Abstract Geometry for Design in an Empirical Modelling Context

Meurig Beynon † Richard Cartwright † Alan Cartwright ‡
Yun Pui Yung †
wmb,carters@dcs.warwick.ac.uk, esraf@eng.warwick.ac.uk

†Department of Computer Science, University of Warwick, Coventry, CV4 7AL, U.K. +44 1203 523193
‡Department of Engineering, University of Warwick, Coventry, CV4 7AL, U.K. +44 1203 523123

Abstract

We review research stemming from an agenda set out in a previous paper [6], which investigated the application of definitive programming principles to the implementation of CAD systems. This has led us to the development of a new Empirical Modelling paradigm. We outline the principles of Empirical Modelling, and explain and illustrate the significance and potential role of definitive notations for shape such as *CADNO* [6, 20], *EdenCAD* [19] and *HyperJazz* [4] in this context. We also describe the motivation for Empirical Modelling as a modelling method for open development that can assist the integration of geometric modelling, non-geometric modelling, and meta-modelling aspects of concurrent engineering.

1 Introduction

The use of families of definitions ("definitive scripts") to represent geometric objects is well-established [12, 13, 3, 10]. An early precedent for their use can be found in [33]. The term *definitive programming* was introduced in [6] to refer to the use of definitive scripts to represent computational state. The main theme of [6] was the application of definitive programming methods to the implementation of CAD support systems, with particular reference to a proposed design for a definitive notation for geometric modelling (*CADNO*). This paper is a sequel to [6], in which the principal subsequent developments are reviewed, and their implications examined.

An agenda for future work was identified in [6]:

a. pragmatic development of definitive notations for application in CAD, and of large software systems based on definitive programming principles;

b. the development of a new programming paradigm, based on the preliminary concept of the Abstract Definitive Machine (ADM);

c. a further study of the principles for interaction involved and their associated applications.

Since [6], there has been further work on the design of definitive notations for geometric modelling both at Warwick and elsewhere [19, 20, 3, 4]. This paper is primarily concerned with clarifying the motivation, context and significance of this work, rather than making a new technical contribution to shape modelling. This reflects the fact that many aspects of the above agenda have been addressed without making use of a sophisticated definitive notation for

geometric modelling. The latest software prototype we have developed (the *tkeden* interpreter) incorporates only the simplest kind of definitive notation for specifying geometry, viz. the definitive notation for line drawing *DoNaLD*. *DoNaLD* has its limitations even as a notation for planar line drawing. For instance, it lacks any concept of layering, and has no built-in support for attributes of geometric entities. Nonetheless, *tkeden* has been the basis for a wide range of case-studies that are distinguished not by the quality of their graphical content, but by the subtlety and complexity of the interrelationship between geometric entities and the "real-world" semantics of the model. (For further illustration and explanation, the interested reader may consult previous case-studies, which include game simulations [23], exercises in mathematical visualisation [8], railway modelling [24], and lathe shaft design [1].)

More importantly, our exploration of the agenda proposed in [6] has led us to a radical change of perspective. This has involved such a re-orientation that it is no longer appropriate to think in terms of definitive programming, but of a new method of computer-based modelling for which we have adopted the term *Empirical Modelling*. The ramifications of Empirical Modelling are broad, and suggest many directions for potential future development. They are characterised by a shift of emphasis towards:

- using the computer as a physical artefact to represent phenomena in which the set of possible states and transitions is uncircumscribed [11];
- accounting for the development of theory through exploration and experiment [9];
- supplying a unifying representation to address many different aspects of the design product in the concurrent engineering process [1, 2].

Such issues can be understood with reference to notations with much more limited geometric capabilities than were envisaged in the CADNO notation [6]. They are particularly relevant to design applications, and to concurrent engineering.

In this paper, we first describe the re-orientation to which the agenda in [6] points, then reappraise the proposals made in [6] in the light of subsequent research that has been directed at developing definitive notations for geometric modelling. This includes the doctoral work of Alan Cartwright on *EdenCAD* [19], of Adzhiev et al on *HyperJazz* [4] and of Richard Cartwright on a variant of *CADNO* [20]. There are four principal sections: section 2 describes the background and motivation; sections 3 and 4 describe the main ingredients of the Empirical Modelling framework as represented in current software tools; section 5 reviews subsequent progress towards developing appropriate definitive notations for geometric modelling.

2 Background and Motivation

Our contention is that shape modelling is most appropriately studied in the context of a new modelling paradigm. Our motivation for this position is twofold, acknowledging the need for

- a meta-modelling framework that embraces shape modelling as a particular aspect;
- modelling principles that transcend the traditional closed-world viewpoint.

Our paper will focus upon design applications, where these considerations are particularly relevant, but our approach has wider applicability. For instance, similar considerations apply to shape modelling in a Virtual Reality setting.

2.1 Meta-Modelling in Design

The specification of geometry has historically been central to tools for CAD support. To this day, commercial modelling packages put their primary emphasis upon geometry, and visual impact remains a major selling point. CAD systems evolved through semi-automation of draughting techniques, but capture conveniently only a part of the intention of a human designer. For instance, a designer is not solely concerned with developing a complete and precise specification of a geometric model. In the early stages of a design, sketches may be used to explore one aspect of the design task, and serve as an impression or metaphor rather than a precise representation. For instance, a preliminary sketch might depict a skeletal view that is to be obscured in the final realisation of an object. To do justice to the role that pictorial representations play for the human designer, a computer system needs to provide support for shape as *metaphor* as well as shape as *product*.

The fact that the core database in a typical CAD system was originally intended to record geometry poses some problems. For instance, to what extent is additional information to be regarded as part of a stored object? Is the lighting model to be set up by the user, and should the lighting configuration be discarded by default when the geometry is stored? How is the scalar information associated with a finite element analysis to be generated and stored? Integrating extra information with a geometric model is a non-trivial task that cannot be adequately addressed simply by attaching new processes to the database.

Putting the primary emphasis on geometry means that what is readily accessible to the computer is necessarily information that is mediated via the geometry database; this limits the scope for constructing representations that truly reflect the rich semantics of real-world objects. Techniques for inferring other characteristics (such as high-level features and scalar physical attributes) from the geometry can help to redress this problem (cf. feature-based approaches to geometric modelling [5]). However, all such techniques are subject to an inherent limitation: abstracting the geometry of an object entails discounting some of the characteristics of the real object.

The computer clearly has unprecedented potential to interpret precise mathematical specifications of geometry (as in CSG or implicit function approaches). Nonetheless, there is a perceived need for prototyping tools for CAD that liberate the designer from the need to specify explicit and detailed geometry initially. It is evident that a good mathematical model of geometric object does not necessarily provide the designer with a useful interface through which to perform exploratory modelling (cf. [4]). What is more, a designer typically conceives and manipulates an object in terms of its combinatorial structure and features rather than as an abstract point set in Euclidean space. The relative success of B-rep in commercial modellers - notwithstanding its limitations as a mathematical representation - may be in part attributable to the scope it allows for incorporating reference to features.

The advent of concurrent engineering approaches to product development further undermines the ethos of the traditional CAD system. In [31], Tomiyama introduces the concept of meta-modelling to express the need for modelling methods that combine the representation of objects with the representation of the design process itself. Tomiyama envisages a design framework in which there are many different descriptions of the design product, all of which are interrelated and draw upon a single source of information. To realise this objective requires an intimate integration of geometric and non-geometric modelling in a context that also supports the management aspects of the design process - meeting the need to record versions, alternative viewpoints, archives and libraries.

The ADDL system, developed by Veerkamp [32], is one proposal that involves integrating knowledge about objects (attributes and characteristics) with knowledge about the design process (designer's goals). In ADDL, non-geometric information is recorded in the knowledge base

for each object. An important characteristic of Veerkamp's conception is that the designer has a creative role to play in the selection of scenarios that shape the evolution of objects. This theme is developed in the next section.

2.2 The Closed-world vs. Open Development Viewpoints

The context for the paradigm shift we propose is supplied by a conflict between two engineering cultures that is well-characterised in these words of Brödner [16]:

> One position, ... the "closed world" paradigm, suggests that all real-world phenomena, the properties and relations of its objects, can ultimately, and at least in principle, be transformed by human cognition into objectified, explicitly stated, propositional knowledge.
>
> The counterposition, ... the "open development" paradigm ... contests the completeness of this knowledge. In contrast, it assumes the primary existence of practical experience, a body of tacit knowledge grown with a person's acting in the world. This can be transformed into explicit theoretical knowledge under specific circumstances and to a principally limited extent only .... Human interaction with the environment, thus, unfolds a dialectic of form and process through which practical experience is partly formalized and objectified as language, tools or machines (i.e. form) the use of which, in turn, produces new experience (i.e. process) as basis for further objectification.

The essential need for interaction between user and computer makes it impossible to construct a computer modelling system that exclusively and faithfully represents a closed-world perspective. Whatever formal propositional account is given for the behaviour of the system, experience is necessarily involved in interaction between the user and the system. This unavoidable open aspect of a computer system is reflected in two ways. It means that the user can attribute interpretations to the system responses that are beyond the scope of what has been formally specified. It can also enable the user to develop skills in interaction with the system that are outside the scope of the abstract system specification. In practice, there are ways in which computer systems can be engineered so as to militate against such open use, viz. by restricting the experiential channels to the user. For instance, numerical data can be presented without visualisation, and user interaction restricted to batch input-output using encoded instructions.

In so far as openness can be suppressed, the products of the closed world culture are systems that offer a preconceived framework for interaction between the designer, the computer model and the external world. In this context, the term *preconceived* refers to the fact that the nature of the content relation that associates the computer model with an external referent is prescribed, and that the manner in which the computer model is shaped by the designer's interaction is predetermined. As will be explored in the following section, our contention is that definitive principles for the representation of state, as investigated in [6], are the key to a new method of computer-based modelling that makes explicit provision for both the content relation and the user-computer interaction to be enriched during the modelling process.

In our view, the challenges of meta-modelling cannot in general be addressed within a closed world system. It is not clear to what extent we can preconceive which information about an object will be significant in the design process, nor what exceptional scenarios of use will need to be considered. Neither can we expect that descriptions of the design product from a variety of viewpoints will be coherent and consistent. Human insight is typically necessary for decision-making for design. For instance, if the profile of a table top is morphed from square to circle, human judgement is required to determine at what point it ceases to have corners.

Openness is associated with essential involvement of the designer and of interpretative activity in the development process. This involvement is reflected both in the development of a computer model and in concurrent ongoing interaction with the external world (if only in the realm of *thought-experiment*).

The problems of efficiency that arise when dealing with shape as product have encouraged the investigation of closed-world frameworks for shape modelling. The algorithmic optimisations that are required restrict the scope for situating shape modelling in a wider context. Greater flexibility and sophistication within conventional paradigms (cf. Djinn [15]) are not enough to overcome these obstacles. The work of Chmilar and Wyvill [21] and Emmerik, Rappoport and Rossignac [22] demonstrates some of the potential for applying spreadsheet-like principles to shape-modelling. However, as Nardi's thought-provoking analysis of the spreadsheet programming culture [26] indicates, such principles can have an even more radical impact when used outside the conventional programming framework. Conceptual design and design in a concurrent engineering context demand a degree of openness that entails such a shift of paradigm.

3 Empirical Modelling

In this section, we review the significant shift of perspective that has occurred in respect of items b. and c. of the agenda established in [6]. Section 3.1 and 3.2 describe how the use of definitive principles for representation of state is connected with the construction of cognitive artefacts such as an engineer or an experimental scientist might develop. Section 3.3 explains how such artefacts differ from conventional programs, with reference to Brödner's closed-world and open development paradigms.

3.1 Communication using Artefacts

Any benefits that stem from the open aspect of what has been conceived as a closed-world system are fortuitous: the user finds significance where none was explicitly intended. In activities such as shape modelling, where the way in which the model is experienced by the user is very important, a greater degree of self-conscious control over how models are experienced is desirable. In this connection, we favour the idea of the computer as an instrument, to be understood in relation to the skills of its user. This begs the question as to whether we can construct systems in such a way as to deliberately encourage openness. Empirical Modelling is proposed as an approach to this problem.

To step outside a closed-world framework means to discount a textual account in a formal computer language. Our aim is rather to communicate information by constructing a *cognitive artefact* [11]: a physical object that directly represents the real-world object or phenomenon to be described. (There are interesting points of contact here with Naur's *constructed models* [27] and Gooding's concept of a *construal* [25].) There are many precedents for such communication, of which the design engineer's use of a physical object as a prototype is the most relevant to our theme. We have ourselves described the application of Empirical Modelling in the design process as constructing a *virtual prototype* [1]. Though our account of Empirical Modelling makes no explicit reference or new technical contribution to shape modelling, it should be apparent that shape in the broadest sense of the word is central to the construction of visual artefacts.

To borrow Naur's terminology, a cognitive artefact represents a phenomenon "by establishing a relation between two parts of the world". The computer-based artefacts we shall construct are not necessarily devoid of symbolic elements (for instance, the computer screen may display textual messages), but our particular interest is in those aspects rooted in a correspondence that is pre-articulate and is not mediated by symbols (cf. the philosophical stance that Brooks has

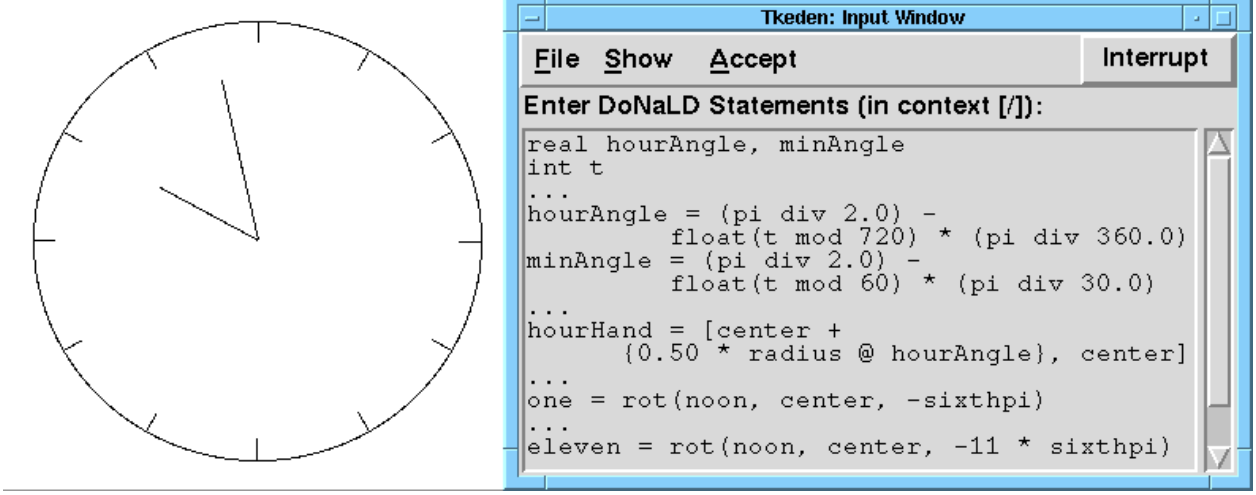

Figure 1: Analogue watch face and script.

adopted so successfully in AI [17, 18] and the discussion in [9]).

Cognitive artefacts as we conceive them differ from Naur's constructed models [27] in that the extent of the correspondence between the artefact and phenomenon is not known in advance. The artefact is associated with an implicit environment for interaction as defined by atomic changes of state, but its true potential to represent a phenomenon is not preconceived, and the possible states in which the artefact may be experienced is initially uncircumscribed. Unlike a formal document, the environment associated with an artefact is open to exploration, and significant relationships to the referent are established on empirical grounds. In this respect, our artefacts have an affinity with the construals that Gooding [25] associates with the activity of an experimental scientist, which represent tentative - and often ephemeral - hypotheses about phenomena.

The scope of the artefact concept is wide. There is no reason why an artefact should be computer-based, as the history of mechanical models of physical phenomena proves. Works of art with an explicit representational quality are arguably conceived as artefacts. Our particular interest is in how the computer can best be exploited in the construction of artefacts. Definitive principles for modelling have fundamental relevance in this study, since their use can be linked with well-established experimental procedures that are widely used in analysing phenomena.

The traditional scientific method analyses a phenomenon in terms of observables. An observable is a feature of a phenomenon to which an identity can be ascribed, that in general manifests different values when the phenomenon is observed in different states. A powerful technique for constructing computer-based artefacts is to use definitive principles to imitate the way in which the observables associated with a phenomenon are indivisibly linked in change. This is essentially the principle by which the formulae that define the cells of a spreadsheet are determined. This has an empirical justification, similar to that invoked by an experimenter who observes that a change to one parameter invariably affects the values of others in a consistent manner.

3.2 Definitive Principles for State Representation

The principle of using a definitive script to describe a geometric object has been extensively described and illustrated in previous papers [10]. By way of example, Figure 1 depicts a simple analogue watch face with a fragment from the associated *DoNaLD* script. The computer model of a watch that is generated in this way can be viewed as an artefact that represents a real-world

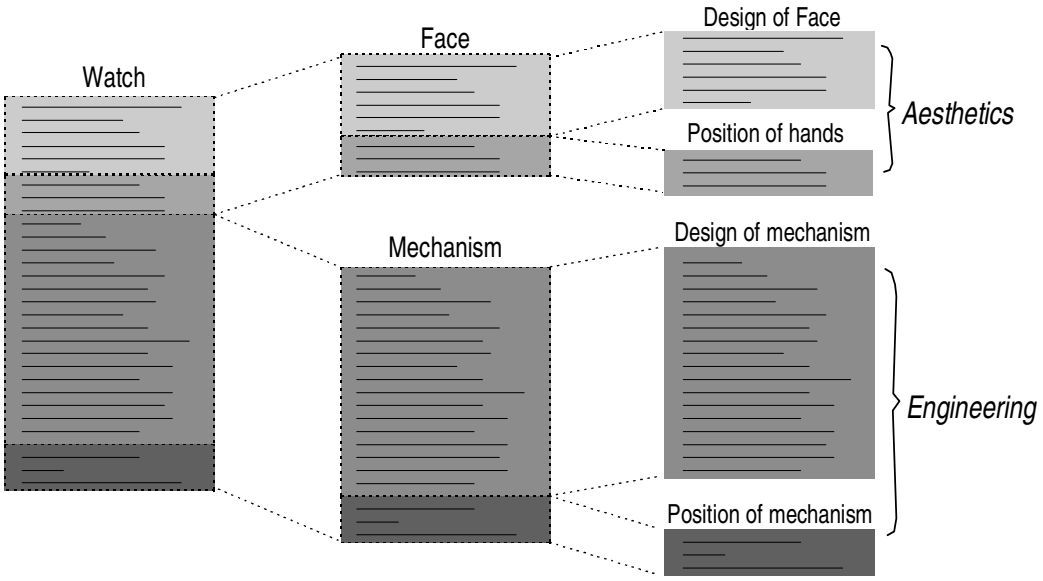

Figure 2: Composition of script for specifying the state of a watch.

watch. The script in Figure 1 is then not to be viewed as a program to generate the picture of a watch, but as specifying one of several possible explicit views of the watch in a particular state. Each variable in the script represents a particular observable associated with the watch, and the current values of variables corporately represent observation of the watch at a particular instant. If the hands on the watch in Figure 1 displayed some other time, this could be construed as displaying the same watch in another state.

The significance of definitive scripts in geometric modelling - as it has emerged through our research on the agenda outlined in [6] - lies in the way in which scripts are composed, interpreted and exploited in interaction. There is no fixed or preconceived pattern for composing, interpreting and interacting with scripts: such constraints as there are upon interaction with a script are imposed by the semantics of its referent. This is best illustrated by how script fragments such as Figure 1 are developed and organised in a typical modelling application.

The development of a model involves several loosely sequential phases:

- the identification of observables;
- the selection of observables to make up viewpoints;
- the identification of constraints and the acknowledgment of commitments governing possible states of the model;
- the development of consistent scenarios for state change and environments for interaction.

The effect of such development may be to create many viewpoints and associated interfaces. These will not necessarily be consistent, but have a similar status to the initially incoherent and independent experimental results and measurements that ultimately inform the design of an engineering product. Where typical modelling methods in a closed-world framework focus on producing a complete model, our modelling activity is an end in itself, as essential insight into the design requirement and the design product is gained throughout the modelling process.

The open character of the modelling activity can be illustrated with reference to Figures 2 and 3. In Figure 2, the box on the left represents a definitive script that includes a family

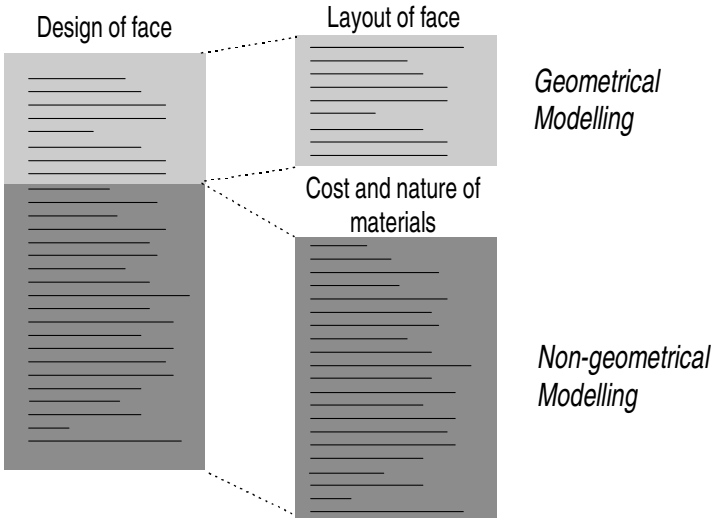

Figure 3: Possible structure of script for design of face.

of observables associated with an analogue mechanical watch. Just what observables might be included here is open. To simulate the watch in operation, it would be sufficient to observe the geometry of the watch face (cf. Figure 1) and of the watch mechanism. The observables can be partitioned as in Figure 2: an aesthetic designer would be interested only in the observables associated with the face, and the mechanical designer with those of the mechanism. More precisely, the aesthetic designer is concerned with those observables that determine the characteristics of the hands, not with the particular time they register. As Figure 2 indicates, the design of the face invokes other non-geometric observables associated with the cost and nature of the materials used in its construction.

The modelling requirements for simulation of watch operation, aesthetic and mechanical design illustrate three possible viewpoints, as defined by a choice of observables. The set of possible viewpoints is itself open to extension and revision. Resolving conflict between viewpoints and negotiating common viewpoints is an important aspect of the concurrent engineering process [1]. Even though two viewpoints may have variables in common, it is not necessarily appropriate or straightforward to combine them. For instance, it is possible to envisage observing the operation of the watch in such a way that the cost of the materials of which it is comprised is monitored in conjunction with stock market prices, but this would be a most unusual way to assess the value of a watch. An inventory of parts of a dismantled watch has many observables in common with the assembled watch, but if one of the parts were to be damaged, assembly might be impossible.

The choice of viewpoint still leaves open questions concerning what system states are feasible, and what sort of interactions are conceivable. In a design context, a change to the configuration of the figures on the watch face may be interpreted as a different view of the same watch. In simulation, repositioning the hands might be interpreted as resetting the time. A reconfiguration of the watch face in which the relative position of the hands was invalid, or in which a hand was missing, might be interpreted as representing the watch in a damaged condition. Where several designers are operating concurrently, there may be a need to constrain interaction to respect commitment to a particular design decision.

3.3 Agent-oriented Modelling

The range of possible viewpoints and transitions associated with a particular family of observables is enormous. To rationalise these viewpoints, it is helpful, if not *essential*, to think in terms of different state-changing agents whose activity is to be modelled. In this context, the concept of agent is broad (cf. [1]). For instance, in respect of different scenarios, we may regard any of the following as agents: the individual components of the watch, the potential users of the watch, the designer(s) of the watch, the managers of the concurrent engineering process and the computer on which the design of the watch is being modelled. In the latter case, the computer serves two functions: executing, whilst simultaneously modelling (at a higher-level of abstraction), the role it performs in relation to the designer and the external phenomenon.

Some simple illustrations of observables associated with agents of the above kinds may be helpful. The interaction between the components of a mechanical watch must be described with reference to whether its spring has run down. The users of watch will observe whether the watch tells the correct time and whether they are late for their appointment. The designer of watch faces will observe a relationship between the dimensions of hands. A manager of a design team will know the current status of the design, and know what privileges particular designers have to redefine parameters of the watch. A simple example of an observable to reflect the characteristics of the computer on which the artefact is being realised is a boolean to indicate whether or not the display has colour, with dependencies to modify the display definitions appropriately.

There are characteristic dependencies between observables associated with each agent's viewpoint. As Figure 1 illustrates, dependencies associated with two such viewpoints can co-exist in the same script. The script in Figure 1 includes a definition to couple the motion of the hands appropriately, together with definitions that enable the designer to change the dimensions of the watch conveniently. Seeking an "objective" viewpoint from which to understand the interaction between the agents active in a phenomenon is an important aspect of moving from an open-development to a closed-world context.

In constructing a computer-based artefact to represent a complex phenomenon, such as the use of a chessclock (cf. [11]), the identification of dependencies is complemented by the identification of agents and scenarios for their interaction. An account of a phenomenon will in general include a specification of the ways in which agents are privileged to observe and conditionally interact to change state. Redefinition of a variable in a definitive script is a means of expressing many kinds of agent action, including voluntary choices of action (chess player chooses to push the button on his clock), stimulus-response patterns (a player's clock stops ticking when its button is pressed), and "acts of God" (a hand falls off a player's clock).

3.4 A New Programming Paradigm?

Our original agenda, as set out in [6], refers to the development of a new programming paradigm. It is clear that the process of developing a script (or family of script fragments) together with an associated agent-oriented model is quite unlike conventional computer programming. By default, the system state model specified in this way has an open uncircumscribed nature, and it is in general necessary to rely upon human intervention to simulate realistic operation of autonomous agents and to resolve conflict in their interaction. Only by putting explicit constraints on how scripts are to be modified can the modeller eliminate open interaction; only by explicitly acknowledging reliable patterns of state change and circumscribing intervention can the modeller construct behaviours similar to those of a conventional procedural program.

Conceptually, the simplest way in which to animate an agent-oriented computer model is to allow the modeller to act as a super-agent who simulates the role of all the agents in the system by explicitly selecting and performing appropriate redefinitions. In practice, from a

suitable viewpoint, the behaviour of certain agents within the system can be circumscribed. This corresponds to using or developing a theory about how agents interact. For instance, if we ignore the possibilities of malfunction, then, for most practical purposes, the components of a watch interact reliably in an entirely predictable manner. In some contexts, the principles of classical mechanics can also be applied. Where such circumscription is possible, the role of such agents is more appropriately delegated to the computer. For example, when simulating the watch in operation, the movement of the hands can be carried out by programmed redefinition. Of course, this simulation is to be executed in a way that does not preclude concurrent intervention by the modeller.

Understanding the status of our models in relation to conventional programs is challenging. The openness of our method derives from being able to augment the model at any stage so as to reflect a new feature of the external phenomenon. This may mean referring to a new observable, or recording a newly identified dependency. So long as we restrict our interaction with our computer artefact to redefinitions whose interpretation involves no new insight into the phenomenon to which it refers, we can realise the resulting limited functionality of our model in a highly optimised way using a conventional procedural program. If, at this stage, we identify new dependencies, and wish to modify our artefact accordingly, there can be no guarantee that our procedural program can be conveniently adapted. In effect, we are faced with the traditional problem of changing the requirement; it is in general hard to adapt a procedural program to perform a function that has not been preconceived. A reader who is sceptical about the claim that our model is more readily adaptable should consider the nature of the analysis of agency and dependency involved in its construction. Arguably, our model encapsulates our understanding - possibly partial, but empirically well-founded - of why particular patterns of behaviour are being observed (cf. [14]). As our everyday experience of learning corroborates, it is exceptional for new insight to undermine empirical knowledge; it normally entails a refinement rather than a wholesale revision of our beliefs about agency and dependency.

4 Software Tools for Empirical Modelling

The Empirical Modelling concept, as outlined above, is supported by a suite of software tools that have been developed at Warwick. The main tasks involved in building computer-based artefacts are:

- agent-oriented modelling and simulation;
- image specification and generation;
- interface specification and generation;
- dependency maintenance;
- version management.

A brief review of these software tools follows; more details can be found elsewhere.

4.1 Agent-oriented Modelling and Simulation

In our agent-oriented analysis of a system, we make use of a notation called LSD. An LSD account of a phenomenon involves identifying the observables associated with the interactions between agents, and representing the potential actions of each agent in terms of them. These observables may well be aspects of state that cannot be directly perceived by the modeller (e.g.

an electric current might be an observable for the crystal in a watch), and may also include states of mind that are not directly accessible (e.g. knowing whose clock is ticking in a game of chess). We cannot in general give a conventional operational semantics to a family of LSD agents, but we can supply artefacts that exploit metaphors to represent their individual perceptions of state in experiential terms. The metaphors we use for this purpose are primarily visual.

The LSD notation has proved useful not only as a route to concurrent systems simulation, but also as a way of analysing meta-level interaction (e.g. between the designer and the computer model) in the construction of an artefact. In principle, it has the expressive power to describe the roles of the design managers, who may be responsible for specifying what kind of agency each designer enjoys [1, 2].

The Abstract Definitive Machine (ADM) is used to synthesise an artefact to represent a system of interacting LSD agents. It supplies a framework within which the current state of a system is specified by a family of definitive scripts that integrates the viewpoints of all the component agents. This model is used for animation and experiment with a view to arriving at a coherent explanation of patterns of behaviour within the system as abstractly discussed above.

Our present version of the ADM relies upon the *tkeden* interpreter for visualisation, and is implemented via a translator. An alternative implementation strategy involves direct hand-translation of embellished LSD specifications into *tkeden*. Both these implementation strategies have their limitations. Translation from the ADM curbs the designer's scope for unconstrained intervention. Direct implementation in *tkeden* uses triggered actions to animate agents and removes some of the designer's power to monitor the interaction at a higher-level of abstraction.

4.2 Image Specification and Generation

To operate as a cognitive artefact, the computer has to generate perceptible states that can be correlated with the states of an external phenomenon. Visualisation using *DoNaLD*, a simple definitive notation for line drawing, is the primary means we have exploited to generate images. As cartoons illustrate, the use of simple drawings with close correlation to real-world state has surprising expressive power. The range of interpretations for line drawings in our case-studies has been wide: they have been used to represent abstract configurations of mathematical lines, to represent railway tracks, digits on an LCD, icons, graphs of functions, mechanical devices etc. In most cases, the characterisation is unexpectedly convincing because of the close correspondence between atomic transformations of the referent and transitions of the drawing associated with redefinition.

The limitations of *DoNaLD* are most clearly revealed where lines have a more complex semantics. The graph of a continuous function is not well-represented by a piecewise-linear approximation, even though what is displayed on a computer screen is a discrete pattern of illuminated pixels. A higher-level abstraction is required to express the way in which such a graph can be conceived as the limit of its discrete approximations. Combinatorial graphs exploit another kind of abstraction, in which lines represent abstract relationships between points. In certain applications, we have used a special-purpose definitive notation *ARCA* to describe such graphs (cf. [8, 24]), but the design and implementation of *ARCA* is not yet as well-consolidated as that of *DoNaLD*.

4.3 Interface Specification and Generation

As explained in section 3.1, the essence of a cognitive artefact lies in the primitive state-changing operations that can be performed on it. This underlines the importance, when generating computer-based artefacts, of being able to modify the interface in a flexible and open manner. In [6], the possibility of using definitive scripts to represent the interface to the designer

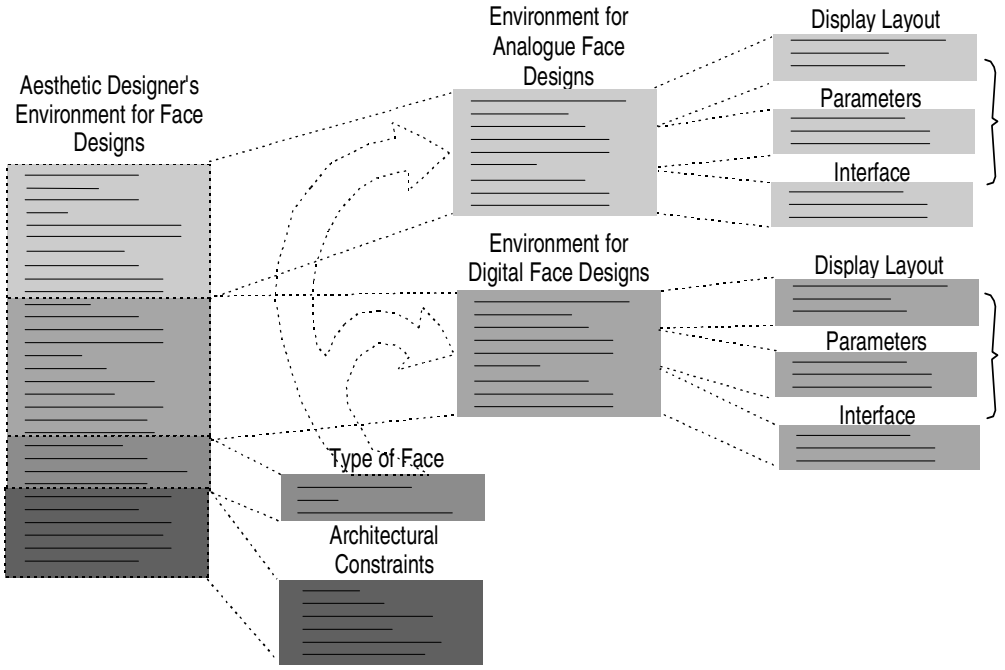

Figure 4: Possible structure of script for a face design interface.

was envisaged; this issue has since been addressed through the design and implementation of *SCOUT* [7], a definitive notation for screen layout. Figure 4 indicates how such a representation assists the design of interfaces for specialised tasks. The use of *SCOUT* is also illustrated in Figure 6.

4.4 Dependency Maintenance

The core interpreter used in our software development is *tkeden*: a *tk/tcl* based extension of the *EDEN* interpreter referenced in [6]. *Tkeden* combines *SCOUT*, *DoNaLD* and *EDEN* within a single environment. The primary role of *EDEN* is to maintain dependencies between variables and to act as an interface between definitive scripts at a high-level of abstraction and the low-level procedural utilities that handle window management and graphical display. As a hybrid programming tool, *EDEN* is less pure than the current version of the *ADM*, but it offers more powerful features for supporting definitions at a higher-level of abstraction. In the interface to *tkeden*, an agent-oriented model is used to rationalise its complex operational semantics.

4.5 Version Management

As explained in section 3.1, cognitive artefacts serve to represent provisional hypotheses about a phenomenon, open to refutation, revision and refinement in the light of experiment. Tools for storing and configuring definitive scripts can greatly simplify the task of managing computer-based artefacts developed by Empirical Modelling. Storing intermediate scripts serves an important function in documenting the design process. Definitive scripts also lend themselves to retrospective correction and refinement that assists a rational reconstruction of the development process.

The *tkeden* interpreter incorporates features to assist the management of scripts. These

Figure 5: A VRML realisation of a CADNO table.

include mechanisms for searching, editing and viewing scripts, and for file storage and retrieval.

5 Definitive Notations for Geometric Modelling

More effective ways of generating images would enhance our Empirical Modelling tools, and help to make them more useful in practical design. Generalising a simple definitive notation such as *DoNaLD* to a more sophisticated notation for geometric modelling raises significant issues. In this section, we describe and illustrate the motivation behind the definitive notation for geometric modelling *CADNO* (as introduced in [6]). We also outline the contributions made by subsequent work on definitive notations for geometric modelling, with particular reference to topical problems for the future development of our tools. These concern implementation, abstraction and interfacing.

5.1 CADNO Revisited

A significant characteristic of the *DoNaLD* scripts we have developed in our previous case-studies is that the variables typically correspond to meaningful features of the referent. There is an important distinction between a *DoNaLD* point that represents the corner of a table, and one of the vertices that make up the wire frame representation of the corner of an actual table, as it appears in a quasi-realistic rendering (cf. Figure 5). In agent-oriented terms (cf. section 3.3), a vertex of a wire-frame model is an observable that relates to the computer, whereas the corner of a table is an observable for the designer. Somewhat paradoxically, there may be no location on a physical table that can properly be called the corner of a table (cf. the table depicted in Figure 5), yet the concept of a corner as a feature of the table is essential for high-level communication about table design.

The design of *CADNO* reflects the need, in an agent-oriented view of shape modelling, to

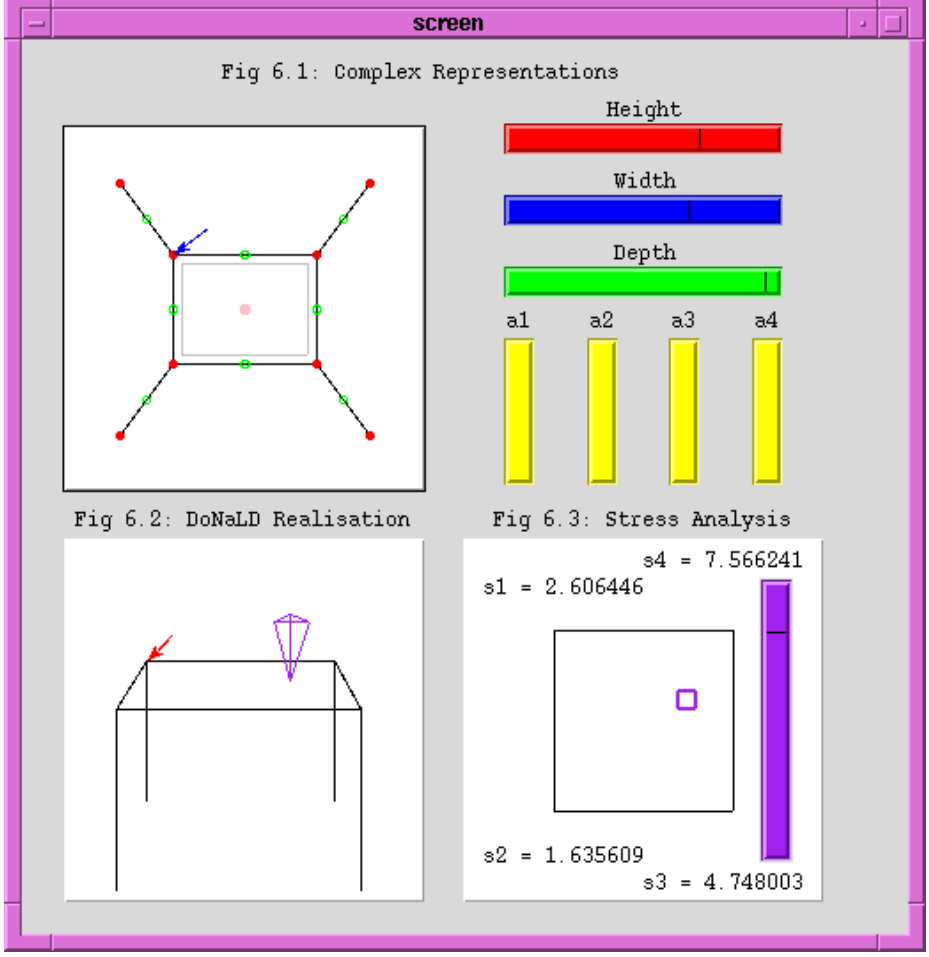

Figure 6: Integrating CADNO into a tkeden interface.

combine real geometry with abstract handles for the shape designer. The choice of the term *abstract geometry* in the title of our paper reflects the fact that the role of geometry in design is mediated through geometric entities that, unlike Euclidean point sets, have a combinatorial and metaphorical character. Though our main emphasis is on the way in which high-level concepts of interest to the designer can be linked to geometry, the general principle that real geometry is specified and manipulated via scalar and abstract geometric parameters applies at many levels of abstraction. For instance, even the use of implicit function specifications requires insight on the part of the designer into how the choice of discrete parameters affects the geometry. The data types in the underlying algebra for *CADNO* are summarised as follows.

complex	list-of label $\times$ list-of list-of label
d-complex	complex $\times$ int
frame	list-of vector-of real $\times$ d-complex
object	list-of frame $\times$ (list-of frame $\rightarrow$ (vector-of real $\rightarrow$ boolean))

In general terms, a complex is a combinatorial structure used to specify the abstract geometric entities from which a geometric object is synthesised. A complex can be realised in Euclidean space as a frame, by attaching coordinates to its points. A geometric object is functionally determined as a point set from a list of frames. A range of different functions can be used for realisation: for instance, the list of frames may be interpreted as the basis of a B-rep, a CSG model, an implicit function specification, or simply a 3-dimensional line drawing in *3D-DoNaLD*.

The listing in Figure 7 is a specification for the geometric models of tables shown in Figures 6.2 and 5. There is as yet no full implementation of *CADNO*, but the specification is faithful to the provisional syntax for *CADNO* developed for [6]. Figure 6 is a simple demonstration screen, constructed using *tkeden*, that is contrived to illustrate how *CADNO* could be integrated with other Empirical Modelling tools.

Figure 6.1 depicts the underlying complex that represents the table: it provides a mode of reference to the *3D-DoNaLD* image of the table in Figure 6.2. The location of a feature of the table, such as the position of the foot of a particular leg, is identified by an arrow on selecting the appropriate component of the combinatorial structure in Figure 6.2.

Figure 6.3 illustrates how the geometric models of the table can be used in conjunction with non-geometric models in an integrated manner. It depicts the distribution of weight on the legs of the table when a load is placed upon the table top. Figure 6.3 also supplies an interface through which the load can be relocated by direct manipulation.

Figure 6 also includes a simple *SCOUT* interface through which the scalar parameters in the underlying complex for the table in the listing in Figure 7 can be manipulated.

Figure 5 is a realisation of the table using the **VRML** modelling notation. The **VRML** program that produces this image incorporates parameters drawn from the underlying complex.

5.2 Complex Definitive Notations: Technical Issues

The technical issues that must be addressed in designing and implementing a definitive notation for geometric modelling are similar to those encountered when handling any complex definitive notation. Three common concerns are briefly discussed in the following sections.

5.2.1 Implementation

The implementation of definitive notations such as *DoNaLD* and *SCOUT* protects the user of our toolkit from having to consider the characteristics of the computer on which the artefact is to be realised. In point of fact, the capabilities of this computer can be quite significant - to return to the theme of section 3.1, not all instruments of a particular kind are equally expressive or easy to use. Sophisticated use of definitive notations sharpens our awareness of the significance of the underlying computer architecture, and obliges us to be more explicit about the computer as an agent. The motivation for this is not simply that we need greater efficiency: the semantics of artefacts may demand control over the characteristics of the physical medium used for representation. This is well-illustrated in [8], where our representation of a configuration of mathematical lines invites us to consider the accuracy of the machine arithmetic as a parameter under the control of the human interpreter (cf. the sense in which only an arbitrarily thin pencil can mimic a mathematical line).

For the most part, *tkeden* sustains the illusion that there is an indivisible link between definitive scripts in *SCOUT* and *DoNaLD* and the screen displays that they represent. With current technology, it is impossible to generate a realisation of a high-level parametric description of a geometric object as fast it is possible to update the *DoNaLD* screen. Such considerations oblige us to examine and interpret the relationship between modeller, artefact and real-world phenomenon more critically (cf. the way we adapt to the slow speed of a cell update in a large spreadsheet).

A variety of techniques can be explored for improving efficiency, or working within the limitations of present models. In geometric modelling, existing packages can be used to handle the realisation of objects. This is the strategy investigated in *EdenCAD* [19]: it leads in general to some loss of control over the representation without a sophisticated interface to the dependency maintainer. The most common problems are concerned with reference to the components of

```
table = 3-complex on [A, B, C, D, E, F, G, H]
                with {[A, E], [C, G], [D, H], [B, F], [E, G, H, F]}

base = 3-complex on [origin]                          # User's handles
angles = 1-complex on [angle1, angle2, angle3, angle4]

dimensions = 1-complex on [height, width, length]  # Designer's handles
radii = 1-complex on [legRad, cornerRad]

table1 = frame on [table, base]
                where   A = B - height * {sin(angle1), cos(angle1), 0},
                        B = origin + {0, height, 0},
                        C = D - height * {sin(angle2), cos(angle2), 0},
                        D = origin + {0, height, depth},
                        E = F - height * {-sin(angle3), cos(angle3), 0},
                        F = origin + {width, height, depth},
                        G = H - height * {-sin(angle4), cos(angle4), 0},
                        H = origin + {width, height, 0}
                        origin = {0, 0, 0}
radii1 = frame on radii
                where   legRad = 3.0,
                        cornerRad = 10.0
                subject to
                        0.5 <= legRad <= 20.0,
                        legRad <= cornerRad <= 25.0
angle1 = frame on angles
                where angle1 = 0, angle2 = 0, angle3 = 0, angle4 = 0
                subject to
                        0 <= angle1 <= 90,
                        0 <= angle2 <= 90,
                        0 <= angle3 <= 90,
                        0 <= angle4 <= 90
dimensions1 = frame on dimensions
                where height = 40, width = 50, depth = 30
                subject to height > 0, width > 0, depth > 0

pict1 = object on [table1, angle1, dimensions1]
                with extent(donald3d) { .... }
pict2 = object on [table1, angle1, dimensions1, radii1]
                with extent(VRML)
                     {  Cylinder { axis A E radius legRad}
                        Cylinder { axis B F radius legRad}
                        Cylinder { axis C G radius legRad}
                        Cylinder { axis D H radius legRad}
                        Profile { base E F G H extrude 10 }
                        .... }
```

Figure 7: Example CADNO script for tables.

an artefact. At other times, it is the very intelligence of the software being invoked that poses problems, since it entails an implicit agency that subverts the intended dependency relationships.

An alternative approach involves new implementations of definitive systems. These potentially include parallel implementation of definitive tools, and the use of definitive principles at very low-levels of abstraction in the machine architecture. The latter concept is being pursued in the definitive assembler maintainer (*DAM*) machine, that aims to exploit the powerful control over graphics afforded by the *ARM* architecture.

Another important direction of research focuses on trading openness for efficiency. As a simple illustration, our billiards simulation in *tkeden* [23] typically executes slowly in such a way that balls and pockets can be relocated or modified in size during play etc. By eliminating such unreasonable privileges for intervention, we can achieve much faster and realistic animation. Related techniques that also depend on imposing some restriction upon the behaviour of the model include wholesale translation into a conventional procedural language, and the use of dynamic algorithms.

5.2.2 Interfacing

The modern concepts of multi-media device and virtual reality environment highlight the limitations of conventional computers as a basis for constructing artefacts. There are intrinsic limitations in typical interaction techniques. For instance, there is no convincing metaphor to convey what it means to pull the sheet in a sailboat [28], and no way to express on the screen display that the motion of the hand of a clock is continuous.

Shape modelling poses serious interface problems, both in respect of manipulation and visualisation. The geometry of a 3-dimensional object cannot be conveyed through a single image, but requires the creation of an environment for interaction and viewing. In the abstract specification of a geometric object (e.g. by an implicit function or a surface model), it is in general hard to predict the geometric implications and visual effect of changing parameters. Whereas in *DoNaLD* the relationship between the defining script and the screen display is easily interpreted, the same cannot be said of typical shape representations. What is more, there are many strategies for manipulation and visualisation, some of which are better suited to certain kinds of objects and applications than others (cf. the *PIRANESI* art package [30]).

Adzhiev's *HyperJazz* [4] is a definitive notation for geometric modelling that allows the shape modeller to investigate parametrisations of shape in an exploratory fashion. It encourages the user to treat the computer as an instrument for realising geometry, developing customised implicit function representations in an empirical fashion, and changing the viewing set-up and the visualisation strategy through interaction with a script. The potential for such an approach is well-illustrated by the range of geometric models and techniques developed by Pasko and his collaborators [29]. Research of this nature is a prerequisite for the effective use of R-functions as a realisation technique in the context of a *CADNO* script, since it is the basis for exercising control over the computer when generating visual artefacts.

5.2.3 Abstraction

Wherever complex structures are to be defined using a definitive notation, the possibility of making definitions at many different levels of abstraction exists. As Listing 1 indicates, it would be tiresome to have to define all geometric objects in explicit detail. There are two aspects to this problem: having control over the mode of definition of definitive variables, and being able to exploit generic abstractions.

Where moding is concerned, it is essential to have means of attaching values to variables in a variety of ways. For instance, one geometric object may have a frame in common with

another, or be specified as the result of applying a transformation to another object. A solution to this problem that involves declaring the mode of definition for each definitive variable has been proposed and implemented for the *ARCA* notation [24].

A related problem is that of specifying generic objects in the spirit of an object-oriented modelling paradigm. Proposals for such specifications have been made in connection with *DoNaLD* in [13], and have been the basis for a variant of *CADNO* designed and implemented by Richard Cartwright [20]. Higher-order definitions (definitions which in turn generate families of definitions at a lower-level of abstraction) are a route to the specification of geometric objects. By way of illustration, a graph abstraction has been introduced into *DoNaLD* that (in particular) allows a piecewise linear arc with an arbitrary number of segments to be specified as a higher-order definition. A more interesting application of the same abstraction is to the specification of a speedometer with an arbitrary number of divisions [24].

Both moding of variables and the introduction of object abstractions into an Empirical Modelling environment place some restriction upon the scope for variable redefinition. It is not difficult in principle to imagine how such restrictions can be observed as codes of practice, but more formal techniques for expressing such commitments in an Empirical Modelling framework are ideally required. Such systems offer exciting possibilities for complementing the exploratory quality of open development with closed-world integrity and efficiency, but more conceptual and practical support is needed before they can be developed.

6 Conclusions

Empirical Modelling has considerable promise as a framework for concurrent engineering. The development of shape modelling capabilities would enhance its usefulness as an approach to practical design. A perplexing aspect of our current suite of software tools is that the unity that Empirical Modelling lends to different design viewpoints is gained at the expense of introducing a plethora of definitive notations for visualisation. It may be that the design of *CADNO* has the potential to resolve this issue.

A striking feature of the abstract geometry underlying the *CADNO* object is that it can be applied in an extraordinarily wide range of contexts. A point in a complex may refer to a construction point in a wire-frame model, to the metaphysical point at the corner of a table, and (via the *graph* abstraction in *DoNaLD*) to the cluster of observables associated with a particular notch on a speedometer dial. As the spreadsheet itself suggests, there is a natural correspondence between partitions of space (square regions on the spreadsheet grid), abstract complexes (the abstract network of spreadsheet cells) and structured sets of symbolic references (the identifiers of the spreadsheet cells). The thesis that underlies the design of *CADNO* - that all geometry is apprehended as derived from combinatorial geometry - reinforces the idea that *CADNO* may provide a generic basis for specifying visual artefacts. Trying to validate this claim by unifying existing definitive notations for geometry within a *CADNO*-like notation may be a useful objective for longer-term research.

7 Acknowledgements

We are indebted to the EPSRC for support under grant GR/J13458 and its research studentship award to Richard Cartwright. We are also grateful to Matra Datavision for its support under the CASE award scheme. We also wish to thank Steve Russ for drawing our attention to several valuable references.

References

[1] V D Adzhiev, W M Beynon, A J Cartwright, Y P Yung *A Computational Model for Multi-agent Interaction in Concurrent Engineering* Proc. CEEDA '94, Bournemouth University 1994, 227-232

[2] V D Adzhiev, W M Beynon, A J Cartwright, Y P Yung *A New Computer Based Tool for Conceptual Design* Proc. workshop "Computer Tools for Conceptual Design", University of Lancaster 1994, 171-188

[3] V D Adzhiev, W M Beynon, A A Pasko *Interactive Geometric Modelling Based on R-functions* Proc. CSG '94 "Set-Theoretic Solid Modelling: Techniques and Applications", Winchester, Information Geometers 1994, 253-272

[4] V D Adzhiev, A A Pasko, A Sarkisov *HyperJazz Project: Development of Geometric Modelling Systems with Inherent Symbolic Interactivity* Proc. CSG '96 "Set-theoretic Solid Modelling: Techniques and Applications", Winchester, Information Geometers 1996

[5] S Ansaldi, B Falcidieno *The Problem of Form Feature Classification and Recognition in CAD/CAM* NATO ASI Series F, Vol. 40, Springer-Verlag 1988, 899-919

[6] W M Beynon, A J Cartwright *A Definitive Programming Approach to the Implementation of CAD Software* Proc. "Intelligent CAD Systems 2: Implementation Issues", Springer-Verlag 1989, 126-145

[7] W M Beynon, Y P Yung *Definitive Interfaces as a Visualisation Mechanism* Proc. Graphics Interface '90, Canadian Information Processing Society 1990, 285-292

[8] W M Beynon, Y P Yung, M D Atkinson, S R Bird *Programming Principles for Visualisation in Mathemtical Research* Proc. Compugraphics '91, First International Conference on Computational Graphics and Visualisation Techniques, 288-298

[9] W M Beynon, P E Ness, S B Russ *Worlds Before and Beyond Words* Computer Science Research Report #311, University of Warwick 1996

[10] W M Beynon, Y P Yung, A J Cartwright, P J Horgan *Scientific Visualisation: Experiments and Observations* Proc. Eurographics: Visualisation in Scientific Computing, 1992, 152-173

[11] W M Beynon, R I Cartwright *Empirical Modelling Principles for Cognitive Artefacts* Proc. IEE Colloquium on Design Systems with Users in Mind - the Role of Cognitive Artefacts, Dec. 1995

[12] W M Beynon *Definitive Principles for Interactive Graphics* NATO ASI Series F, Vol. 40, Springer-Verlag 1988, 1083-1097

[13] W M Beynon *Evaluating Definitive Principles for Interactive Graphics* New Advances in Computer Graphics, Springer-Verlag 1989, 291-303

[14] W M Beynon *Agent-Oriented Modelling and the Explanation of Behaviour* Proc. "Shape Modelling: Parallelism, Interactivity and Applications, TR 94-1-040, Dept of Computer Software, Univ. of Aizu, Japan 1994, 54-63

[15] A Bowyer et al *Introducing Djinn, A Geometric Interface for Solid Modelling* The Geometric Modelling Society and Information Geometers, 1995

[16] P Brödner *The Two Cultures in Engineering* Skill, Technology and Enlightenment: On Practical Philosophy, Springer-Verlag 1995, 249-260

[17] R A Brooks *Intelligence without Reason* Proc of 12th IJCAI, 1991, 569-595

[18] R A Brooks *Intelligence without Representation* Artificial Intelligence 47, 1991, 139-159

[19] A J Cartwright *Applications of Definitive Scripts to Computer-Aided Conceptual Design* PhD Thesis, University of Warwick, April 1994

[20] R I Cartwright *CADNORT: An R-function Enhancement of the Definitive Notation CADNO* Third Year Computer Science Project Report, University of Warwick 1994

[21] M Chmilar, B Wyvill *A Software Architecture for Integrated Modelling and Animation* New Advances in Computer Graphics, Springer-Verlag 1989, 257-276

[22] M van Emmerick, A Rappoport, J Rossignac *Simplifying Interactive Design of Solid Models: a Hypertext Approach* The Visual Computer (1993) 9, Springer-Verlag, 239-254

[23] M Farkas, W M Beynon, Y P Yung *Agent-Oriented Modelling for a Billiards Simulation* Computer Science Research Report #260, University of Warwick 1993

[24] D Gehring, Y P Yung, R I Cartwright, W M Beynon, A J Cartwright *Higher-order Constructs for Interactive Graphics and Design in a Definitve Programming Framework* Proc. Eurographics UK '96, Vol. 1 1996, 179-192

[25] D Gooding *Experiment and the Making of Meaning: Human Agency in Scientific Observation and Experiment* Kluwer Academic, 1990

[26] B A Nardi *A Small Matter of Programming: Perspectives on End-User Computing* The MIT Press, Cambridge MA, 1993

[27] P Naur *Knowing and the Mystique of Logic and Rules* Studies in Cognitive Systems, Vol. 18, Kluwer Academic 1995

[28] P E Ness, W M Beynon, Y P Yung *Applying Agent-oriented Design to a Sailboat Simulation* Proc. ESDA '94, Volume 6 1994, 1-8

[29] A A Pasko, V D Adzhiev, A Sourin, V Savchenko *Function Representation in Goemetric Modelling: Concepts, Implementation and Application* The Visual Computer 11, Springer-Verlag, 1995, 429-446

[30] S Schofield *Piranesi: A 3-D Paint System* Proc. Eurographics UK, 1996, p91-100

[31] T Tomiyama *Meta-model: a Key to Intelligent CAD Systems* Research in Engineering Design, Vol.1, No. 1, 1989, 19-34

[32] P Veerkamp *On the Development of an Artefact and Design Description Language* CWI Amsterdam 1992

[33] B Wyvill *An Interactive Graphics Language* PhD Thesis, Bradford University, 1975