ELSEVIER

Computer Integrated Manufacturing Systems Vol. 11, No. 4, p. 267–283, 1998

Printed in Great Britain
0951-5240/98/$ — see front matter

PII: S0951-5240(98)00024-X

An object oriented approach to design with modules

Peter O'Grady and Wen-Yau Liang

Department of Industrial Engineering, University of Iowa, Iowa City, IA 52242, USA

The modular design of products leads to a large number of different products by creating distinct combinations of modules and components. This can give each product distinctive functionality, features, and performance levels. The design of modular products is of considerable importance in enabling companies to respond rapidly to changes in the market environment. This paper is concerned with the area of design with modules (DwM), which involves selecting the module combination to best satisfy the given set of requirements. The aim of this paper is to develop an approach for DwM, to meet customer requirements, using modules that come from suppliers that may be geographically separated and on differing computer platforms. An object-oriented approach for DwM, termed object-oriented design with modules (OODwM), is proposed where modules are represented as objects. The proposed OODwM approach is described and the approach is illustrated with an example involving the design of personal computers using the Internet as an implementation environment. The exchangeability inherent in OODwM is shown by exchanging the original selection object in the example implementation with a selection object that includes new constrained evolutionary algorithms.

The use of an object-oriented approach for DwM offers several important potential advantages in that the model developed is readily computable, in the reusability of objects, and in the exchangeability of objects with similar interfaces.

The main contributions of this paper are fourfold. First, an object-oriented approach to DwM is described. Second, a formalism for DwM using this object-oriented approach is presented. Third, the use of this formalism is illustrated with an Internet-based implementation showing how the formalism can be used for a specific problem, and how objects can be readily exchanged. Fourth, new constrained evolutionary algorithms are presented, together with some initial testing.

Symbols and notation

$\{\}$	set
$[]$	subset
$[]_f$	feasible subset
$< >$	order set
$\{\}_u$	unsatisfied set
$\{\}_s$	satisfied set
C_i	constraints
k	generation/iteration
z	location of design object
$o^l_{i,j}(z)$	design object i with attribute j from location z selected at design model l
$M^l_{i,j}(z)$	design module i with attribute j from location z selected at design model l
n	number of design objects in S^k_l
m	number of design models in population
$R_{o,p}$	functional requirement o with attribute p
S	design model S
New S (S')	new design model S
V^k_l	objective function, where $V^k_l = f(S^k_l)$.
ξ	object operator: inheritance
ψ	object operator: import
ω	object operator: message passing
O	object: physical design objects
A	object: design algorithms
R	object: requirements
C	object: constraints
E	object: evaluation schema
P	object: selection

Introduction

The need for industry to deliver high quality products at low cost has long been recognized. What is also becoming clear is that the further requirements of high variety and rapid product development are gradually being superimposed on these older requirements, so that, for example:

> 'The complex product markets of the twenty-first century will demand the ability to quickly and globally deliver a high variety of customized products.'
> Earl Hall quoted in Davidow and Malone[1].

One promising way for companies to address this difficult set of requirements is to use a modular approach, where products are built using standard modules. This modular approach promises the benefits of high volume production (that arises from producing standard modules) while, at the same time, being able to produce a high variety of products that are customized for individual customers. Such modular design is starting to become a focus of attention and has been stated as a goal of good design practice[2].

A modular product design is one in which the input and output relationships between components, that is, the component interfaces, in a product have been fully specified and standardized. The modular design of products may lead to a large number of different products by creating distinct combinations of modules and components, thereby providing each product with distinctive functionality, features, and performance levels.

Modularity in a product design permits the processes of developing components for that design to be partitioned into tasks[3] that can be performed autonomously and concurrently[4]. The distinctive nature of modular product design can perhaps best be explained by contrasting it to more 'conventional' product design as shown in *Table 1*[4].

Modular products refer to products, assemblies and components that fulfil various functions through the combination of distinct modules[5]. Modules refer to components whose functional, spatial, and other interface characteristics fall within the range of variations allowed by the specified standardized interfaces of a modular product design. The mixing and matching of modules in a modular product design can potentially generate a large number of different products, each consisting of distinct combinations of components that give each product distinctive functionalities, features, and/or performance levels[6–8]. Thus modular product design can lead to an important form of strategic flexibility[9], that is, flexible product designs that allow a company to respond to changing markets and technologies by rapidly and inexpensively creating product variants derived from different combinations of existing or new modules.

The potential benefits of modularity include[10,5,11]:

(a) economies of scale

Since each module will usually be produced in relatively high quantities, natural economies of scale arise.

(b) increased feasibility of product/component change

Since each module;s interface is strictly specified, changes can be made to a module independently of other modules, provided the interfaces remain within specification.

(c) increased product variety

The use of modules mean that a high product variety can be achieved using different combinations of modules.

(d) reduced order lead-time

Since modules are manufactured in relatively high volume, production can be organized to reduce manufacturing lead time. Hence the order lead time can be reduced.

(e) decoupling tasks

Since the interfaces of the modules have been standardized, these interfaces enable design tasks and productive tasks to be decoupled. This decoupling can result in reduced task

Table 1 Comparison between conventional and modular product design

	Definition	Design	Development
Conventional product design	Attributes of an 'optimal' product are determined by market research.	Desired functionality is decomposed into components, but component interfaces are not specified in detail.	Component development and product design co-evolve in a reiterative process. Product architecture is defined by the final product design, that is, as the output of the development process.
Modular product design	Product is conceived as a platform for leveraging product variations and improved models.The modular product architecture fully specifies component interfaces and constrains subsequent component development		Component development processes are concurrent, autonomous, and distributed. Product architecture is defined at the design stage and does not change during development.

complexity and in the ability to complete tasks in parallel.

(f) the ease of product upgrade, maintenance, repair, and disposal

Since a product is decomposed into modules, only certain modules need to be replaced when a repair is made. For the same reason, upgrades, maintenance, and disposal become simpler.

The use of standardized component interfaces means defining the functional, spatial, and other relationships between components that will remain constant during, and perhaps beyond, the product development process[4]. It has been argued that specifying standardized component interfaces in a modular product design creates an information structure that can coordinate the overall development process[4].

Modular product design can therefore imply a new model for managing information flow and knowledge in product development[12,13]. By contrast to the evolving information structures characteristic of the sequential problem solving models for managing product development, a modular product design creates a complete information structure by means of the specified module interfaces that, in essence, define the desired outputs of the module development tasks.

Product variations based on mixing and matching modules are now appearing in product markets as diverse as aircraft, automobiles, consumer electronics, household appliances, personal computers, software, test instruments, and power tools[14,7,15,9].

For example, Swatch produces hundreds of different watch models, but can achieve this variety at relatively low cost by assembling the variants from different combinations of standard modules[16]. A large number of different hands, faces, and wristband can be combined with a relatively small selection of movements and cases to create seemingly endless product combinations.

Another example of design with modules is the Nippondenso panel meter[17], where six standard modules have been established. A small variety of each module is sufficient to support 288 different models, of which about 40 are currently being produced.

The main research issues associated with modular products can be divided into those associated with the identification of modules, the design of modules, and design with modules. This paper is concerned with the latter area of design with modules which involves selecting the module combination to best satisfy the given set of requirements.

Important research issues in the area of design with modules (DwM) center around the central theme of selecting modules to satisfy customer requirements. The need to allow modular product design for environments where design is carried out at geographically separate locations, and on differing computer platforms, adds further complications to the research issues.

However, while modular product design would appear to be an attractive proposition, little work has been done on these research issues[5,18,19], particularly for the relatively common design environment where modules come from geographically separate locations and differing computing platforms are used by those involved.

This paper aims to address the resulting research issue:

'How can DwM be carried out to meet customer requirements using modules that come from suppliers that may be geographically separated and where suppliers operate on differing computer platforms?'

The difficulties associated with DwM are compounded when the geographical separation of personnel is taken into account, and this geographic dispersion of personnel is increasingly the case in many industrial concerns. Some research has been done to facilitate remote collaboration. For example, FLECSE (Flexible Environment for Collaborative Software Engineering) is a multimedia environment designed to facilitate the communication of two or more geographically dispersed software engineers[20]. Unfortunately, there is no work reported, as yet, that examines the collaboration with suppliers in general, or using the Internet in particular.

The central problem of DwM can be stated as:

Given a set of candidate modules $\{M_i,.(z)\}$, at design iteration k, produce a design S_1^k that is composed of a subset of the candidate modules and which satisfies both a set of functional requirements $\{R_{o_i}.\}$ and a set of constraints $\{C_i\}$ while minimizing (maximizing) an objective function V_1^k, where $V_1^k = f(S_1^k)$.

A promising approach to the research issue above is that of object-oriented design:

'Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notion for depicting both logical and physical as well as static and dynamic models of the system under design'[21].

The emphasis of the object-oriented design is decomposition and representation. Decomposition is the breakdown of the functions of a product so that the lowest level of the function structure consists exclusively of functions that cannot be sub-divided further while remaining generally applicable. Decomposition differentiates object-oriented design from conventional structured design, while representation is concerned with defining an object and organizing objects.

Work done on object-oriented design systems includes that by Kusiak[22], Takeda et al.[23], and

Dopplick[24]. While the potential advantages of object-oriented design has been recognized[25,26] the previous object-oriented design systems appear to have been built for the most part in an ad-hoc manner for a specific design domain and are not based on a formal methodology. Therefore, the results are difficult to extend to other design domains, such as DwM.

The work presented in this paper is focused on how to represent DwM, where the participants may be remote, and how to implement it in a formal and systematic way. A representation formalism is proposed for that purpose. This paper first overviews the DwM process. A formalism for object-oriented DwM (OODwM) is then presented. This formalism uses an object-oriented approach, where modules are represented as objects. Given this formalism, the question remains of how can this be implemented? Certainly, it can be implemented on a single processor with a single user. However, such an isolationist approach, ignores the geographic distribution of personnel. To address these aspects, an Internet-network based implementation of this formalism is described that uses the Internet to support OODwM. An example of personal computer design is used to illustrate the implementation.

An object oriented formalism for DWM

Functional requirements

Design is often defined as the creation of synthesized solutions in the form of products, processes or systems that satisfy perceived needs through mapping between functional requirements (FRs) in the functional domain and the design parameters (DPs) in the physical domain, through the proper selection of DPs that satisfy FRs[27], i.e.,

$$[FR] = [A].[DP]$$

where $[A]$ is the design matrix. The relationship between requirements and functions is discussed by Kota and Ward[28], and Kusiak and Szczerbicki[29].

In theory, it is possible to decompose the functions of a product so that the lowest level of the function structure consists exclusively of functions that cannot be sub-divided further while remaining generally applicable. The relationship between sub-functions and overall function is often governed by certain constraints, in as much as some sub-functions have to be satisfied before others. The meaningful and compatible combination of sub-functions into overall function produces a functional structure[5].

The functional structure may be represented in the form of a network where the blocks correspond to the sub-functions and the flow connects the blocks (see *Figure 1*). The functional structure can be decomposed into multiple levels to separate the overall function into multiple sub-functions.

The breakdown of functions into sub-functions is one of the first steps in design with modules. The remainder of this section is devoted to the development of an approach for design with modules.

An object oriented approach

The proposed Object-Oriented DwM (OODwM) approach views design objects as being objects that represent not only physical entities, such as modules or components, but also non-physical entities, such as the design history or vendor information. Each design object can have the following features:

- Methods: which are procedures that act on data or on design objects.
- Data (or property): which is a named attribute of a design object. Data (or property) describes design object characteristics such as the size, the weight,

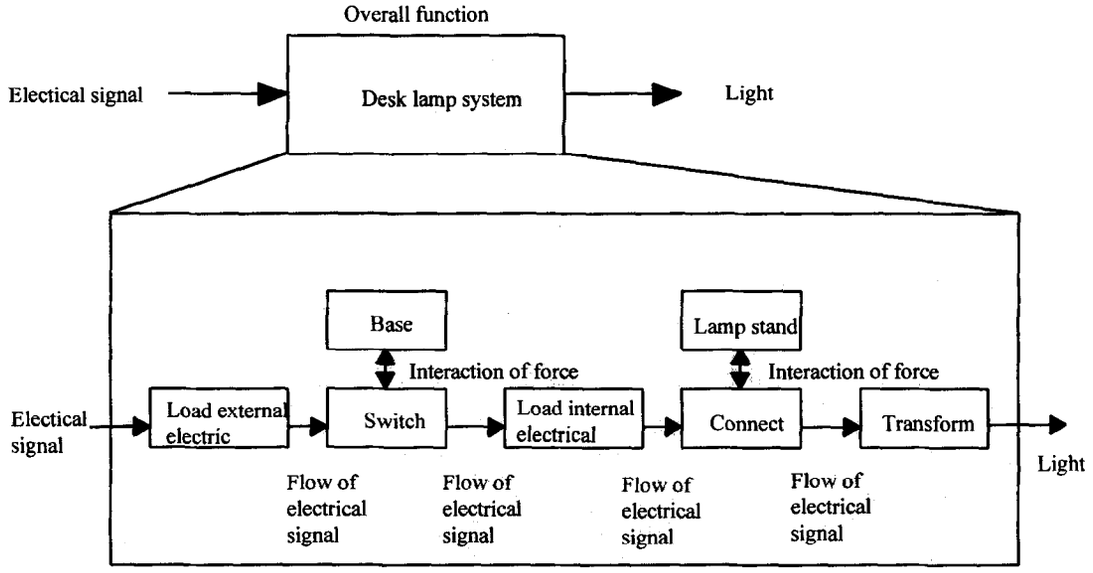

Figure 1 Functional structure of a desk lamp.

the state of a design object, or the state of progress of the design model.
- Interface (or structure): this defines the basic structure and interface of a design object.

Given the above view of a design object, the relationship between design objects can be established by using the following object operators:

- Inheritance (ξ): which creates an instance B from a design object A. The instance B then can be thought of another design object which inherits the basic structure and methods of A but which can be altered and which can contain its own data.
- Import (ψ): this allows design object B to import methods from another design object A. The design object B can now use the methods which are imported from design object A in a similar manner to A.
- Message passing (ω): this allows data to be passed from one design object B to another design object A. This could involve the data or methods associated with A and/or B.

These operators are shown diagrammatically in *Figure 2*.

The use of design objects in OODwM has several potential advantages:

1. Computability: design can be thought of a computable design process model. The implication for DwM is that a design process model obtained using OODwM is not just a descriptive model but a computable model, which computes new stages in the design process using the object operators described above.
2. Reusability: once a design object in OODwM has been established, it can be used repeatedly.
3. Exchangeability: because the design object in OODwM is modular in concept, it can be replaced by another design object, provided the interfaces of the two objects are similar. This would allow, for example, for the relatively straightforward upgrading of portions of a computerized design system.

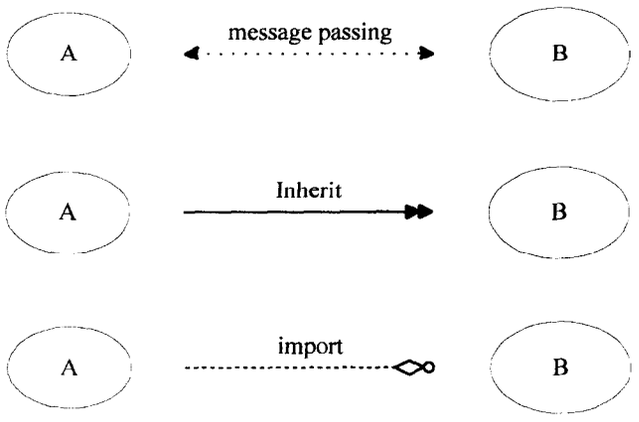

Figure 2 Object operators.

While the above has described the underlying concepts behind design objects in OODwM, such concepts are of limited use unless they can be incorporated into a design process model. Such an OODwM design process model would have the potential advantages of design objects, described above, but would also possess some other potential advantages:

1. Improved communication: a design process model can potentially reduce the effort of translating thoughts from one design to another, and therefore reduce misunderstandings as an idea passes from one person to the next. This could result in an increase in productivity and quality.
2. Modeling the design process: human cognition often recognizing things first, and the functions that connect them afterward. Since our minds are naturally practised at thinking about things in the everyday world, why not parlay that practice into design activity? This would let the design jobs easier.

The design model S

A design model (S) is the representation of an artifact to be designed. For an OODwM process then:

$$S = [o_i,.(z)]$$

S represents a model that consists of one or more design objects. These design objects can be physical or non-physical objects.

The exact form of S will vary with the design process model and with the artifact being designed. For example, for evolutionary design where the design goes through a number of generations and where there may be several design models at each generation, then

$$S_1^k = [o_i^l,.(z)]^k = (o_1^l,.(z), o_2^l,.(z), o_3^l,.(z) \ldots o_n^l,.(z))^k$$

where S_1^k represents the design model l at generation k, and $o_1^l,.(z)$ represents a design object from location z included in design model l at generation k, while n represents the number of design objects in the design model. Note that all design objects are not necessary come from the same location z.

OODwM methods

An OODwM method is a mechanism for changing or creating one or more design objects. In order to apply a OODwM method, both the design object(s) to be manipulated and the model to which the method is applied should be specified. We can define two basic OODwM methods namely add and delete, and these are defined as follows:

Definition (add method Φ_{add}). An add design object OODwM method on a model S is a unit operator

which adds exactly one design object $o_{i_1}.(z)$ to the design model S.

$\Phi_{add}(o_{i_1}.(z)):S \rightarrow S'$ where $S' = S + o_{i_1}.(z)$

Definition (delete method Φ_{del}). A delete design object OODwM method on a model S is a unit operator which deletes exactly one design object $o_{i_1}.(z)$ from S

$\Phi_{del}(o_{i_1}.(z)):S \rightarrow S'$ where $S' = S - o_{i_1}.(z)$

It is readily apparent that any change in the design model S can be obtained by applying a finite number of these two basic OODwM methods to S.

The flow of design operations in the proposed OODwM approach is, in essence, represented in terms of OODwM methods. A combination of OODwM methods is an ordered set of OODwM methods that are successively applied to the model.

The overall architecture of OODwM

The central design process inherent in OODwM can be represented as the architecture, shown diagrammatically in *Figure 3*, with five main types of objects involved: namely design models (S), design objects (O), design algorithms (A), requirements and constraints (R and C), and the evaluation schema (E). Object operators can express the relationship between these objects as follows:

Relation 1: instance of design model $S_1^k \xi$ design model S
Relation 2: design algorithm A ω design model S
Relation 3: design algorithm A ψ evaluation schema E
Relation 4: instance of design object $o_{i(2)}$ ξ design object O
Relation 5: design model S_1^k ω $o_i(z)$
Relation 6: requirements and constraints R and C ω design algorithm A
Relation 7: design algorithm A ω $o_i(z)$
Relation 8: design algorithm A ω design model S_1^k

The architecture in *Figure 3* shows how the particular instance of a design model, S_1^k, is obtained from the design algorithm, evaluation schema, requirements, constraints and the design model object.

For pure formulation design (or creative design) a new design model object S is defined that describes the form of the model. A specific instance, S_1^k, of this design model can then be created. For pure parametric design then the design model object S has already been defined and the design process therefore only involves the determination of a specific instance, S_1^k, of this design model.

An important point to note is that additional objects can be defined within the overall architecture shown in *Figure 3*. For example, in the example below, the design algorithm object, A, includes a selection object, P.

In particular, a design object, o_i, can be part of a tree structure with sub-objects, o_i^- and super-objects o_i^+. Modules can be thought of as objects o_i so that:

$o_{.,.} \equiv M_{.,.}$

The design object and module relationship is shown in *Figure 4*. Each instance of a design object and module, and each sub-object in the tree can have the OODwM architecture, shown in *Figure 3*, associated with it. In this manner there is a hierarchy of instances of design objects and modules with the OODwM architecture potentially operating at each level of the hierarchy.

OODwM design process model

The above has described OODwM in terms of the basis of design objects, design object methods and the architecture of OODwM. From this basis it can then

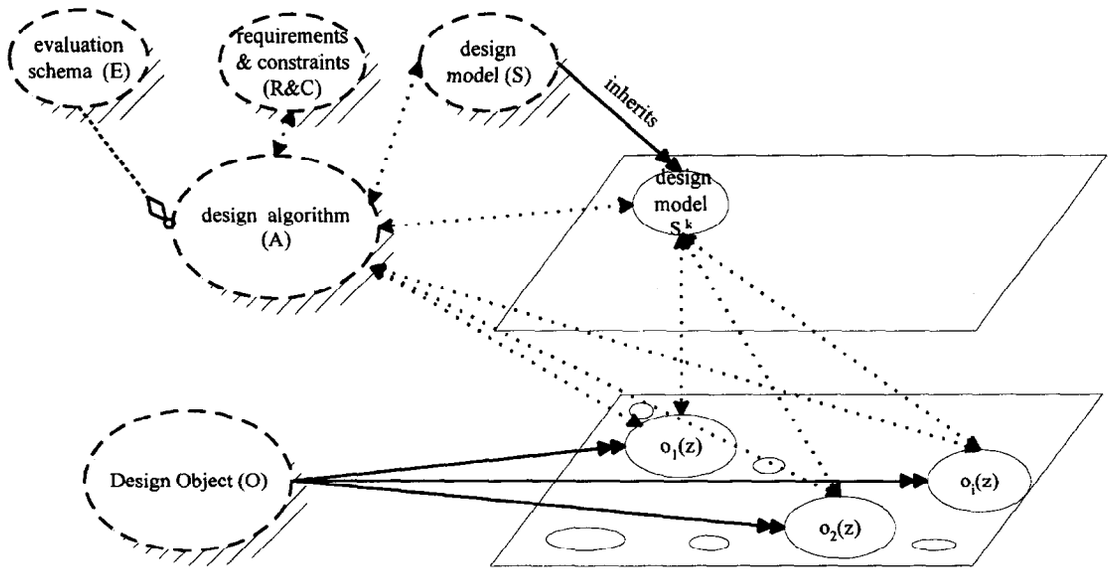

Figure 3 The overall architecture of OODwM.

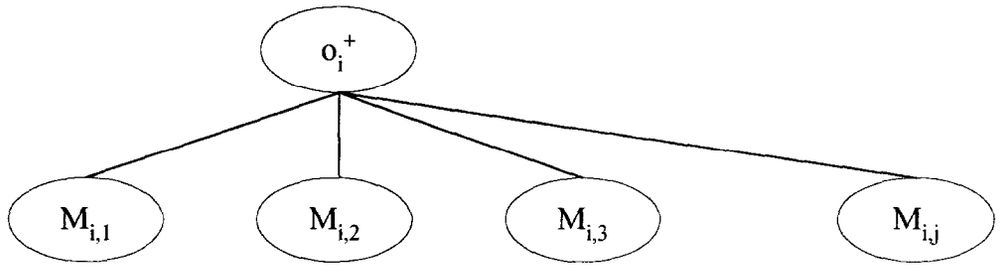

Figure 4 Hierarchy of design object and module instances.

be possible to formulate different design process models. By doing this, the inference is that the resulting design process models would have the advantages described above, namely: computability, reusability, exchangeability, and improved communication.

With the design objects (with relations) and design methods defined, the OODwM design process model can be thought of a computable model. The implication of this is that the design process model is not just a descriptive model but a computable model that computes new stages in the design process using the design methods described above.

The central problem of OODwM can be stated, as in the Introduction, as:

> Given a set of candidate modules $\{M_i,.(z)\}$, at design iteration k, produce a design S_1^k that is composed of a subset of the candidate modules and which satisfies both a set of functional requirements $\{R_o,.\}$ and a set of constraints $\{C_i\}$ while minimizing (maximizing) an objective function V_1^k, where $V_1^k = f(S_1^k)$.

For this work, the design model S_1^k which in turn is made up of a sub-set of modules $[M_i,.(z)]$. The problem then becomes one of selecting a suitable sub-set of modules $[M_i,.(z)]$.

This section describes one instance of a OODwM design process model which consists of four steps (with associated sub-steps). Note that in the following description, a design modules $M_{i,j}^1(z)$ is regarded as a design object $o_{i,j}^1(z)$. In essence, it is a sub-object of $o_{i,j}^{1+}(z)$, similar to the hierarchy shown in *Figure 4*.

Step 1 (refinement). Refine the set of initial design specification into an equivalent set of functional requirement $\{R_{o,p}\}$ such that each $R_o,.$ is ready to be mapped into a subset of objects. $R_{o,p}$ is functional requirement o with attribute p. This step is described by Liang and O'Grady[30,31].

Step 2 (design initialization)

- $A \ \psi \ E$ (design algorithm A imports evaluation schema E)
- $A \ \omega \ R$ and C (design algorithm A gets data from requirements and constraints R and C)

The design algorithm, A, used in this example is described below.

The design model, S, for this example consists of a set of design modules $M_{i,j}^1(z)$.

- $S_1^k \ \xi \ S$ (instance of design model S_1^k inherits from design model S)
- $M_{i,j}^1(z) \ \xi \ O$ (instance of design modules $M_{i,j}^1(z)$ inherits from design object O)

The design object, O, consists of a description of the artifact.

$\{R_o,.\}_u = \{R_o,.\}$
$\{R_o,.\}_s = \text{Null}$

*Step 3 (selection object **P**).* In this step, Object P, containing the selection algorithm, is invoked. A variety of selection procedures are possible. Below is one particular selection procedure encapsulated in selection object P_1. Object P_1 involves selecting a functional requirement and then selecting a design object that satisfies the requirement. When all functional requirements have been satisfied then the main design stage has been completed.

Step 3.1 If the set of unsatisfied requirement $\{R_o,.\}_u$ is empty, stop since the design is finished. Otherwise, arbitrarily select one requirement $(R_o,.)$ from $\{R_o,.\}_u$.

Step 3.2 Find a set of design modules $\{M_{i,j}^1(z)\}$ which satisfies the selected functional requirement $(R_o,.)$ with each attributes. The selection is done by an attribute matching process in a 1:1 relationship where attribute j in design modules $M_{i,j}^1(z)$ satisfies attribute p in requirement $R_{o,p}$.

Step 3.3 If the set of design modules $\{M_{i,j}^1(z)\}$ is empty, stop. In this case there is no design modules that will satisfy the selected functional requirement $(R_o,.)$, and the design process can only proceed if either the selected functional requirement is relaxed or if alternative design modules can be made available.

Step 3.4 Ranks set of design modules $\{M_{i,j}^1(z)\}$ into an ordered set $< M_{i,j}^1(z) >$, according to the evaluation scheme E, such that the $M_{i,j}^1(z)$, with higher ranking comes first. This ranking is based on the objective function V_1^k, where $V_1^k = f(S_1^k)$. Select the highest ranked $M_{i,j}^1(z)$. Using add object methods (Φ_{add}), add the selected design modules into the current instance of the design model

Table 2 Sample constraint feasibility calculations[32]

Number of problem variables (X)	Feasible percentage of each problem variable (Y)	Probability of generating feasible solution	Expected number of trials to get one feasible solution
10	0.99	0.9044	1.10
	0.75	0.0563	17.76
	0.50	0.0010	1,024
50	0.99	0.6050	1.65
	0.75	5.70×10^{-7}	$1.76 \times 10^{+6}$
	0.50	8.88×10^{-16}	$1.12 \times 10^{+15}$
100	0.99	0.3660	2.73
	0.75	3.21×10^{-13}	$3.11 \times 10^{+12}$
	0.50	7.89×10^{-3}	$11.27 \times 10^{+30}$

(S_1^k). This results in a new design model (New S_1^k).

Step 3.5 Checks that none of the set of constraints $\{C_i\}$ are violated when the method is applied. This procedure is called validity-preservation.

Step 3.6 If the model violates the constraints, undo Step 3.4 and delete $M_{i,j}^1(z)$ from the ordered set of design objects $<M_{i,j}^1(z)>$. Then repeat from Step 3.4 and choose $M_{i,j}^1(z)$ with the next rank.

Step 3.7 Delete the functional requirement $(R_o,.)$ from the set of functional requirements $\{R_o,.\}_u$. Go to 3.1.

Evolutionary search and OODwM

The above has described a design process model using OODwM. Steps 1–3 above constitutes the design algorithm A that underlies the design process model. The selection algorithm object P selects the modules to be included in the design. It is therefore a crucial element in achieving a good design. In the example design process model above, object P_1 is used which ranks set of design modules $\{M_{i,j}^1(z)\}$ into an ordered set $<M_{i,j}^1(z)>$ and then selects the module with the highest ranking. The result of this process is that the instance of the design model obtained will be one that does not violate the set of constraints $\{C_i\}$, but the instance of the design model obtained may not have a high ranking based on V_1^k. This section describes the use of an evolutionary search mechanism to identify higher ranked designs.

Preserving feasibility during operations

A blind search and selection of modules may not guarantee that the resulting design model, S, will satisfy both a set of functional requirements $\{R_o,.\}$ and a set of constraints $\{C_i\}$ as required in the problem considered. This may not be of great concern in problems that are only loosely constrained and where any infeasible designs that are generated can be eliminated(using, for example, the backtracking approach in the example design process model above). However design problems tend to be heavily constrained[32] and there is a consequence that a high percentage of the resulting designs produced may be infeasible. When the number of decision variables is X and the percentage of values in the domain which are feasible, Y, is the same for all variables, then the probability of randomly generating a feasible solution is YX. *Table 2*[32] shows the effects of different values of X and Y.

It can be seen that for even relatively small design problems ($X = 50$) with fairly loose constraints ($Y = 0.75$) then the probability of randomly generating a feasible solution is very small (5.70×10^{-7}). The repercussion of this is that for such problems, generating solutions without considering their feasibility will lead to a high degree of inefficiency. There is therefore a need to devise algorithms that generate feasible solutions.

Evolutionary approaches under constrained conditions

Much of the work in evolutionary approaches has as its basis the work on Genetic Algorithms. The Genetic Algorithm operates on a population $P(k)$ of solutions rather than a single solution[33,34]. The operation of the standard Genetic Algorithm is shown in *Figure 5*.

The crossover function aims to achieve genetic diversity in the population by exchanging some genetic material in the relevant population members while the mutation function aims to introduce an element of randomness.

Despite its promise, a quite serious limitation of the Genetic Algorithm is that it is intended primarily for unconstrained search. However, as indicated above, the DwM problem is replete with constraints that pertain to such aspects as cost, delivery lead time, module compatibility, and reliability. Modules have to be selected therefore, not only on the basis of satisfying requirements and minimizing (maximizing) an objective function, but also of satisfying these constraints. The presence of these constraints can severely reduce the probability of a randomly selected design being feasible[32] and, consequently, a mechanism for handling such constraints is needed. A number of techniques have been proposed for handling constraints in the Genetic Algorithm.

One such technique is to penalize infeasible solutions when evaluating each member of the population. An example of a well known design system which uses penalty functions is Engineous[35].

```
Begin
        set generation k = 0
        Initialization: Initialize population, P(k)
        Evaluation: Evaluate each population member of P(k) according to some
                measurement criterion
        Do the following
                Selection: Select part of P(k)
                Crossover: Crossover (usually) each in turn two members of the
                        selected part of P(k) to form a new population member
                Set k = k+ 1
                Mutation: Mutate at random the new population members P(k+1)
                Evaluation: Evaluate P(k+1)
        Until Termination Condition. This can be set for a variety of conditions.
End
```

Figure 5 Operation of the standard genetic algorithm.

Constraints in Engineous have a goal, actions, a weight, and conditions[36]. While penalty functions are useful when all variables are measured in the same unit, extreme care must be taken to avoid problems in scaling when applying penalty functions[37].

Perhaps, the primary difficulty with penalty functions is that slightly infeasible solutions which otherwise would be of high quality can contain a great deal of useful 'genetic' material. With high penalties, these solutions and their genetic material are lost. With low penalties, the population may become filled with infeasible solutions. This problem has led to a number of penalty functions which increase with time[33,14].

For order-based problems such as scheduling or routing, a different approach has been taken for handling constraints. Rather than using penalties (or in addition to them), the problem is transformed into a format such that specially designed genetic operators do not violate the constraints. Upon completion, the solutions are transformed back into their original structure[38].

A third basic technique for handling constraints is to use backtracking whenever constraints are violated. During initialization, crossover, or mutation, any attribute values that cause constraint violations are retracted and new values are assigned. The cycle repeats itself until a feasible solution is obtained. This is the basic approach used in the Genetic State Space Search (GSSS)[39]. The GSSS is the first method to use local constraint propagation and backtracking in the Genetic Algorithm as a means of preserving feasibility.

One limitation of relying solely on local constraint propagation, however, is that inequality and database variables cannot be supported. Since local constraint propagation cannot handle these types of constraints, there is significant potential for violations on heavily constrained problems. This leads to a great deal of backtracking which is computationally time consuming.

When the problem of concern to this paper is considered then the potential presence of large numbers of database constraints means that the approaches described above are of limited applicability. A proposed approach to handling problems of this type is described below.

Proposed evolutionary methods

In addition to the two basic object methods defined above, we can define two extended methods that address evolutionary aspects. These are the Crossover Method and the Mutation Method. Each extended method is composed of the successive application of the basic methods add and delete.

Crossover method. As indicated above, the crossover operation aims to achieve genetic diversity in the population by exchanging some genetic material in the relevant population members. In regard to this work, then the main aim is to achieve this genetic diversity while, at the same time, maintaining feasibility in the resulting design model, S.

Without this need to maintain feasibility, and the underlying constraints that impact the issue of feasibility, then we can define an unconstrained crossover method as follows: Definition (Unconstrained Crossover Method)
This is a module method that is applied to two design models S_1, S_2, which exchange module subsets at an arbitrary crossover point j.

$$\Phi_{cro}(j,S_1,S_2):S_1 \rightarrow S_1' \text{ where } S_1' = S_1 - \Sigma_{i=j}^{n}\Phi_{del}(M_i^1,.(z)) + (\Sigma_{i=j}^{n}\Phi_{add}(M_i^1,.(z)), \text{ where } M_i^1,.(z) \text{ is from } S_2)$$

Also

$$\Phi_{cro}(j,S_1,S_2):S_2 \rightarrow S_2' \text{ where } S_2' = S_2 - \Sigma_{i=j}^{n}\Phi_{del}(M_i^1,.(z)) + (\Sigma_{i=j}^{n}\Phi_{add}(M_i^1,.(z)), \text{ where } M_i^1,.(z) \text{ is from } S_1)$$

Note that the add method here is a little different from the original object method, here we indicate the source of modules. The Unconstrained Crossover Method, as defined here, is a single point crossover method but other crossover methods can be readily defined. The main problem with unconstrained crossover methods, such as that defined above, is that they

will tend to produce a high percentage of infeasible designs under a moderately constrained design environment (see *Table 2*), since there is no mechanism to preserve feasibility. They are consequently of very limited usefulness in DwM problems.

In order to produce feasible designs a new Constrained Crossover Method is proposed. The operation of this Constrained Crossover Method is shown in *Figure 6*.

In this proposed constrained crossover method, a crossover point is selected and modules are exchanged between S_1^k and S_2^k one at a time, and the validity is checked after each module is exchanged (Step 4). If the result is infeasible then the exchange is cancelled (Step 4) and another module is selected. In this manner the final result will be a population of design models that are feasible.

Mutation method. As indicated above, the mutation operation aims to introduce some element of randomness into the population. For an unconstrained design environment then an Unconstrained Mutation Method can be defined as follows:

Definition (Unconstrained Mutation Method)
A mutation module method is applied on a design model S, to delete exactly one module at certain point and add one module from the feasible subset of modules, $[M_j^l.(z)]_f$

$\Phi_{mut}(j,S):S \rightarrow S'$ where $S' = S - \Phi_{del}(M_j^l.(z))$

$+ (\Phi_{add}(M_j^l.(z))$, where $M_j^l.(z)$ is from $[M_j^l.(z)]_f)$

The arguments expounded for the Unconstrained Crossover Method above also apply to the Unconstrained Mutation Method, namely that in the absence of any mechanism for ensuring that the resulting design models are feasible, in moderately constrained design environments, a high percentage of the resulting design models will be infeasible (*Table 2*).

To help overcome this issue of feasibility, we propose a Constrained Mutation Method as shown in *Figure 7*.

In this proposed constrained mutation method, a design model S_1^k and mutation point j are selected (Step 1). The selected module is then replaced by a member of the feasible subset of modules $[M_j^l.(z)]_f$ (Step 2) and the validity of the resulting new design model is checked (Step 3). If it is invalid then the replacement is cancelled and a new design model S_1^k and mutation point j are selected.

Example implementation of OODwM—personal computer design

The OODwM design process model described in Section 3.6 can now be applied to an example product design (Personal Computers (PC)). Compared to other computer products (e.g., a supercomputer or a minicomputer), a PC typically has a simpler operating system, a smaller amount of

Begin
At generation k, with a population of design models S_1^k, ... S_m^k, then:
Step1: Randomly selects two individuals designs S_1^k, S_2^k from the populations of designs and randomly select a crossover point j
Step2: Start from crossover point j
Step3: Use the basic module methods to obtain a new design model $S_1^{k'}$
Where $S_1^{k'} = S_1^k - \Phi_{del}(M^l_{j,.}(z)) + (\Phi_{add}(M^l_{j,.}(z))$, where $M^l_{j,.}(z)$ is from S_2^k)
Step4: Check the validity of $S_1^{k'}$. If $S_1^{k'}$ is not valid, undo step 3 and add 1 to j. Go to step3
Step5: If $S_1^{k'}$ is valid, add 1 to j. If j<n+1, then go to step 3
Step6: $S_1^{k'}$ is a new design model
Step7: repeat Step1 to Step6 to get $S_2^{k'}$.
End

Figure 6 Proposed constrained crossover method.

Begin
At generation k, with a population of design models S_1^k, ... S_m^k, then:
Step1: Randomly select one design model S_1^k and mutation point j
Step2: Obtain a new design model $S_1^{k'}$ where $S_1^{k'} = S_1^k - \Phi_{del}(M^l_{j,.}(z)) + (\Phi_{add}(M^l_{j,.}(z))$, where $M^l_{j,.}(z)$ is from $[M^l_{j,.}(z)]_f$)
Step3: check the validity of $S_1^{k'}$, if $S_1^{k'}$ is valid, then stop. Otherwise, undo step 2 and go to step 1
End

Figure 7 Proposed constrained mutation method.

memory and primary storage space, limited input and output devices, and is a single user system[40]. However, the rapid change in the number of software products and their inherent incompatibility problems, as well as platform-specific requirements, has added to the complexity of PC configuration, so that a modern PC can have literally billions of possible configurations, but with only a much smaller number of feasible configurations[40].

The PC configuration includes a motherboard with a CPU mounted, e.g., Pentium II 266 MHz, and other objects such as RAM, monitor, and hard disk. Where a limited set of each of these is available with little overlap, then this can be a relatively simple design task since each of these elements is modular in nature. This most straightforward design task has been implemented as such by several PC manufacturers on their web sites for potential customers to use. However the complexity rapidly rises where elements may be obtained from different suppliers with both differing costs and differing reliabilities. Under these circumstances, design can involve selecting the best combination of elements to best meet overall requirements with the more complex tradeoffs that result.

The architecture of example OODwM PC design system

The overall architecture of the Example OODwM PC design system, shown in *Figure 8*, follows that of the OODwM architecture shown in *Figure 3*. For the PC design system the objects are distributed over the Internet so that data and objects can be in different locations. The implementation consists of a web server with links to databases servers and clients that communicate with the web server over the Internet. This particular implementation was carried out using a testbed in the Iowa Internet Laboratory (IIL). The IIL has a number of web server and client platforms including Netscape Fasttrack servers and Microsoft Internet Information Servers, each of which run on either Windows NT and 95 operating systems. For this example system the users can access the system through a conventional web browser.

The requirements (R) are of the form:

{processor, hard drive capacity, monitor size, memory}

so that a user would need to specify values for each of these attributes. For example the attribute 'memory' may be set to 'at least 32 MB'.
The set of constraints {C} are

{budget, overall reliability}

and, as above, values can be set by the user for each attribute.

The design Model, S, includes the following design objects:

[mother board, hard drive, monitor, RAM]

Specific attributes of each design object are:

Mother board	{processor, pin type, reliability, cost...}
Hard drive	{capacity, reliability, cost...}
Monitor	{size, reliability, cost, manufacturer...}
RAM	{memory, pin type, reliability, cost...}

An instance of this design model (S_1^k) inherits this structure from S, so that each specific instance may have different values of each of the design object attributes. The I/O constraints, e.g., the compatibility of RAM pins, are checked by the object methods during the search for feasible solutions.

The evaluation schema (E) is one of lowest cost. The design algorithm, A, involves another object (the selection object, P) as described in a previous section.

The set of design modules $\{M_{i,}.\}$ includes data on design objects and this is contained in vendor databases that may be scattered over the Internet.

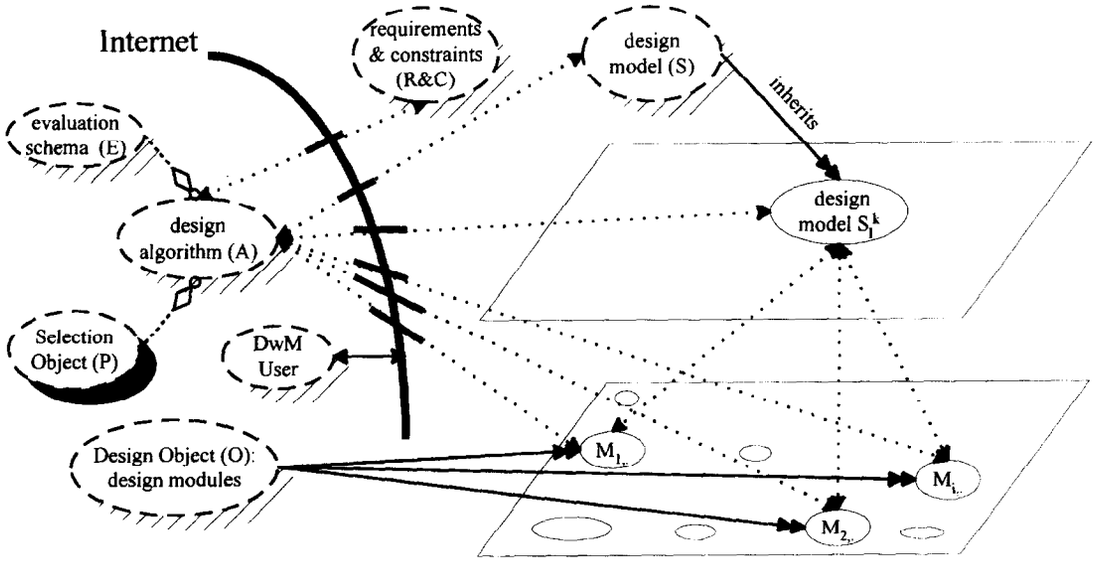

Figure 8 The architecture of OODwM PC design system.

Example operation—personal computer design

In this section, an example implementation of the OODwM is implemented using the Internet to result in a prototype system for OODwM where suppliers may geographically dispersed. Let us consider a situation where constraint C, represented by {budget, overall reliability} has attribute values arbitrarily selected as {\$2,000, 0.985}.

Step 1 (refinement). The initial design specifications are refined into functional requirements. For this scenario, let us assume that the functional requirements are as follows:

$$\{R_{o,p}\} = \{R_{1,1}, R_{2,1}, R_{3,1}, R_{4,1}\}$$

where

$R_{1,1}$: Motherboard with attribute 'processor spec.' 233 MHz or faster
$R_{2,1}$: Hard disks with attribute 'capacity' at least 8 GB
$R_{3,1}$: Monitor with attribute 'screen size' at least 15 in
$R_{4,1}$: RAM with attribute 'memory' at least 48 MB

Note that in this scenario each functional requirement has only one attribute. This can be extended readily to many attributes (for example, the monitor can have additional attributes such as 'pitch', 'maximum resolution', etc.). The entry of these requirements is done by the OODwM user using a web browser as shown in *Figure 9*.

Step 2 (design initialization)

$A \psi E$ (design algorithm A imports evaluation schema E)
$A \omega R, C$ (design algorithm A gets data from requirements and constraints R and C)
$S_1^k \xi S$ (instance of design model S_1^k inherits from design model S)
$M_i,.(z) \xi O$ (instance of design module $M_i,.(z)$ inherits from design object O)

$$\{R_o,.\}_u = \{R_1,.,R_2,.,R_3,.,R_4,.\}$$

$$\{R_o,.\}_s = \text{Null}$$

Data is contained in databases that are scattered over the network, each data item being at location z.

At this stage, the design model is represented as follows.

$$S_1^k = [M_i^1,.(z)]^k \text{ and } \{M_i^1,.(z)\}S_1^k = \text{null}$$

Step 3 (selection object P_1)
Design Session 1

Step 3.1 $\{R_i\}_u$ is not empty; arbitrarily select R_1,. which is to have a motherboard with attribute 'processor spec.' 233 MHz or faster

Step 3.2 The feasible motherboards are therefore those with attribute 'processor spec.' 233 MHz or faster and a portion of the feasible subset is shown in Table 3 and includes {AE2, P6, P3, S5, S6 or S4}. Slower motherboards are eliminated from consideration.

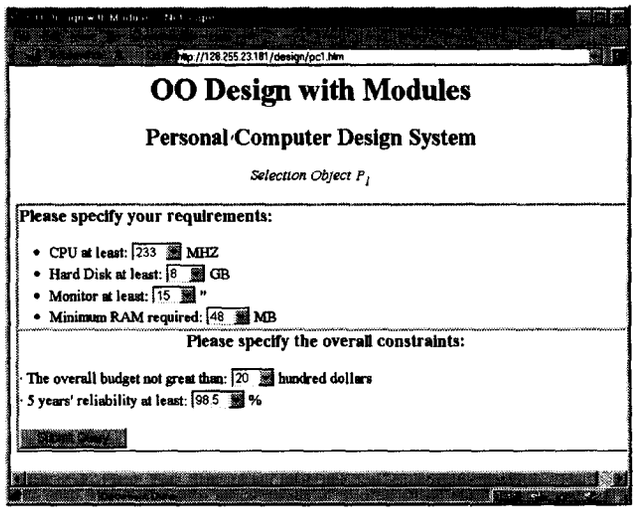

Figure 9 OODwM user interface.

Session 2: Table

Motherboard	Hard Drive	Monitor	RAM	Cost
233	8	0	0	575

Session 3: Table

Motherboard	Hard Drive	Monitor	RAM	Cost
233	8	17	0	1165

Session 4: Table

Motherboard	Hard Drive	Monitor	RAM	Cost
233	8	17	128	1624

Figure 10 The results produced by the OODwM algorithm.

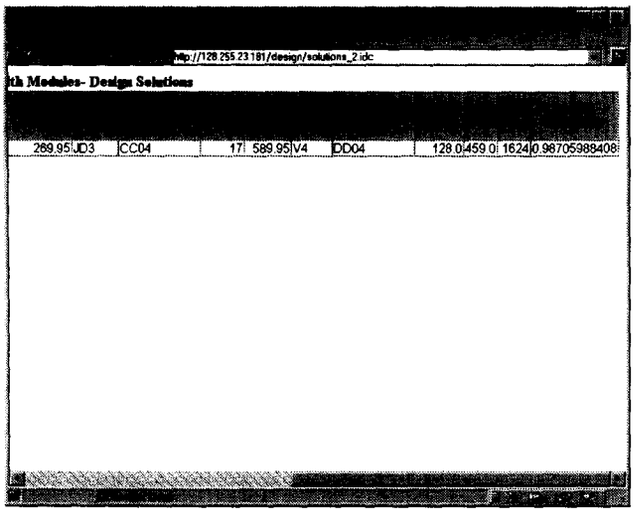

Figure 11 Part of the output screen of OODwM using P_1.

Step 3.3 Continue.

Step 3.4 Using the add method, the new model is $S_2^k = S_1^k + \{\text{AE2, P6, P3, S5, S6 or S4}\}$ module. The modules are ranked by cost. The first rank (i.e. lowest cost) is P3, a Pentium 233 motherboard, with a cost of \$285.

Step 3.5 Evaluate the set of overall constraints. The cost of the first rank must be smaller than the total budget \$2000. There is no constraint violation. Check the constraints:

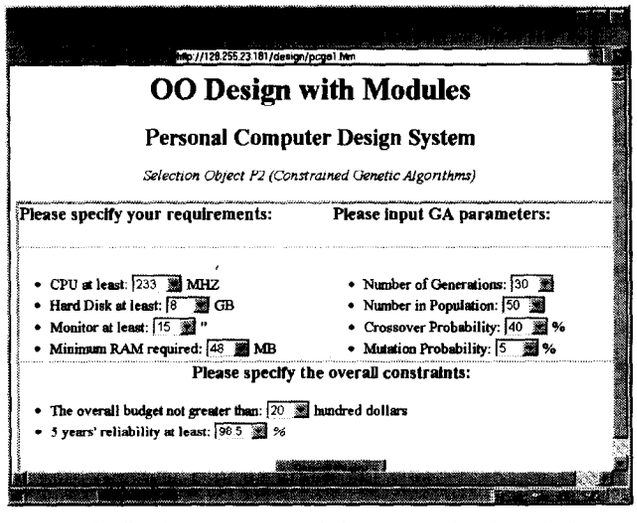

Figure 12 The input screen of the constrained evolutionary OODwM system.

The reliability of the P3 motherboard is 0.965, which is less than 0.985, so there is a violation.

Step 3.6 Undo Step 3.4 and delete P3 from the ordered set. Then Repeat from Step 3.4 and choose $M_{i,j}^l(z)$ with the next rank.

Step 3.4 With the add method, the new model is $S_2^k = S_1^k + \{\text{AE2, P6, S5, S6 or S4}\}$ module. The function evaluation ranks the functions based on cost. The lowest cost is the P6 motherboard with a cost of \$305.

Step 3.5 Evaluate the overall function, the cost of the first rank must be smaller than the total budget of \$2000. No violation occurs. Check the reliability constraints: The reliability of the P6 is 0.994, which is greater than 0.985, therefore there is no violation.

Step 3.6 $\{R_o,.\}_u$ and $\{R_o,.\}_s$ are updated. $R_1,.$ is deleted from $\{R_o,.\}_u$ and added to $\{R_o,.\}_s + \{R_o,.\}_s$ (new) $= \{R_o,.\}_s$ (old). $\{R_1,.\} = \{R_1,.,\}$ and $\{R_o,.\}_u$ (new) $= \{R_o,.\}_u$ (old) $- \{R_1,.\} = \{R_2,., R_3,., R_4,.\}$. At this point, only $R_1,.,$ is satisfied.

Design Sessions 2, 3 and 4

The process above can be followed for Design Sessions 2, 3 and 4 with the results at the end of each session shown in *Figure 10*. Note the incremental manner in which the design is built up.

Design Session 5

Step 3.1 $\{R_o,.\}_u$ is empty; go to Step 4.

At this point a feasible design has been selected

Step 4 (detailed design)

When the example OODwM design process model described above is used, the result, for the particular data sets used, is shown in *Figure 11*. A single instance of the design model (S_1^k) is obtained that satisfies the set of constraints $\{C_i\}$ namely an overall reliability of at least 0.985 (0.987 is obtained), with a maximum cost of \$2,000 (a cost of \$1,624 is obtained). P_1 is a simple selection procedure and has been included to show the operation of OODwM. However much more complex selection procedures are possible to be readily incorporated into OODwM as shown in the next section.

Exchangeability: selection object P_2

Perhaps the main potential advantage of the OODwM approach is the reusability and exchange-

Table 4 Portion of the initialization results for the constrained evolutionary OODwM system

Motherboard model	Processor spec.	Hard drive	Capacity (GB)	Monitor model	Screen size (in)	RAM model	Memory (MB)	Cost	Reliability
AE2	300	AD2	16	MS2	15	MV2	128	2444	0.9850
AE2	300	AD0	8	MS4	20	MV2	128	2964	0.9850
AE2	300	KO1	8	V1	17	DP5	512	3144	0.9850
P2	333	ST3	8	PS2	21	V4	128	3009	0.9850
P2	333	AD0	8	171	17	MV2	128	2009	0.9850
P2	333	ST5	8	MG1	17	MV2	128	2298	0.9850

ability of an object, namely one or more objects can be easily replaced provided the interfaces remain within the same specification. To examine this aspect, in this section the original selection object P_1 in the above example is replaced with another selection object P_2.

The example above uses selection object P_1 which simply selects the design module with the lowest cost even though there may be a better design that involves some trade off between reliability and cost. In addition P_1 uses a simple backtracking approach to handling constraints, whereby when a constraint is violated then the decision that led to the violation is reversed. As indicated above, this is potentially very inefficient in even moderately constrained environments.

As an alternative to P_1, we can use a search mechanism, specifically the constrained genetic algorithm search mechanism, described above, which we can include in a new selection object P_2. The nature of OODwM allows for the exchangeability of objects so that we can simply exchange selection object P_2 for selection object P_1. This section describe the elements of selection object P_2 and the resulting design process model.

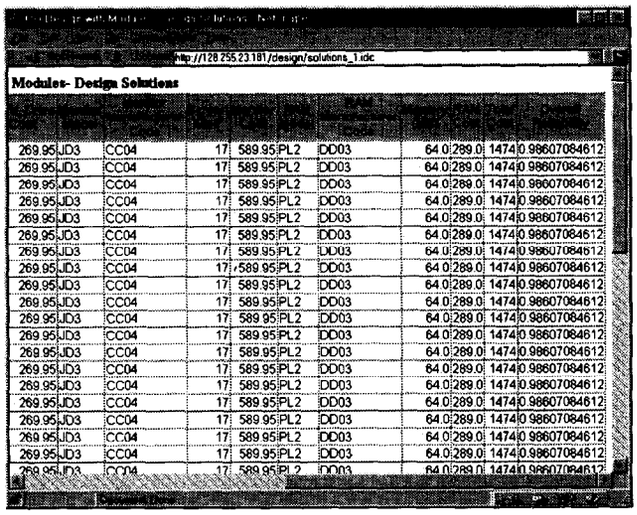

Figure 13 Part of the resulting population from P_2.

Constrained evolutionary OODwM—selection object P_2

This section describes a new selection object P_2 which can be used to replace P_1 for Step 3 during the design process model described in Section 3.6. To avoid confusion with P_1, P_2 sub-steps will be indicated by (P_2).

Step 3.1 (P_2) initialization. A variety of selection schemes are possible, including random and user invoked schemes. The one presented here is based on P_1, with changes to ensure a random selection of modules.

(a) If $\{R_o,.\}_u$ = Null, then set $\{R_o,.\}_u = \{R_o,.\}$ and $\{R_o,.\}_s$ = Null. Add 1 to population number
(b) If population size = m + 1 (m is from user input), then go to Step 3.2 (P_2)
(c) Arbitrarily select one requirement $(R_o,.)$ from $\{R_o,.\}_u$. Find a set of design modules $\{M^1_{i,j}(z)\}$ which satisfies the selected functional requirement $(R_o,.)$ with each attributes. The selection can be done by an attribute matching process in a 1:1 relationship where attribute j in design modules $M^1_{i,j}(z)$ satisfies attribute p in requirement $R_{o,p}$.
(d) If the set of design modules $\{M^1_{i,j}(z)\}$ is empty, stop. In this case there are no design modules that will satisfy the selected functional requirement $(R_o,.)$, and the design process can only proceed if either the selected functional requirement is relaxed or if alternative design modules can be made available.
(e) Randomly select one $M^1_{i,j}(z)$ from $\{M^1_{i,j}(z)\}$. Using add object methods (Φ_{add}), add the selected design modules into the current instance of the design model (S^k_1). This results in a new design model (New S^k_1).

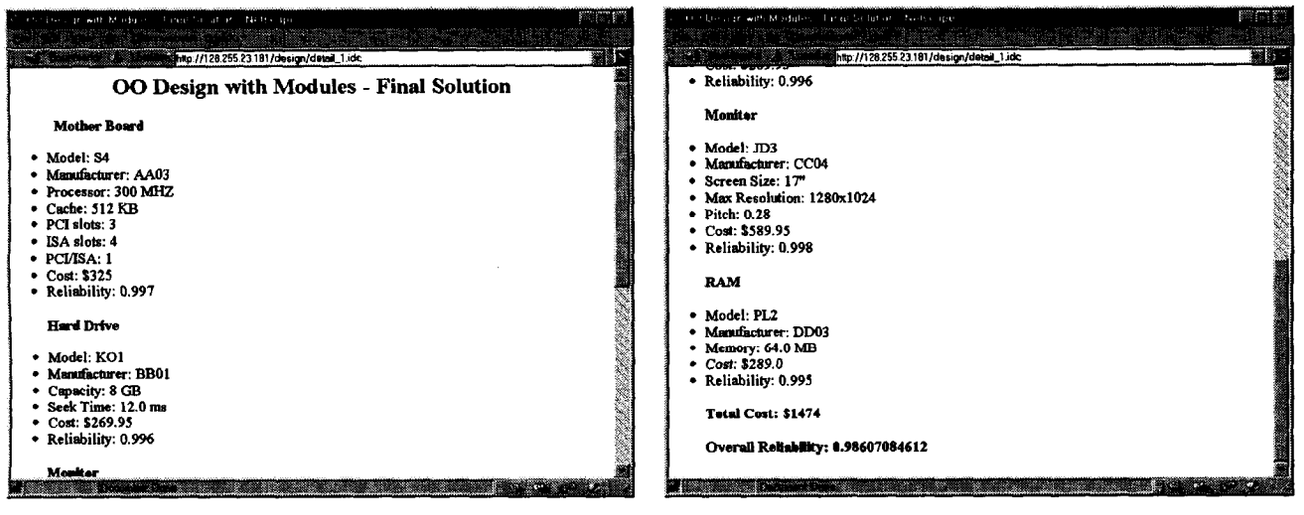

Figure 14 The final solution obtained by OODwM System.

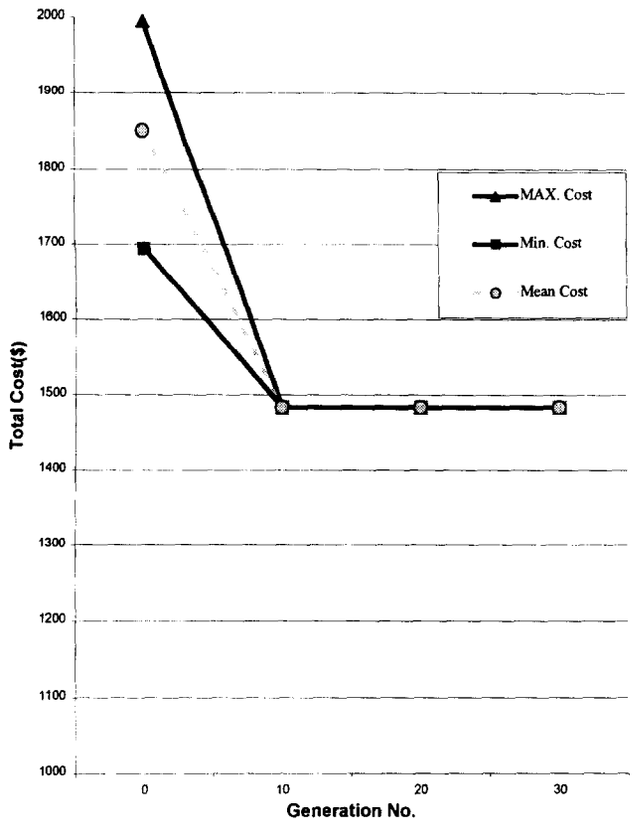

Figure 15 The overall cost vs. generation number (Set 1).

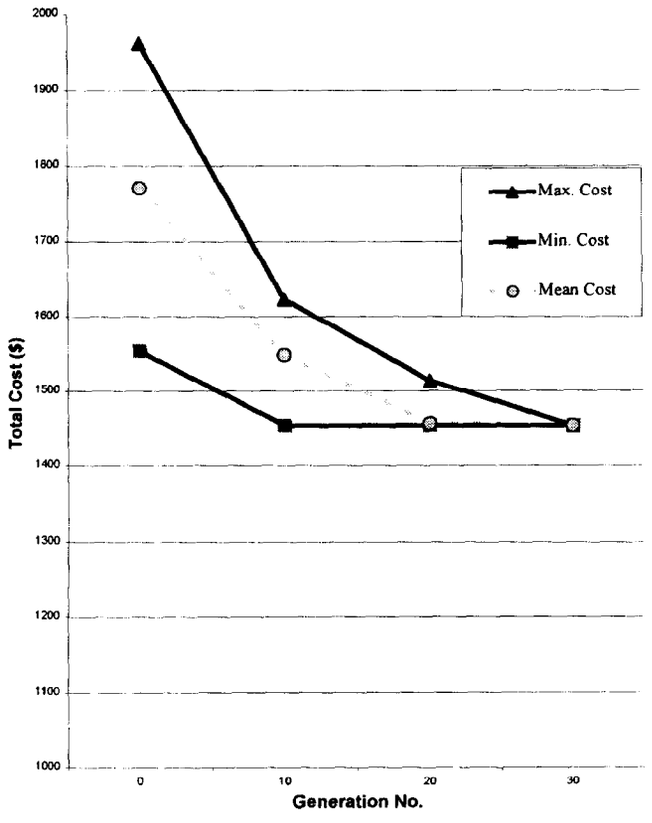

Figure 16 The overall cost vs. generation number (Set 2).

(f) Checks that none of the set of constraints $\{C_i\}$ are violated when the method is applied. This procedure is called validity-preservation.
(g) If the model violates the constraints, undo 5 and delete $M^1_{i,j}(z)$ from $\{M^1_{i,j}(z)\}$. Then Repeat from 5 and choose another $M^1_{i,j}(z)$. Otherwise go to 8.
(h) Delete the functional requirement $(R_o,.)$ from the set of functional requirements $\{R_o,.\}_u$. Go to Step 3.1 (P_2).

Step 3.2 (P_2) selection

(a) Add 1 to the generation number. If the generation number is greater than k, stop.
(b) Evaluate each design model using the objective function V^k_1, where $V^k_1 = f(S^k_1)$.
(c) Use an appropriate selection strategy to select the new population.

Step 3.3 (P_2) crossover. Use the Constrained Crossover Method (*Figure 6*) to the design models which are selected for crossover.

Step 3.4 (P_2) mutation. Use the Constrained Mutation Method (*Figure 7*) to the design models which are selected for mutation. Go to Step 3.2 (P_2).

Example illustration

The revised input screen is shown in *Figure 12* where the various parameters required for selection object P_2 have been added.

As the design process outlined above is invoked with selection object P_2 and its constrained evolution algorithms then the initialization step [Step 3.1 (P_2)] aims to determine a population size of m, if that is possible. In this case the user has specified that the population size will be set to 50 (*Figure 12*). The result is 50 initial design models and a portion of these models is shown in *Table 4*.

These can then pass into the selection, crossover and mutation stages [Steps 3.2–3.4 (P_2)] for a total of 30 generations as specified by the user (*Figure 12*). Note that in this example the selection strategy is based on the total cost. The resulting total cost is lower for the PC system with higher probability being selected.

The resulting final population is shown to user via a web browser as shown in *Figure 13*. (Note that in this case the population has converged to a single design solution) and the user has the option of examining further details of each solution, using a web browser, as shown in *Figure 14*.

The result of using this search mechanism, shown in *Figure 13* and *Figure 14*, indicate that a family of design models (S^k_1) are obtained (albeit consisting of a single member in this case). The improvement that selection object P_2 gives over selection object P_1 is

Table 5 The input parameters of the constrained evolutionary OODwM system

	Processor spec.	Hard drive capacity	Monitor size	RAM memory	Population number	Crossover probability	Mutation probability	Budget	Minimum reliability
Set 1	233	8	15	48	20	40%	5%	2000	0.985
Set 2	233	8	15	48	20	40%	5%	2000	0.980

shown in that the minimum cost solution obtained is \$1474 with a reliability of 0.986 (*Figure 14*) compared to a cost of \$1624 with a reliability of 0.987 using selection object P_1. However these figures are illustrative only: the main principle is that of exchangeability in that the selection object P_1 has been readily exchanged with selection object P_2.

Convergence of the constrained evolutionary algorithms

The performance of evolutionary approaches is notoriously difficult to evaluate but we can gain some indication of such aspects as the convergence of the algorithm for specimen problems. The convergence of the constrained evolutionary OODwM algorithm is shown in *Figures 15* and *16* for the two specimen user input data sets shown in *Table 5*. The main differences in the data sets are in the required system reliability.

The results shown in *Figures 15* and *16* indicate that there is convergence with the mean cost reducing from \$1852 and \$1771 to \$1484 and \$1454 respectively. The resulting cost is higher for the PC system with higher reliability.

Summary and conclusions

Modular design is becoming increasingly important and raises some important research issues. This paper has addressed a central research issue in the area of DwM, namely to determine a method for a structured search for feasible designs, where the DwM problem is being addressed using modules that come from sources that may be geographically separated, where data is kept in remote databases and where differing computer platforms may be used. The use of OODwM for DwM offers several important advantages in computability, in that a design process model obtained using OODwM is not just a descriptive model but a computable model, in reusability, in that once a design object in OODwM has been established, it can be used repeatedly, and in exchangeability, in that objects can be readily exchanged with other objects with similar interfaces. The constrained evolutionary DwM algorithm includes new crossover and mutation algorithms were presented. The implementation of the approach in an example problem domain area has been described. This domain is that of PC design where modules are searched using the constrained evolutionary approach over the Internet, thereby allowing for remote DwM. The specimen results show good convergence but further work needs to be done in testing the proposed constrained evolutionary DwM algorithm for stability and convergence.

The contribution of this paper is that it presents an approach for design with modules that can operate on a computer network to allow for remote systems to be included. Such an approach, once determined, can be relatively easily computerized. This approach can lay the groundwork for further research in the area of object-oriented design with modules.

References

1 **Davidow, W and Malone, M** *The Virtual Corporation*, Harper Collins (1992)
2 **Kidd, K T** *Agile Manufacturing: Forging New Frontiers*, New York: Addison Wesley (1994)
3 **Von Hippel, E** *The Source of Innovation*, Oxford University Press, New York (1988)
4 **Sanchez, R** 'Strategic product creation: Managing new interactions of technology, markets, and organizations', *European Management Journal* Vol 14 No 2 (1996) 121–138
5 **Pahl, G and Beitz, W** *Engineering Design*, Springer-Verlag, New York (1988)
6 **Langlois, R N and Robertson, P L** 'Networks and innovation in a modular system: Lessons from the microcomputer and stereo component industries', *Research Policy* Vol 21 No 4 (1992) 297–313
7 **Sanderson, S W and Uzumeri, V** 'Strategies for new product development and renewal: Design-based incrementalism', *Working Paper*, Center for Science and Technology Policy, Rensselaer Polytechnic Institute, Troy, New York (1990)
8 **Ward, W, Liker, J F, Cristiano, J J and Sobek, D K** 'The second Toyota paradox: Delaying decisions can make better cars faster', *Sloan Management Review* Vol 36 No 3 (1995) 43–61
9 **Sanchez, R and Sudharshan, D** 'Real-time market research', *Marketing Intelligent and Planning* Vol 11 No 8 (1993) 29–38
10 **Nevins, J L and Whitney, D E** *Concurrent Design of Products & Processes: A Strategy for the Next Generation in Manufacturing*, McGraw-Hill, New York (1989)
11 **Corbett, J, Dooner, M, Meleka, J and Pym, C** *Design for Manufacturing: Strategies, Principles, and Techniques*, Addison Wesley, New York (1991)
12 **Clark, Kim B and Wheelwright, S C** *Managing New Product and Process Development*, Free Press, New York (1993)
13 **Takeuchi, H. and Nonaka, I** 'The new product development game', *Harvard Business Review* Vol 62 No 1 (1986) 34–39
14 **Morris, C R and Ferguson, C F** 'How architecture wins technology wars', *Harvard Business Review* Vol 71 No 2 (1993) 86–96
15 **Sanchez, R** 'Strategic flexibility, real options, and product-based strategy', *Ph.D. Dissertation*, Massachusetts Institute of Technology, Cambridge, Massachusetts (1991)
16 **Pine, B J, I** *Mass Customization: The new Frontier in Business Competition*, Harvard Business School Press, Boston, Massachusetts (1992)
17 **Aoki, K** 'High speed and flexible automated assembly line—Why has automation successfully advanced in Japan?', *Proceedings of the 4th International Conference on Production Engineering, Japan Society of Precision Engineering, Tokyo* (1980) pp 1–6

18 Shirley, G V 'Models for managing the redesign and manufacture of product sets', *Journal of Manufacturing and Operations Management* Vol 3 No 1 (1990) 85–104

19 Ulrich, K and Tung, K 'Fundamentals of product modularity', *Issues in Design/Manufacture Integration 1991*, Sharon A (Ed.), ASME, New York, DE 39 (1991) pp 73–79

20 Dewan, P and Riedl, J 'Toward Computer-Supported Concurrent Software Engineering', *Computer* Vol 26 No 1 (1993) 12–16

21 Booch, G *Object Oriented Design with Applications*, Benjamin/Cummings (1994)

22 Kusiak, A, Szczerbicki, E and Vujosevic, R 'Intelligent design synthesis: an object-oriented approach', *INT. J. PROD. RES.* Vol 29 No 7 (1991) 1291–1308

23 Takeda H, Tomiyama, T and Yoshikawa, H 'A logical and computable framework for reasoning in design'. In: *Design Theory and Methodology—DTM92*, ASME (1992)

24 Dopplick, T A 'Science Users Guide to the EOSDIS Core System (ECS) Development Process', *Technical Paper* 160-TP-003-001, Science Office, EOSDIS Core System Project (1995)

25 Coad, P and Yourdon, E *Object-Oriented Design*, Yourdon Press, NJ (1991)

26 Liu, C *Smalltalk, Objects, and Design*, Hardbound (1996)

27 Suh, N P *The Principles of Design*, Oxford University Press, Oxford, UK (1990)

28 Kota, S and Ward, A C 'Functions, structures, and constraints in conceptual design'. In: Rinderle I (Ed.), *Proceedings of the 2nd International Conference on Design Theory and Methodology*, ASME Press, New York (1990) pp 239–250

29 Kusiak, A. and Szczerbicki, E 'A formal approach to specifications in conceptual design', *ASME Journal of Mechanical Design* Vol 114 No 12 (1992) 659–666

30 Liang, W Y and O'Grady, P 'Genetic Algorithm for Design for Assembly: The Remote Constrained Genetic Algorithm', *Computers and Industrial Engineering* Vol 33 No 3–4 (1997) 593–596

31 O'Grady, P and Liang, W Y 'Remote Collaborative Design and Manufacture With Modules: Internet-Based Collaboration', *Proceedings of 4th International Conference on Concurrent Enterprising*, 8–10, October 1997, Nottingham, UK, pp 315–326

32 Cormier, D 'A Constraint-Based Genetic Algorithm for Concurrent Engineering', *Ph.D. Thesis*, Dept. Industrial Engineering, North Carolina State University (1995)

33 Michalewiczm, Z *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin (1994)

34 Goldberg, D *Genetic Algorithms In Search, Optimization, and Machine Learning*, Addison Wesley, Reading, MA (1989)

35 Powell, D and Skolnick, M 'Using Genetic Algorithms in Engineering Design Optimization with Non-Linear Constraints', In: Forrest S (Ed.), *ICGA-93*, Morgan Kaufman, San Mateo, CA (1993) pp 424–43

36 Powell, D, Tong, S S and Skolnick, M 'EnGENEous domain independent, machine learning for design optimization'. In: Schaefer, D (Ed.), *Proceedings from the Third International Conference on Genetic Algorithms*, Morgan Kaufman (1989) pp 151–159

37 Richardson, J, Palmer, M, Liepins, G and Hilliard, M 'Some Guidelines for Genetic Algorithms with Penalty Functions'. In: Schaffer D (Ed.), *ICGA-89*, Morgan Kaufman, San Mateo, CA (1989) pp 191–197

38 Tamaki, H, Kita, H, Shimizu, N, Maekawa K and Nishikawa, Y 'A Comparison Study of Genetic Codings for the Traveling Salesman Problem', *Proceedings of the First IEEE Conference On Evolutionary Computation*, IEEE, Piscataway, NJ (1994) pp 1–6

39 Paredis, J 'Genetic State-Space Search for Constrained Optimization Problems', *13th International Joint Conference On Artificial Intelligence*, Morgan Kaufman, San Mateo (1993) pp 967–972

40 Fohn, S M, Liau, J S, Greef, A R, Young, R E and O'Grady, P J 'Configuring computer systems through constraint-based modeling and interactive constraint satisfaction', *Computer in Industry* Vol 27 (1995) 3–21