1

A Modelling Method for Experiments in Design

1.1 Introduction: Design as an experimental activity

Design presupposes a designer. That argument for a Creator is philosophically attractive. It is also an argument that suggests that engineering design cannot be automated to the exclusion of the human designer.

The link between design and designer is *choice*. Given the myriad possibilities of building the desired product from the synthesis of ideas with intransigent physical reality, decisions have to be made. The conceptual stage of the design process cannot be reduced to methodical procedures, despite some attempts to generalise methods that were devised to deal with certain routine tasks. One must carefully distinguish routine tasks from non-routine. Failure to do so leads to the belief that it is possible to automate design totally. Roth, following the German tradition for methodical procedures, thought that within ten years, methodical design would dominate and pure intuition become the exception. [Roth, 1981]. His interest lay in automating tasks that are well de-fined and for which solutions may be formulated that enable speedy choice of a relatively few options. But, as David Pye of the Royal College of Art put it

> "It is said that by the aid of computers we can arrive at the correct solution .. with certainty. They are clever, these computers! They are going to show us the cheaper answer. But if they think their clients are going to be satisfied with that, they are not so clever as they think." [Pye 1983].

Contrasted with the 'methodical' school is recent American research on design systems. At Carnegie and Stanford "non-routine" tasks within the design process

are defined by Piela *et al,* as "ill-defined, because they involve incomplete task descriptions and non-deterministic solution paths." They also support the notion that human interaction is essential to design: "Because non-routine design requires on-going human intervention, facilitating designers in this work will depend crucially on how well systems support use" [Piela, 1992]. Piela' s work is significant in highlighting the experimental nature of conceptualisation. In the days before computer models replaced the difficult and expensive process of physical prototyping it was well understood that the nature of the physical world is such that frequently more could be extracted from a physical model than might be anticipated when the model is conceived. The act of building the model may reveal hidden manufacturing or feasibility problems, or simpler ways of achieving the same functionality.

In this work a design philosophy is developed that shows that designs are built on the designer' s perception and experience of the physical universe. By experiment those perceptions are changed and improved. Thus they are based upon observation, both personally acquired and passed on.

The number of observations of the physical universe that can be made is infinite, both in category and within category. Thus one speaks of "design space", or "search space", by analogy with linear programming, (see, *e.g.* [Starkey, 1992]) by which attempts are made by specification to make limits within which the search for a feasible solution can be investigated in the design time allowed. In pre-computer days the design space was often narrowed to the extent of allowing only one solution: the first one that could be done that looked feasible. Despite that the environment for the search was still experimental. Indeed the word "search" well illustrates the experimental flavour of design. The search may not be done according to a preconceived specification but may be modified in the light of each finding. As the observation base grows so the problem definition evolves and with it the product. A client comes to a designer wanting a wall for his garden and goes away happy with a hawthorn hedge because the designer probed the reasons for the wall rather than merely discussing the type of wall.

A further strand to experimental design is developed in this work, namely the idea of *agents* in design. Design is normally the function of more than one person or agent, each interacting with the design, and often independently. An understanding of the way that agents operate is essential to a “theory of interaction” in design. The agent may be a person, making inspections or changes in the state of a design,

or the agency may be built into the design itself. For example a simple 4-bar linkage may have alterations made to its geometry by the designer: however a change in the angle of the driving lever will cause the whole mechanism to change its state by virtue of the connectivities of the bars. Agents often have choices: a warning light can be disregarded, constraints can be violated or redefined. Indeed these choices often provide the way that design progresses. Ways need to be found for coping with that level of interaction. In particular the issues of simultaneous and conflicting interactions are crucial in cases where there is a number of independent agents.

This thesis is concerned with the experimental aspects of conceptual design and how to support observation and experiment on computer-based systems. It is argued that any attempt to create such design support systems must provide an environment with the following properties.

- It is able to record observations of the real world.
- Computer models can be set up with attributes of physical prototypes, *i.e.* they have states that can be manipulated in infinitely many ways.
- It puts no constraint upon invention, particularly in the early stages.
- It allows interaction with the design by different agents acting concurrently.

On the other hand, in experimental investigations on how designers do design, it was observed that there is constant alternation between searching for a solution and fixing a solution; fixing the solution proceeds rapidly from success in the search for a solution [Ehrlenspiel & Dylla, 1989]. Access to standardised or automated procedures becomes advantageous at the point of fixing solutions, so methodical approaches need to be integrated. The assertion that the attempt to automate design is futile does not invalidate advances in methodical design. The two areas are complementary. The scope of the thesis is the conceptual stage of design.

1.2 A New Modelling Method

Interaction is fundamental to most CAD systems. It is that which has made them useful in conceptual design. However that is one of the few points in their favour. The modelling of a design on conventional CAD leaves the interpretation of the model firmly with the user. Modelling a design with real ‘meaning’ and enabling the design process are extremely difficult to do and are a major area of current research. Recently developed design support systems point up the importance of

appropriate computer-languages for design tasks, but also expose deficiencies in programming methods. For example, the modelling of state and state change is an important aspect of design but a traditional computer program *describes* state rather than itself being thought of as having state.

In this work a new programming language has been developed by the author to support the central thesis that computer models to help design should be situated in reality rather than being abstracted away from reality. Current programming techniques rely upon pre-selection of a set of observations with their own logical integrity in order to create modelling methods to aid the designer. Those methods mean that the designer effectively enters a logically predictable world where autonomous agents (such as, for example, those that are familiarised by the phrase "Murphy' s Law") do not operate. In contrast, I created the language EdenLisp to permit a more realistic design world. It is developed from a programming method (paradigm) called the Definitive Programming framework, first developed at Warwick University. Using this method, it is possible to make a computational model as a metaphor of physical reality that allows one to examine that real world as one might by experiment and observation.

The definitive programming concept may be introduced by reference to one of the antecedent languages of EdenLisp developed by [Beynon & Yung, 1987] called EDEN.

> "The key idea behind definitive programming is the representation of computational state by a set of definitions of variables and of a transition between states by a set of redefinitions. A simple application of this principle underlies the spreadsheet. By way of illustration (when augmented by definitions of voltage *etc.*) the set of definitions
>
> ```
> resistance = resistance_of_lamp + cable_length * coeff_of_resistance
> current    = if switch_on then voltage / resistance else 0
> light_on   = switch_on and current >= threshold
> switch_on  = false
> ```
>
> can be interpreted as describing the state of a simple electrical circuit. In this context an appropriate transition might involve the redefinition
>
> ```
> switch_on   =   true."
> ```
>
> [Beynon, 1990]

Each new definition is interpreted in the light of previous definitions. If dependent variables are without current values the relationship remains unevaluated; only

when they all have values is the definition evaluated. If any variable acquires a new value at any time the definition is immediately re-evaluated. That point is significant. In conceptual design an object may be an instantiation of an idea defined initially in very abstract terms and only at the last "put into flesh". The definitive method allows such an approach. An object can be defined initially by formulae using labels that have no current values, nor even references to values. Nevertheless the formulae are binding and remain so when other definitions are made which build on them. In principle one could build up design specifications without needing to give values to important features, merely referring to them by labels: indeed one can construct a veritable algebra of labels in which ideas gradually attain substance.

As the definitive method was applied to design it is shown that it has the outstanding feature of being the means of creating computational objects that are each a kind of metaphor of a physical prototype. States defined by a set (or *script*) of definitions can be examined as one might physical states of real objects: no definitive statement is merely an adjunct to the computation. A redefinition is akin to changing a feature of a physical prototype or its state

For the interpretation and evaluation of definitive statements the techniques common to creating new languages are used. The original Evaluator of DEfinitive Notations (EDEN) was created by Beynon and Yung [Yung, 1987], and a number of derivatives of Eden were also developed. These definitive methods were the starting points for developing the main ideas described in this thesis, ideas that underlie the tools that are proposed for supporting design. For the engineering designer, a tool that uses Definitive methods would have little application without access to geometrical modelling: so *EdenLisp* was conceived. EdenLisp is a definitive CAD notation containing the basic Eden engine, but considerably enhanced with geometric extensions. As the name suggests, it is written in Lisp. It is specified by identifying an underlying algebra of sorts and operators to describe many different structural aspects of a geometric object.

Engineering design problems using EdenLisp show that the properties of the definitive method are indeed analogous to Engineering Design prototyping. Engineering models written in EdenLisp can be manipulated in a similar way to parametric models that sit on top of most existing CAD systems. The difference is that because it is a language approach there is no point at which the parametric

concept breaks down: it is definitive all the way down! Indeed it is conceivable that eventually all computation can be expressed using definitive scripts.

With that introduction the main "thesis" of this work may now be stated.

1.3 Thesis: Design Prototyping with Definitive Scripts

The process of engineering design, defined as a synthesis of physical observations and deductions with incremental refinement, may be modelled accurately on a computer by definitive, or agent-oriented, programming.

The author' s Definitive Script interpreter EdenLisp provides a computer-based approach whereby models of systems can be originated that have state and on which state change can be made by independent agents

Computer models produced that way can have state that is characteristic of engineering prototype designs of physical systems. The explicit representation of state at the lowest levels permits experimentation, observation of properties and addition of further observations. Development of the design exploits interactions, constrains certain directions and allows concurrent agents of change.

The interactive construction of definitive scripts is analogous to the conceptual design process. It may be used to illustrate and test design meta-theories for modelling conceptual design. It is also an excellent vehicle for design education.

1.4 Overview

The ordering of the chapters following reflects the intertwining strands of design and computation. First, the experimental nature of conceptual design is considered, highlighting the modern requirement for rapid prototyping of products and hence the need for design product models that exhibit state and state change behaviour similar to physical prototypes. Trends in concurrent engineering and the growing interaction of non-engineering experts are examined. The aim is to identify the kind of computer-based tools that can cope with very different kinds of interaction on the same design.

Second, the novel idea of computation as experiment is introduced. Properties of older definition-based methods used in engineering are examined and the development of Definitive methods described and contrasted with them. Methods

for coping with multi-agent systems are still emerging but promising progress is described.

Definitive tools for graphical interaction are described next, defining the requirements for design in terms of abstract object definition. An algebra based upon these abstractions is developed out of which grew EdenLisp. The design philosophy and structure of the language of EdenLisp are described. The link with the CAD system AutoCAD® illustrates the flexibility of the method in accommodating traditional programming styles whilst exploiting the power of prototyping. In contrast experiments using definitive notations developed by others in the research group are reported to reinforce particular issues in prototyping and to show the generality of the method.

In the next two chapters the implementation of, and experiments in EdenLisp are described in detail. EdenLisp is implemented as a strongly typed language that is easily extended as new operators are introduced. The lexical analyser, parser and type checker are described, together with the evaluator. The environments for dealing with geometrical objects are then described. Finally programming examples are used to show the input style and possible user environments.

Design Management is very much the issue when there are design teams consisting of people with very different skills. The idea of definitive scripts is readily adapted to prototype definitive designs that have multi user, multi task design scenarios. The possibilities are explored in that chapter.

Applications of definitive methods in design education are very attractive. Parametric design, optimal design, animation and self-teaching are each illustrated with examples that show the power of the method

The work is completed with a discussion of the topics dealt with, evaluating the thesis from the perspective of likely developments and future research.