Representing design knowledge in a definitive programming framework

W M Beynon and A G Cohn
Department of Computer Science
University of Warwick
Coventry CV4 7AL
Telephone:+44 (203) 523089 *Telex:* 31406 COVLIB G
FAX: +44 (203) 461606 *E-mail:* wmb@uk.ac.warwick

Abstract

Previous research suggests that a programming paradigm based upon the use of definitions is well-suited for the implementation of design support systems. This paper considers whether the choice of such a paradigm has implications beyond rationalising the implementation task.

This paper re-examines the thesis advanced in [5] that definitive ("definition-based") programming provides an appropriate framework for the implementation of CAD systems. Previous work on applying definitive principles to CAD has generated many technical ideas to assist implementation [3,4,5], but has been focussed primarily upon interaction at a rather low-level of abstraction. In this paper, we make a preliminary assessment of the merits of definitive principles from the perspective of "what is required of the system at the highest level of abstraction". Our eventual aim is to address such issues as: How can we most effectively represent functionally significant entities e.g. in such a way as to simulate behaviour within our model? In what sense can we represent and reason about the design process? To what extent can we develop an effective graphical interface? Characteristic of these concerns is the need to incorporate design knowledge in our internal representation and external presentation of design entities. There are two aspects to our theme: firstly, we shall review and further develop some of the key ideas introduced in [3,4,5]; secondly, we consider how these ideas are related to current thinking about representing and supporting the design process [15,16,17,18,19].

Our paper comprises three sections: a brief introduction, in which we review the background to the problems addressed by design support systems, and give a brief synopsis of the main ideas underlying the definitive programming paradigm; an examination of various ways in which definitive principles support knowledge representation, extending and elaborating the proposals made in [3] and [5]; a brief discussion of some relevant issues for intelligent CAD systems, with reference both to design methodologies, and AI techniques.

§1 Background

1.1 Problems encountered in design support systems

Systems for computer-aided design and modelling are centrally concerned with the representation of a complex geometric object in a manner that reflects many possible interpretations and functions. The mechanisms that are required to reference the components of such an object have to be very subtle, and there must also be ways to support many varieties of abstraction. An effective CAD system must combine convenient methods for representing both static relationships between the components of an object and the dynamic changes such as might correspond to reconfiguration of an object, or progressive stages in the articulation of a design. In developing a CAD system that deals "intelligently" with the user, there are major problems both in choosing an internal representation for user knowledge about geometric objects, and an external representation through a suitable user-interface. Solutions to these problems must also take account of performance considerations constraining the inference and data processing methods that can be effectively employed, and the implications of numerical approximation and singular behaviour familiar in computational geometry.

Previous research on the implementation of CAD software has focussed on two primary themes: language issues, concerned with the underlying programming paradigms to be exploited, and user-interface issues, concerned with the mode of interaction and data presentation.

Where languages are concerned, it is clear that there is scope within CAD systems for the exploitation of a variety of programming paradigms, including procedural methods, to express dynamic aspects of the design process and of user-computer interaction, and declarative methods, to express equational constraints and establish functional relationships. Current research projects in Intelligent CAD have typically combined procedural and declarative principles, synthesising procedural object-oriented techniques primarily suited to particular modes of abstract representation e.g. of geometric objects, and declarative techniques adapted from constraint-based programming and logic programming (c.f. [1]). Such a synthesis poses many technical difficulties: object-oriented techniques based upon message-passing prove cumbersome as a means of implementing static relationships between geometric objects for instance [20], and techniques to support constraint maintenance and geometric reasoning are notoriously hard to analyse in respect of termination and computational complexity.

Most of the work on user-interfaces has been focussed on the mechanics of interaction, and on graphical techniques intended to provide the user with a convenient framework within which to view and manipulate data. Current approaches deal very effectively with the routine aspects of dialogue sequencing and presentation graphics, but are not so well-adapted for deeper semantic concerns. For instance, it has proved hard to provide an adequate

representation for the stages through which the design process passes as a result of dialogue transactions, so that attempts to model the design process more formally have had to deal with a transaction history. Constructing adaptive interfaces poses similar problems.

Research on the CAD/CAM interface, and on Design Support Systems (such as the Edinburgh Designer System) indicates that a key problem in the traditional approach to design is that of retaining and distributing the knowledge about a particular product generated during the design process. An excellent account of the main issues appears in [16,17]. In the context of this paper, several particular aspects of representing design knowledge are of especial importance. The design process typically proceeds sequentially, but design knowledge is declarative in nature. Where "design to product" is concerned, there are several stages in the design process, and major problems of data consistency and management have to be resolved. Above all, there is a need to develop unifying principles to integrate the diverses data representations commonly associated with "islands of automation". We examine the prospects for addressing these issues through adopting a different underlying programming paradigm for design.

1.2 A programming paradigm based upon definitions

Our approach makes use of a novel programming paradigm based on definitions - "definitive programming". (In this paper, we assume familiarity with the potential applications of this paradigm to CAD systems as set out in detail in references [2,3,4,5], and merely summarise the principal ideas.) The simplest example of a definitive programming framework is provided by a spreadsheet that has been stripped of its tabular interface. Viewed in this way, each cell in a spreadsheet is a scalar variable whose value is either explicitly defined, or implicitly defined by an arithmetic expression in terms of constants and other variables. More sophisticated notations ("pure definitive notations") for interaction can be developed in a similar fashion. To this end, it is only necessary to devise an appropriate underlying algebra of data types and operators (analogous to the scalars and arithmetic operators underlying a spreadsheet), and to introduce variables of similar types whose values can be specified by algebraic expressions. Previous research has established effective methods both to design and implement pure definitive notations for the most complex applications [2,3,4,5,6]. For instance, a prototype line drawing system that has been applied to design furniture layouts within a room has been implemented, and work on the design and implementation of a definitive notation for general geometric modelling for CAD applications is already well advanced [4,5].

Pure definitive notations provide a framework within which it is easy to specify relationships between entities within the same semantic categories: for example, to assert that "the lamp is placed at the centre of the table", or that "the cable to the lamp is taut" (c.f. [4]). In developing more sophisticated modelling systems, it becomes essential to establish functional relationships between more disparate types of entity: for example, to express the fact that "the

moment of the door about the hinge, expressed as a function of its volume and density cannot exceed the load bearing capacity of the hinge" or "whilst the table obstructs the door, an appropriate error message will be displayed in a textual window, and the region in which the door and the table intersect will be highlighted in a graphical window". In yet more ambitious applications, it may be that the mode of user-computer interaction itself is determined via a functional relationship from a user profile and the current context. To support such applications of definitive principles, a more complex general-purpose programming paradigm based on definitions is required [5]. The key idea behind this generalisation is that the state of a suspended interaction or computation can be effectively represented by a system of variable definitions over an appropriate underlying algebra, and that this state is changed - subject to preconditions being satisfied - by the action of agents that corporatively represent the user and the computer. In this way, any computational step or dialogue transaction can be represented by a sequence of actions that either redefine existing variables or reconfigure the agents that are currently privileged to affect the current state.

The use of a definitive programming paradigm leads to a much closer integration of issues concerned with the user-interface and the abstract application. A characteristic feature of our approach is the use of an acyclic system of variable definitions for the subtle representation of state information, whether in respect of the abstract design process, or of the more ephemeral aspects of user-interface management. We expect that further development will have implications beyond solving technical problems of implementation at a relatively low-level of abstraction. Within a definitive programming framework, the manner in which the system state evolves through declaration or redefinition of variables is very different in nature from the state transitions in a procedural program. Its special qualities are well-suited to the representation of the "non-algorithmic" aspects of the design process, and we believe that it can provide an alternative methodology for CAD that departs from the essentially sequential nature of traditional engineering design. To justify this view, we must do more than apply definitive principles to the implementation of current design support concepts and techniques - challenging as this task may prove. We must identify respects in which the adoption of a definitive programming paradigm can alter our perspective upon design at the highest levels of abstraction.

Sections's 2 and 3 approach this objective from two directions. In §2 (a "bottom-up" approach), we examine the prospects for extending existing prototypes (such as the DoNaLD line-drawing system [3,4]) to support interaction at higher levels of abstraction. In §3 (a "top-down" approach), we consider how the use of definitions is related to current thinking about high-level design techniques and the abstract design process.

§2. *Applying definitive principles*

2.1. Definitive principles for data representation

The basic themes behind definitive principles for data representation have been set out in detail elsewhere (c.f. [2,3,4,5]). As discussed in [3,4], the DoNaLD and ARCA systems illustrate how definitive notations can be used to describe graphical objects in ways that support an underlying interpretation. An ARCA **diagram**, for instance, can be interpreted both as a combinatorial graph for display, and as encoding an abstract group structure. An important feature of the use of definitive notations is that it becomes possible to describe objects at different levels of abstraction. For example, some features of a complex diagram may be defined explicitly component by component, whilst others are defined implicitly in their entirety (see below). The use of definitions is also helpful in representing incomplete design specifications.

The range of techniques that have so far been conceived for representing graphical objects within a design environment will be illustrated with reference to DoNaLD. The basic data types in DoNaLD are the **point**, the **line** and the **shape**, where a **shape** comprises a set of **point**s and **line**s. The operators of the underlying algebra include constructors such as: finding the point of intersection of two **line**s, forming the union of two **shape**s, applying a rotation to a given **shape**, etc. A variable of type **shape** can either have its value defined in its entirety by an expression of type **shape**, or be defined incrementally through the declaration and definition of its constituent **point**s, **line**s and **shape**s. By way of illustration of basic definitive principles, consider the following DoNaLD description of a revolving door:

```
openshape rev_door
within rev_door {
    point axis
    openshape E_door
    within E-door {
        point hinge, lock
        line body
        real aperture, width
        body = [hinge, lock]    % hinge & lock are the endpoints
        lock = hinge + { width*cos(aperture), width*sin(aperture) }
    }
    E_door/hinge = axis
    shape N_door, S_door, W_door
    N_door = rotation (E_door, π/2, axis)
    W_door = rotation (E_door, π, axis)
    S_door = rotation (E_door, 3π/2, axis)
}
```

In this description, the variables declared as **shape** variables (viz N_door, S_door and W_door) have implicit definitions, whilst variables declared as **openshape** (rev_door and E_door) are explicitly defined. The role of a **within** {} clause is purely syntactic, so that, for instance, the declaration

point hinge

is to be interpreted as declaring a variable "rev_door/E_door/lock".

As outlined in §1.2, there are a variety of ways in which the use of definitions illustrated above can be elaborated. We can use definitions to establish relationships between objects, to specify simple parametric objects and linkages, or to describe the regions of shadow associated with a light source. By augmenting the underlying algebra, we can envisage handling more sophisticated line drawings e.g. admitting the possibility of **shapes** with infinitely many lines and points that can be recursively defined. For this purpose, we should need to introduce a fixpoint operator into the underlying algebra, in the spirit of functional programming. As discussed in detail in [5], we can also extend the use of the definitive paradigm in a consistent fashion so as to encompass broader issues, so that (for instance) monitoring geometric constraints is handled through defining the screen location and content of an error box appropriately. To handle such issues of constraint management as are involved in revoking a user dialogue action that obstructs a door, or in automatically closing an open door in so far as it is unobstructed, we adopt a general-purpose programming paradigm whereby several agents corporatively act through "definitive interfaces" as and when they are privileged [5,6].

In reviewing these developments, our present purpose is not to examine the technical problems posed by such a programme for the design and implementation of an intelligent CAD system (c.f. [5]). The issues we consider here lie beyond that stage. Let the indulgent reader imagine that our present DoNaLD prototype has been embellished to such a point that it is possible to describe revolving doors that cast shadows / are reflected in mirrors / cannot be obstructed by permanent architectural features / can be mechanically configured so that the N_door, W_door and S_door components are folded away in a locked position and the E_door acts as a conventional door etc. That the systems of definitions we conceive can in principle be used to support exceedingly rich and subtle data representations is at least very plausible. The deficiency in our proposed data representations is apparently not a lack of expressive power: what is problematical is how to achieve the information hiding required for successful use of the design system at higher levels of abstraction. Informally, our data representations fail to achieve the degree of abstraction that is commonly expected of knowledge engineering techniques to support the design process.

2.2 Definitive principles for knowledge representation?

The sophisticated extension of our DoNaLD prototype system conceived above merits closer examination. If we consider what is involved in interpreting actions such as introducing shadows and mirrors, changing the mode of operation of the revolving door, or simulating the passage of a person through it, it is clear that relatively complicated processes of redefinition are required. In effect, complex actions and concepts at a high level of abstraction are supported by our definitive data representation, but only in such a way that our definitions serve as an intermediate code for interaction.

By way of illustration, were we to simulate opening the revolving door by indicating the intended movement by means of a graphical interface (i.e. by "pushing the door open"), we should need to interpret "pushing the N_door, S_door or W_door" as "changing the

aperture of the E_door". This is of course a consequence of the particular choice of acyclic dependencies between the components introduced in specifying the revolving door. Following [5], we might have ensured that the dependency relationship between the E_door, N_door, S_door and W_door components was determined by which component of the door was pushed. That is, the internal representation of the revolving door is at all times matched to the expectations (the "intelligent view") of the agent acting upon the door. But whatever the nature of our implementation we choose, an abstractly simple interaction with the model typically requires a complicated translation.

What are the objections to using definitive principles merely as a vehicle for implementation, as "an intermediate code for interaction"? The consequences of such a strategy are thought provoking. In effect, the concepts underlying the adoption of a definitive approach will be invisible at the higher-levels of abstraction, so that other paradigms for data representation must then apply. In understanding the role that definitive principles can play in design support systems, it will be important to clarify whether this is the case - and, if so, why.

We may conjecture that the application of definitive principles is most relevant for the specification of precise functional relationships that are most naturally defined algebraically, and that such relationships are commonly to be found at lower levels of abstraction. For instance, we can readily perceive the functional relationships between components that describe a rocking chair, a folding table or a filing cabinet in algebraic terms, but cannot describe the constraints that govern the disposition of these objects within a room without recourse to a more abstract representation based upon Boolean algebra and geometric predicates. We can gain further insight by considering alternative ways in which definitive principles might be employed in applications such as architectural design. In principle, we could after all design an underlying algebra specifically for such an application, and include data types such as doors and windows as primitives. The relative paucity of operators that can be applied to construct objects such as doors and windows from components at the same level of abstraction is surely connected with the difficulties of describing the characteristics of a furniture layout in algebraic terms identified above. It seems unlikely that a definitive notation over such an algebra would prove more effective than our extended DoNaLD system at describing relationships between architectural features.

This is not to say that other kinds of abstract representations for doors and windows are inappropriate. A door can be modelled, but only as a set of points and lines with a rather complex protocol for manipulation. It would obviously be desirable to introduce more complex variables into DoNaLD to facilitate the definition and manipulation of doors. The full technical details of such an enhancement do not concern us in the present context, but the declaration of **openshape** variables to represent conventional and revolving doors respectively based upon macro expansions of the definitions of E_door and rev_door given in §2 might be an appropriate ingredient. Such definitions could be complemented by appropriate suites of actions to simulate opening doors, as outlined above. The significant point here is that the paradigm we are using in this process of abstraction is no longer "definition-based"; it is the

type of procedural abstraction commonplace in object-oriented systems, albeit implemented on top of a definitive framework. Such observations bring to mind the well-recognised qualities of frame-based representations for modelling objects at a high level of abstraction, and their limitations as a means of abstractly formulating precise relationships between objects (c.f. [20]).

In the context of other design environments, it is possible that we should wish to introduce definitive principles at a higher level of abstraction. Were we to apply an extended DoNaLD system to support (say) gear design, we could perhaps conceive appropriate relationships between gears that could be modelled algebraically at a higher level of abstraction. In that case, we would not wish to be confined to the particular framework for data representation imposed by the choice of the DoNaLD underlying algebra. Our strategy would be to devise a "gear algebra" whose data types operators could be compiled into complex shapes and functions defined upon them (c.f. the DoNaLD to EDEN translation described in [4]). Without such enhancement, all our evaluation would be carried out at a fixed level of abstraction, leading to a data representation too flat relative to that supported by a typical object-oriented or functional programming system. Note carefully that our concern is not just that our data representation is otherwise at too low a level of abstraction, but that we need the flexibility to work at many levels. Nor is the existence of techniques for describing values in the underlying algebra at different levels of abstraction relevant: it is the nature of the values themselves that is at issue.

§3 Clarifying the role of definitive principles in design support

3.1. Low-level definitions meet high-level abstractions

The arguments advanced above indicate that we cannot reaonably expect to build design support systems that use definitive principles in an orthodox way at every level of abstraction. We have acknowledged that at some level of abstraction it becomes impossible to specify the relationships between design objects simply by formulating variable definitions over a sophisticated underlying algebra. Our objective in this section is to propose a framework within which definitive principles can perhaps most appropriately assist the modelling of higher-level abstractions.

In investigating the role that definitive principles can play it is helpful to contrast the methods of data representation discussed above with those conventionally used in a Design Support System, as described in [16]. The distinction between generic knowledge about the application (the "Encyclopaedia") and knowledge specific to the product being designed (the "Design Description Document") have obvious counterparts in the underlying algebra and the representation of a product through a system of definitions and constraints in our design dialogue ([16] §2.1). Data tables may be seen as methods of specifying operators in the underlying algebra, and the usefulness of relational databases such that "some tables have symbolic entries" clearly hints at a definitive paradigm. The emphasis in [16] is very clearly upon formulating design decisions as predicates, and applying inference engines to derive the consequences of assumptions made by the designer. What seems clear is that the explicit representation of algebraic and geometric

relationships that definitive principles allow has very considerable advantages of efficiency (c.f. [16] §2.4.2 where the need "to reduce complexity arising from the existence of multiple, rather simple, relationships" is acknowledged).

The issue of how our approach relates to knowledge representation techniques based on inference bears closer examination. In one respect, it is clear that we are concerned with the relative advantages of formulating constraints as equations rather than as functional relationships - a theme explored at length in [16]. For instance (c.f. [16] 2.4.2) we should typically expect to guarantee that a bearing seating *bs* of a shaft fits the inner ring of a bearing *bg* by formulating one or more acyclic systems of definitions relating the variables *bs* and *bg*, and using whichever system was appropriate to model the consequences of a changes in parameter values that might affect this relationship. In effect, the use of definitive principles aims to impose some discipline upon the methods of constraint management invoked, substituting the identification and direct application of a preconceived strategy for constraint maintenance for an open-ended search for a solution. Closer consideration suggests that definitive principles can mimic inference in more unexpected ways, however.

Let us return to the consideration of applying definitive principles to a system that supports the design of furniture and room layouts, as discussed in §2. We may imagine that our objective is to build a system that not only allows us to model and experiment with the objects and locations of objects within a room, but can also be used to simulate their behaviour as faithfully as possible. Amongst the objects we should expect to describe are: doors, windows, cupboards, desks with drawers, table lamps, sockets, cables etc. In designing these objects we should expect to perform actions such as relocating doors, changing the size of a table, or changing the mode of operation of a window etc. In simulation, we should expect to be able to open windows, place the table lamp on the table, clamp the table lamp on the desk, change the socket at which the cable is plugged in, display the shadows that result from switching on the light, move the table across the room etc. We should expect to have a simulation that was realistic in so far as it was impossible to move the table through the wall for instance, to carry the table lamp beyond the range of its cable, or to directly open the door when it was obstructed by the table etc.

The techniques we shall use to represent objects and their current locations, and to monitor and impose constraints, have already been discussed in detail. We may assume that the mechanisms required to manipulate objects at a high level of abstraction have been implemented, so that e.g. we can speak of relocating the table at a specified position. Superficially, there should be a big distinction between the way in which we represent the consequences of parameter changes (such as changing the width of the door), and those of carrying out other procedural actions (such as placing the table at right angles to the wall). The effect of changing the width of the door will probably be to change the values of a few variables whose dependence upon the door width is expressed through explicit formulae. The effect of relocating the table should in some sense be very much harder to establish, but in the context of our system will correspond quite simply to changing certain parameters that describe the position of the table, and observing what

implications these changes have for other objects. If the centre of the table lamp is located at the centre of the table by an appropriate definition, the lamp will move with the table, and the cable will take up an appropriate new position (c.f. [4]). The key point here is that the low-level definitions that prescribe the relative locations of objects such as the table, the table lamp and the cable, perform what should by rights be a very abstract function. To elaborate this point: what kind of inference would be required to determine the implications of relocating the table? There is perhaps no satisfactory answer to this question - in effect, what happens when the table is relocated depends entirely upon what is in the designer's mind - whether the intention is that the lamp should remain on the table, or should retain its present position, and whether the fact the cable is too short should obstruct the relocation. The purpose of the low-level definitions is to this extent simply to articulate the designer's intention.

Suppose now that we operate in a context within which inference can be appropriately invoked: that the table lamp is placed at the centre of the table, and we simulate movement of the table a short distance in some direction. There are many potential complications to be considered. Is the surface of the table assumed frictionless? If the table is obstructed by a chair will the chair slide in the same direction, or tilt on its rear legs? Perhaps the cable is taut. Will the cable stretch? Will the lamp move towards the plug? Will the plug come out of the wall? Will the wall move? All these scenarios assume that the context doesn't radically change, so that the cable doesn't become taut during the course of the movement, or the table encounter a second table tight against the wall. What is clear is that the use of inference is fraught with problems, and we may doubt whether a rational person would attempt the classical recipe for tooth extraction that involves a string and a door, for fear that the door would come off its hinges. (This problem has been well explored in AI within the context of defining the effects of actions - where it is called the *frame problem* [7] - and in the related context of choosing extensions to nonmonotonic theories[11].)

The use of definitive principles for simulation is in contrast far from logical, even if it may be deemed *a*logical rather than *ill*ogical. To simulate the movement of the table we have first to decide what frame of definitions we should adopt to suit the action we intend to perform. Whether there are good prospects for determining the appropriate frame of definitions - that dynamically changes as we carry out a simulation - without re-invoking the complexities faced by an inference approach is an issue beyond the scope of this paper. Clearly there are no prospects unless we make plausible assumptions regarding the uniformity of the environment being simulated.

To give more credibility to the application of definitive principles in simulation, we may briefly consider a simple example that is illustrative of a method that may have wider applicability. Consider a set of blocks free to slide in a uniform tube. If we select a block, and propose to move it left or right through a specified distance, we can see how to model the implications of such a move applying definitive principles. We first refer to the disposition of the blocks, and from this determine the system of functional relationships between positions of the blocks that should correspond to an infinitessimal movement of the selected block in the prescribed direction. For instance: suppose that the unit-size blocks A, B and C are adjacent,

that they are centred at the points a,b and c respectively, and that there is no block to the right of C. On moving A to the right, the appropriate system of functional relationships is

"b=a+1; c=b+1"

whilst on moving B right it is simply "c=b+1". To simulate the entire movement of A through a specified distance, we must incrementally move A to the right until the entire movement is completed, invoking each new definitive context as it is encountered. Thus if C comes into contacts with the block D, the modified system of functional relationships

"b=a+1; c=b+1; d=c+1"

is invoked etc. The significance of this example, simple as it is, is that it illustrates that it is sometimes possible in the course of a simulation to derive the current definitive context directly from a 'map' (because ABCD are adjacent, therefore the system of functional relationships

"b=a+1; c=b+1; d=c+1"

applies), so that the process of simulation consists of repeatedly: consulting the map, determining the current context, updating the map accordingly. It remains to be seen whether similar principles can be applied to more complex situations, but it seems plausible that this is characteristic of at least some of our cognitive models.

The above discussion leaves many issues unresolved. To handle the evaluation of definitive contexts, there is patently a need for abstract methods for describing and manipulating systems of definitions - a need that is independently motivated by the interpretation of editing operations such as the creation of symbols or "cutting and pasting", by the consideration of objects whose mode of operation can be changed (c.f. the revolving door), and by the abstract specification of operators that e.g. "introduce the shadows when the light is switched on". It will also be important to relate the use of definitive principles to the theory of databases, and to the challenging critique of conventional database technology in [12].

3.2 Definitive principles and the design process

The application of a definitive programming paradigm to CAD systems can be viewed from many different perspectives. From an engineering viewpoint, our work is most closely connected with the use of parametric objects, and with paradigms for design in which such parametrisation is a central feature [18,19]. The L.E.G.O language [13], and the "retrospective planning" paradigm for design developed by Takala both use similar principles to express relationships that can be captured very effectively - and with less complicated semantics - by using definitions. Our methods are particularly relevant to the concerns of CAD/CAM, in which it is important to integrate geometric modelling with additional information such as is required for manufacturing process planning. From a cognitive psychology viewpoint, the thesis that a system of variable definitions is a useful metaphor for a user's intelligent view of a system is supported by previous research, in which the merits of spreadsheet principles are established. The concept that an agent's perception of a system, and his/her expectation of its dynamic behaviour at any time can be represented by such a system of variable definitions (c.f. [5]) is relevant both in the design of the user-interface, and in providing support for dynamic interaction with the model.

Of most potential interest are the implications for those issues addressed by Engineering Design Support Systems viz the intelligent manipulation and interpretation of design products, and the management and logical analysis of the design process itself. There are several ways in which current AI techniques can in principle be integrated into our framework. As discussed in §3.1, definitive models for designed artifacts can be readily specified in such a way as to model expected or intended behaviour. If the lamp is defined to be in the middle of the table, the lamp will move with the table - provided that the cable to the lamp is long enough. If the cable becomes taut, then the relationship between the lamp and the table has to be modified, and a new system of definitions invoked. There is a clear connection here with research on qualitative reasoning and qualitative physics [9], in which the objective is to develop methods of reasoning and simulating physical systems in the absence of precise quantitative information, without resorting to probabilistic or "fuzzy" logic. The application of qualitative reasoning in this area would itself be an important innovation. Existing applications have primarily been concerned with electronic or pneumatic systems, and there is considerable interest in extending these ideas to cope with mechanical systems. We anticipate that the use of qualitative reasoning principles within the definitive programming framework will lead to new insights into qualitative spatial descriptions to support a wide variety of inferences, and suggest novel methods for integrating qualitative and quantative reasoning. The conceptual apparatus of Forbus' Qualitative Process Theory (QPT) is possibly the most appropriate in this context, not only because it has a rich ontology, but because it incorporates a directionality resembling that implicit in an acyclic system of definitions [10]. Furthermore, the notion of *individual views* to be found in QPT has similarities with our idea of alternative systems of definitions. Moreover, the explicit notion of process in QPT may also be used to model various transformations to be performed on a design.

Some of the most interesting issues concern the potential use of logic concepts and principles - an innovation that will require careful evaluation, as the discussion of inference in §3.1 illustrates. Logic is the cornerstone of contemporary approaches to intelligent CAD. For reasoning about the design process, the introduction of an appropriate temporal logic system to support a user-dialogue is often advocated. For managing suites of designs, as is appropriate e.g. when comparing, contrasting and evaluating alternative designs, it is important that a design support system should allow convenient simultaneous representation of multiple partial designs. For this purpose (c.f. [16]), we may wish to apply truth maintenance system (TMS) principles [14]. An alternative possibility that naturally comes to mind in this context is that of applying definitive principles to the formulation of design constraints. In effect, as we can to some extent simplify the equational constrant maintenance through using definitions, we may be able to improve efficiency in inference through applying similar principles to the resolution of logical constraints. There may also be a useful role for symbolic manipulation of defining expressions, and the application of deductive techniques concerned with inference over a many-sorted algebra (c.f. [8]).

Is it possible that definitive principles can aid our understanding of the abstract process of design? Clearly, definitive principles are related to the simple paradigm of parametric design, in which we vary parameters and monitor design constraints (compare the

traditional use of the spreadsheet). Definitive methods allow such frameworks to be constructed conveniently, and could easily support activities such as the adaptation of parameter values through automatic response to constraint analysis.

Though such parametric design may be an important ingredient in the design process, it is clear that much more than this is envisaged in the work of designers such as Lansdown [15], and in projects such as the Edinburgh Designer System [16]. Some of the most suggestive evidence for the relevance of definitive principles even at high levels of abstraction in a design may perhaps be seen in music, in the fugues of Bach, or in compositions that exploit motif development. By way of illustration, if Beethoven's 5th symphony began with the motif "CCCE*b*" and not "GGGE*b*", we should expect that the entire movement would need to be revised, but could probably be revised consistently, in such a way that the musical development springing from the original motif was appropriately transformed. For instance, we might expect the introductory bars ("GGGE*b pause* FFFD *pause*") to be transformed to "CCCE*b pause* DDDF *pause*". An intelligent appreciation of Beethoven's 5th involves recognising how many musical fragments in the first movement are derived from the opening motif; if we change the first motif, we must also change these subseqent references and elaborations. In a definitive programming setting we could perhaps - in principle - go further and articulate the precise correlation between the opening motif and its derivatives, but if this is possible it is a very sophisticated kind of musical exercise (viz writing a parametrised symphony of which Beethoven's 5th is one instance!). Nonetheless the latter is probably closer in spirit to Beethoven's musical insight in "designing" the 5th symphony: it is conceivable - if doubtless untrue - that earlier sketches for the symphony began with the motif "CCCE*b*", and were subsequently systematically revised to reflect inversion of the original motif.

In this context, it can be argued that musical appreciation involves apprehending a design (if not necessarily through erudite analysis, through acquired familiarity and subconscious association of ideas). Definitive principles offer prospects for explicitly representing "the design", so that a musical score can be formally annotated by cross references indicating possible dependencies between motifs and musical fragments. But just as Beethoven's 5th symphony is not just the result of applying an operator to the motif "GGGE*b*", and there are many different plausible ways to model the score using definitions, we have to accept the need for many parallel alternative interpretations by means of definitions, not necessarily consistent, as when a composer derives a musical fragment that can be ambiguously interpreted as an elaboration of more than one previous motif.

Musical composition - within some idioms at least - typically involves processes such as: defining new ideas from old through the use of operators such as augmentation and inversion; assembling contrasting musical episodes; reworking episodes ("tinkering with parameters") whether to attain greater unity; or to introduce deliberate ambiguities of interpretation into the design ... to some extent such devices used by classical composers could be usefully represented in a definitive programming framework. This is perhaps the best evidence that definitive principles are relevant even to abstract design.

References

1. P Bernus, P J W ten Hagen, P J Veerkamp, V Akman, *IDDL: The Language of a Family of IIICAD systems*, in Intelligent CAD Systems 2: Implementation Issues, Springer Verlag (to appear 1988)
2. W M Beynon, *Definitive Notations for Interaction*, Proc hci'85, CUP 1985, 23-34
3. W M Beynon, *Definitive Principles for Interactive Graphics*, NATO ASI Series F:40, 1987, 1083-1097
4. W M Beynon & Y W Yung, *Implementing a Definitive Notation for Interactive Graphics*, 456-468
5. W M Beynon & A J Cartwright, *A Definitive Programming Approach to the Implementation of Intelligent CAD systems*, in Intelligent CAD Systems 2: Implementation Issues, Springer Verlag (to appear 1988)
6. W M Beynon, M D Slade, Y W Yung, *Parallel Computation in Definitive Models*, Proc CONPAR'88 (to appear)
7. F M Brown (ed) The Frame Problem in Artificial Intelligence, Morgan Kaufmann, Los Altos (1987).
8. A G Cohn, *A More Expressive Formulation of Many-sorted Logic*, J Aut Reasoning Vol 3, 1987, 113-200
9. A G Cohn, *Qualitative Reasoning*, in Special Topics in Artificial Intelligence (Proc ACAI88), ed R Nossum, Springer-Verlag (to appear 1988)
10. K D Forbus, *Qualitative Process Theory*, Artificial Intelligence, Vol 24, 1984
11. S Hanks and D McDermott, Default Reasoning, Non Monotonic Logics and the Frame Problem, in Readings in Nonmonotonic Reasoning, ed M L Ginsberg, Morgan Kaufmann, Los Altos (1987).
12. W Kent, *Data and Reality*, basic assumptions in Data Processing, North Holland (1978).
13. N Fuller, P Prusinkiewicz, *Geometric Modelling with Euclidean Constructions*, New Trends in Computer Graphics, ed. Magnenat-Thalmann, D Thalmann, Springer-Verlag 1988, 379-391
14. J de Kleer, *An Assumption-based TMS*, Artificial Intelligence, Vol 28, 1986

15: J Lansdown, *Graphics, Design and Artificial Intelligence*, NATO ASI Series F:40, 1987, 1153-1174

16. R Popplestone, T Smithers et al, *Engineering Design Support Systems*, IKBS/Mailshot 7, 1986

17: T Smithers, *AI-Based Design v Geometry-Based Design*, Workshop on AI in Civil Engineering, AI Applications Institute, Edinburgh University, November 1987.

18: T Takala, C D Woodward, *Industrial design based on geometric intentions*, NATO ASI Series F; Computer and Systems Sciences, Vol 40, 953-964

19. T Takala, *Design Transactions and Retrospective Planning Tools for Conceptual Design*, in Intelligent CAD Systems 2: Implementation Issues, Springer Verlag (to appear 1988)

20. T Tomiyama, *Object-oriented Programming for Intelligent CAD Systems*, ibid.