Chapter 2

Empirical Modelling Principles and Geometry

2.1 Introduction

This chapter presents the conceptual basis and background to the work in the rest of the thesis. Geometric modelling on computer systems is commonly used in applications that support engineering and architectural computer aided design, computer games and computer generated animations. These are all well-covered fields of research with a significant volume of associated literature. The principal aim of this thesis is to find a way of uniting some of these well-developed research areas with empirical modelling principles. In comparison with tools developed with procedural and declarative programming techniques, many case studies [BBY92, NBY94, BJ94, BSY95] have shown that uniting these principles with various application areas leads to tools for modelling and simulation that exhibit an unprecedented degree of interactivity.

The background to this thesis can be classified into two distinct areas. These areas are discussed independently in Section 2.2 and are briefly described below:

1. Relevant literature on geometric modelling and computer-aided design. Particular attention is paid to notations for geometric design that allow for the textual description of geometric shapes.

2. History and background to the Empirical Modelling Project and its relevance to geometric modelling.

The empirical modelling process on a computer system benefits from strong visual metaphors for the representation of models, to aid a modeller's interpretation of the current state of a model and to assist their conceptualisation of model behaviour through interaction with a model. The construction of computer-based *cognitive artefacts* [BC95] with empirical modelling principles is introduced in Section 2.3. These artefacts are contrived to imitate interaction with a real world referent. The process of animating artefacts benefits from strong geometrical representations, as does support for the collaboration agents by providing many different views and interfaces for interaction with an artefact. In Section 2.3.2, a digital watch and other timepieces are used as the basis of a case-study to demonstrate the construction of cognitive artefacts using empirical modelling principles.

The process of constructing geometric models, as well as the process of describing and reasoning about abstractions associated with geometric models, can be supported by applying empirical modelling principles and definitive programming techniques. Many dependencies arise as constraints between the components of geometric entities, such as:

- two lines have a common end-point;
- two lines are set at perpendicular angles to one another;
- sides of a regular polygon all have the same length.

Section 2.4 examines the process of modelling abstract geometry on a computer system from an empirical modelling perspective, as well as the realisation of this geometry by lines and faces, vertices and point sets. In Section 2.4.1, consideration is given to the use of abstraction by designers in the design process. The CADNO definitive notation for the representation of geometry is introduced in Section 2.4.2. A case-study demonstrates the use of the CADNO notation to represent the abstract geometric relationships in a model of a table[1].

In the final section of this chapter (2.4.4), a brief description of the *EdenCAD* tool [Car94a] is given. This tool, implemented to work as a modular part of the *AutoCAD* computer-aided design package from AutoDesk [Aut92a] allows a user to establish dependencies between defining parameters for geometry from within the package and offers support for empirical modelling. The relationship between EdenCAD and the work in this thesis is discussed[2].

2.2 Background to Empirical Modelling with and for Geometry

Geometric modelling on a computer system is a well-established area of research that has developed alongside advances in the capabilities of computer hardware, particularly computer-graphics hardware. Many geometric modelling software tools are available for Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM) and Computer-Aided Engineering (CAE). They are often expensive and sold into a competitive market, usually on the strength of their graphical user-interface. Empirical modelling, in contrast, can be viewed as a novel approach to programming a computer system that is established as a research project located at one university.

[1] The kind of table used here is a dining table or a desk, rather than that used to tabulate data or organise information in a thesis.

[2] It is coincidence that EdenCAD is developed by my namesake Alan Cartwright.

As a result, there is a wealth of literature relating to CAD/CAM/CAE and only a relatively small local group of papers relating to empirical modelling.

This thesis discusses issues related to bridging the gap between geometric and empirical modelling through the cross propagation of concepts. This process essentially involves improving data representations and efficiency of the implementation of empirical modelling methods, to bring them into line with geometric modelling methodologies. It is not possible to approach this work from the opposite perspective, by adapting existing CAD/CAM/CAE tools for empirical modelling, as these tools exploit aspects of conventional programming paradigms that conflict with empirical modelling priciples. The approach adopted here is to explore the scope for extending empirical modelling to encompass abstractions in geometry.

A brief background and introduction to current CAD techniques and tools is presented in Section 2.2.1. The two main data representation techniques (B-rep. and CSG) for geometric modelling are described, along with recent work on implicit shape representations. Empirical modelling involves the identification and expression of observed dependencies between data. In existing geometric modelling tools, techniques exist to support similar expressions of observed dependency and these are known as *parametric* and *variational* modelling. In Section 2.2.2, these techniques are distinguished from the representation of dependency in a definitive script.

Section 2.2.3 describes the history of the Empirical modelling project. The work described is based at the University of Warwick and the associated references are, therefore, mainly local to Warwick. Although there are significant similarities between spreadsheets and programming with definitive notations, definitive programming methods form a paradigm for programming whereas spreadsheets are applications written with more conventional procedural programming techniques. The types of data stored in spreadsheet cells are generally simple types such as

integers, floating point numbers, dates and strings. The definitive programming paradigm has the potential to support more complex data types[3].

2.2.1 Geometric Modelling

The development of geometric modelling on computer systems has developed alongside the development of computer graphics hardware. One of the earliest tools for two dimensional drafting on computer systems is Sutherland's *Sketchpad* [Sut63], the first proposal for a human/computer interface through graphics. Manufacturing industry developed many Numerical Control (NC) systems for controlling manufacturing equipment, such as lathes and milling machines, by computer program. This prompted much commercial work on *sculptured surface modelling*. This included Bézier's work for Renault on the development of *Bézier Surfaces* [Far90]. At the time, there was a need to be able to simulate the NC process on computer systems prior to manufacture because of the potential for errors in the NC code. This initiated study into how to model and represent three-dimensional geometry of apparently solid material on computer systems.

The addition of the third dimension proved to be a challenging problem. During the 1970s, there were two distinct research groups investigating the representation of three-dimensional geometric shape on computer systems. The product of this work is two separate representations:

CSG *Constructive Solid Geometry* techniques represent shape on computer systems as a finite number of boolean set operations (union, intersection, set difference) on half spaces defined by algebraic inequalities. Based on work carried out at the University of Rochester by Voelcker and Requicha, the seminal journal paper on CSG techniques is by Requicha [Req80]. Similar half-space represen-

[3]This is demonstrated in the ARCA, DoNaLD and SCOUT implementations, described in Section 2.2.3.

tations were developed independently by Okino et al [ONK73].

B-Rep *Boundary Representations* of shape consisting of facets that are subsets of planar, quadric or toroidal surfaces were developed at the University of Cambridge by Braid [Bra79].

Requicha et al developed the PADL-1 notation [VRH+78] (the *Part and Assembly Description Language*), implemented as a computer based tool for constructing CSG models. Shape is constructed by textual description, where the construction sequence is represented by a sequence of assignments to variables in the model. These variables can be associated with defining parameters of the model and its component shapes. PADL-1 is mainly regarded as a research tool that does not support the full coverage of geometry required in industry[4]. This has led to the development of PADL-2 notation [Bro82] and its prototype implementation P2/MM, a computer-based tool for solid modelling with support for both CSG and boundary representation models. PADL-2 supports data types for primitive solid shapes (block, wedge, cylinder, sphere, cone), primitive faces (plate, disk, cylinder face, sphere face, cone face), half spaces defined by surfaces (planar, cylindrical, spherical, conical) and line segments.

The PADL-2 notation was widely adopted and developed by many industrial companies during the 1980s, as noted by Sheridan in [She87]. McDonnell Douglas and other industrial partners devoted a large amount of effort to improving the rather academic and text-based implementation to create an application with graphical user interface support for the modelling process. This application allows a non-programmer to select points by choosing them on the screen representation of the model and click on icons to select shape primitives and operations. This removes the need to write code in order to construct models.

[4]See "*A tale of technology transfer*" by Voelcker, an inset to [Bro82].

Cadetron developed *The Engineering Works* [She87], the first solid modelling package to run on the PC. The package incorporates PADL-2 as its kernel modeller and data representation. This implementation dispenses with the textual construction of models, in favour of an interactive graphical interface with multiple windows.

The data representations for PADL-2 only allow geometric data and some standard attributes (colour, bounding box) to be associated with geometric entities. Geometric data may be appropriate for the design of shape, but for other processes such as the generation of NC machining code and finite element material stress analysis, information relating the features in the geometry with its components is also required. For example, a machine tool may be able to machine edges accurately to a certain tolerance but can drill holes with much greater accuracy. It is therefore necessary to know which edges in the geometric model relate to drilled holes and which edges to the boundaries of the object. A data representation technique for feature based modelling is described by Ansaldi et al in [AFF85, AF88].

The development of graphical user-interfaces has significant benefits for mechanical engineers who are not expert programmers. However, it is not possible to program with a graphical interface for the purpose of integrating solid modelling within other applications. Bowyer developed the sVLIs [Bow94] geometric modelling kernel modeller as a C++ library to support integration of CSG modelling into bespoke computer-based modelling tools. The *ACIS* kernel modeller and programming toolkit from Spatial Technology is widely used for B-rep modellers [Cor97]. Bowyer et al also developed *Djinn* [BCJ+95, BCJ+97], an *application programming interface* (API) to standardise procedure calls to libraries of procedures where the representation is independent of language or point sets, regardless of the underlying software or hardware. Paoluzzi et al [PPV95] have carried out research into the use of functional programming techniques in the construction of geometry. Programming techniques

and interfaces to solid modelling allow a programmer to construct solid geometric models, build new application domain specific user interfaces for geometric modelling and perform simulations or generate animations from solid geometric models. The *CAS.CADE* C++ programming library[5] from Matra Datavision allows a programmer to construct bespoke modelling tools and is also used by Matra Datavision as the underlying library for their latest family of modelling software products.

Sculptured surface modelling techniques (Bézier surfaces, B-spline surfaces) can be merged with solid modelling techniques using boundary representation. It is not yet possible to completely integrate these techniques completely with CSG modelling and there is a divide between CSG solid and B-rep surface modellers. This has been partially achieved by Kirshnan and Manocha [KM96] (by plugging NURBS bounded solids into the CSG-tree) and by Berchtold and Bowyer [BB98] (by integrating support for Bezier surfaces and CSG modelling). It is interesting to note Shah and Mäntaylä's comment on the future of modelling programs and the separation of CSG modelling and sculptured surfaces [SM95]:

> "More lately, *implicitalization techniques* have been introduced that may eventually make it possible to merge sculptured surfaces also in CSG models."

Implicit techniques are adopted in this thesis, and implicit and CSG techniques are merged in the case-studies in Chapters 7 and Chapter 8. Implicit surface representation techniques, the *function representation* of shape, are a current topic for research by Pasko, Savchenko and their colleagues [PSA93, SP94, PASS95, MPS96]. In general, point sets constructed from any representation of shape in Euclidean space can be combined into this uniform representation of shape that includes all the standard CSG operations and many more. Traditional CSG mod-

[5] See `http://www.matra-datavision.com/`.

els, closed boundary representations, volumetric objects [KCY93] and skeletal-based implicits [BBCG$^+$97] (also known as *blob-tree* models [WvO97]), in common use in computer generated animations, can all be combined with the function representation of geometric shape. The rendering of shapes represented by function representation require the arbitrary and repeated sampling of mathematical functions at points in Euclidean space, a procedure that is difficult to optimise.

The definitive notation "*HyperJazz*", developed by Adzhiev et al [APS96], is of particular interest in this context. This notation allows a modeller to design shapes using function representation and express dependencies between the defining parameters of the shapes. The notation requires its users to have a good understanding of the function representation of point sets and it would be advantageous if tool included a library of geometric primitives that could be drawn upon during modelling. It would also be beneficial if it were possible to express dependencies between shapes as well as between defining parameters. The latest version of HyperJazz, entitled "*HyperFun*", contains better support for geometry but no longer supports the expression of dependency between parameters.

2.2.2 Variational and Parametric Modelling

The expression of dependency between defining parameters and geometric entities is supported by many geometric modelling tools. Mathematical expressions of desired relationships between numerical variables express the constraints between component entities of a model. These include:

- constraining two points so that they are the same distance apart;
- constraining two lines to be parallel to one another in a particular plane;
- constraining the radii of two circles to be proportional to one another (circle A has twice the radius of circle B).

The terms *parametric model* and *variational model* are used almost interchangeably to describe models containing geometric constraints. Their differences are explained later in this section.

The overall design process for parametric and variational systems is similar. It is described by the following four stages[6]:

1. A user creates the nominal topology of a design using standard geometric and solid modelling techniques. The result is a model with the same underlying combinatorial structure as the final geometry, but without exact dimensioning. In other words, all component entities are connected in the way that they will be connected in the final model.

2. The user describes the relationships between entities in terms of geometric constraints.

3. Once the constraints are specified, the modelling system applies a general constraint satisfaction procedure to produce an evaluated model with dimensions. The system may be over-constrained leading to no solution, or under-constrained leading to many possible evaluations.

4. The user can create variants of the model by changing the values of constrained values, or even the constraints themselves. Each change creates a new evaluation of the model through the re-execution of the complete constraint solving procedure of stage 3.

It can be seen from the stages above that this process is very different from propagation of change in a spreadsheet, where only values in cells that require re-evaluation are computed. If every change in a spreadsheet required the re-evaluation

[6]Process as described by Shah and Mäntaylä [SM95].

of every cell, the system would be seen as inefficient. The definitive programming process is closer to the construction and use of a spreadsheet model than the parametric/variational design process, yet it can represent dependencies similar to those expressed through geometric constraints using definitions[7]. Moreover, with the definitive approach it is possible to express relationships between the topology of the models during the conceptual design phase, integrating stages 1 and 2. This integration process is discussed further in Section 2.4.

PADL-1 [VRH+78] implements a constraint satisfaction mechanism in a procedural way. A sequence of assignments to variables of a model constructs the model. Changing one of the values in the construction sequences and re-executing this sequence results in a new evaluation of the shape. The order in a procedural script of definitions is important and the flow of computation describes models that are *unidirectionally parametric*. The description of the topology of the geometry has to be expressed with references to its defining parameters, leading to the rigid coupling of stages 1 and 2 of the design process.

Flexible constraint satisfaction is independent of the construction order of the geometric model. Parametric systems apply sequential assignments to variables in a model, where every assignment is computed as some function of previous assignments. The ordering of the sequence of assignments is determined by an algorithm. For example, to constrain a point (x, y) in two dimensional space to lie on the line passing through the origin $(0, 0)$ and point $(2, 1)$, the mathematical constraint $y = \frac{1}{2}x$ is used. This system works well unless the value of x depends on the value of y in another constraint. One solution algorithms is the graph method devised by Serrano [Ser87]. Another, called *DeltaBlue*, is credited to Freeman-Benson et al [FBMB90].

[7] The expression of some constraints may require the use of *Higher-order dependency* as described in Section 3.2.2 of Chapter 3.

Variational modelling systems represent geometric constraints using sets of equations. These equations are solved simultaneously to evaluate the dimensions for a model. To constrain a point (x, y) as above, the equation $x - 2y = 0$ is solved by numerical or symbolic methods along with the equations for all the other constraints. One possible numerical algorithm is given by Press et al [PFTV92] — this is based on the popular *Newton-Raphson* solution method. The algorithm iterates through a series of estimated solutions until the estimation is accurate within a certain tolerance level. This algorithm requires an initial guess for the solution. Another approach is to use symbolic manipulation to solve the system of equations, as described by Kondo [Kon92]. In both the symbolic and numerical cases, it is required that there are mechanisms to deal with sets of equations that are over- or under-constrained.

Whether variational or parametric modelling techniques are implemented in a computer-aided design system, the system solves a set of constraints with an algorithm and is conceived in the framework of conventional procedural or declarative programming paradigms. For instance, the GEOMAP-III parametric modelling tool [KSW86] allows a user to represent constraints between geometric entities using a logic programming notation, solving the constraints using inference rules. In contrast, the definitive programming paradigm, founded on empirical modelling principles, represents dependencies as observed from an agent viewpoint and not as logical propositions. This representation encourages open development at the conceptual design phase without the need for early circumscription to topological or mathematical relationships. Stage 3 of the parametric/variational design process imposes a barrier to open development resembling the separation established by compilation of source code to executable binary. Compilation is not required in the definitive programming paradigm.

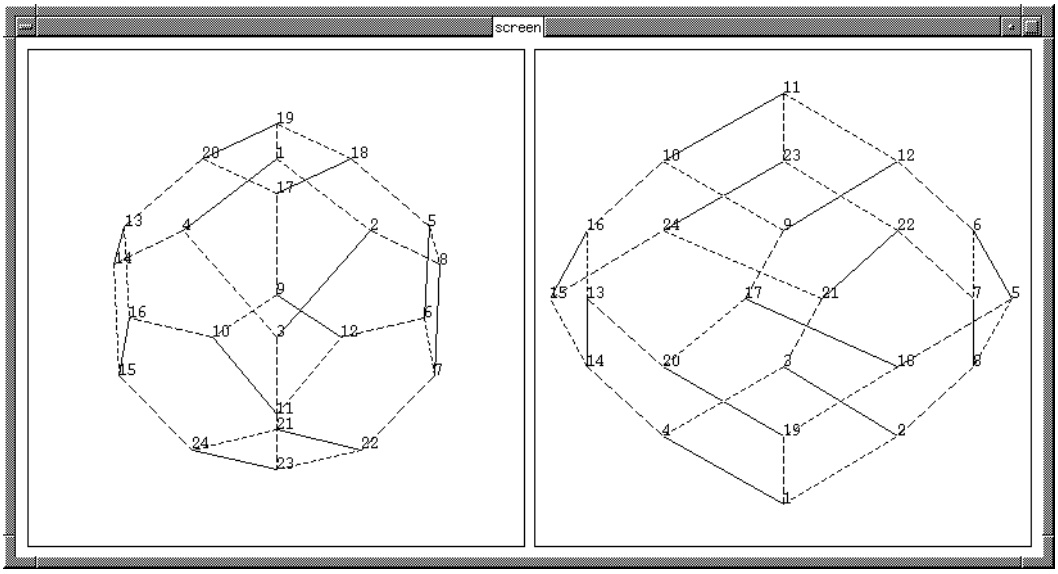


Figure 2.1: Three dimensional Cayley Diagrams generated by the ARCA tool.

2.2.3 The Empirical Modelling Project

The Empirical Modelling Project (EMP) at the University of Warwick originated from a study by Beynon in 1983 of a tool interactively to display and manipulate Cayley diagrams [Cox65], entitled ARCA [Bey83, Bey86a]. Two Cayley diagrams rendered by this tool are shown in Figure 2.1. The research report describing ARCA contains a detailed description of a context free grammar for a notation that supports both definition and declaration of variables. These variables can be of type integer, vertex, colour and diagram. This notation is of interest in the context of this thesis as it includes a *moding* mechanism for handling the dependency between components of data structures in the underlying algebra of the notation.

Interest in the general concept of definitive notations by Beynon resulted in the development of EDEN (the **E**valuator for **DE**finitive **N**otations), a computer program that is an interactive evaluator for generic definitive notations [Yun90] (see Section 1.2.1). This tool is a script interpreter which in use is similar to interaction with a shell program on a UNIX system. Users of this interpreter can at any point

in their interaction:

- create a variable or modify the value of an existing variable;
- create a dependency between variables by introducing a new definition or modifying an existing definition;
- create code for a function or modify a function that is or will be used as part of a definition;
- create or modify code for a procedural triggered action;
- inspect the current values of variables, their current definition and the current code for functions and triggered actions.

The syntax of EDEN has control constructs that mimic those in the ANSI C programming language, with *for*, *if-then-else* and *while* statements. Triggered actions are necessary to reveal the current state of the value of definitive variables in the script. When the value of a variable is updated, an action can be set to be triggered as a result of this update. By analogy with a spreadsheet, actions perform an equivalent operation with the updating of one spreadsheet cell causing the display of other cells dependent on it to be redrawn, once they have had their value recalculated. Triggered actions are required as an interface between a definitive script in the definitive programming paradigm and a procedural operating system based in a traditional procedural paradigm, through textual and graphical input and output.

The design of a definitive notation for interactive graphics called DoNaLD (the **D**efinitive **N**otation for **L**ine **D**rawing) is described in [ABH86]. The data types of the underlying algebra of DoNaLD are geometric entities such as points, lines and circles. Dependencies between these entities can be expressed using definitions, as

illustrated in Figure 2.2[8]. An implementation of the notation through translation into EDEN definitions is described by Beynon and Edward Yung in [BY88b]. Communication between the translator and the EDEN interpreter in this implementation is via the UNIX command pipeline. For many years, this translation methodology remained a mechanism that could be used to implement newly designed definitive notations or improved versions of existing ones. For example, there is a new implementation of the ARCA notation with translation into EDEN on-the-fly [Bir91] that uses the drawing procedures of DoNaLD.

The SCOUT notation (notation for **SC**reen Lay**OUT**) for screen layout by Simon Yung [Yun93] is another definitive notation that is implemented using a similar translation mechanism. On initialisation, SCOUT opens a window within the screen of the current window-manager. It then becomes its own definitive window-manager within this space. SCOUT can control the layout of sub-windows, buttons, text and provide a *viewport* for DoNaLD drawings. Viewports provide zoom and pan facilities for DoNaLD 2D drawings. SCOUT has no built-in capability to draw shapes and lines.

Issues in the design of DoNaLD and its use as an interactive graphics language were evaluated in [Bey89], a report that addresses the problems of creating and referencing a recursively described shape, such as an infinite staircase. Apart from the first stair, each stair of the staircase is defined to be the same as the one below it, translated by a suitable vector. The report poses an open question: How is it possible to make reference to the n-th step of the staircase?[9] The implementation method for DoNaLD cannot support recursive shape description or reference.

[8] In this screen snapshot figure, the DoNaLD definitions are shown in the window labelled "`Tkeden: DoNaLD Definitions`". The section of code shown is the definition of the flag on the chess clock's face that is pushed up by the minute hand and then drops the instant that the minute hand reaches the vertical position.

[9] For example, how can I hang a bucket on the 56th step, or animate a ball rolling down the stairs from the 109th step?

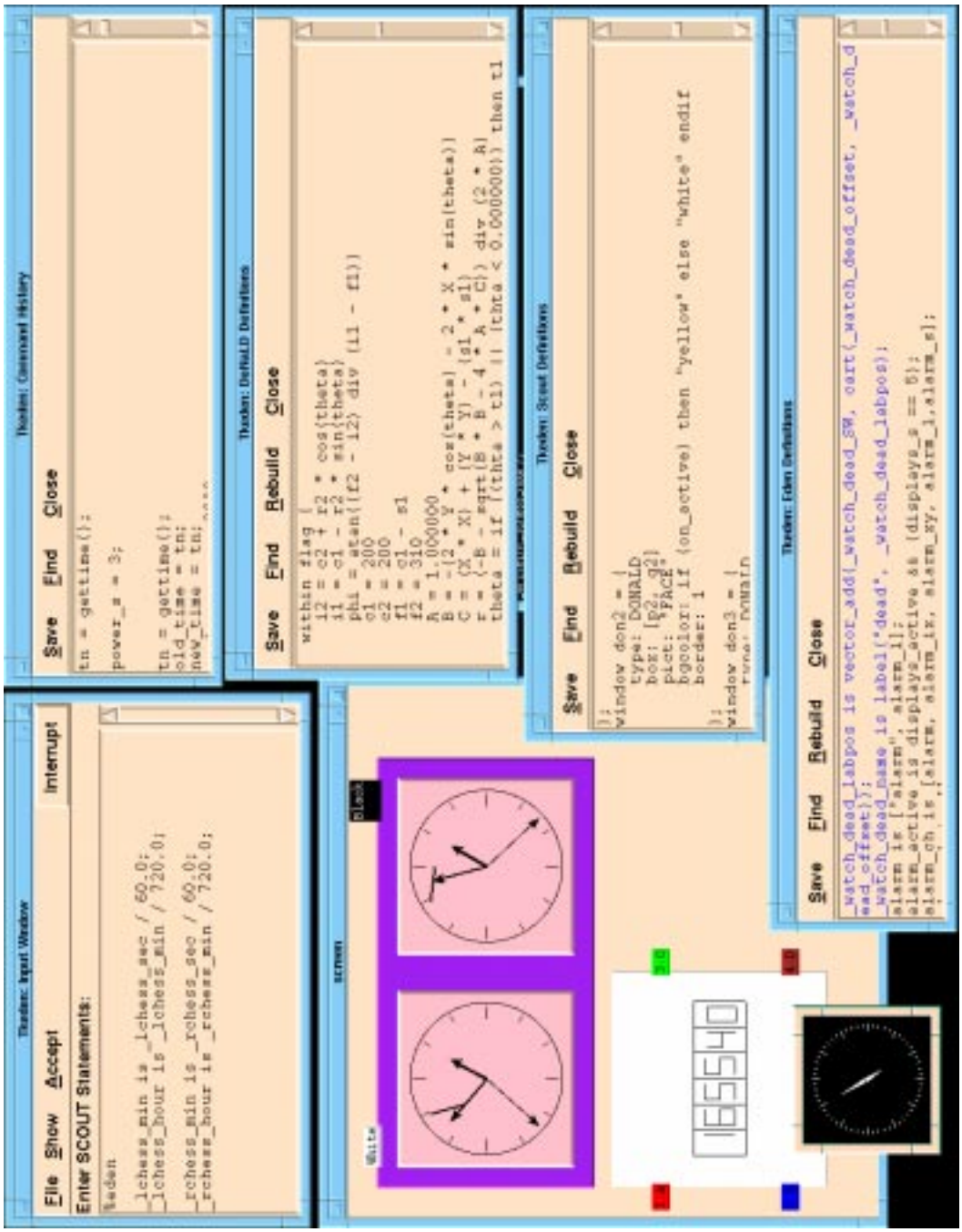


Figure 2.2: Chess clocks constructed from the digital watch model.

Ideally, DoNaLD should also incorporate definitions for shape description like those used by van Emmerick et al in their hypertext approach to the interactive design of solid models [vERR93].

DoNaLD has potential to be used in the engineering design process as a tool for drafting and displaying models, where the dependency between geometric elements can be linked to observed dependency in the model's real world referent[10]. This resulted in research by Beynon and Alan Cartwright into new definitive notations which were specifically aimed at CAD, rather than just simple line drawing tools [BC89]. The CADNO definitive notation (**C**omputer-**A**ided **D**esign **NO**tation) for the description of combinatorial structure and coordinate geometry is described in [BC88] and in Section 2.4.2. A partial implementation of the notation was made by Stidwill for his third year project [Sti89]. This implementation is based on translating CADNO definitions to EDEN definitions. The tool produces three dimensional, wire-frame, black and white graphical output of the geometry represented by scripts of CADNO definitions.

More recent research into the use of definitive notations for CAD software examines the design of definitive notations that support interactive conceptual design [BC92] and concurrent engineering design [BACY94a, BACY94b]. Redefinitions in a definitive script can represent the actions of different agents. This means that the privileges for redefinitions can represent the roles of different designers in the design process. The LSD specification language [Bey86b] is used to describe agent privileges for redefinition in a script. This process is called *agent-oriented modelling*. An agent has definitions in a script that correspond to its current state, definitions of the states of other agents that in can observe, its privilege to change other agents definitions and protocols for action. LSD specifications can be animated through the use of the Abstract Definitive Machine (ADM) [Sla90]. The roles of different

[10] For example, if a lamp is resting on a table, then moving the table will also move the lamp.

designers in the design process can be specified using LSD. Adzhiev examines the roles of different agents in the implementation of CAD systems in [Adz94].

Collaboration between Beynon and Adzhiev initiated further research into definitive notations for interactive shape modelling using implicit function representation techniques. The *Hypersurf* tool [PSA93] developed by Adzhiev and his colleagues provides an environment for experimenting with implicit function representations of shapes. The textual description of shape in Hypersurf has many similarities with a definitive script: the association of a left-hand-side identifier for a function representation with its definition. This motivated further discussion of the benefits of definitive programming techniques for the representation of geometry using function representation.

In my undergraduate project work, I designed and implemented a new definitive notation entitled *CADNORT* [Car94b]. This notation is based on a combination components of the syntax of CADNO and *Hypersurf*. The implementation supports textual interactivity with a notation for the design and encapsulation of the function representation of shape on-the-fly. It is possible to create libraries of shape primitives. Only limited graphical output of these models is available in the form of visualisations of two-dimensional slices through point sets, a task for which the EDEN interpreter is particularly inefficient. CADNORT raises many issues concerning the use of complex data types in definitive notations. These are discussed in Chapter 3.

The line of development of the tools SCOUT, DoNaLD and EDEN has culminated in an amalgamated tool entitled *Tkeden*[11]. This tool integrates the SCOUT, DoNaLD and EDEN front-ends with the EDEN engine for definition maintenance. It offers an attractive window interface environment for the interactive management

[11] The **Tk** in *Tkeden* refers to a switch in implementation strategy made by programmer Y. P. Yung from basing EDEN on the X Windows programming libraries to using Ousterhout's *Tk/Tcl* library. See [Ous90, Ous91].

of definitive scripts, as well as being less demanding on system resources than previous versions of EDEN. Figure 2.2, to be discussed in detail in Section 2.3.1, shows the *Tkeden* tool in use. The current implementation of the graphical tool does not support the UNIX shell pipeline. The result of this is that writing new definitive notations becomes a complex software development task, requiring a developer to have an in-depth understanding of the *Tkeden* source code.

The term *empirical modelling* was adopted for the work in 1995. This change was intended to make a clear distinction between conventional mathematical modelling and the phenomenological, experiential models described by definitive scripts. The adoption of the term reflects the fact that there are some primitive underlying concepts that bring coherence to the diverse activities of modelling using definitions, which has a deeper purpose in trying to model the world. These primitive underlying concepts are common to *Definitive Programming*, *Observation-Oriented Modelling*[12] and *Agent-Oriented Modelling*.

2.3 Cognitive Artefacts

A *cognitive artefact* is an object or an environment that is contrived to represent (through metaphor) interaction with some other (typically real-world) system[13]. An artefact exhibits different states, each of which corresponds to a state of the system that it represents. Interaction with an artefact in this mode of representation allows a human interpreter to explore the correspondence between the artefact and its referent beyond preconceived limits. This interaction is used as a well-established means of communication in many disciplines and is distinguished from communication via documents expressed in a formal language. Examples of artefacts include

[12] For an explanation of *Observation-Oriented*, see `http://www.dcs.warwick.ac.uk/modelling/hi/principles/observation.html`.

[13] Definition for *Cognitive Artefacts* given in [BC95].

models of engineering systems, architectural models, molecular models, maps and globes.

Cognitive artefacts serve a useful function in the general design process (rather than the geometric design process in particular) in two different ways, listed below. Both kinds of representation are especially significant in relation to modern trends towards concurrent engineering [BACY94a, CB91] and several different artefacts are necessary to take account of a whole range of design participants, users and scenarios. The two purposes of artefacts in the design process are:

- artefacts with which the designer can interact in order to develop an insight into the nature and current status of an object being designed;
- artefacts that inform the designer about the current status and progress of the design process.

The purpose of discussing cognitive artefacts in this thesis is to explore the empirical modelling process and its requirement for good metaphors for interaction that can be provided through multi-media interfaces. Solid geometric models can provide interaction with computer-based artefacts that closely imitates interaction with the real world referent. Current tools that support the empirical modelling process provide only simple line-drawn graphical models. Through understanding the relationship that exists between artefacts and their graphical representation, there is potential to support empirical modelling for richer interaction with and more detailed construction of artefacts. This process will benefit from better geometric models to assist in the communication of the model. The relationship between empirical modelling and cognitive artefacts is explored in Section 2.3.1 and a case-study examining cognitive artefacts for timepieces is presented in Section 2.3.2.

Many different agent views of the same artefact can exist. For example, in the mechanical design process a component part may be the responsibility of a

geometric designer, an electrical engineer, a materials expert and others. Each agent has its own viewpoint of the artefact and the nature of their interaction with the artefact is dependent on their viewpoint. Issues in the use of artefacts as a means of communication between agents are discussed in Section 2.3.3.

2.3.1 Cognitive Artefacts and Empirical Modelling

Empirical modelling can be used for constructing artefacts by specifying each state of a referent in terms of the current values of a collection of observables[14]. Primitive interactions with the referent by manipulation of the observables are specified as synchronised changes to these observables. The artefact has its own collection of observables and primitive interactions that correspond with the observables and primitive interactions of the referent. This presumes a metaphor for the representation of the current state of the referent, whereby observables (variables) of the computer model correspond to observables of the referent. The use of such metaphors can be illustrated by considering different representations of time. For example, consider the following three metaphors for time:

- the position of hands on a clock;
- the value on a numerical display;
- the position of the sun in the sky inside some virtual environment, such as a computer game.

The exact and explicit correspondence between states and transitions in artefact and referent plays a fundamental role in the empirical modelling method, and is the basis for being able to recognise the association between them. It is through experimentation with the artefact that the characteristic relationships between ob-

[14] This content of this section is based on work previously presented in [BC95].

servables that determine the identity of the referent are revealed. Traditional programming techniques render the current state of the screen that represents a model using procedural algorithms. The empirical modelling approach is to treat the state of the screen as part of the computational state of a model. The state is defined by a system of variables, whose values are appropriately synchronised in change to represent the dependencies between graphical elements. These graphical elements are often of a geometric nature, such as lines, circles and polygons.

It is the representation of dependency, both between graphical elements and the observables that form computer-based artefacts, and the linking of these two kinds of dependency, that facilitates construction and animation with artefacts. Through the use of this dependency, it is possible to represent the effect of spontaneous human agent interaction in a system, continuous processes and instantaneous events. The geometric primitives available in current empirical modelling tools such as DoNaLD do not allow for the representation of geometric shape in more than two dimensions by techniques such as boundary representation and CSG methods[15], or for adequate representation of dependencies between components of shapes. As a result, current support of metaphor for the representation of artefacts using empirical modelling on a computer system is limited, and not generally suitable for tasks such as mechanical design. These tasks may require three-dimensional models of shape to achieve the goal for the construction of artefacts: to place into correspondence the states and state transitions of the artefact and referent, and to be able to recognise the association of the artefact with the referent through interaction with the artefact.

[15] See Section 2.2.1.

2.3.2 Case-Study - *Timepiece Artefacts*

Figure 2.3 depicts three artefacts representing aspects of timepieces: a digital watch, an analogue clock and a statechart (cf. Harel [Har87b, Har87a]), which represents the functionality of the digital display. The clock faces are artefacts for communicating time and the statechart serves to tell a designer how the display functions of the digital watch are affected by the pressing of watch buttons and the level of charge of the battery. In Figure 2.3a, the digital watch face is surrounded by four buttons that respond to mouse button clicks. The buttons are linked by definition to the statechart in Figure 2.3c, where the current internal state of the watch is given by boxes with solid lines rather than dotted lines.

In a real analogue clock, the hands of the clock may be coupled mechanically so that when the hour hand moves, the minute hand moves with it. This dependency can be expressed with a definition for *angle_hour_hand*[16], the angle between vertical and the hour hand measured clockwise, in the following way:

$$angle\_hour\_hand = \frac{angle\_min\_hand}{2\pi} \times \frac{\pi}{6} \tag{2.1}$$

This dependency is represented in the analogue clock face in Figure 2.3b in a graphical line drawing representation, where the angle of the second hand *angle_sec_hand* is linked to the angle of the minute hand in the same way. The time on the digital watch shown in 2.3a is linked by a definition of *angle_sec_hand* to the number of seconds through a day. This is calculated from the time display of the digital watch.

Sundials are timepiece instruments that could be represented with the support of three-dimensional graphical representations of artefacts. Time is represented on a sundial by the shadow of a pointer cast by the sun on to a graduated disc. The pointer is raised above the flat disc. An environment could be constructed for the

[16] The angle of the minute hand *angle_min_hand* is measured modulo 24π.

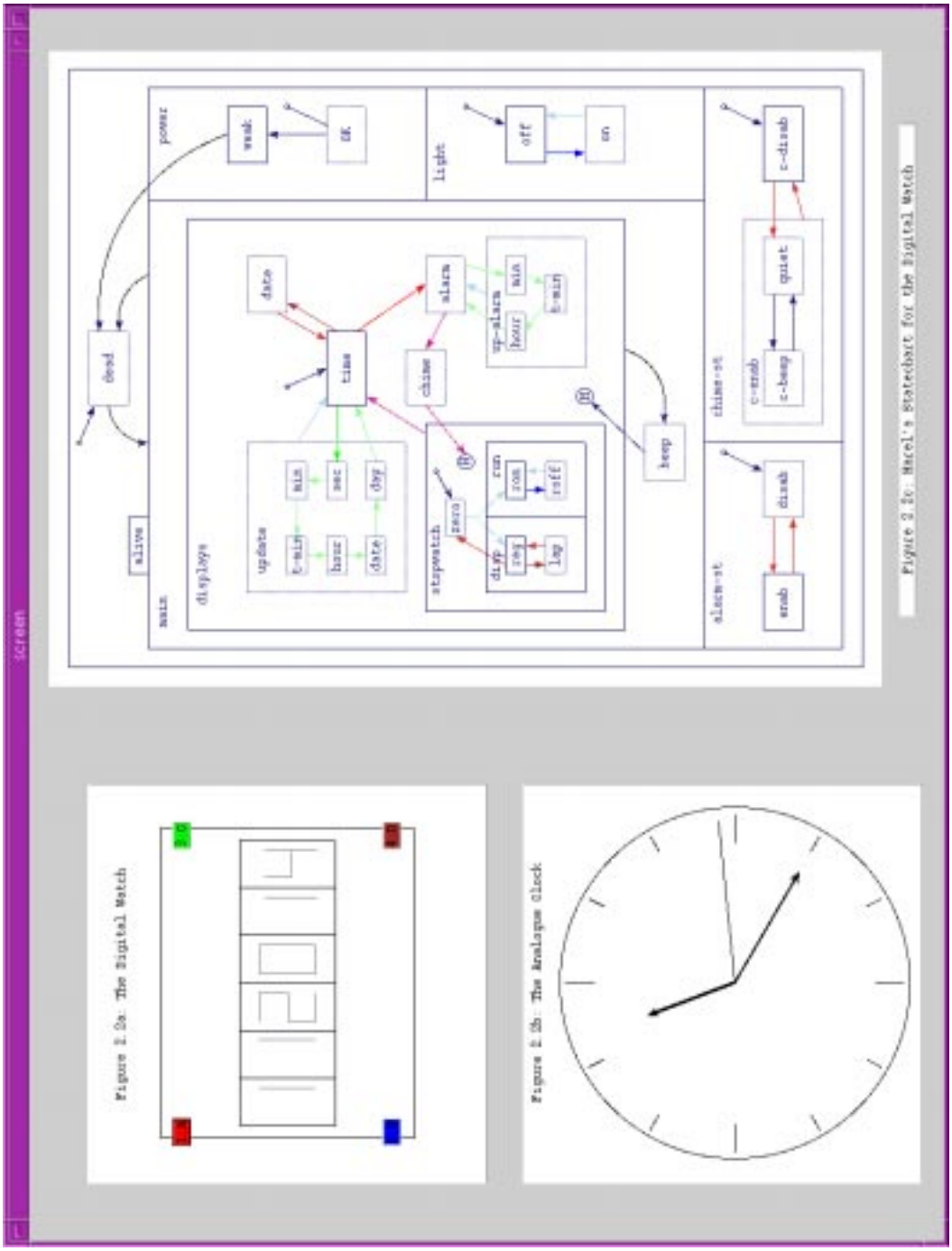


Figure 2.3: View of the *Tkeden* cognitive artefact for a digital watch, analogue clock and statechart.

exploration of sundial artefacts, where the rotation of the earth causes the apparent motion of the sun throughout the day. In this environment, it is possible to explore agent views of time by:

- correctly and incorrectly positioning and orienting sundials;
- the effect of breaking or modifying the geometry of the pointer of a sundial;
- examining an agent's misconceptions of time when glancing at the graduations on the disc from an oblique angle;
- comparing the margins for error in viewing time during different seasons, in bad weather and in different countries of the world.

Characteristic patterns of change in observables in the referent model are identified through observation and experiment, and are always subject to revision in the light of subsequent observation. The process of developing an artefact naturally involves incremental construction that is guided by evolving knowledge about the properties of the referent. In this process, it is essential to be able to record the current status of partial models, and revise these to take account of new information and perspectives on the referent as they are encountered. Empirical modelling principles can be used to support this process.

Figure 2.2 shows the *Tkeden* development environment in use for the construction of, and interaction with, the model of a chess clock[17]. The figure shows the definition input window (top left), SCOUT definition viewing window (top right), DoNaLD definition viewing widow (middle right), translated to EDEN definition window (bottom right), model viewing window (middle left) and an independent

[17] The nature of development and interaction with the *Tkeden* tool is difficult to convey in a snapshot image. It is only the experience of the tools over time that demonstrates the full potential of the tool. Note that only the interpretation and maintenance of definitions has occurred during the construction of the model and no compilation has taken place

computer system clock (inset bottom left). The model was developed on-the-fly directly from the digital and analogue clock models, reusing the components (analogue clock face, relationship between the hands) of the time artefacts to construct the new chess clock cognitive artefact. The existing clock mechanism is linked to the chess clock model to make the clocks tick and runs in real time, simultaneously with the digital watch model. This on-the-fly construction of models in *Tkeden* supports computer-based rapid prototyping methods for the incremental development and reuse of components of artefacts.

With support for good geometric representations, the tools can provide much better virtual environments for interaction and rapid prototyping. Support for solid geometrical models that can be used as visual metaphors for the communication of the current state of a cognitive artefact and can assist the incremental construction of a modeller's evolving insight into a model. For a clock model, a three dimensional model can be used to experiment with the design of the clock casing or the design of the clock hands. It is also possible to consider inventing a completely new spatial method for the representation of time, such as a spinning globe representing the world with the location of dawn and dusk indicated by illuminated and dull areas. This is well beyond the scope for data representation in the current *Tkeden* tool, which supports only data types for two-dimensional line geometry through translation from DoNaLD definitions to EDEN definitions. The data structure used for the representation of DoNaLD data is a general EDEN data structure that is not specifically designed for the representation of geometry[18].

2.3.3 Agent Views of Cognitive Artefacts

The timepiece models demonstrate how empirical modelling techniques can support three kinds of agent interaction with a model:

[18] Further discussion of the DoNaLD to EDEN translation process is presented in Section 3.4.1.

- Interaction with the model through interfaces provided to represent the interface of the referent. This is illustrated in Figure 2.3a, where virtual buttons that represent the real buttons of a digital watch can be pressed by mouse clicks.

- Revision of the model through open-ended redefinition to represent revision and incremental construction of the model. This is demonstrated by the on-the-fly construction of the chess clocks model, as shown in Figure 2.2.

- Autonomous agents who take action to change the model without the intervention of a user modeller. An autonomous time-updating agent exists in the timepiece examples.

It is also possible to treat the components of a model and their relationship as interacting agents. For example, rather than the *analogue* definition of the relationship between the hour hand and the minute hand expressed in Equation 2.1, we can conceive an autonomous agent that observes the position of the minute hand. Every ten minutes, this agent updates the position of the hour hand.

Harel's vision of software development for reactive systems [Har92] emphasises the role of visual formalisms, in particular the use of statecharts [Har88]. The widespread adoption of statecharts in software development methodologies (cf [Rum91]) is a motivation for interpreting statecharts as cognitive artefacts within the empirical modelling framework. The concepts of state *depth* (a state that can be one of many possible sub-states) and *orthogonality* (a state with several coexisting sub-states) are central to the construction of statecharts. The depth concept can be represented by an agent who plays more and more specific roles in a model, the orthogonality concept by agents cooperating simultaneously. These roles and privileges of agents can be described by the LSD specification notation [Bey86b].

Statecharts as cognitive artefacts can play an important role in the concurrent engineering process [BACY94b, CB91], where several cooperating human agents are collaborating in the construction of an engineering model. The artefact represents the current internal states of the model and allows for the discussion of states and of how the state model can be collaboratively improved by the agents. The statechart can also be used to represent and to compare several prototype models for the same artefact. Visual formalisms assist design agents in the process of bringing together several different versions of a design artefact and committing to components of these to form a new artefact.

Each human agent may have a particular focus on an aspect of the design, e.g. the materials used or the electrical properties of a referent. Practical support for the concurrent engineering process across distributed systems is a current topic for research. No support for associating variables with agents and controlling their privileges to redefine or reference other variables is provided by EDEN. There is further discussion of LSD and support for multi-user modelling in Section 6.3.

2.4 Abstract Geometry from an Empirical Modelling Perspective

Commercial modelling packages place their primary emphasis on geometry and only recently have efforts been made to capture a greater part of human design intent in the construction of models[19]. Feature-based approaches to geometric modelling (cf. [SM95, AF88]) allow the association of high-level information, such as the result of *cutting* a cylinder away from a block of material being a *hole*, with the good mathematical representation of the geometry of the model formed by the boolean set-difference operation for the cut operation. In general, abstracting the

[19] The work presented in this Section (2.4) has previously appeared in a technical research report [BCCY97].

geometry of a real-world object entails discounting some of the characteristics of the real object. It seems that there is a need to liberate a designer from the obligation to specify explicit geometry at the initial, conceptual phase of the modelling process. A designer typically conceives and manipulates an object in terms of its combinatorial features rather than as an abstract point set in Euclidean space.

Geometric modelling can be viewed as the construction of cognitive artefacts for the exploration of shape, both as *metaphor* for communicating structure in a model in the manner of a skeletal sketch, and as a virtual reality object that can be viewed as a *product*. Empirical modelling can be used to support the conceptual and initial phases of the geometric design process, as well as in the specification of mathematical models for geometry. Empirical modelling techniques can also be used to establish dependencies between feature information, combinatorial structure, attributes, design processes and geometric models in an open-ended way. Section 2.4.1 examines the relationship between design process, geometric modelling and empirical modelling. This is followed by the description of the CADNO definitive notation for geometric modelling (Section 2.4.2) and a case-study in the use of the CADNO notation for a *table* (Section 2.4.3).

2.4.1 Meta-Modelling in Design

The concept of *meta-modelling* is introduced by Tomiyama in [Tom89]. He expresses the need for modelling methods that combine the representation of objects with the representation of the design process itself. His vision is a design framework in which there are many descriptions of the design product, all of which are related and draw on 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. Empirical modelling

principles have the potential to represent the dependency between the geometric and non-geometric data and scripts (or sections of scripts) of definitions can be used to record versions, viewpoints and create libraries of parts. These scripts can be managed in the same way that text documents are managed by a word processor or desk top publishing system.

The ADDL system, developed by Veerkemp [Vee92], is one proposal that involves the integration of two kinds of knowledge:

1. Knowledge about objects - their attributes and characteristics.

2. Knowledge about the design process - a designer's intention.

Non-geometric information is recorded in the knowledge base for each object. An important characteristic of Veerkemp's conception is that the designer has a creative role to play in the selection of scenarios that shape the development of the evolution of objects.

The context for the paradigm shift proposed in the work in this thesis is supplied by a conflict between two engineering cultures that are well characterised in the words of Brödner [Brö95]:

> 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 knowldge 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 observation.

The essential need for interaction between user and computer makes its impossible to construct a computer modelling system that exclusively and faithfully represents a closed-world perspective. This is because experience is necessarily involved in interaction between the user and the systems. A user can interpret system responses in ways that are beyond the scope of what has been formally specified. The openness of interaction between user and computer system can be suppressed by restricting the experimental channels available to the user. The products of the closed world culture are systems that offer preconceived framework for interaction between the designer and the computer. The nature of the content relation that associates the computer model with an external referent is prescribed and the manner in which the computer model is shaped by the designer's interaction is predetermined.

During the modelling process, definitive programming techniques support computer-based modelling that makes explicit provision for both the content relation and the user-computer interaction to be enriched beyond preconceived limits. Is it possible to preconceive which information about an object will be significant in the design process and what exceptional scenarios will need to be considered? In general, the closed world approach to modelling cannot address these and other

concerns of meta-modelling. Description of the design of a product from a number of different viewpoints of the design of an object cannot be expected to be coherent and consistent. Interaction with definitive scripts allows for the introduction and testing of new scenarios for designs on-the-fly, with support for "*what-if?*" experimentation, as well as the representation of interdependencies between different viewpoints of a design.

If the profile of a table top is transformed from a square to a circle, at what point does it cease to have corners? Moreover, if reference is made to the corners of the table to connect its legs, what is the new location of the table legs during and following the transformation? Human insight is typically necessary for decision making in design and computer-based tools for design should support this essential involvement of this interpretive activity in the development process. The problems of computational efficiency that arise when dealing with shape as a product have encouraged investigation of closed-world frameworks for shape modelling, where optimisation of algorithms takes priority over situating shape modelling in a wider context.

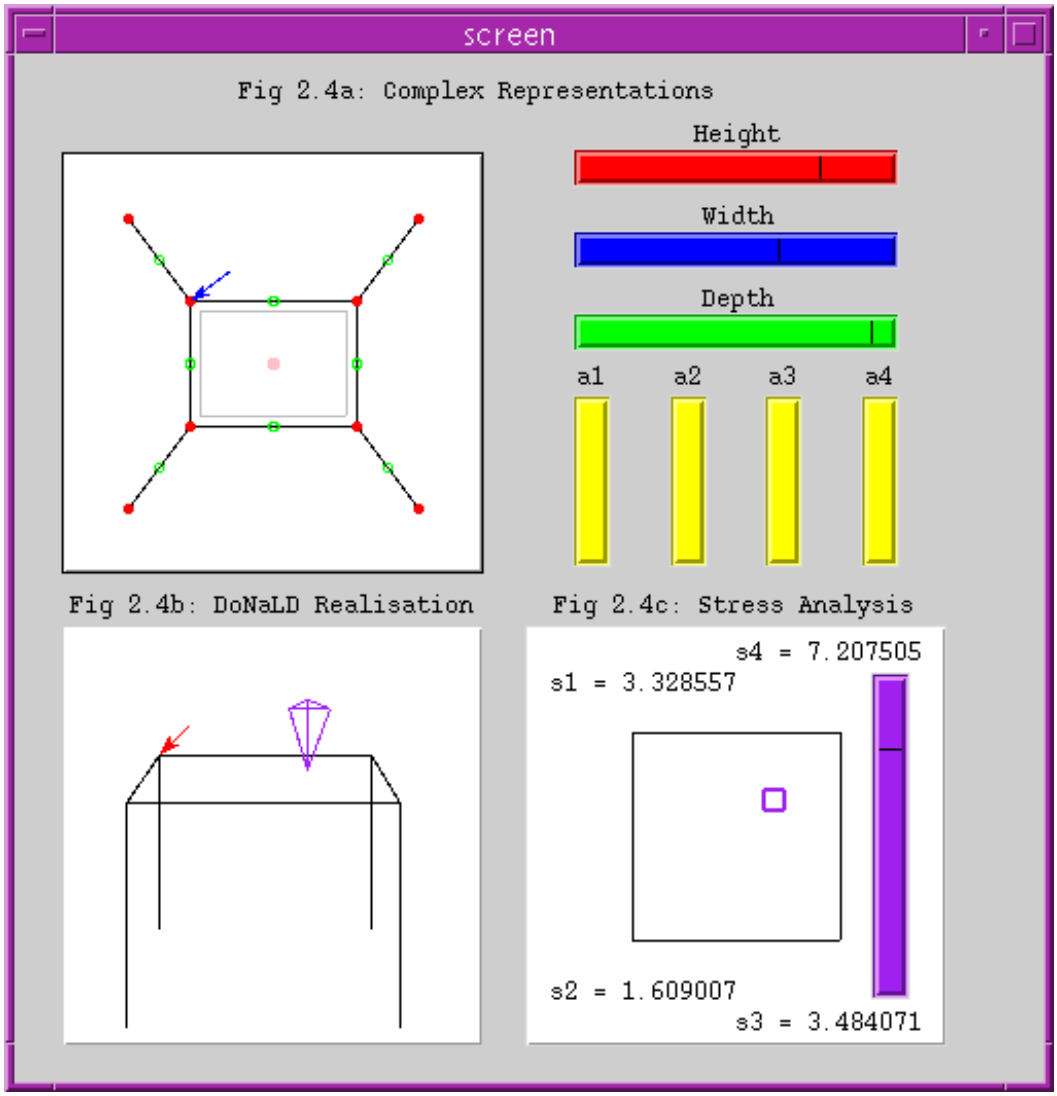
Definitive programming can be thought of as a programming paradigm that provides a spreadsheet-like environment for interaction with computer systems. Chmilar and Wyvill [CW89], and Emmerick, Rappaport and Rossignac [vERR93] demonstrate the potential for applying spreadsheet principles to shape modelling. Nardi's thought-provoking analysis of the spreadsheet paradigm culture [Nar93] indicates that 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 programming paradigm. The research in this thesis examines the enabling technologies for and the implications of such a radical paradigm shift to support geometric shape modelling.

2.4.2 The CADNO Notation

The definitive notation for computer aided design (CADNO) introduced in [BC89] can be used to model *abstract geometry*, which places emphasis on the combinatorial and metaphorical aspects of geometry rather than Euclidean point sets. In variational and parametric design, dependencies can be expressed between parameters that define the location of geometry by formulating constraints. These dependencies are determined after commitment to the topological / combinatorial structure of the geometry. The CADNO notation allows for both the expression of dependency between components of combinatorial structures and between scalar parameters. Dependencies can also express the relationship between these two areas and the realisation of geometric objects as graphical representations of Euclidean point sets.

Variables for the representation of combinatorial structure do not necessarily have a realisable counterpart in geometric shape. In an agent view of the DoNaLD notation for line drawing, a designer's mental model of a table object may have corner observables that are end points of lines and each corner can be clearly identified (see Figure 2.4b, to be discussed in Section 2.4.3). These corners are distinguished from points in a wire frame quasi-realistic rendering of a geometric table (as shown in Figure 2.4d) that are observables for the computer application used in rendering algorithms. 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 2.4d), yet the concept of a corner as a feature of the table is essential for high-level communication about table design.

The data types in the underlying algebra for CADNO are summarised in the table below:



a, b, c

d

Figure 2.4: DoNaLD and VRML realisation of an abstractly defined table.

complex	list-of label × list-of list-of label
d-complex	complex × int
frame	list-of vector-of real × d-complex
object	list-of frame × (list-of frame → (vector-of real → boolean))

In general terms, a complex is a combinatorial structure used to specify the abstract geometric entities from which a geometric object is synthesised, such as points, lines, faces and scalar values. A complex can be realised in Euclidean space as a frame, by attaching coordinates and values to its vertices. 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, a list of frames can be interpreted as the basis of a boundary representation, a CSG model, an implicit function specification, or simply a three-dimensional line drawing[20].

No complete implementation of CADNO yet exists[21] and this thesis does not describe one. CADNO is viewed here as an ideal notation that fits well within the framework of definitive notations conceived prior to CADNO. This thesis examines new ways to approach the implementation of definitive script notations with reference to the technical problems encountered when trying to use existing definitive notations for geometry. This approach leads to the new *empirical worlds* definitive notation for computer-aided geometric design. This notation exhibits many of the same qualities as the CADNO notation. The CADNO notation is compared to the new approach in Chapter 9.

2.4.3 Case-Study - *Table*

To demonstrate the potential for the use of the CADNO notation, a case-study for the modelling of a table is presented. Table 2.1 shows a listing of definitions in

[20] The three-dimensional version of DoNaLD I developed for this case-study was further improved by MacDonald in his third year undergraduate project [Mac97].

[21] An attempt to implement a tool to support the CADNO notation exists. It is written by Stidwill as part of his undergraduate degree project [Sti89].

the CADNO notation that captures the combinatorial structure, point coordinate locations and relationships between scalar parameters of a designer's model of a physical table. This table has four folding cylindrical legs and a square profile table top with rounded corners.

In the following list, some of the definitions in the CADNO script for the model are described:

`table` A three-dimensional combinatorial structure representing eight identifiable reference points `A` to `H` for the table, as well as their interconnections. The legs of the table are represented by the pairs "`[A, E]`", "`[C, G]`" etc. and the table top face as the quadruple "`[E, G, H, F]`".

`angles` The four scalar quantities representing the current angle of fold for each of the four legs.

`table1` An association of coordinate points with the labels in the combinatorial structure "`table`". Notice the dependencies expressed between labels from complexes not yet associated with this frame. These associations occur during the realisation process.

`angle1` A frame that associates a scalar value with the fold angles from complex "`angles`" and represents limits ("`subject to`") for the range of values the scalar values can take.

`pict1` A geometric object on three frames with a realisation as a line drawing in a three dimensional version of DoNaLD. The dots in the braces ("`{ .... }`") should be replaced with a template description of how to realise the geometry from the frames that the object is based on.

The listing in Table 2.1 is a specification for the geometric models shown in Figure 2.4. The figure is a simple demonstration screen, constructed using *Tkeden*,

```
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 }
                        .... }
```

Table 2.1: CADNO script for a table model.

that is contrived to illustrate how CADNO could be integrated with other empirical modelling tools. Figure 2.4a depicts the underlying complex that represents the table: it provides a mode of reference to the three dimensional DoNaLD image in Figure 2.4b of the figure through mouse sensitive spots. The location of a feature of the table in Figure 2.4b, such as the position of the foot of a particular leg, is identified by an arrow when the appropriate spot on the combinatorial structure (`complex`) is selected.

Figure 2.4c 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 (values `s1` ... `s2`) when a load is placed on the table top. The tool supplies an interface through which the load can be relocated by direct manipulation and includes a slider bar to allow its mass to be scaled. The load is also represented by an inverted prism in Figure 2.4b. Other *slider* controls allow for experimentation with the scalar parameters that define the geometry for the table, such as its height, width, depth and the angle by which its legs are folded (`a1` ... `a4`) within the range limits specified in the CADNO script.

A more realistic rendering of the table geometry is shown in Figure 2.4d. The image is generated by first creating a section of VRML code[22] to represent the geometry consistent with its template definition in the CADNO listing, and then rendering this in a VRML browser. Existing dependency maintenance tools do not support good renderings of objects and the most complex graphics currently available in *Tkeden* is three-dimensional line drawing with labelling and attributes (cf Figure 2.4a, b and c).

[22] VRML, the *Virtual Reality Modelling Language*, is discussed further in Chapter 7 and [ANM96].

2.4.4 The EdenCAD Tool

With standard two-dimensional computer graphics hardware, it is impossible to generate a realisation of a high-level parametric description of a geometric object as fast as it is possible to update a line drawing in a DoNaLD image. To improve efficiency, there is a CADNO-like tool developed at the University of Warwick by Alan Cartwright as part of his doctoral research [Car94a], which links a definitive notation with a geometric modeller. *EdenCAD* is a Lisp-based dependency maintainer that makes use of the AutoLISP interface [Aut92b] to the AutoCAD draughting design package [Aut92a]. The idea is that by utilising an existing package to handle the realisation of objects, efficiency in the realisation of geometry can be improved. Dependencies can be established between the defining parameters of the AutoCAD geometry by EdenCAD definitions.

In general, this strategy leads to some loss of control over the representation without the construction of a sophisticated interface to the dependency maintainer. The most common problems are concerned with reference to the components of 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. For example, consider the problems of implementing a definitive notation for screen layout when windows are subject to relocation to respect "intelligent" constraints (such as visible, non-overlapping, automatically scaled).

The work in this thesis takes a different approach to efficient implementation from EdenCAD. It investigates how greater efficiency can be achieved by improving data representations and algorithms for dependency maintenance. Consideration is given to the exploitation of three-dimensional computer graphics hardware, now becoming commonplace in high-end computer workstations and personal computers alike. This hardware can be utilised through software interfaces, such as *OpenGL*

programming libraries [WND97].