Creativity and Design as Exploration

Brian Logan and Tim Smithers
Department of Artificial Intelligence
University of Edinburgh
Forrest Hill
Edinburgh, UK

Abstract

This paper considers the problem of *creative* design, and in particular the role of *a priori* knowledge or 'prototypes' in the design process. A design problem is characterised as one in which both the objectives and the means available for achieving these objectives are (of necessity) initially only poorly defined. Some observations concerning the nature of design process based on this characterisation are presented, and a model of the design process as a knowledge-based exploration task described. The role of prototypes in organising this knowledge is examined, and the widely accepted view that prototypes can form the principal source of knowledge for creativity in design is challenged. In a final section we outline the structural principles of a representation scheme which aims to overcome some of these difficulties and describe a design support system which uses this scheme to support the design process.

DAI Research Paper No 597, August, 1992

This is a revised version of a paper to appear in "Modelling Creativity and Knowledge Based Creative Design", J. S. Gero and M. L. Maher eds., Lawrence Earlbaum, 1992

1 Introduction

There have been a number of attempts to define 'creative' design. Several authors have identified different kinds of design processes associated with different levels of problem difficulty or the novelty of the resulting solution (Brown & Chandrasekaran, 1984; Gero & Rosenman, 1989; Westerberg et al., 1989). For example, Gero and Rosenman (1989) identify three different types of design: *routine* design in which the design variables and their range is known and the problem is one of instantiation of variable values; *innovative* design in which the space of known solutions is extended by making variations or adaptions to existing designs; and *creative* design in which the state space of possible solutions is extended or a new state space created. Other authors have argued that any such categorisation must be based on the nature of the designed artefact irrespective of how it was produced (Cagan & Agogino, 1989; Coyne & Subrahmanian, 1989; Woodbury, 1989). For example, Cagan and Agogino (1989) emphasise the characteristics of the product rather than the process used to produce it, reserving the term 'creative' for those artefacts so judged by the designer's (human) peers and leaving open the possibility of creative designs produced by 'mundane' means. These two approaches appear to be irreconcilable. The first, process-oriented view, characterises creativity as an absolute, inhering in the nature of particular kinds of processes, while the latter, product-oriented view is essentially relativistic in emphasising that creativity is recognisable only within a particular context. Either these definitions are equivalent, or one (or both) of them must be wrong. However assuming their equivalence involves making a number of very strong assumptions.

If this claim is to be taken seriously, it must mean that the application of a given set of operators to a particular body of design knowledge always results in designs which are judged to be creative by the designer's human peers. The body of knowledge and/or operators are presumably specified relative to some (current) context, as different designs will be judged creative in different contexts. (The alternative involves a set of operators and/or prototypes which produce creative designs in all contexts, which violates the second definition.) Note that the relevant design knowledge doesn't have to correspond to the current design practice against which creativity is judged, although this would be the most obvious interpretation. There may be other bodies of knowledge (and sets of operators) which inevitably produce creative results. We could, for example, imagine a stock of 'creative solution' prototypes to which the identity operator is applied to produce a creative solution. Such a set would, however, still be defined relative to the current design context. Clearly, this must be a *sufficient* condition, otherwise it would rule out the production of creative designs by other means.

However, this is a very strong claim. Presumably what you want to say if you take the process-oriented view is (something like) the production of creative designs involves the use of certain operators, i.e., that the use of such operators is a *necessary* condition for creativity. But this is just what the equivalence of definitions rules out. The operators must be sufficient for the production of creative designs if they are to constitute any sort of a *definition* of creativity. (The only other interpretation, that the operators *may* produce creative designs hardly amounts to a claim at all; it would be strange indeed if no creative designs could be produced by the application of such operators, whatever they are.) There have been several attempts to define such a set of operators in the literature: as some transformation of one or more class variables of a prototype (Oxman & Gero, 1988); as mutations of existing designs (Murthy & Addanki, 1987); in terms of reasoning by analogy (Maher, Zhao & Gero, 1989); or as design from first principles (Rosenman & Gero, 1989). However for the definition to be complete requires a characterisation of the corresponding context for which these operators will produce creative designs. As far as we are aware, no such characterisation has been proposed. Nor has any adequate metric for the determination of creative designs been developed (Cagan & Agogino, 1989). All this seems to suggest that our intuitions about creativity are problematic.

In this paper we adopt a different approach based on distinguishing between different kinds of *problems*. We define a *design problem* as one in which either the objective to be achieved or the means of achieving it (or both) are initially only poorly defined. Design in this sense is explicitly not a *search* process in which the task is essentially one of selection or optimisation over a completely defined space. To qualify as a *search problem*, a problem must be well-defined; that is, it must be possible to specify completely both the goal to be achieved, and the means available (in terms of legal moves or operators) to achieve it. More precisely a search problem is defined by a triple consisting of: an initial problem description; a set of operators for transforming problems into sub-problems; and a set of primitive problem descriptions (Barr & Feigenbaum, 1981). A solution to a given problem consists of a sequence of operators which successively transform the initial problem description into a set of primitive problems whose solution is immediate. Search problems have been extensively studied in the artificial intelligence literature, and a wide range of computational techniques have been developed for solving them. This type of problem solving corresponds approximately to the "routine design" identified by Gero and Rosenman (1989).

In this view *design* is the activity of 'solving' *design problems*. Design is a much less well understood activity than search, involving the modification of both the problem goals (the design requirements) and the means available to achieve these goals (the design knowledge). In fact one might almost say that the defining characteristic of design problems is that they are not amenable to purely search-based problem solving techniques. This is not to deny that problem solving forms part of the design process. However, it is not what characterises design as a distinct kind of intelligent behaviour. Neither should it be interpreted as claiming that all design problems are in some sense equivalent. Within the class of design problems there are wide variations in complexity and scope. Clearly design problems do differ in terms of the skill and effort required to solve them; some problems will require considerably more creativity or ingenuity on the part of the designer than others. However, even in the simplest of design tasks, the fundamental problem of how a solution is *generated* (as opposed to selected) remains. Our use of the term 'design' is similar to the "non-routine" design identified by Gero (1989).

This approach has the advantage of classifying different kinds of design processes in terms of the kinds of problems they attempt to solve, thereby avoiding *a priori* decisions about the nature of design processes or the character of designed artefacts. We believe it offers a fruitful perspective on non-routine design in general and serves to highlight some of the problems of attempting to extend the concept of problem-solving to non-routine design.

In the next section we develop our characterisation of design problems in more detail. We then present some observations concerning the nature of design process based on this characterisation, and develop a model of the design activity as a knowledge-based exploration task. The role of 'prototypes' in organising this knowledge is examined, and we challenge the widely accepted view that prototypes can form the principal source of knowledge for creativity in design. In a final section we outline the structural principles of a representation scheme which aims to overcome some of these difficulties and describe a design support system which uses this scheme to support the design process.

2 The Nature of Design Problems

We begin by presenting our characterisation of design problems in more detail. At its most general level, a design problem is concerned with the production of a description of of an artefact or process which meets a given set of objectives or requirements. These requirements are often initially ill-defined and may be in conflict. In general they will not all be equally important; legislative controls are often value free, whereas user requirements may be modified during discussions with the client, and constraints generated by the designer may be extensively revised, or even abandoned altogether during the design process (Lawson, 1980). However, the real difficulty is that these objectives cannot easily be related to one another. Lawson (1980) states: "the relative importance of the various requirements change constantly during the design process as the designer's value system is itself affected by the exploration of objectives and what he finds to be possible". The value judgments regarding 'trade-offs' between criteria are therefore context dependent, and the balance of satisfaction for such requirements may not be clear until the designer explores the various possibilities in appropriate detail. Such value judgments apply not only to the 'qualitative' criteria such as aesthetics, but also to the relative importance of quantitative criteria which themselves may be susceptible to objective measurement. Questions about which are the most important problems and what kinds of solution most successfully solve these problems are also value laden, and the answers given by designers to these questions are therefore frequently subjective and highly context dependent.

The nature of the 'real' problem is thus often not apparent but must be discovered; problems may suggest certain features of solutions, but these solutions in turn create new and different problems. The initial expression of the problem is often misleading, and designers must typically expend considerable effort in identifying the actual nature of the problem which confronts them. Design problems have no obvious or natural boundaries, but rather seem to be organised roughly hierarchically. Many elements of the problem cannot be expected to emerge until some attempt has been made at generating solutions. Given the essentially subjective nature of design it is inevitable that some aspects of the problem will remain either unrecognised or undeveloped for much of the design process. As a result design problems are full of uncertainties both about objectives and their relative priorities, and both priorities and objectives are likely to change as solutions emerge. Simon (1973) calls such problems 'ill-structured' and argues that any problem with a large base of potentially relevant knowledge falls into this category. The design task is ill-structured in this sense in a number of respects. There is initially no definite criteria to test a solution, much less a formal process to apply the criteria. In addition the problem space cannot be completely defined due to a radical lack of knowledge. Also, while the set of alternative solutions may be given in a certain abstract sense, it is not given in the only sense that is practically relevant. As a result there can never be an exhaustive list of all the possible solutions to such problems.

Design problems are therefore often multidimensional and highly interdependent. It is rare for any part of a design to serve only one purpose, and it is frequently necessary to devise a solution which satisfies a whole range of requirements. Design decisions may have results other than those intended, which highlight previously unrecognised criteria and relationships. In many cases the stated objectives are in direct conflict with one another and the designer cannot simply optimise one requirement without suffering losses elsewhere. For example, though enlarging a window may well let in more light and give a better view, it will also result in greater heat loss and may create greater problems of privacy. Different trade-offs between the criteria result in a whole range of acceptable solutions, each likely to prove more or less satisfactory in different ways to different clients and users. It is the very inter-relatedness of these factors which is the essence of design problems rather than the isolated factors themselves, and it is the structuring of relationships between these criteria that forms the basis for the design process (Lawson, 1980). The fundamental objective thus becomes one of understanding the structure of the problem (rather than the solution), and analysing the inter-relationships between criteria to gain some insight into the relationships between each individual design decision and all of the other decisions which together define the solution.

3 The Nature of the Design Process

The designer's exploration of this structure begins with the initial formulation of the problem. To a large extent, a design problem has no inherent structure; it acquires structure as solutions are proposed and problems are reduced to subproblems. In a very real sense the relationships between criteria can be seen as a function of the approach to design embodied in the proposed solution rather than as inherent in the problem itself. This initial formulation forms the basis of subsequent exploration of the problem.

Consider, for example, the problem of providing a particular view from the living room in designing a house. (This example is based on one given by Bazjanac (1974).) Such a requirement may have been specified in the original problem description or it may have been generated during the design process. In either case, an architect might choose to formulate the associated design problem in terms of some standard solution, as a problem of the arrangement of the living room and the placement of a window in a way which will provide the desired view from the relevant areas of the room. Through such a formulation of the design problem the designer has also formulated the general form of the solution; any particular design solution is determined by a specific placement of the window and a specific disposition of the living room. The designer then proceeds to explore the implications of this particular design decision. In doing so he or she may make further design decisions and consider a number of design alternatives. This process of exploration may lead to the discovery that in providing the desired view it becomes impossible to maintain the relationship between the living area and the entrance to the house, or that that the basic structural system of the house will not allow the positioning of a window of the required size in the desired location. As a result the designer comes to realise that the problem of providing a view from the living room has other aspects and its solution may involve finding a more appropriate layout for the house on the site, or the redesign of the structural system in a way which will accommodate the desired window. The formulation of both the problem and the solution may therefore change as a consequence of an attempt to solve a particular problem.

It might be argued that in this case the original formulation of the problem was unrealistic and that an experienced designer would approach the problem at a more appropriate level. However such an argument misses the point. In trying to develop a solution to a particular design problem even the most experienced designer will gain new insights which necessitate the redefinition of the problem and suggest alternative solutions. The need to understand (at whatever level) the details of a particular case and how they interact are in a sense what makes a design problem a design problem.

As a solution develops it provides an increasingly detailed context against which to test the designer's hypotheses, and the evaluation of a proposal can result in the discovery of previously unrecognised relationships and criteria. In a sense later decisions are constrained by earlier decisions in that they are taken within the context of an existing partial solution, and each decision further limits the range of possible alternatives. Solutions to particular subproblems are apt to be disturbed or undone at a later stage when new aspects are attended to and the considerations leading to the original solution are forgotten or not noticed. Such side effects accompany all complex design processes. As a result, while the final solution may satisfy all the requirements that are evoked when it is tested, it may violate some of the requirements that were imposed (and temporarily satisfied) at an earlier stage in the design. The designer may or may not be aware of these violations. Other appropriate design criteria may simply remain dormant, never having been evoked during the design process. The development of a design is thus constrained by what best fits the knowledge the designer has at that time.

The formulation of the problem at any stage is not final; rather it reflects the designer's current understanding of the problem. As the design progresses the designer learns more about possible problem and solution structures as new aspects of the situation become apparent and the inconsistencies inherent in the formulation of the problem are revealed. As a result, designers gain new insights into the problem (and the solution) which ultimately result in the formation of a new view; the problem and the solution are redefined. This process of exploration and redefinition continues until one or more of the following conditions is met (Bazjanac, 1974).

1. The incremental gain in knowledge has become insignificant and the understanding of the problem (and the solution) cannot change enough to warrant further redefinition. (i.e. the designer has reached the limits of his or her understanding).

2. The incremental gain in knowledge has become too costly.

3. The available resources (primarily time) have become exhausted.

There is no meaningful distinction between *analysis* and *synthesis* in this process; problems and solutions are seen as emerging together rather than one logically following from the other. The problem is explored through a series of attempts to create solutions and understand their implications in terms of other criteria. The designer comes to understand the critical relationships and possible forms as a solution evolves. Between generic solutions planning is less a search for the best solution than an exploration of the compromises that give sufficient solutions. These explorations help the designer appreciate which requirements may be most readily achieved. As part of this process, the designer learns which criterion values will achieve the design requirements and how much variation of these values can be tolerated while still achieving acceptable performance, the implications of achieving the current goal, and any other decisions required to make the attainment of these goals consistent with the existing solution.

Discovering more is the most important part of this process. The fundamental objective becomes one of understanding the structure of the problem, with a major part of the effort in design being directed towards structuring problems and only a fraction of it devoted to solving them once they have been structured (Simon, 1970). The design process can be viewed more generally as a process of discovering information about problem structures that will ultimately be valuable in developing possible solutions.

4 Knowledge in Design

In this section we describe a model of the design process which has been developed as part of the AI in Design research programme at the Department of Artificial Intelligence, Edinburgh University. In this model design is viewed as a knowledge-based exploration task (Smithers et al., 1990) (Figure 1). According to this exploration-based model, a design task starts with the construction of an initial requirement description R_i. R_i is usually a weak statement; it is typically incomplete in the sense that it fails to identify some necessary aspect of a possible solution, and it may be inconsistent in the sense that it may specify two or more mutually exclusive properties of a solution or conditions which must be met.

It is not possible to say in what way R_i is incomplete and inconsistent as it requires knowledge of the structure of the space of possible designs (SPD) to identify any omissions or inconsistencies. The initial requirement description cannot therefore serve as a specification for a search problem—since if it is incomplete and inconsistent it cannot define a *goal-state* in a SPD. The design process, E_d, is one of exploring the space of possible designs. This process draws on the designer's experience of previous designs together with information gained within the context of the current problem. The problem is analysed through a series of attempts to create solutions and understand their implications in terms of design requirements. As part of this process, the designer discovers those sets of design decisions (if any) which jointly achieve the design goals and their implications in terms of design criteria. The designer acquires knowledge about the nature and structure of parts of the SPD, and thus of possible design specifications which perhaps partially meet the current requirement description, or which indicate why aspects of it cannot be satisfied. As a result incompletenesses and inconsistencies in R_i are discovered. The requirement description is then modified in a way that resolves

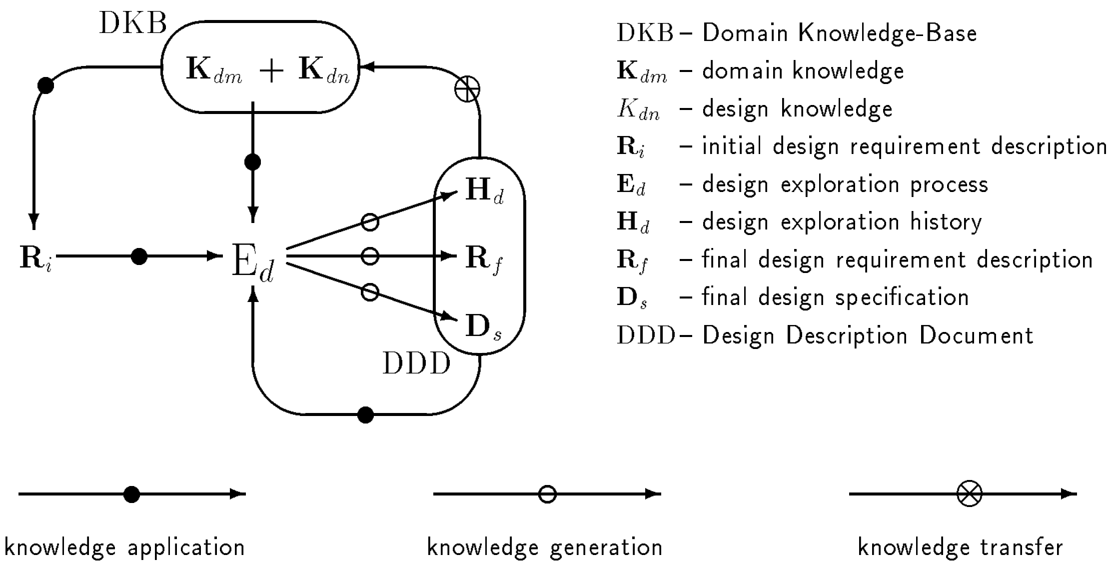

Figure 1: An exploration-based model of design

identified incompletenesses and inconsistencies, and the exploration is continued until a point in the design space is found that fully specifies a design which satisfies the evolved requirement description. The result is a final, complete and consistent, requirement description R_f, and an associated design specification, D_s, which is consistent with it.

The requirement description thus evolves from an initial statement which only incompletely defines a region in the SPD (and may not even do that) to a final requirement description which completely and consistently defines a region of the SPD. The design specification represents a sub-region of the region of the SPD defined by R_f, or, in the case of more than one design specification they represent sub-regions of the region of the SPD defined by R_f. The record of the designer's exploration of the SPD constitutes a history, H_d, of the design process. It is a description of the nature and structure of the space surveyed—in other words the paths taken, the reasons for doing so, and what it looked like on the way.

To start the exploration process both knowledge of the domain, K_{dm}, and knowledge about how to design in the domain, K_{dn} is required. Domain knowledge is knowledge about the characteristics of the domain: the nature of the underlying physical reality, the materials and techniques available to the designer, legislative controls, codes of practice and other norms and standards which are largely invariant across designs (and designers) within the domain. For example, when designing the structural frame of a building, a decision to use steel rather than reinforced concrete requires the application of domain knowledge about such things as the maximum economic span possible in steel, the availability of standard steel sections and the legislative controls governing the use of exposed steelwork in loadbearing structures. This knowledge ultimately constrains the form of any solution by defining the boundaries of the space of possible designs. It gives *structure* to the exploration space, and thus the problem, and identifies and locates interesting regions and important discontinuities.

Design knowledge is knowledge about how the space can be explored. This consists of design methods and strategies, and might include, for example, knowledge of different structural layouts or how the system of forces set up by the wind and imposed loads is best solved to determine the size of the structural members. This knowledge indicates paths through the terrain of the design space. K_{dm} and K_{dn} are not orthogonal kinds of knowledge, there is often an important dependency relationship between domain knowledge and design knowledge which means that one cannot sensibly be expressed without reference to the other. This is why they are not indicated as being two separate, and thus independent, inputs to the exploration process in Figure 1.

In the following sections we consider the form of this design knowledge and its role in the generation of solutions in more detail.

5 Prototypes in Design

The generation of solutions draws on an extensive knowledge of design methods, strategies and solutions to previous problems. Design proposals are not produced blindly but result from a general understanding of the kinds of solutions which may be appropriate in a given situation, and how these solutions may be pursued. An important component of this knowledge is what might be termed 'compiled experience'. The role of *a priori* knowledge derived either from a familiarity with related problems or, in the form of published guides and standards, has been widely recognised in studies of design. Akin (1978) discusses the use of 'problem transformations', which "make the current solution more specific, such as a precompiled solution, an analogous solution, a generic solution etc. ... if explicit transformations are not possible at the time, use previous experience to assume that certain aspects of the current solution can be further specified". Foz (1972) has shown how exploration of the problem evokes previously known solutions from memory. These examples are used as 'guides' or 'templates' for analysing or developing possible solutions in terms of the problem requirements. More recently, Gero et al (Gero, 1987; Gero, Maher & Zhang, 1988; Oxman & Gero, 1988) have proposed that design knowledge is stored and retrieved in a series of abstract schemata called 'prototypes' — "generalised groupings of elements in a design domain ... from which instances of elements can be derived".

These approaches to design may be broadly characterised as 'knowledge-based', in viewing the design process as a series of problem transformations governed by 'rules' or 'codes' linking design solutions and abstract requirements. There are clear parallels between, for example, Akin's 'problem transformations', Foz's 'templates' and Gero et al's concept of 'prototypes'. Common to all these approaches is the idea that design proceeds through the utilisation of an organised body of *a priori* knowledge, which is used both to structure and understand the design problem and which forms the basis of design hypotheses.

Steadman (1979) has argued that in architectural design this body of general or collective knowledge is perpetuated through architectural education, architectural journals and publications and the study of existing buildings. However it is not, with certain exceptions, of an organised, explicit or scientific nature. Rather, empirical experience of a range of related designs provides a body of knowledge and understanding on the basis of which it is possible to build a generalised theory (or theories) of a class of artefacts, which is used to extrapolate, beyond the tried cases, to hypothetical but related designs yet to be constructed. It is this body of knowledge, concerning, for example, the relation of physical performance to shape, which informs the creation of solutions.

However, when we turn to the application of this knowledge in the design process, a major difficulty immediately becomes apparent. A design problem is by definition unique. (If an artefact already exists which is recognised as satisfactorily meeting all the design requirements, there is by definition no design problem). The question then arises: how can an infinite number of designs be generated from a presumably finite set of schemata? A complete answer to this question is impossible given our current understanding of the design process, however we shall attempt to develop a small part of such an answer in the remainder of this paper. Briefly, we shall argue that, in general, design can only proceed through the development of new schemata and strategies within the context presented by the current design problem. In this view the primary function of *a priori* knowledge or cases is to structure the problem space, not to instantiate a solution or act as an initial starting point for one. These ideas form the basis of a simple model of discovery in design.

To the extent that the role of schemata in design has been made *computationally* explicit, they have tended to be seen as as 'parameterised design descriptions' or 'parameterised design description generators' within the context of a 'routine' or 'variant' design task. The most developed of these treatments is probably the work on prototypes (see for example (Gero, 1987; Oxman & Gero, 1988)). For example, Gero (1987) identifies four components of a prototype: a parameterised design description; a set of goals or requirements expressing the functions provided by the prototype; a vocabulary of design elements; and knowledge relating vocabulary, requirements and design description. In this view design involves a search for a prototype (or set of prototypes) that potentially satisfy the design requirements. Once a suitable prototype has been found, an instance in the form of a parameterised design description is generated using the knowledge and the vocabulary associated with the prototype. This involves reasoning from function or behaviour to structure using knowledge of the relationships between function, structure and behaviour which forms part of the prototype. This in turn may involve the selection of a value from a range of specified values, or searching for a 'lower-level' prototype that satisfies the design requirements associated with a particular design parameter. In the PRODS system (Oxman & Gero, 1988), this knowledge is encapsulated in the form of procedures or 'methods' attached to the prototype. The 'instantiation' of a prototype ultimately results in a set of assignments to design variables which further specify the design description. This process is complicated by the requirement that the resulting values and/or prototypes be 'globally' consistent, that is, consistent with the rest of the emerging design solution.

While the resulting problem is a search problem, it is not a trivial one, and there is no guarantee that even if a solution exists it will be found. One of the difficulties of much of the published work on routine design is a failure to be explicit about the kinds of control strategy required to find a consistent set of assignments to design variables. However, in principle, such problems should be amenable to conventional goal-directed problem solving techniques. A more fundamental problem is that there appears to be nowhere within such a prototype-based model that control information can be expressed.

This omission becomes more problematic when we turn to non-routine design and the problem of control comes to assume a major role.

In more recent work, Rosenman and Gero (1989) have attempted to extend their prototype-based model to include creative design. This has involved the introduction of two new classes of prototypes: 'joint prototypes' which "allow the configuration of new designs"; and 'functional prototypes' which represent the relationship of functions to each other and to behaviours to provide "an inference structure for general goal decomposition and the provision of general requirements". In addition, the relationship between function, behaviour and structure is now explicitly represented, and forms the basis of the generation of new prototypes. New prototypes can be produced in a number of ways including combination, analogy, mutation and design from first principles. However we are given no computational interpretation of these processes. Furthermore, although the problem of 'configuring' prototype instances representing partial solutions to form an overall solution is identified, there is no attempt to address the problem of what happens when a prototype fails to meet its local or global goals. Presumably this becomes a new design problem to which the same processes apply.

However while such an approach may be adequate for routine design problems, a moment's reflection shows that it cannot work in general. Even assuming that a prototype, as an abstraction from known cases or previous solutions, is capable by definition of achieving its own 'local' goals (a very strong assumption—much of the knowledge linking function to structure is abductive or heuristic in nature), it is incapable of resolving interactions between criteria. When an irreducible conflict between two or more criteria arises then either: i) a solution exists but the search strategy is incapable of finding it; or ii) no solution exists within the solution space defined by the prototypes. This happens when no way can be found of achieving a particular requirement or set of requirements within the constraints imposed by the rest of the solution context. This may either be because while the available prototypes are capable of achieving their immediate goals, the resulting 'local' solutions have unacceptable consequences elsewhere (solutions can be found for each of the sub-problems considered in isolation, but these partial solutions are mutually inconsistent), or because no way can be found of achieving a particular goal even in the absence of other constraints (the problem lies outwith the scope of any known prototype). Such conflicts are common in design, indeed we have identified the existence of conflicts between criteria as one of the characteristics of design problems.

In such a situation the designer must either modify the problem, i.e., relax one or more constraints until the current solution meets the revised design requirements, or modify the means available to solve the problem by modifying an existing prototype or creating a new prototype. (In many situations, it will be necessary to modify both the requirements description and the available problem solving strategies to achieve a consistent solution.) However neither of these strategies appears to be available within a prototype framework. We cannot modify the problem as prototypes contain no explicit representation of the current problem requirements (as distinct from the functional properties of the prototype). Requirements are represented implicitly in the initial problem definition and are elaborated and refined as the instantiation of prototypes implicitly creates new goals to be achieved by lower-level prototypes. If instead we decide to modify the solution, there are two problems to be solved: i) deciding which part of the current inconsistent solution is to be modified; and ii) deciding how to modify it to achieve the design requirements. In the prototype model, this corresponds to: i) deciding which prototype to modify (or to create a new prototype); and ii) determining how it is to be modified (or created). While Rosenman and Gero identify a number of ways in which new prototypes can be produced, we are given no formal definition of the individual operators. More importantly, the control of this process—which operators to use and to which part of the solution they should be applied—is not made explicit. Control information does not form part of the definition of a prototype and in Rosenman and Gero's model there appears to be nowhere else to put it, as the prototype model itself contains no explicit meta-level or control strategy. In the terminology of section 4, prototypes contain only domain knowledge, they have no design knowledge.

Prototypes are only a representational framework and as such are provably equivalent to many others in terms of expressive power. Ultimately the question is not one of how the knowledge is organised (although this is important), but what knowledge is represented. Until we address the problem of what should be represented we will not have a model of creative or any other sort of design.

These difficulties are a result of a series of mistaken assumptions about the role of *a priori* knowledge in design. Viewing prototypes as 'parameterised design descriptions' or 'design description generators' results in an emphasis on local problem solving. Design problems *are* solved piecemeal, but this is often the easiest part of a design task—the hard part is integrating the solutions to the various sub-problems to form an overall solution.

Underlying this model of design are the assumptions that:

1. the design problem can be broken down into a series of independent sub-problems, i.e. that there is no, or only weak, interaction between the prototypes;

2. each of the resulting sub-problems can be solved by search.

In dealing with design problems, neither of these assumptions is valid. The defining characteristic of design problems is that they are highly inter-dependent. Attempting to solve one problem will, in general, produce new problems elsewhere. The designer must explore the space of possible designs. However, a prototype seen as simply a 'parameterised design description' or a 'parameterised design description generator' is incapable of supporting such exploration. Note that the designer cannot simply select another prototype—the problem is not solvable by search by definition. Even if this were possible, the designer would still require knowledge about which of the possible alternatives to choose, which can only be obtained from knowledge about the relationship between the prototype and the solution context, i.e. the structure of the space in which they are embedded.

A prototype is more usefully characterised as a tentative hypothesis about how to proceed with the design. The evocation of a prototype from memory provides the designer with two types of information: it tells the designer what is possible within a given set of constraints, i.e. that an artefact or class of artefacts with a given set of attributes is *feasible*, and, more importantly, it contains information about the structure of the space of possible designs — a way of looking at the current problem which proved successful in dealing with similar problems in the past. In other words, it is knowledge of what previous explorations were carried out and how that is important, not what the specific results were. In effect a prototype can be seen as a rule which states "given a situation in which you are trying to achieve A, use B", where B can be seen as a tag or a pointer to a set of relationships, strategies, problem decompositions, standard or generic solutions, default values etc. The evocation of the prototype makes these strategies and relationships available for use in achieving the current goal. However, the fact that the prototype may subsequently be used to infer some aspect of the design from the design requirements (in the form of a default value or through the application of some procedure or algorithm), should not obscure the fact that the prototype itself must be modified in the event that the solution attempt is unsuccessful. Similarly, in the event that it becomes necessary to modify the current requirement description, this structural information is essential in identifying the goal(s) to be abandoned.

Design therefore becomes a process of modification or refinement of the designer's general strategies and prototypes within the context of the current problem. More generally we can see the problem of design as one of discovering how to develop and represent these basic prototypes to solve a particular problem. Starting from a set of prototypes or models representing, for example, a previous solution to a similar problem or the underlying physical reality of the problem domain, certain relationships are emphasised while others become less important or are ignored as the design process progresses. New relationships may be introduced, either empirically or through the introduction of facts or relations which had previously been suppressed.

Hillier and Leaman (Hillier & Leaman, 1974; Hillier & Leaman, 1976) have characterised this process as one of the elaboration and modification of cultural stereotypes or 'templates'. They argue that that the designer is situated in "a richly connected universe whose connections are those dissimilar domains that must be related in design; activity and space, psychology and climate and so on". These structures are embedded in the language the designer uses and in the instrumental set—the technologies or kits of parts (concepts) and typical design solutions (prototypes or 'prestructures') to which his systems of representation refer (Hillier & Leaman, 1974). Even to name an architectural problem—say, 'design a school'—implies a whole range of solutions which will be more or less immediately activated by the designer's prestructures. These structures form an evolving typology of standard solutions to recurring problems in design, modified by the designer's experience, ideology and the physical, social, and cultural environments which form the context of design.

Such deep cultural structures may be transmitted unchanged through several generations yet produce great variety at the observable level. Hillier argues that such underlying stable structures or 'manifolds' correspond to what biologists call 'genotypes' as opposed to the 'phenotypes' (or variably developed observable forms) which constitute individual solutions. The cultural genotypes of architectural and urban form mutate gradually or are suddenly altered to reflect much more profound mutations in the relations between changes in society and changes in its spatial form. This evolutionary process is concerned with "the gradual unfolding of a socio-spatial morphology, which includes the development and stabilisation of technologies, social processes of environmental control and all constructions of the mapping from social form into spatial form" (Hillier & Leaman, 1974). Over a much shorter timescale the genotype is, in turn, elaborated into a phenotype appropriate to a particular set of local conditions, through the activity of designers, users and others.

In this view design is seen as the process of discovering “the appropriate transformation or ‘unfolding’ of prestructures (genotypes) in relation to the constraints imposed by the environment of the problem” (Hillier & Leaman, 1974). Both the transmission and transformation of prestructures form a process of elaboration and discovery which underlies the active formation of relationships and within which every solution may be unique. The development of individual designs is, in general, of less significance in the overall design of the artificial environment than events that take place less obviously in evolutionary time. These changes have the effect of altering the general form of the prestructures, which designers bring to bear on individual problems of phenotype production, effectively acting as a framework of autonomic assumptions within which they operate (Hillier & Leaman, 1974).

6 Representing Design Knowledge

The problems identified with prototypes in the previous section suggest that another approach is required. In this section we outline an attempt to develop a declarative representation scheme for domain and design knowledge and describe how this representation is used by a design support system to assist the designer in exploring the space of possible designs. We use the terminology introduced in Figure 1.

Both the DKB (Domain Knowledge-Base) and the DDD (Design Description Document) constitute knowledge about a particular SPD (space of possible designs), but each represents knowledge of a different kind.

The DKB partially defines a SPD (for a particular domain) is terms of domain knowledge, K_{dm}, and design knowledge, K_{dn}. The domain knowledge expresses facts about the domain, such as the heat conduction properties of different building materials, and the properties and attributes of basic objects and processes associated with the domain—its ontology—such as walls, windows, doors, etc. It also reflects scientific, technological, sociological and statutory knowledge and constraints relevant to the domain. This knowledge must be represented declaratively so that it can be freely used in general inferencing, not encapsulated in procedures that are only applicable in certain circumstances. It is this domain knowledge which determines the basic structure of the space of possible designs. In our representation scheme domain knowledge is organised hierarchically using *specialisation* relationships between representation objects.

Design knowledge derives from previous designs and is concerned with how domain knowledge is used to define and solve design problems; how the space of possible designs is explored and how a developing design problem structure is created, modified, and refined. Design knowledge includes useful decomposition criteria and strategies, synthesis and analysis methods and techniques, and required validation, documentation, and presentation procedures, for example. While domain knowledge governs the underlying behaviour of a design problem, design knowledge is used in deciding how it is to be configured—what priority to put on requirements and constraints, what is to be ignored, and what is to be included etc. Since, during the design process, this problem structure is often modified and revised as its nature is explored and understood, design knowledge must also be represented declaratively. It must not be wrapped up in preconceived and fixed approaches to design problems, as a 'parameterised design description' and its associated 'methods'. As more designs are carried out so more ways of approaching, structuring, and solving the problems that occur in a particular domain will be discovered. It therefore needs to be related to the domain knowledge. In our representation scheme an *aggregation* relationship is used to 'chunk' design knowledge expressed in terms of domain knowledge.

Using the *specialisation* and *aggregation* relationships the representational objects used to declaratively define domain and design knowledge are organised in a way that can support design tasks. In general this means that the DKB partially defines a SPD in terms of 'regions of interest' and the relationships between them. These regions of interest represent regions in the space of possible designs which have been explored as a result of doing previous designs, and in which design knowledge has been embedded in the domain knowledge—regions in which the relationships between the structure of the space of possible designs and possible design problem structures and their development and exploration have been previously worked out, and which can therefore be used to start and subsequently support new exploration processes.

The DDD resulting from a particular design task defines a region of interest in the SPD. It typically represents an elaboration and/or extension of the region defined by the DKB. Its structure and content directly reflects the nature and results of a particular design task; its exploration history, evolved requirement description, and final design specification, or specifications if more than one design results. The structure and content of a DDD is thus particular to each design task undertaken, but each DDD represents an important source of knowledge for extending and modifying the DKB.

The relationship between a DKB and the DDDs resulting from a series of design tasks is one of abstraction and generalisation. Carrying out individual design tasks does not just result in solutions to particular design problems, it also results in greater knowledge and understanding of a class, or type, of design problem. Initially some of this additional knowledge is expressed in the DDDs of particular designs. In order to make it available for use during other design tasks it needs to be transferred to the DKB built to support the class of problems concerned. This transfer process represents a kind of design problem in itself; new domain and design knowledge structures and relationships need to be designed and incorporated within an existing DKB so that it can be used to advantage in future design problems. As design is practised today this process of abstraction and generalisation of knowledge from particular design tasks is largely implicit in the practice of designers—it is part of becoming an experienced designer; of becoming an expert. Since the abstraction and generalisation process of transferring knowledge from DDDs to a DKB is described as a design problem, it should also be characterised by our exploration-based model described in section 4. The details of this await elaboration and are currently the subject of research.

In the remainder of this section we briefly outline how this declarative knowledge representation scheme can be used to support the exploration process described in section 3. We describe the architecture and operation of the Edinburgh Designer System (EDS), a design support system for mechanical engineering design developed as part of the Alvey large scale demonstrator 'Design to Product' (Smithers, 1987) which uses this knowledge representation scheme. A *design support system*, unlike a CAD system, attempts to actively assist the designer throughout the design process. EDS doesn't actually design anything—rather it attempts to support the designer in exploring the space of possible designs. This support is manifest at two distinct levels:

1. support for the overall control of the design process; and
2. support for the production of alternative design solutions.

These might loosely be described as *strategic* and *tactical* support. Strategic support is primarily concerned with the overall organisation of design process: the generation and maintenance of alternatives or solutions to particular sub-problems; and the selection of the most promising candidates for further development. Tactical support is concerned with the solution of particular problems: the assignment of values to design parameters; and the evaluation of the resulting partial design solutions. To the extent that the exploration of the space of possible designs is carried out through the development of possible solutions, the overall control of the design process is dependent on the production of alternative solutions. For example, the comparison of alternative proposals relies on there being some means of generating and evaluating alternative solutions.

The process of analysis-through-synthesis outlined above typically results in the generation of a large number of alternative designs or variants. These alternatives may embody radically different approaches to the problem or they may be variations on a common theme or both. Alternatively a problem may be broken down into a set of simpler sub-problems (for example, the design of an assembly may be reduced to the design of its constituent components), together with the problem of integrating the resulting part solutions, or it may be broken down into a set of associated functions. A major task of any design support system will therefore be one of complexity management—keeping track of the various alternatives design proposals and the information required to solve particular problems, both as a record of the design process (which alternatives have been tried and what was learned) and to allow the designer to partition the problem appropriately (Logan, 1989).

At the level of individual design problems the designer requires both information about the consequences of design decisions and assistance with the solution of problems. Of particular concern are inconsistencies which arise in attempting to satisfy multiple goals. The system should therefore provide support for context specific inference based on design decisions, constraints and the requirements defining the design problem, together with physical laws, heuristics and other domain knowledge relating the parameters of the design. In addition the system should, where possible, provide assistance in solving particular design problems, drawing on the large amounts of knowledge encoded in design handbooks, codes of practice and in the expertise of individual designers.

EDS consists of four main functional elements which together embody the model of design support outlined above:

1. Consistency Maintenance
2. Context Management
3. Knowledge Representation

4. Inference

In EDS, the context management and consistency maintenance sub-systems provide strategic support, and the knowledge representation and inference sub-systems provide tactical support. In what follows we shall concentrate primarily on the consistency maintenance and inference sub-systems of EDS which are implemented as an assumption-based truth maintained blackboard system. The user interface and the overall design of the system are discussed in more detail elsewhere (Smithers et al., 1990).

The production of a large number of alternative design solutions results in a major consistency maintenance problem. If the system is to effectively aid the designer at the strategic level, the various incompatible design alternatives must be considered in isolation. This, together with the need to retain alternatives for comparison and/or possible later development, led to the adoption of an assumption-based truth maintenance system (ATMS) as the basis of EDS (de Kleer, 1984). The ATMS forms the core of the system and all of the other system components are implemented using the facilities it provides. The ATMS builds and maintains the Design Description Document and provides an interface between the contents of the DDD and the other sub-systems, passing out relevant pieces of information to them as required and incorporating new information which it receives from them into the dependency structure. Its central role in the system architecture has resulted in a number of extensions to the 'conventional' ATMS model which are discussed in more detail below.

At the tactical level a number of particular tasks within the overall exploration process can be identified for which partial support is possible. In our work on mechanical design, for example, we have identified algebraic equation solving, constraint manipulation, reasoning about spatial relationships, shape and space occupancy and reasoning about relational information as common to several different activities in the design process. Support for particular design tasks is provided by a series of general purpose support systems — forward chaining or data-driven inference engines which attempt to infer by means of constraint satisfaction the consequences of the designer's decisions. In general the inferences engines derive necessary consequences of what is currently known about the design and can be allowed to proceed relatively unhampered by the designer, with the sole constraint that they should refrain from re-deriving information which is already present in the DDD.

Control of the interactions between these inference engines is in the style of a blackboard system (Hayes-Roth, 1985), with the inference engines acting as knowledge sources. The system maintains an agenda of Knowledge Source Activation Records (KSARs) which mark an intent to perform some particular task. The ATMS truth maintains the KSARs within the DDD on an equal footing with existing data and its justifications. This benefits the system by automatically removing flawed bids as they come to light as a result of other inferences and the operation of the ATMS. The agenda control mechanism therefore does not need to be vigilant on this matter. A similar arrangement can be found in (Jones & Millington, 1986). Blackboard systems are usually designed and implemented as 'single context' problem solving systems in which the knowledge sources work together to construct one consistent solution. Truth maintaining the blackboard has therefore resulted in a number of departures from the conventional practice in blackboard systems, notably the absence of deletions (except for KSARs) or amendments as it is unclear how these fit into an assumption-based truth maintenance scheme.

EDS represents domain knowledge in a Domain Knowledge Base containing definitions of domain objects, called *module classes*, related by *kind_of* (specialisation) and *part_of* (aggregation) relations. Each module class declares a set of parameters, variables and constraints which define a particular class or type of object. The creation of an instance of a module class results in datum nodes being created in the ATMS for each of the module's parameters, variables and constraints. *Justifications* link new data introduced into the DDD as a consequence of applying a knowledge source to some particular set of domain data already present in the DDD. This existing data is also pointed to by the justification. Note that a particular datum can be supported by more than one justification—for example if it can be derived in different ways or from different sets of assumptions.

Design proceeds by creating instances of module classes and assigning values to their parameters to define one or more possible solutions. (This is something of an oversimplification—the user can also define new parameters and constraints and assemble novel designs from existing modules.) When the user makes an assumption one (or more) datum nodes are created in the DDD to hold the new information. As new information becomes available, it is examined by the knowledge sources to see if it, together with any information already in the DDD, can be used to make further inferences. If a knowledge source indicates that it can make an inference, it generates a bid in the form of a Knowledge Source Activation Record which is scored and merged into the agenda. When a KSAR reaches the front of the agenda, it is executed and the results are claimed into the ATMS. This information may in turn form the basis for a new round of bids and this cycle continues until no executable KSARs remain in the agenda. As new values for parameters or bounds on them are assumed or derived, consistency checks are performed between constraints and values by the *valueConflict* KS. Conflicts result in the creation of a justification for the distinguished node ⟨*false*⟩ recording the fact that the assumptions involved are mutually inconsistent and causes the ATMS to partition the assumptions into mutually consistent sets. If there is no conflict, EDS marks this by justifying the datum ⟨*consistent*⟩ and proceeds as usual. The user is viewed as a knowledge source whose 'bids' are always processed first. This allows the system to follow several lines of reasoning as it attempts to infer the consequences of the user's design decisions, while still giving priority to user input.

To conclude this section, we present a simple example to illustrate the operation of the system. This is not intended to explain in detail all of what the system does, but to indicate how the various sub-systems described above work together to support the exploration of the space of possible designs. The example we have chosen is the design of a beam supporting a precast concrete floor in a small office development. The beam spans 7.4m, the total dead load is 292.7kN and the imposed load is 362.3kN, both uniformly distributed. It is assumed that the beam is simply supported and is adequately restrained against lateral buckling.

The design process begins with the designer searching the DKB to see if it contains anything relating to the problem in hand. If a complete design closely satisfying the requirements exists in the DKB the designer may decide to use it and modify it by changing the existing values of various parameters. Alternatively, and more usually, object(s) representing a class of some useful kind of functional mechanism is used and an instance of it is assumed, and named. This results in instances of all the constraint expressions represented by the chosen object class, and all that it inherits from its parents (if any) being passed to the ATMS which builds them into the Design Description Document. Having loaded all the instances of object classes thought to be required, at least for the moment, the DDD will effectively represent a particular part of the space of possible designs selected to be explored in more detail. The designer might then start exploring the possible solutions to the particular problem in hand by choosing values to assign to parameters appearing in the instances of constraint expressions, to see what the consequences of such values, or combinations of values are.

In this case we assume that the DKB contains an appropriate module class representing a simply supported beam with uniformly distributed load, `UB001` (see Figure 2). (It should be emphasised that this is a rather unrealistic example in that it presupposes the existence of a module class appropriate to the problem. In reality, such a problem would probably be solved by combining instances of a number of simpler and/or more general modules relating to particular beam types and loading conditions. The standard 'beam set' consists of about 10 module classes covering various beam types, end support and loading conditions. However, it serves to illustrate the operation of the system and highlights the differences between the approach taken here and that taken in, for example (Gero, Maher & Zhang, 1988).) The module class definition consists of three main sections: the module class header; parameter, variable and constant declarations, and constraints, each of which in turn consists of one or more subsections.

The module class header contains information relating to the creation of the module and its modification history, its place in the *kind-of* and *part-of* hierarchies, and the names of any diagrams or figures relevant to the module class which should be displayed by the DKB browser. The declaration `[B] : universalBeam` declares the *typical name variable* of the module, `B`, which is used to refer to the module relative to its parts or in describing its position relative to other objects and is similar in many respects to the notion of 'self' in object oriented programming languages. The next section declares the module's parameters, variables and constants. *Parameters* describe some property of the designed artefact such as its size, shape or the material of which it is made. The set of parameters can be viewed as properties of the object for which values need to be determined in order to form a description of the object that is sufficient to consider this area of the design complete. *Variables* are properties of the design which relate the parameters of its description, and may change during the operation of the designed object, such as its speed or power output. *Constants* are symbols which will appear in the constraints and for which no substitution of numerical values will be performed. Examples of these are symbols denoting things like 'aluminium' or 'pi'. Each sub-section consists of zero or more entries declaring a parameter, variable or constant, one per line. Each declaration consists of the name of the parameter, variable or constant followed by its type, dimension and an identifier consisting of a '`#`' and an integer (used by the system to uniquely identify the parameter variable or constant). The remainder of the line, up to the 'double hash' (`##`) symbol, is treated as a comment and ignored by the system for the purposes of inferencing. The '`$`' symbol is an operator used to create references to the parameters or variables of a particular named instance of a module class from the instance name and the declared name of the parameter or variable.

The format of the constraint declarations is similar to that for parameters, variables and constants. Each constraint consists of an expression containing parameters, variables or constants, followed by a relational operator, an expression (involving other parameters, variables or constants) or value, followed by a constraint number and a comment. Tables are viewed as a special kind of constraint, relating a 'dependent variable' (a parameter or variable) to one or more 'independent variables'. Each table is introduced by the `functionTable` or `catalogueTable` keyword, followed by the name of the table, its dimension, the type of interpolation to be used by the table engine and the independent and dependent variables in order. The system contains a syntax checker and translator which converts this human-readable form of the module class definition into an internal form used by EDS.

We assume that the designer begins by creating a named instance of the module class `UB001`. This results in EDS constructing, in the DDD, a uniquely identified *instance* of `UB001` using the knowledge declared in the module class definition. This is done using the following user instruction:

```
USER: assume [b1] : universalBeam
```

which says create in the DDD an instance of the module class `UB001` called *b1*. The inner workings of the system can be viewed in one of the EDS output windows, and looks like the following:

```
EDS: NEW > N   1: [b1] : universalBeam
     NEW > J   0: assumption
     NEW > K   0: findSuperClass (125)
     NEW > K   1: findDirectParts (122)
     NEW > K   2: findDirectConstraints (121)
     NEW > K   3: declareLabels (124)
     NEW > K   4: declareDirectAttributes (123)
     NEW > J   1: findSuperClass
     RIP > K   0: findSuperClass (125)
     NEW > N   2: Declaration of b1 as a label
     NEW > J   2: declareLabels
     RIP > K   3: declareLabels (124)
     NEW > N   3: Declaration of g$b1 as a parameter
     NEW > N   4: Declaration of py$b1 as a parameter

                  .
                  .
                  .

     NEW > N  26: Declaration of e$b1 as a constant
     NEW > J   3: declareDirectAttributes
     RIP > K   4: declareDirectAttributes (123)
     NEW > J   4: findDirectParts
     RIP > K   1: findDirectParts (122)
     NEW > N  27: wif$b1  =  1.6 * wi$b1
     NEW > N  28: wdf$b1  =  1.4 * wd$b1

                  .
                  .
                  .
```

```
     NEW > N  41: functionTable(~, 2, 0, ~)
     NEW > N  42: catalogueTable(~, 2, 0, ~)
     NEW > J   5: findDirectConstraints
     RIP > K   2: findDirectConstraints (121)
     NEW > K   5: bbTermination (50)
     RIP > K   5: bbTermination (50)
yes
```

Here `NEW` means a item has been created in the DDD, `N` means that it is a datum *node*, i.e. a piece of information, `J` means that it is a *justification*, the method by which the information was introduced into the DDD (used by the consistency maintenance system to record dependencies), and `K` means it is a Knowledge Source Activation Record, which marks the intent to perform some particular piece of work (used by the agenda control mechanism to schedule the work done by EDS). (The "~", in "`NEW > N 41: functionTable(~, 2, 1, ~)`" indicates that a structure has been compressed to allow the output to fit on a single line. In this example the first "~" stands for a table.) `RIP` indicates that the work associated with a particular KSAR has been carried out. The numbers following these type identifiers are the *tags* that the system uses to identify the corresponding nodes in the ATMS. `RIP > K 5:  bbTermination (50)` (the last line in the EDS output listed above) indicates that the EDS Blackboard system has terminated, EDS has done all the work it can, and that it is awaiting further user interaction. At this stage there are no inferences that the system can make, other than to simplify the constraints into an internal 'canonical' form.

As the design proceeds the consequences of the designer's assumptions are derived by the support systems. Such derived information typically relates to the predicted performance of the designed artefact (including any constraints violated by the proposed design) and any parameter values which can be inferred from the designer's assumptions or their consequences and the constraints linking these values.

For example, if the designer makes further assumptions about the the yield stress and the self weight of the beam:

```
USER: assume py$b1  = 275
      assume wbf$b1 = 0
```

and the problem requirements (the imposed load, dead load and the span):

```
USER: assume wi$b1  = 326.2
      assume wd$b1  = 292.7
      assume l$b1   = 7.4
```

the system can infer values for the maximum bending moment, shear force and plastic modulus:

```
EDS: NEW > N  48: w$b1  = 654.9
     NEW > N  49: mc$b1 = 605.8
     NEW > N  50: s$b1  = 2202.9
     NEW > N  51: fv$b1 = 327.5
```

This information can be used to assist in selecting a beam size from the standard sections represented in the catalogue table in the module class UB001. For example, if the designer decides that a $457 \times 191 \times 98$ beam of type 43 steel would be suitable:

```
USER: assume g$b1  = 43
      assume sd$b1 = 457
      assume sb$b1 = 191
      assume wb$b1 = 98
```

EDS will re-derive the values for self weight and yield stress assumed earlier and, using information from the catalogue table in the module class, can check the proposed design for consistency against the other constraints:

```
EDS: NEW > N  67: s$b1     = 2320.8
     NEW > K  27: valueConflict (150)

     !!! > New nogood set: [[1 45 46 47 52 53 54 55]]
     !!! > J  49: valueConflict

     OUT > J  49# valueConflict
     OUT > K  27# valueConflict (150)
     IN  > N   0: <false>
     RIP > K  27# valueConflict (150)
     NEW > N  68: pv$b1    = 847.2
     NEW > N  69: defx$b1  = 12.7
```

The beam is satisfactory in shear and deflection, but fails in bending (constraint 7 is violated) as the required plastic modulus, *s$b1*, is greater than that of the selected section. Conflicts are noted by creating a justification for a distinguished datum ⟨*false*⟩ which results in the ATMS partitioning the assumptions made so far into sets which, internally, are mutually consistent. If there is no conflict then EDS marks this by justifying the datum ⟨*consistent*⟩ and proceeds as usual. A inconsistent set of assumptions is called a *nogood*. In the above example, the nogood set consists of the constraints declared in the module class and loaded when the instance *b1* was created (assumption 1) which are jointly inconsistent with the assumed problem description (assumptions 45, 46 and 47) and the chosen section (assumptions 52, 53, 54 and 55).

When an inconsistency arises or the design is found to be unsatisfactory (as is typically the case), the designer can attempt to to modify the offending parameter values using information on which assumptions are jointly inconsistent provided by the ATMS. To assist the designer in understanding the dependencies between assumptions and derived results, EDS provides various utilities which allow the user to examine the contents of the DDD. For example, using the *graphical explanation* facility, the user can display a graphical representation of how a parameter was inferred by viewing its *justification*, *environment* and *inferencing method*. In particular the user can discover the reason for the inconsistency signaled above by requesting that the system display the mutually inconsistent assumptions (and their consequences) which led to the derivation of ⟨*false*⟩. Alternatively, the designer can ignore the inconsistency and continue to pursue the development of the inconsistent design based on what can be coherently derived. The opportunistic nature of the blackboard control strategy means that whatever can be consistently derived from a set of assumptions will be derived. This approach may be appropriate when, for example, the inconsistency is considered minor or peripheral and the main interest lies with the consequences of some (consistent) set of assumptions which are considered central to the proposed solution. Even if if the design fails to violate any constraints the designer may elect to pursue several different designs in parallel in an attempt to determine which gives the best overall performance, or to determine the sensitivity of the derived performance to the values of the design parameters. This will typically involve trying several alternative values for parameters until the constraints are satisfied or the relative performance of the various alternatives is understood. Multiple assignments to parameters (giving rise to multiple alternative solutions) and their interactions are handled automatically by the ATMS as are inconsistencies between parameter values and any assumed constraints.

In this case we will assume that the designer elects to continue with the current approach and selects a larger beam with an appropriate plastic modulus, for example a $533 \times 210 \times 92$ beam with a plastic modulus of 2366 cm^3:

```
USER: assume sd$b1 = 533
      assume sb$b1 = 210
      assume wb$b1 = 92
```

EDS again derives the consequences of the decision and this time the constraints are satisfied; the section meets the design requirements (although it is not necessarily optimal). However, the chosen section is over half a meter deep and although it solves the 'structures problem' a beam of the is size may be unacceptable for other reasons, for example the cost of the structure may be too great or the size of the structural zone may add unacceptably to the total building volume which must be air-conditioned. (This is when the 'real' design problem begins to emerge. Unfortunately, we can do little more than indicate the nature of the problem here. For more details, see (Logan & Newton, 1989).) If the size of the structure is deemed to be unacceptable, there are several options open to the designer: the designer can change the bay size, reducing the span and hence the size of the structure; alternatively the designer could change the structural system, perhaps to reinforced concrete or a loadbearing masonry construction drawing on knowledge about these structural systems represented in the hierarchy of module classes. However, to avoid complicating the example, we shall assume that the designer decides to explore the possibility of using a higher stress steel (type 55):

```
USER: assume g$b1 = 55
```

The system re-derives the parameter and variable values for the current $533 \times 210 \times 92$ beam size using data about the yield stress of type 55 steel obtained from the *functionTable* in the module `UB001`. (Note that the system derives this information for both the $457 \times 191 \times 98$ beam and the $533 \times 210 \times 92$ beam. If this is undesirable, the designer can create a new instance for each alternative design. However this is unnecessary as the ATMS maintains consistency of the information and can be cumbersome when we wish to exploit information which has already been derived for an existing design.) The maximum bending moment is unchanged, but the higher yield stress of type 55 steel means that the plastic modulus is reduced to 1365.5 cm^3. From the table of beam properties, the smallest section with an adequate plastic modulus is a $406 \times 178 \times 74$. However if we check this section by assuming new values for the beam depth, width and self weight, we discover that although satisfactory in bending and shear the beam fails in deflection:

```
EDS: NEW > N 104: defx$b1 = 21.3
```

```
NEW > K  87: valueConflict (150)

!!! > New nogood set: [[1 45 46 47 93 94 95 96]]
!!! > J  91: valueConflict
```

etc. This difficulty can be overcome by using a deeper beam. For example, a $457 \times 152 \times 67$ beam meets the deflection criterion. (Another option would be to relax the deflection criterion.) However the decision to use a larger section results in a corresponding increase in the depth of the structure and offsets one of the principal advantages of the more expensive, high stress steel. Whether the reduction in depth achievable with the high stress steel is worth the increased cost will depend on a number of factors: the relative cost of the various grades of steel; the implications of a smaller structural bay for the flexibility of the building in use; the consequences of the increased construction time associated with a reinforced concrete structure for the financial viability of a speculative development etc. etc.

The example could be further elaborated by introducing additional criteria, or utilising modules representing different components (such as a reinforced concrete beam), or the analysis of complete structural systems consisting of a number of modules. However, while the example described above presents an oversimplified view of the design process, it serves to illustrate the use of the system. The designer can pursue the development of the design without worrying about inconsistencies except insofar as they indicate the shortcomings of a proposed solution. The ATMS ensures that only valid inferences are drawn from the current design description by automatically detecting inconsistent sets of assumptions and violated constraints and partitioning the design description to restore consistency. One of the advantages of the exploration based model of design and the truth maintained blackboard system control structure used in EDS is that it is easy to make different assumptions corresponding either to design choices available within one module class definition or to choices between module class definitions. By choosing different types of structure, i.e., loading different module class definitions and assigning values to their parameters, the implications of different approaches to the structures problem become apparent. The investigation of design alternatives leads to a better understanding of the design problem. These explorations help the designer appreciate which requirements may be most readily achieved and those that may be neglected without loss. As part of this process the designer learns which criterion values will achieve the design goals and how much variation in these values can be tolerated while still achieving acceptable performance. The designer also discovers the the implications of achieving the current goal and any other decisions required to make the attainment of the goal consistent with the current solution.

We believe this provides a flexible environment for the exploration of the structure of design problems. However, making the best use of the exploration based model of design and control structure of EDS relies heavily on the designer's understanding the problem and knowledge of the contents of the module class definition files.

7 Conclusion

We have argued that the design process can be seen as one of exploring the space of possible designs to discover its structure, and hence the nature of possible designs. The designer proceeds by proposing solutions and evaluating their implications in terms of solution criteria. Decisions regarding any single criterion often have widespread implications, highlighting previously unrecognised criteria and relationships. These interactions between one requirement and another define what we have termed the 'structure' of a design problem. It is the pattern of constraints defining the problem space and the relationships between them and the basic structure of the space of possible designs that forms the essence of design problems, and the discovery of appropriate relationships between these criteria that forms the basis for the design activity. The fundamental objective in design is therefore to understand the structure of the problem, with a major part of the effort in design directed at structuring problems, and only a fraction of this effort directed at 'solving' them once they are structured. The technique of 'analysis-through-synthesis' discussed in the preceding sections can be viewed more generally as a process for gathering information about problem structure that will ultimately be valuable in discovering a solution.

An appreciation of these relationships is fundamental to the design process and necessary for the production of new designs. The adaption of prototypes within the framework of the exploration based model of design presented above can be viewed as a simple model of this (in reality complex) process. The modification of a prototype within the context of the current problem can be seen as a very crude model of, or perhaps a metaphor for, what happens when a designer begins to 'understand' the structure of a problem, and its relationship to the space of possible designs. It is at this level that the problem of creativity in design—or simply design—must be addressed.

The most important limitation of our exploration-based model is its failure to say anything about this process of prototype adaption and in particular how it is controlled—the problem of deciding which prototype to modify and determining how it is to be modified. This limitation is reflected in the kinds of support EDS is capable of providing. EDS attempts to support the designer's exploration of problem structure by automatically deriving the necessary consequences of the designer's decisions, detecting inconsistencies in the requirements or the problem description and maintaining the justification structure which expresses the dependencies between the designer's assumptions and their consequences. This strategy has two main advantages: it allows us to tackle more complex (though still not very complex) problems; and it allows us to concentrate on those parts of the problem we think we do understand (e.g. knowledge representation and a few limited forms of inference) while offloading the rest onto the human designer. However, although it is possible to modify a module class by introducing *ad hoc* assumptions into the DDD or by creating a new module class in the DKB, the system offers little or no support in this area beyond tools to assist in the creation of new modules. This lack of understanding also limits the effectiveness of the kinds of support which the system can provide. Because it has no understanding of what the user is trying to do or the information relevant to a particular design task and consequently of where it should concentrate its efforts, the system is capable of providing only weakly directed support unless the user exercises close control. We are currently working on an extension to this framework which is intended to help better understand this aspect of design (Logan, Millington & Smithers, 1990).

Acknowledgements

We are grateful to Nils Tomes who read an earlier draft of this paper and made a number of helpful comments, and to her and the other members of the AI in Design research programme at Edinburgh for many discussions during the development of these ideas.

The AI in Design research programme being conducted at the Department of Artificial Intelligence, Edinburgh University, is partially funded by the UK Science and Engineering Research Council under grant number GR/F/6200.1.

References

Akin, O. (1978). "How do Architects Design", in *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, J. C. Latombe, ed., North Holland, 65–98.

Barr, A. & Feigenbaum, E. A., eds. (1981). *The Handbook of Artificial Intelligence (Volume 1)*, Pitman.

Bazjanac, V. (1974). "Architectural Design Theory: Models of the Design Process" , in *Basic Questions of Design Theory*, W. R. Spillers, ed., North Holland, Amsterdam, 2–19.

Brown, D. C. & Chandrasekaran, B. (1984). "Expert System for a Class of Mechanical Design Activity", in *Proceedings of the IFIP WG5.2 Working Conference on Knowledge Engineering in Computer-Aided Design*, Budapest, Hungary.

Cagan, J. & Agogino, A. M. (1989). "Why AI-Design Researchers Should Distinguish Between Creative, Innovative and Routine Levels of Design", Intelligent Systems Research Group, Department of Mechanical Engineering, University of California at Berkeley, Working Paper 89-0202-0.

Coyne, R. F. & Subrahmanian, E. (1989). "Computer supported creative design: a pragmatic approach", in *Preprints of the International Round-Table Conference: Modelling Creativity and Knowledge Based Creative Design.*, Department of Architectural and Design Science, University of Sydney, Sydney, 35–68.

Foz, A. T. K. (1972). "Some Observations on Designer Behaviour in the Parti", MIT Press, MA Thesis, Cambridge, Mass.

Gero, J. S. (1987). "Prototypes: a new schema for knowledge-based design", Architectural Computing Unit, Department of Architectural Science, University of Sydney, Working Paper.

Gero, J. S. (1989). “Routine and Non-Routine Design: a prototype-based approach” , in *International Conference on Expert Systems in Engineering Applications*, Huazhong University of Science and Technology Press, Wuhan, China, 369–371.

Gero, J. S., Maher, M. L. & Zhang, W. (1988). “Chunking structural design knowledge as prototypes”, in *Artificial Intelligence in Engineering: Design*, J. S. Gero, ed., Elsevier, Amsterdam, 3–21.

Gero, J. S. & Rosenman, M. A. (1989). “A Conceptual Framework for Knowledge-Based Design Research at Sydney University’s Design Computing Unit”, in *Artificial Intelligence in Design, Proceedings of the fourth International Conference on the Applications of Artificial Intelligence in Engineering, Cambridge UK*, J. S. Gero, ed., Springer-Verlag, Berlin, 363–382.

Hayes-Roth, B. (1985). “A Blackboard Architecture for Control”, *Journal of Artificial Intelligence* 26, 251–321.

Hillier, W. & Leaman, A. (1974). “How is design possible”, *Journal of Architectural Research* 3, 4–11.

Hillier, W. & Leaman, A. (1976). “Architecture as a discipline”, *Journal of Architectural Research* 5, 28–32.

Jones, J. & Millington, M. (1986). “An Edinburgh Prolog Blackboard Shell”, Edinburgh University, Department of Artificial Intelligence, DAI Research Paper No. 281.

de Kleer, J. (1984). “Choices without backtracking”, in *Proceedings of the National Conference on Artifical Intelligence*, Austin, Texas, 79–85.

Lawson, B. (1980). *How Designers Think*, Architectural Press, London.

Logan, B. S. (1989). “Conceptualizing Design Knowledge”, *Design Studies* 10, 188–195.

Logan, B. S., Millington, K. & Smithers, T. (1990). “Assumption-Based Context Management in the Edinburgh Designer System”, Department of Artificial Intelligence, University of Edinburgh, DAI Research Paper No. 494.

Logan, B. S. & Newton, S. (1989). “The intractability of design: is design too complex for expert systems?”, in *Proceedings of the Australasian Conference on Expert Systems in Engineering, Architecture and Construction*, J. S. Gero & F. Sudweeks, eds., University of Sydney, Sydney, 283–302.

Maher, M. L., Zhao, F. & Gero, J. S. (1989). “An Approach to Knowledge-Based Creative Design”, in *Preprints of the NSF Engineering Design Research Conference*, University of Massachusetts, Amherst, 333–346.

Murthy, S. S. & Addanki, S. (1987). “PROMPT: An innovative design tool”, in *Proceedings of AAAI Sixth National Conference on Artificial Intelligence*, Seattle, 637–642.

Oxman, R. & Gero, J. S. (1988). "Designing by prototype refinement in architecture" , in *Artificial Intelligence in Engineering: Design*, J. S. Gero, ed., Elsevier, Amsterdam, 395–412.

Rosenman, M. A. & Gero, J. S. (1989). "Creativity in design using a prototype approach", in *Preprints of the International Round-Table Conference: Modelling Creativity and Knowledge Based Creative Design.*, Department of Architectural and Design Science, University of Sydney, Sydney, 207–232.

Simon, H. A. (1970). *The Sciences of the Artificial*, MIT Press.

Simon, H. A. (1973). "The Structure of Ill Structured Problems", *Artificial Intelligence* 4, 181–201.

Smithers, T. (1987). "The Alvey Large Scale Demonstrator Project Design to Product", in *Artificial Intelligence in Manufacturing, Key to Integration*, T. Bernhold, ed., North Holland, Amsterdam, 251–261.

Smithers, T., Conkie, A., Doheny, J., Logan, B., Millington, K. & Tang, M. X. (1990). "Design as Intelligent Behaviour: An AI in Design Research Programme", *Artificial Intelligence in Engineering* 5, 78–109.

Steadman, P. (1979). "The History and Science of the Artificial", *Design Studies* 1, 49–58.

Westerberg, A., Grossmann, I., Talukdar, S., Prinz, F., Fenves, S. & Maher, M. L. (1989). "Applications of AI in Design Research at Carnegie Mellon University's EDRC ", in *Artificial Intelligence in Design, Proceedings of the fourth International Conference on the Applications of Artificial Intelligence in Engineering, Cambridge UK*, J. S. Gero, ed., Springer-Verlag, Berlin, 335–361.

Woodbury, R. F. (1989). "Design genes", in *Preprints of the International Round-Table Conference: Modelling Creativity and Knowledge Based Creative Design.*, Department of Architectural and Design Science, University of Sydney, Sydney, 133–154.

```
beginSection: moduleClassHeader

    beginClassName
      class name [B] : universalBeam
      created :        November 1990
      created by :     bsl
    endClassName

    beginModification
      modified:    none
      modified by: none
      reason:      none
    endModification

    beginIsAkindOf
      none
    endIsAkindOf

    beginHasAsParts
      none
    endHasAsParts

    beginFigureReferences
      none
    endFigureReferences

endSection: moduleClassHeader

beginSection: parametersVariablesConstants

    beginParameters

      g$B    integer    -     #1    steel grade                           ##
      py$B   real    N/mm**2  #2    minimum yield stress                  ##

      sd$B   integer    mm    #3    nominal depth of beam                 ##
      sb$B   integer    mm    #4    nominal width of beam                 ##
      wb$B   integer    kg/m  #5    beam weight (linear)                  ##
      d$B    real       mm    #6    beam depth                            ##
      b$B    real       mm    #7    beam width                            ##
      tw$B   real       mm    #8    web thickness                         ##
      tf$B   real       mm    #9    flange thickness                      ##
      i$B    integer    m**4  #10   moment of inertia (x-x axis)          ##
      s0$B   integer    m**3  #11   plastic modulus (x-x axis)            ##

    endParameters

    beginVariables

      wi$B   real       N     #1    imposed load                          ##
      wd$B   real       N     #2    dead load                             ##
      wif$B  real       N     #3    imposed load (factored)               ##
      wdf$B  real       N     #4    dead load (factored)                  ##
```

```
    wbf$B  real       N     #5    self weight (factored)                   ##
    l$B    real       m     #6    beam length                              ##

    mc$B   real       N m   #7    maximum bending moment (mid-span)        ##
    s$B    real       m**3  #8    plastic modulus                          ##
    fv$B   real       N     #9    shear force                              ##
    pv$B   real       N     #10   shear capacity                           ##
    def$B  real       mm    #11   maximum permissible deflection           ##
    defx$B real       mm    #12   beam deflection                          ##

  endVariables

  beginConstants

    e$B    real  N/mm**2      #1  young's modulus                          ##

  endConstants

endSection: parametersVariablesConstants

beginSection: constraints

  wif$B   =   1.6 * wi$B                                         #1      ##
  wdf$B   =   1.4 * wd$B                                         #2      ##
  wbf$B   =   1.4 * l$B * wb$B * 9.81 * 10 ** (3)                #3      ##
  w$B     =   wif$B + wdf$B + wbf$B                              #4      ##

  mc$B    =   w$B * l$B / 8                                      #5      ##
  s$B     =   mc$B / py$B                                        #6      ##
  s$B     <   s0$B                                               #7      ##

  fv$B    =   w$B / 2                                            #8      ##
  pv$B    =   0.6 * py$B * a$B                                   #9      ##
  a$B     =   d$B * t$B                                          #10     ##
  fv$B    <   pv$B                                               #11     ##
  fv$B    <   0.6 * pv$B                                         #12     ##

  def$B   =   l$B / 360                                          #13     ##
  defx$B  =   (5 / 384) * (wi$B * l$B * 10 ** (12)) /
                          (e$B * i$B * 10 ** (4))                #14     ##

  functionTable   yieldStress                                            ##
  dimension: 2
  interpolation: match

  g$B     tf$B    py$B

      0       17      41      64      100
  43  275     265     255     245     245
  50  355     345     340     325     325
  55  450     430     415     undef   undef                         #1 ##
```

```
    catalogueTable  sectionProperties                                           ##

    sd$B    sb$B    wb$B    d$B     b$B     tw$B    tf$B    i$B     s0$B

    914     419     388     920.5   420.5   21.5    36.6    718742  17675
    914     419     343     911.4   418.5   19.4    32.0    625282  15474

    914     305     289     926.6   307.8   19.6    32.0    504594  12583
    914     305     253     918.5   305.5   17.3    27.9    436610  10947
    914     305     224     910.3   304.1   15.9    23.9    375924  9522

                                        .
                                        .
                                        .

    254     102     28      260.4   102.1   6.4     10.0    4008    353.4
    254     102     25      275.0   101.9   6.1     8.4     3408    305.6
    254     102     22      254.0   101.6   5.8     6.8     2867    261.9

    203     133     30      206.8   133.8   6.3     9.6     2887    313.3
    203     133     25      203.2   133.4   5.8     7.8     2356    259.8 #1 ##

endSection: constraints
```

The UB001 Module Class Definition File