Engineering with Computers (2011) 27:183–192
DOI 10.1007/s00366-010-0188-4

ORIGINAL ARTICLE

A theoretical framework for an intelligent design catalogue

Paul Winkelman

Received: 5 December 2007 / Accepted: 3 February 2010 / Published online: 25 March 2010

Abstract Product catalogues constitute a valuable source of information for engineers engaged in design activities. Unfortunately, these catalogues provide only limited support to engineers in the earlier, conceptual stages of design. This research proposes the intelligent design catalogue consisting of a virtual design environment linked to catalogues of standard components. Engineers develop their design concepts within the virtual environment and refer to the catalogues as these concepts are refined. The selected components are assembled within the design environment. The intelligent design catalogue provides search aids as well as assessment tools. The theoretical framework draws on several engineering areas. Manufacturing demonstrates how process plans can be developed in a virtual environment independently of the machines on the shop floor just as products can be conceptually designed independently of the standard components available. The standard components themselves can be grouped borrowing from classification schemes of group technology. Object-oriented programming (OOP) provides an environment for the development of the software that runs the intelligent design catalogue. As the objects of OOP parallel standard components, OOP also serves as a design paradigm after which the catalogue can be modelled. Design theory suggests frameworks for developing a (semi-) hierarchical structure for cataloguing parts.

Keywords Standard components · Object-oriented · Virtual design · Catalogue design

P. Winkelman (✉)
Department of Mechanical Engineering, University of British Columbia, 6250 Applied Sciences Lane, Vancouver, BC V6T 1Z4, Canada
e-mail: pwinkel@mech.ubc.ca

1 Introduction

Engineers frequently refer to catalogues when designing products. By carefully selecting standard components, they are able to create their own unique systems that meet their particular needs. The selection process itself is an important aspect of the design activity and considerable effort may be spent locating the appropriate component. The burden of this task has been eased to some extent with the transfer of component data from hard-copy catalogues to computer databases and their supporting search engines. Unfortunately, whether computerized or on a hard copy, product catalogues offer limited assistance to the development of the design ideas themselves that precede the catalogue search. The assistance that is available normally comes in the form of design examples, which demonstrate typical uses of common components. These examples may or may not resemble the design task at hand.

Some limited research has been carried out, which attempts to make the computer a more integral part of catalogue design. Carlson-Skalak et al. [1] describe an evolutionary algorithm. They view catalogue design as a two-step process (1) the design of the configuration using generic parts, and (2) the selection of manufacturers' components. During the first step, the designer selects generic components from a "component type database" to form the initialization set. All input and output ports (through which materials, signals or energy flow) are connected. The second step is carried out by the computer. Manufacturers' components are randomly selected from the catalogues, which satisfy the requirements of the generic components. Due to the vast number of manufacturers' components available, the computer is seen as being particularly well suited to this task. The ports of the selected parts must match in a "positive–negative" fashion. Design refinements are made via various forms of mutation. Designs are evaluated, assigned a fitness score and ranked. The fitness score is determined by summing weighted performance criteria. These weights are supplied by the designer.

One of the major limitations of this system is that the user must carry many tasks to completion before any assistance is offered. For instance, the user must complete the configuration design (initialization set) of generic components in order to initialize the system. Later, to allow the computer-generated configurations to be assessed and listed, the designer must enter weights for all the performance criteria.

The *intelligent design catalogue* proposes an alternative approach to the computerization of catalogue design. Its primary aim is to provide designers with computer support early in the design process, prior to the completion of any particular task. This is accomplished by linking the catalogue database to a virtual design environment. Within this environment, designers are able to develop partial assemblies, in terms of both configuration and specification.

Partial *configuration* refers to an assembly which is missing some of the components necessary to make it a viable system. For instance, when designing a hydraulic system, there may be some valves or fittings not present in the assembly but which must eventually be included if the system is to have any real functionality.

Partial *specification* refers to components within the assembly that are not completely specified. A single component can thus span a range or spectrum of possible specifications. At one end of the spectrum is the fully specified component which, by definition, corresponds to a component in the catalogue database. At the other end is the minimally defined component. We can think of such a component as one which has the minimal specification which allows it to occupy a meaningful position in an assembly. Functional descriptions may fulfill this purpose. The task of the designer is to gradually refine the components of the configuration until all components are fully specified. The refinement process is also supported by computer aids which can be invoked at the discretion of the designer, such as offering suggestions for suitable components. The extent of the support is largely determined by the sophistication of the features which link the design environment with the catalogue database.

Once the selection and assembly process is complete or nearing completion, additional aids are made available to the designer by linking the design environment to existing engineering software systems. A component of sufficient specification would allow a solid modelling program to render the item. Once two or more components are assembled together, the entire assembly and can be rotated as a single unit in three dimensions. An evolutionary algorithm such as that proposed by [1] can be used to find alternatives to the selected components. Other links may allow designers to analyze the created system or run simulation tests.

The proposed features will provide several major benefits. First, it reduces the need for the premature specification of an anticipated component. Second, essential parts that do not exist in the catalogue can still be represented in the design environment. Finally, it presents an important first step in the development of a design environment suitable for novices with limited knowledge of the standard components. Here, the idea is to take knowledge normally residing in the mind of designers and place it within the computerized system.

The challenges that arise in the development of a comprehensive intelligent design catalogue are many. In this paper, the discussion will be limited to outlining some of the underlying concepts that provide the initial inspiration for the research as well as the theoretical framework aimed at developing a suitable computerized system.

2 Objective

The objective of this research is to develop an intelligent design catalogue, which combines a traditional catalogue with a virtual design environment. The proposed environment must allow designers to assemble parts with varying degrees of specification and offer assistance to designers as they refine their designs from concepts to assemblies of fully specified components.

3 Theoretical framework

The initial inspiration for the intelligent design catalogue came from the development of a virtual process planning model in engineering manufacturing. Additional concepts are drawn from the areas of object-oriented programming (OOP) and more general theory in design. These three areas will be discussed individually and then brought together to form a coherent theoretical framework. At this point of development, possible internal mechanisms, such as may be offered by graph theory or relational algebra, are not examined. The intent of postponing their inclusion is to ensure that the logic of the design catalogue framework is not prematurely constrained by the logic of the mechanisms.

For the sake of this discussion, I wish to distinguish between a part and a component. A *part* refers to a generic engineering object, such as a bolt, a motor or a length of pipe. A *component* refers to a part that is fully specified and may be unambiguously identified within a catalogue.

3.1 General design theory

3.1.1 The design process

The engineering design process is typically modelled as passing through several phases or stages. Pahl and Beitz [2], for example, describe the process with respect to four phases. During the clarification of the task, information is gathered, leading to detailed problem specifications. The conceptual phase results in solution variants, establishing functions and principles. During the embodiment phase, form is assigned to the concepts and functions. In the detailed design phase, the required properties of the individual components are analyzed and perhaps optimized.

Based on this model of the design process, the traditional design catalogue only becomes a viable resource during the last, or perhaps the last two, design phases as it is only then that the design begins to take on a physical, more concrete character. Hence, a significant amount of the design process is already carried out before any reference is made to product catalogues. Thus, catalogues make no direct contribution to the early stages of design. If the information contained in catalogues is to impinge on these early stages, designers should be able to quickly access the data even while working on design concepts. There needs to be a link between the concepts and the data or, alternately, between function and form.

3.1.2 Connecting function to form

Gero et al. [3] have attempted to bridge the gap between form (what they call structure) and function, proposing behaviour as the intermediary step or layer. They refer to this as the function–behaviour–structure (FBS) design model. Within this model, function represents the teleological part of design, capturing the designer's intent. Behaviour states how the structure hopes to achieve the desired function. Structure refers not only to the individual components, but how these components are interconnected. The designer starts with a function in mind and, via teleological knowledge, conceives of some desired set of behaviours. Through knowledge of the structural composition of various components and the behaviours associated with these structures, a design is synthesized. Using causal knowledge, the actual behaviour of the synthesized design is determined and then compared to the desired behaviour. Any mismatch leads to refinement.

Gero et al. develop a F–B–S hierarchical network. Some of the features of this network include two layers between behaviour and structure, namely, behaviour variables and structure variables, resulting in five layers in all. Behaviour variables are akin to the performance descriptors or performance variables of the artefact. Structure variables refer to the geometry and material of the artefact. There are also crossovers within the network, meaning that one can arrive at any given node from more than one direction. This is to be expected as a given component can fulfill more than one function. The links within the network are also bidirectional. One can move in one direction from function to structure to select a component. Having selected a component, one can then move in the other direction to determine the implications of that selection.

Gero et al. elaborate on the model using the example of the design of a window. One function is that of providing daylight. The corresponding behaviour is light transmission. A variable associated with this behaviour is the light flux transmitted. The structure consists of glazing and a frame. The structure variables fall into several categories. Examples of structure variables include window length (regular structure variable), glazing length (structure component variable) and glazing area (structure component behaviour variable).

The designer may postpone assignment of values to variables almost indefinitely. When assignment is finally carried out, there is normally a particular order that follows as the implications of the assigned values are propagated throughout the design space. For instance, if glazing area and glazing length are assigned, then glazing width is necessarily defined and the window frame itself is also largely defined.

In a later work, Gero et al. [4] build on the FBS model. They outline eight steps in the design process. Of particular interest are the final three steps. These three steps are all reformulations, dealing with a refinement of the design if the initial design (or design from the previous cycle) is found unsatisfactory. The first reformulation is concerned with changes to the product's structure; the second, with changes to the product's behaviour; the third, with changes to the functions. The model assumes that if the product is found unsatisfactory, designers first assume the shortcomings can be addressed by finding alternatives to the structure. Designers are unlikely to move onto behaviour issues unless structural changes prove insufficient. Functional changes are then the last resort.

The concept of behaviour as an intermediary between function and structure is not without its critics. Dorst and Vermaas [5] see several problems with Gero et al.'s approach. They view Gero et al.'s concept of behaviour as providing a link between intentional descriptions and structural descriptions. Where this shift from intentional to structural descriptors occurs is rather unclear. They also note that certain behaviours do not depend solely on the properties of the window itself. Window behaviour, for instance, is partly attributable to external factors such as wind pressure. They question whether logic is actually sufficient to determine function from a given structure.

Finally, if cost is part of structure, then structure, in addition to function, has intentional features for the cost of an item depends upon, to some extent, what people are willing to pay for it.

Despite the difficulty of the concept, behaviour can provide a suitable building block for the intelligent design catalogue. As with Gero et al.'s network, behaviour can take the form of several layers to facilitate a smoother function-form transition. These layers also facilitate movement in the opposite direction as designers assess the functional qualities of the selected component. Crossovers ensure that this bi-directional movement can cover large areas of the design space, attesting to the fact that a single component can fulfill several functions and a single function can be achieved by several components. This implies that if the intelligent design catalogue is built on a hierarchical structure, this structure must allow for crossovers.

The designer's prerogative to postpone value assignments to parts almost indefinitely, as Gero et al. [3] point out, attests to the importance of working with partial designs. At the same time, the intelligent design catalogue should facilitate the refinement process. Thus, as an individual part within the design environment is refined to a more specific component, these changes automatically invoke a search in the catalogue for suitable components to replace the ill-defined parts remaining in the assembly. Those parts deemed to be suitable replacements can then be presented to the designer.

3.2 Manufacturing

Considerable research has been conducted within manufacturing. Of interest here are those research questions focusing on the arrangement of machines on the shop floor and the integration of process planning with machining operations. Through analogical models, the solutions that address these manufacturing problems also suggest directions for developing the intelligent design catalogue. These models will be drawn from Group Technology and Reconfigurable Systems.

3.2.1 Group technology

Group Technology was developed in the USSR during the 1950s [6] and was further developed over the next two decades in Europe [7]. The idea was to organize manufacturing floors in work cells where similar parts could be machined within a small area. Thus was born the idea of part families.

The machines appropriate for each cell were determined using a matrix, where the rows contain the parts to be manufactured and their operations, and the columns contain the machines that carry out the necessary machining operations. By rearranging the rows, smaller clusters within the matrix are formed and these clusters identify the machines within a given cell as well as the parts to be manufactured there [8]. Well-formed clusters allow all the machining operations for each part to be carried out in a single cell. Typically, some outliers remain and certain parts must travel between cells.

These concepts can be used to develop the intelligent design catalogue by positing the machine on the shop floor as an analogue of a specific component. The operations performed by a machine parallel the functions performed by a component. There are two possible models that follow from this analogue.

The first model views the shop floor as analogous to the design environment. A particular machine is brought onto the shop floor, just as a specified component of the catalogue is brought into the virtual design environment. The manufacturing cell of GT becomes the subassembly within the design environment. The problem GT addresses, using part families, is deciding which machine should be brought to the shop floor and where or in which cell it should be placed. In the case of the design environment, the designer must decide which parts or components to select and on what basis the subassemblies are formed. How might a designer form "part families"?

One possible approach comes from the work of Alexander [9]. Using set theory, he forms groups (design modules) based on design requirements such that the interaction between the groups is minimized. By minimizing the interactions, design changes made to one module have minimal impact on other modules. The design modules parallel the manufacturing cells of GT and the (minimized) interactions between modules correspond to the outliers that must visit more than one manufacturing cell. Although promising for the intelligent design catalogue, Alexander's method, like GT, requires that designers do considerable work and complete several tasks prior to activating the computerized system. Thus, this approach does not facilitate partial designs.

The second model can perhaps suggest a way to ease the design overhead. This model sees the machine shop floor as the catalogue database with the manufacturing cells representing a particular organization of the components within the catalogue. A designer using the intelligent design catalogue to develop a new design is akin to the production of a new part being added to a pre-existing GT shop floor. If the new design maps closely to the particular structure of the catalogue, the design process is greatly simplified. Thus, a designer selecting parts such as a hydraulic pump and some hoses, can be presented with a standard design pattern which includes related parts such as a tank, valves, fittings and hydraulic cylinders.

If the new design does not map closely to the catalogue structure, then the assistance that can be offered to the designer is limited. However, one of the great strengths of the virtual environment, compared to the shop floor of GT, is that the component database need not be restricted to a single classification scheme, for a single component can be classified into many categories. Within the virtual environment, we can very easily relocate the "machine" to another "cell" on the "shop floor", as it were, to accommodate the new "part". The classification schemes themselves can be modelled after a range of standard design practices or developed heuristically from historical data. For example, after extensive use of the virtual system, it may be found that 90% of designers using Component "X" in their assembly also use Component "Y".

Both of these models posit the machine as the component and the machine's operations as the functions of the component. These models also differ in important respects. In the first model, the component is an object within an assembly; in the second, an object within a catalogue. Similarly, in the first model, the function refers to the function of the component within the assembly (actual function); in the second, the function refers to the function as stated in the catalogue (potential or assumed function). These two distinct models represent an important goal of the intelligent design catalogue, namely, a shifting of design knowledge from the designer's head (the shop floor as design environment) to the computerized system (the shop floor as catalogue). In other words, by organizing the components in the catalogue, less knowledge and effort will be required by the designer to organize the components in the design environment.

3.2.2 *Reconfigurable systems*

Oldknow and Yellowley [10] describe a reconfigurable, open-architecture system to control machine tools on a manufacturing shop floor. One of their main goals is to develop a system that can accept hardware (e.g., CNC machines) from any number of vendors (vendor neutrality). The system consists of two sides: a virtual machine tool (hardware independent) and hardware-dependent operations (which describe how a particular machine carries out its tasks). Between these two sides is the binding table, which allocates hardware components to the functions demanded by the virtual environment. Thanks to the binding table, one can design a manufacturing process using the virtual machine tool without any specific knowledge of the machines available. The idea of reconfigurability refers to the ease with which a machine in the system can be removed or added. Their system offers dynamic reconfigurability, which means that reconfiguration can be carried out without necessitating a complete shut-down of the system. Another added feature of the system is that it allows for constraints to be added to the machining process. The machining speed, for example, can be altered if torque limits are exceeded. Thus, two machines can carry out slightly different operations using identical process plans.

The intelligent design catalogue can be modelled after the reconfigurable system where, once again, the machine parallels the component and the shop floor becomes the catalogue. Vendor neutrality means that the catalogue need not be confined to a single source of components. Dynamic reconfiguration allows new components to be added to the catalogue without interrupting catalogue use. The virtual machine tool becomes the virtual design environment where the designer can carry out conceptual and some embodiment design, sometimes unaware of the actual components that are available. As machining operations demanded by the virtual machine tool are assigned to the existing machines via the binding table, design concepts in the virtual design environment can be assigned to components from the catalogue. As one adjusts the machining process to account for process constraints, one can also restrict the available components in the catalogue in keeping with designer-imposed constraints (e.g., cost, weight). The open architecture of the system allows sponsors of the intelligent design catalogue, such as suppliers, to change the components within the database in-house without resorting to outside expertise.

The model just presented is actually composed of two submodels, as there are two distinct selection processes. One process is concerned with the selection of *operations*. This process is carried out by the process planner within the virtual environment. The second process selects the *machines* to perform these operations. This process is carried out by the binding table. These same two processes are contained the GT matrix, where the rows represent the selection of operations, and the columns represent the selection of machines. Although the goals are different, one can draw a parallel between the GT matrix and the binding table.

To a certain degree, these two processes parallel the two-step process of catalogue design outlined by Carlson-Skalak [1], with some additional features. As the process planner assembles operations in the virtual environment independently of the machines available, the designer assembles parts in the virtual design environment independently of the components available. Similar to the binding table, the process of assigning components to the parts can potentially be automated. (However, in support of partial design, designers must have the option of selecting components manually.) The two selection processes demands that the intelligent design catalogue consist of (at least) two databases, one containing the detailed specifications of components (the catalogue itself) and another containing ill-defined parts (and perhaps functions as well). By having two databases rather than one, the designer need not refer to the component catalogue and vendor neutrality can be achieved.

The distinction between these two selection processes can be further demonstrated by the analogies they suggest. In the case of machine selection, the component or part is analogous to the *machine*. However, in the case of operation selection, component or part aligns more closely with the machining *operation*. A process planner selects operations, and places these operations in a particular sequence in the hope of producing a part with the desired features. The mechanical designer selects parts, and places these parts in assemblies in the hope of producing a certain functionality. Thus, the part features of process planning are analogous to the functions of mechanical design, and part families, used to construct the manufacturing cells of GT, become functional families. It is also through this second analogy that design constraints can be aligned with process constraints. Process constraints normally result in making modifications to the operation (rather than a change in machine) just as design constraints components lead to a change of component.

The system proposed by Oldknow and Yellowley [10] can be considered to be a reconfigurable manufacturing system (RMS). Mehrabi et al. [11] and Koren et al. [12] compare RMSs to flexible manufacturing systems (FMSs). FMSs are designed to accommodate a wide range of possible machining processes. Unfortunately, in most FMS installations, the full flexibility offered by the system has gone unrealized as the number of parts manufactured in the long run failed to justify the initial investment [11]. RMS was developed to address this shortcoming of FMS. Although less flexible than FMSs, RMSs do provide a reasonable range of manufacturing capabilities at a reduced cost. RMS is thus an attractive alternative for many manufacturers.

The failure of many FMSs and the rise of RMS can serve as a warning to the intelligent design catalogue. The catalogue, though virtual, need not be comprehensive with every possible part listed. Catalogues may better serve their clients if they are confined in scope. This is particularly true of routine designs where designers tend to restrict themselves to a limited range of fairly predictable components. As the catalogue content is extended to include a wider range of components, navigation becomes more difficult. However, as unique machining operations may justify an FMS, non-routine, more innovative design, requires a more comprehensive catalogue. A comprehensive catalogue is thus akin to an FMS, a limited catalogue to an RMS.

3.3 Object-oriented programming

Object-oriented programming, as the name implies, uses the metaphor of an object to develop a programming language. Hence, OOP focuses on the *what* of programming. By way of contrast, procedural languages attempt to address the question *how* [13].

An object is a conceptual unit, combining both data and methods. A method is defined as a procedure or function that alters the state of an object or causes the object to send a message (i.e., return values) [14]. A well designed object can serve many purposes, and fosters reusability, often cited as one of the main benefits of OOP (some, however, remain skeptical of this claim, e.g., [15]).

Another important feature of OOP is encapsulation. Encapsulation refers to the hiding of information within an object, information not required by other objects. Encapsulation allows changes to be made to the internal workings of the object (implementation) while leaving its interface, where it receives messages from other objects, intact. Encapsulation thus promotes modularity.

Objects receiving the same message need not respond in the same way. This is referred to as polymorphism. The message “addition”, for example, may mean arithmetic addition for one object (addition of numbers), but concatenation to another (addition of words).

Another important feature of OOP is inheritance. Inheritance means that certain objects acquire attributes (but not data) from others. Inheritance necessarily creates various levels of abstraction, leading to a hierarchy consisting of superclasses and subclasses. (An object is not a class itself, but an instance of a class.) According to [13], attributes should be as high up the hierarchy (more abstract) as possible. This ideal is promoted by allowing some of the attributes of the lower objects to override the inherited values, leading to polymorphism.

Typically, an object inherits all of its attributes from a single class (which may inherit from yet another class above it). Multiple inheritance refers to an object or class inheriting from two or more superclasses within the same level of the hierarchy. Multiple inheritance, however, should be used sparingly as it may introduce some ambiguity [16]. For example, if an object inherits from two classes that are both on the same level with methods that bear the same name, which method does the object inherit?

Many of the features of OOP map well to the intelligent design catalogue, for many of the issues OOP tries to address are design problems. Perhaps the most obvious feature of OOP is that of the object, analogous to a fully specified component. The close conceptual coupling of the object to the standard component suggests that OOP is a good choice for programming the virtual environment of the intelligent design catalogue. The programming environment of OOP is the design environment and the program, as an assembly of objects, is the final assembled engineering product.

Reusability, an important feature of OOP, maps well to the intelligent design catalogue as a single component is expected to be part of many different assemblies. Components can be classified as belonging to part families, the classes of OOP. Increasing the level of abstraction leads to a functional description of the part. Inheritance within classes and objects describes how a single function can be fulfilled by any one of a number of components. Unlike OOP, the intelligent design catalogue must make use of multiple inheritance if it is to foster any kind of creative design. Within design, multiple inheritance refers to a single component fulfilling more than one function. A bolt, for instance, can be used as a fastener, or it can be used to position a mechanism.

Encapsulation refers to hiding the internal workings of a standard component. Generally, when purchasing a component, the buyer considers the specifications (attributes) of the component, not its precise internal mechanisms. An important specification is the interface of the component as it determines to which other components it can be connected, just as the interface of an OOP object determines how it communicates with other objects.

As objects have methods inherited from classes, so components have functions inherited from functional or part families. Polymorphism refers to instances where functional attributes within a given family are modified or overridden. Equivalently, mechanical design must take into account design constraints. Constraints may compromise the functional quality of a normally viable component, or render its function invalid. For example, the normal functional attribute of a diesel engine, such as power output, may be compromised in the face of noise limitations. A solid rivet, which normally has the function of fastening, cannot perform that function if access is limited to one side of the assembly. Within an OOP-like class structure of the catalogue, one can place all rivets within a given family (e.g., fastening and deformation) and then provide functional overrides that distinguish between those rivets which require access to both sides and those which do not.

In OOP, objects are organized in object libraries in the same way that components are organized in catalogues. In order for reusability to be realized, programmers must be able to find those objects in the library which meet their needs, just as designers must find components within a catalogue which meets their needs. As the catalogue database uses a hierarchical class structure as an aid to locate a desired part, one would expect that, within OOP, a hierarchical structure can be used to find an object. Hence, the hierarchical structure which defines an object (i.e., which led to its creation) would be the same structure which would allow one to find that same object within a library. Searching is hampered, however, if the logic of the one searching differs from that of the one who created the object, or if the one searching has only partially defined the required object. Hence, regardless of the way an object was created, a flexible organization is necessary to facilitate programmers (or designers) to actually find the object in the library.

Alternatively, the catalogue itself can also be modelled after a single object. In this model, encapsulation refers to the ability of designers to carry out much of their design activities without necessarily being aware of the contents of the catalogue. Access to the catalogue is carried out through the interface, such as through “behaviour”, freeing the designer from the need of a thorough understanding of the catalogue contents. Polymorphism is enacted when designers run programs to assess their designs. The actual analysis program called will depend on the type of assembly.

4 Linking of manufacturing, OOP and design

Concepts from design, manufacturing and OOP have much to offer the development of an intelligent design catalogue. A more coherent picture can be created by simultaneously aligning the equivalent features (analogues) of each of these three areas. As presented, however, these three are not conducive to this end and they will be re-divided into the four categories of machining systems, process planning, OOP and the intelligent design catalogue itself. A separate category for the general design theory would be redundant as the concepts essentially match those of the catalogue category.

The analogous features of all four categories are summarized below and compiled in Table 1. The alignment of the analogues is by no means perfect and some licence has been taken for the sake of greater completeness and as I believe that they provide some insight concerning the development of the intelligent design catalogue. Nevertheless, some incompleteness remains as three of the four categories have missing or inconsequential analogues as indicated by the dashed entries in the table. Two of the analogies previously presented, namely, the shop floor as design environment and the OOP object as catalogue, have been omitted from the discussion and do not appear in the table. This has been done for the sake of simplicity and not as a result of shortcomings of the analogies themselves.

The basic unit of interest (that which is ultimately selected) is the (CNC) machine in machining systems, the machining operation in process planning, the object in OOP and the standard component within the intelligent design catalogue. A machine performs a certain set of

Table 1 Mapping manufacturing and object-oriented programming to the intelligent design catalogue

Machining systems	Process planning	Object-oriented programming	Intelligent design catalogue
Machine	Operation	Object	Component
Operation	Part feature	Method	Function
Shop floor	–	Object library	Catalogue
Manufacturing cell	Part family	Class	Functional family
–	Virtual machining environment	Programming environment	Virtual design environment
–	Process plan	Program	Assembly
–	Sequence of operations	Module	Subassembly
Binding table (matrix)	List of operations	–	Behaviour
Reconfigurability	–	Modularization	Part replacement
–	Process constraints	Polymorphism	Design constraints
–	–	Interface	Interface
–	–	Encapsulation	Internal component mechanism unknown Component/part family unspecified
One operation, many machines	One feature, many operations	Simple inheritance	One function, many parts
One machine, many operations	One operation, many features	Multiple inheritance	One part, many functions

operations, an individual operation produces a certain range of forms, an object has particular methods and a component performs certain functions. A machine is selected from the shop floor, an object from an object library and a component from a catalogue. The machines are grouped to form manufacturing cells, machining operations are grouped according to the part families that they produce (from whence comes the manufacturing cell), objects can be grouped according to their class, and components can be grouped according to their functional families. Once all operations, objects and components have been selected and assembled, they form the process plan, the program and the assembly, respectively.

A good manufacturing system allows for reconfigurability, where one machine can be replaced with another without rendering the entire shop floor inoperative. In OOP, objects can be replaced with other similar objects while leaving the rest of the program intact. The intelligent design catalogue allows a component to be changed in the catalogue while minimizing disruption to the selection process.

From a process planning perspective, the virtual machining environment allows for operations to be assembled without specifying the machine. The assignment, in this case, is carried out by the binding table (the matrix of GT can also be considered to assign operations to machines). Within the OOP programming environment, one can conceivably develop a program (template) to perform certain operations before the actual objects are created or selected from a library. The virtual design environment allows for some specification of certain parts or functions without naming a particular component. Behaviour, like the binding table, is used to translate the required function to a candidate component.

The process plan to manufacture a specific component can be modified in keeping with process constraints. As a machining tool becomes dull, for example, the feed rate may be reduced to keep torque requirements below acceptable limits. Process constraints allow the normal process setpoints to be overridden. Within the class structure of OOP, methods can be overridden within particular classes or objects, allowing for polymorphism. Within the intelligent design catalogue, functional specifications can be overridden if a given component violates certain design constraints, e.g., the component is too heavy.

When designing a machining process for the manufacture of an individual part, one may assign any one of a number of machines to carry out the operations. Conversely, an individual machine can perform a number of operations. Inheritance in OOP means that a superclass can pass its attributes to any one of a number of subclasses. Multiple inheritance means that a subclass can inherit attributes from any number of superclasses. Within design, a function can be carried out by many different components. Conversely, a single component may fulfill a number of different functions.

5 Summary

The intelligent design catalogue is a design aid which combines a virtual design environment with a catalogue (or catalogues) of standard engineering components. Components are selected from the catalogue and assembled within the design environment. The theoretical framework seeks to facilitate the selection process by providing a bridge between the design environment and the catalogue.

In addition to drawing on design theory, the theoretical framework was developed with inspiration from manufacturing (process planning and machining systems) and OOP. Design figures prominently in these areas and the intelligent design catalogue stands to benefit from some of the solutions to their particular design issues.

An important feature of the intelligent design catalogue is that it supports *partial* designs. As the design process is not strictly linear and information is often missing, an effective design aid must accommodate incompleteness. Rather than presenting the designer with a string of fields, all of which must be filled in prior to activating the system, the intelligent design catalogue guides the designer, effectively filling in the fields along the way.

Designs may be partial with respect to configuration or specification. Partial *configuration* refers to an assembly that is missing parts or components, particularly when the missing elements render the system functionally unviable. Partial *specification* refers to a part whose specification is incomplete and can therefore not be unambiguously identified in the catalogue. A minimum level of specification must be maintained if a part is to fulfill a meaningful role in the assembly; this minimum level may correspond to a function or a behaviour.

As the design is refined, moving from partially specified parts to fully specified components, the parts, and the assembly as a whole, move through several levels or layers of detail. These levels suggest that a hierarchical structure may be beneficial. Refinement of the assembly may be uneven, meaning that adjacent parts may be at different levels of specification.

Partial specification implies that the catalogue of standard parts may be kept at a distance from the design environment. This (temporary) hiding of catalogue contents is a form of *encapsulation*. The designer can carry out part of the design, unaware of the actual contents of the catalogue. Encapsulation also refers to the hiding of the contents of a component. When creating an assembly, the designer is generally concerned with the function and physical interfaces of the component, not its internal workings. Encapsulation thus allows information not relevant to the design at hand to be kept at arm's length, reducing the "clutter" in the design environment.

By keeping the catalogue at a distance from the design environment, *reconfiguration* becomes possible. This means that components or even catalogues can be removed or added without disabling the entire system. The design environment remains intact, allowing designers to create functionally complete assemblies independently of the catalogues. A second database serves to bind or bridge selected behaviours or functions to the catalogues. This is the same database that allows the designer to pass through several levels during the refinement stage.

The database must be sufficiently well structured to provide adequate assistant, yet flexible enough to accommodate different designer types. The flexibility allows potentially relevant parts to be *grouped* and presented to the designer in any number of ways, and possibly filtered in keeping with designer-imposed *constraints*. Flexibility should also allow the designer to use a standard component in an unconventional assembly and, hence, outside the standard groupings. This implies that the designer must be able to *override* (part of) the system. The ability to override means, in effect, that the designer can go directly to the catalogue for the purposes of browsing.

Those designers overriding the system are likely to be more experienced; those requesting help, less experienced. This suggests that the intelligent design catalogue has significant potential to be used for *educational* purposes. Learning can be facilitated by *restricting* the catalogue domain, and thereby limit the amount of information the student must deal with. Smaller catalogues may also serve the needs of specialized industries with a limited number of standard components.

Future developments will investigate the internal mechanisms of the intelligent design catalogue. These mechanisms must be carefully selected and implemented to ensure that they do not compromise the intended features of the system.

Potentially, additional tools can be added to the system to assess the performance of the newly created product. These tools may be analytical in nature or allow the designer to run simulations.

References

1. Carlson-Skalak S, White MD, Teng Y (1998) Using evolutionary algorithm for catalog design. Res Eng Des 10(2):63–83
2. Pahl G, Beitz W (1996) Engineering design: a systematic approach (trans: Pomerans A, Wallace K). Springer-Verlag, London
3. Gero JS, Tham KW, Lee HS (1992) Behaviour: a link between function and structure in design. In: Brown DC, Waldron M, Yoshikawa H (eds) Intelligent computer aided design. Elsevier, Amsterdam, pp 193–220
4. Gero JS, Kannengiesser U (2004) The situated function-behaviour-structure framework. Des Stud 25(4):373–391
5. Dorst K, Vermaas P (2005) John Gero's function-behaviour-structure model of designing: a critical analysis. Res End Des 16:17–26
6. Mitrofanov SP (1966) Scientific principles of group technology (English translation). National Library for Science and Technology, Washington
7. Opitz H (1970) A classification to describe workpieces. Pergamon Press, Oxford
8. Gallagher CC, Knight WA (1986) Group technology production methods in manufacture. Ellis Horwood Limited, Chichester
9. Alexander C (1964) Notes on the synthesis of form. Harvard University Press, Cambridge

10. Oldknow KD, Yellowley I (2001) Design, implementation and validation of a system for the dynamic reconfiguration of open architecture machine tool controls. Int J Mach Tool Manuf 41:795–808
11. Mehrabi MG, Ulsoy AG, Koren Y, Heytler P (2002) Trends and perspectives in flexible and reconfigurable manufacturing systems. J Intell Manuf 13(2):135–146
12. Koren Y, Heisel U, Jovane F, Moriwaki T, Pritschow G, Ulsoy G, van Brussel H (1999) Reconfigurable manufacturing systems. CIRP Ann 48:527–540
13. Wirfs-Brock R, Wilkerson B, Wiener L (1990) Designing object-oriented software. P T R Prentice Hall, Englewood Cliffs
14. Graham I (1991) Object oriented methods. Addison-Wesley Publishing Company, Wokingham
15. Marston T (2006) What is object-oriented programming? http://www.tonymarston.net/php-mysql/what-is-oop.html. Accessed 14 Nov 2007
16. Lutz M, Ascher D (2004) Learning python. O'Reilly, Sebastopol