hill. It also eliminates backtracking, without eliminating the possibility of moving back to a previous position. Most importantly, it recognizes that often what appears to be a weak alternative can become, given sufficient time for refinement, the most viable alternative.

At any time, several alternatives co-exist and compete for resources (including computational resources). When the number of competing alternatives becomes too large for the available computational resources, the weakest members are eliminated.

4.2.2. Time-memory window.

In order to support a non-serial model of the design process, we propose a time-memory ‘window’. This concept implies that, at any given time, several alternatives to either the whole design or to sub-parts are ‘alive’ within the system and are being worked on by the IPs independently. If they reach a point when merging is possible (because certain common conditions are met) within the time-memory window, they are recognized and kept by the system. If not, they are removed and replaced by other alternatives, also modified in parallel and observed within this window.

In addition to determining the parametric definition of objects through concatenation and instantiation, the design genes are directly linked to the diagrams.

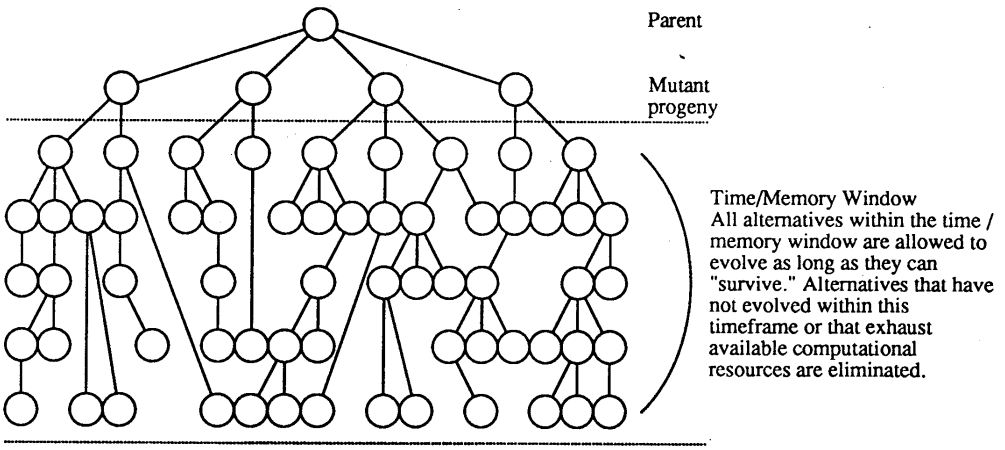


Fig. 3.

4.3. Parametric types

As we have seen, the Design Machine is implemented in a three-tiered architecture, a layer of diagrams representing reconfigurable neural connections (additional sub-layers within this layer provide task subdivisions), a layer of genetic design, and a layer of types. The layer of types consists of four full-parameter modules, corresponding to the four categories of possible designed artifacts (specifically, objects, sequences, relational systems, and reference systems), which we have called *designata.* For each *designatum,* we distinguish the following matrix:

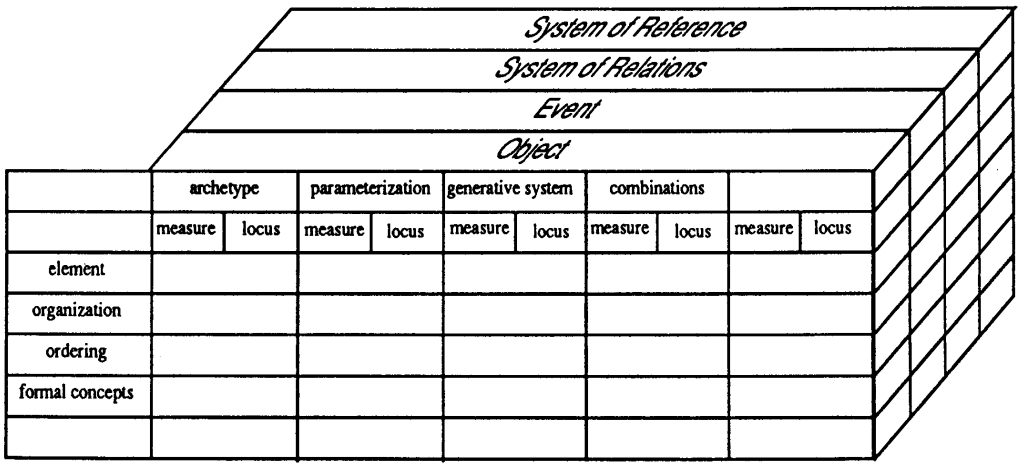


Fig. 4.

This is to say that are at least four levels of primitives that are of interest to the designer, as shown in the rows of the matrix. For instance, an architectural design may be described as consisting of *elements*--such as columns and walls- *organized* on a grid, *ordered* symmetrically, and *exploring the formal idea* of having analogous plan and section relationships. The columns of the matrix show how the variation of these elements is structured. First, for each row-item we can recognize existing *archetypes,* such as different column types (Doric, Ionic, Corinthian) in the element row, grid types (radial, central, orthogonal, triangular) in the organization row, ordering principles (symmetry, asymmetry, clustering) in the ordering row; and formal ideas (plan to section relations, overlap, transformation) in the formal idea row.

Each pre-existing type, and any that are added to the system through its use, can be computerize*d,* as implied by column two. Column three suggests that at this level the system knows how to systematically vary those parameters by employing a variety of *generative systems.* Finally, column four suggests that the system has provisions for keeping track of *combinations* of simpler types at each level.

One more distinction is necessary. For each item in the matrix, we recognize that variations can deal with relative or absolute values, or, stated differently, with the *locus* or the *measure* of a primitive. Depending on the kind of artifact being designed, this distinction may apply to *spatial relations* (hence to topology vs. geometry or topometry), to *temporal relations* (thus to chronology vs. chronometry), and finally to *relations of energy* (and thus to energy loci, such as equilibrium vs. energy measures, such as particular input values to a circuit).

The four broad categories that constitute the designata help ensure completeness. For each item in the matrix, however, we expect that the initial entries will be domain- dependent, at least initially. It is our intention that the organization of the system encouraged cross-disciplinary interactions, of course, and thus no other partitioning of archetypes is attempted.

In terms of implementation this layer is built of fairly standard software modules, such as parametric solid modelers (PADL-2), object-oriented page description languages, etc., to which the DM acts as a front-end.

4.4. IPs learning

An important implementation concept is that of ‘overlay/underlay,’ understood as the extraction/generation of new patterns from old ones through operations of comparison and distinction,

addition/ deletion, elaboration. These take place through the association of a ‘vector-set’ (as defined in 4.1.4) with each diagram. This set of vectors provides control information to a neural net layer. Through initial training and subsequent learning, the network develops the ability to associate particular IPS with specific tasks, while general tasks are propagated throughout the entire network.

An IP receiving inputs that refer to its specialized knowledge does not distribute the information further along the same level, but rather to a lower level, the parametric variation/resolution level. In this intermediate level, neural nets are used to vary related components of the design while ensuring that corresponding dimensions and attributes do not conflict.

5. Functioning

The activity of the Design Machine is based on the parametric interrelation of interactive diagrams and the manipulation of underlying ‘design genes.’ Diagrams as well as strips function parametrically to control the type layer. Let us explain some details.

5.1. ‘Design Gene’ coding strip

In order to explain this mechanism, let us examine the following example. The decomposition of an object will usually yield a structure of abstract types (as shown below).

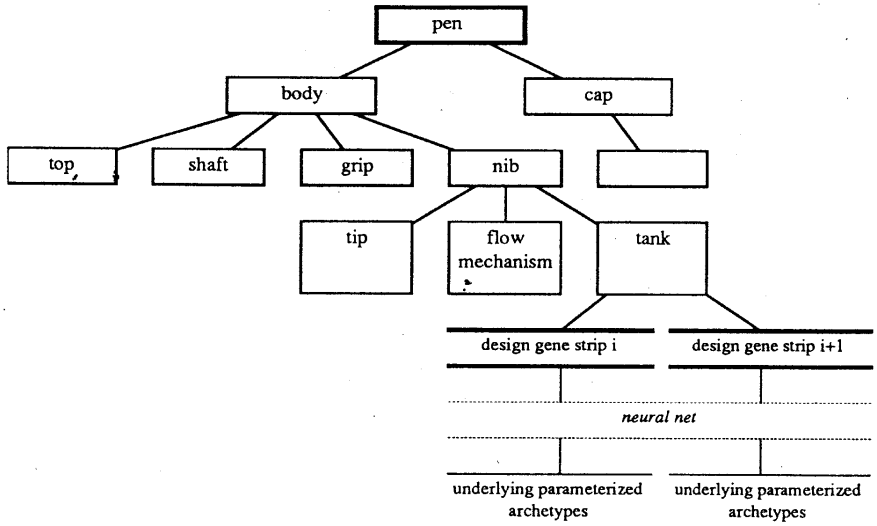
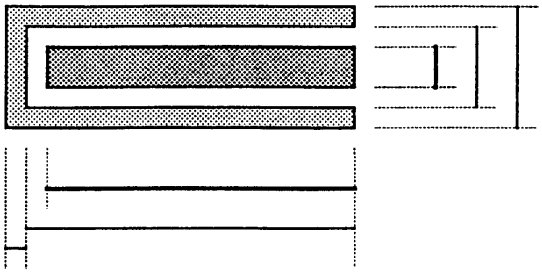


Fig. 5.

The archetypal constructs that form the branches of this tree are parameterized and placed in the systems initial knowledge base, with provisions for modification, reparameterization, addition, deletion, and so on. Each such abstract type is controlled by a ‘gene’ strip of data, which we call a ‘design gene’. The design gene strip specifies which dimensions can be modified and contains the particular values that determine each instance of this type. New types can be invented and parameterized by the designer, and design gene strips can be spliced to create new objects. The overall object is determined by a long strip of design genes created by merging all the strips from the lower levels.

5.2. Specification of modes of operation

The modes of operation discussed in 4.2.1 can be specified as follows:



name	position	parameter list	operation list	function list
tank	x1,y1,z1,R1,P1,Y1	(d h)	2, contained, body	$d_t = f(d_b)$, $h_t = f(h_b)$

name	position	parameter list	operation list	function list
body	x2,y2,z2,R2,P2,Y2	(d_n, d_{out}, h, m)	2, attached, cap	functions of cap

Fig. 6.

Direct Mode $D \rightarrow S\ ;\ D' \rightarrow S'$

Delegation Mode

$$
\begin{aligned}
D + V_i \rightarrow\ & S(V_i)_i \\
& S(V_i)_{i+1} \\
& S(V_i)_{i+2} \rightarrow D'(V_i)_{i+2} \\
& S(V_i)_{i+3} \rightarrow D'(V_i)_{i+3} \\
& \cdot \\
& \cdot \\
& S(V_i)_{i+n} \rightarrow D'(V_i)_{i+n}
\end{aligned}
$$

Evolution Mode

$$
\begin{aligned}
& S + rm_i \rightarrow S'_{rm_i} \\
& \textit{if } S'_{rm_i} \neq \text{nil } \textit{then} \\
& \quad S'_{rm_i} + rm_{i+1} \rightarrow S''_{rm_i + rm_{i+1}}
\end{aligned}
$$

in which D stands for diagram, S for design chromosome or 'strip,' V for vector, and rm for random mutation.

This formalism explains how the design evolves from diagrams to strips and how changes in diagrams result in changes in the genetic strips in each mode. What is not immediately evident is the influence of the vector set on the generation of alternative design strips, and, correspondingly, diagrams. Vector sets are used here for implementation of associative memory neural nets. Finally, random mutation follows the genetic model. With the help of diagrams, we would like to offer additional explanation of the modes defined above.

5.3. Simulation

The diagram submitted below exemplifies the functioning of the machine. Important here is not the level of detail, but the structural suggestion.

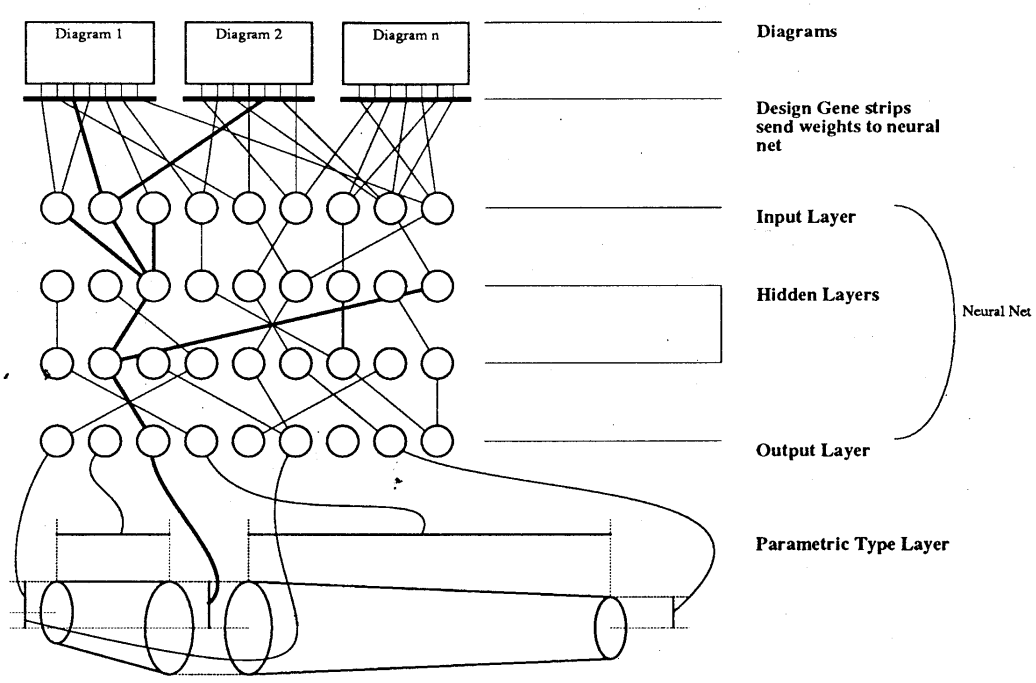


Fig. 7.

As a global representation, this figure is meant to suggest how diagrams, strips, IPs implemented as neural nets, and types together generate designs. Details concerning the genetic strips, the way diagrams are applied as inputs into the IPS, and how types are generated can be noticed.

Finally, the manner in which the constitution of IPS as neural networks is supported is explained by the following discussion.

We have concentrated on the implementational platform upon which IPs will act. The actual implementation of IPs follows rather closely from the neural network model, in that activities such as specialization and delegation can be accomplished simply by training separate neural modules to make particular associations. Particular design team interactions (modeled as input vector values) directly require particular connections ('knows' links and 'needs' links, for instance, modeled as the resulting neural output values). However, many problems remain.

The weighted inputs and outputs and the non-linearity, as well as the network topology involving feedback and the possibility to employ various rules by which weights are adjusted (i.e. by which learning, self-organization and self-adaptation occur) require further clarification. We are evaluating which specific models of networks are more appropriate for the type of intelligence characteristic of design. We are also evaluating the resulting characteristics of the IPs network during training, which we see as equivalent to a transfer of experience. And we know that the massively parallel model of the IPs network in the Design Machine presupposes algorithms different from those that we were initially prepared to implement. As of this writing, we have not come up with reasonable answers to such questions.

References

1. Anderson, J.A. and Rosenfield, E. *Neurocomputing: Foundations of Research.* London/Cambridge MA: MIT Press, 1988.
2. Bush, Vanevar, As We May Think, *Atlantic Monthly,* no. 176, pp. 101-108, 1945.

3. Brown D.C. and Chandrasekaran, B. “Expert Systems for a class of mechanical design activity,” *Knowledge Engineering in Computer-Aided Design, Proceedings of the IFIP WG 5.2 Working Conference 1984 (Budapest),* Gero J.S., (Ed.). Amsterdam: North Holland, pp. 259-290.
4. Dawkins Richard, *The Blind Watchmaker. Why the evidence of evolution reveals a universe without design. New York:* W.W. Norton & Co., 1987.
5. Feldman, J.A., Fanty, M.A., and Goddard, N.H. Computing with Structured Neural Networks,” *Computer,* vol. 21, no. 3, 1988, pp. 91-103.
6. Mitchell, J.W., Ligett, R.S., and Kvan, T. *The Art of Computer Graphics Programming.* New York: Van Nostrand Rinehold, 1987.
7. Nadin, M. and Novak, M. MIND: A design machine. Conceptual framework, *Intelligent CAD Systems I, Theoretical and Methodological Aspects (* P.J. W. ten Hagen and T. Tomiyama, Eds.). Berlin: Springer Verlag, 1987, pp. 146-170.
8. Peirce, Charles Sanders. *Manuscript letter to J.M. Hantz.* Watertown WI: Northwestern University, 29 March 1987.