# Chapter 1

# Introduction

Empirical modelling is concerned with the exploration of artefacts through experimentation and the construction of new instruments that support interactive elaboration of these artefacts. These artefacts inform the process of discovering new concepts through the use of metaphorical representations of state. By exploiting novel computational abstractions and through a broad concept of what constitutes a computer — any reliable, interpretable, state-changing device — empirical modelling can be used to create computer-based artefacts that explicitly imitate phenomena as observed. Artefact construction is similar to the process of developing an engineering prototype [Rus97]; it is open-ended and does not require early circumscription to a models components or parameters. Further exposition of empirical modelling principles in relation to education and learning can be found in [Bey97], and in relation to the foundations of artificial intelligence in [Bey98a].

Empirical modelling has the scope to support early conceptual development, concurrent engineering, reactive systems design, computer-aided design and software requirements capture. Geometry in its broadest sense plays an important role in establishing metaphors for the representation and communication of the states of artefacts. In this thesis, aspects of geometry that support metaphorical state

representations are studied with reference to the principles of empirical modelling. This process necessarily involves the design and implementation of tools that allow empirical modelling to exploit richer geometry. These tools can provide appropriate and flexible interfaces for empirical modelling that support concurrent engineering, possibly on distributed computer systems, in an efficient manner. The process also involves the development of empirical modelling tools and techniques to support design. This requires re-appraisal of existing implementation techniques to better support script management, data structure, agent privileges and solid modelling.

The representation of artefacts benefits from strong visual metaphors that closely imitate their observed real world referent. Empirical modelling aspires to the use of photo-realistic images and virtual reality environments for the real-time animation of models, although empirical modelling on a computer system can be adequately carried out without these benefits.

## 1.1 Motivations for Empirical Modelling

Human agents play many different roles in design and the creative process. Some of these many different roles are categorised in the triangle in figure 1.1. In Section 1.1.1, an analogy with the composition and performance of music is drawn to illustrate how modern technology supports the conflation of these many roles. In this thesis, empirical modelling and geometric shape modelling are central themes. The processes of modelling shape and composing and performing music have a significant affinity as both involve establishing metaphors for aspects of the real world that have an analogue character. The figure refers to the roles of agents in the design of computer hardware and software systems, to assist in the explanation of the relationship between empirical modelling and the development and the use of computer systems in Section 1.1.2.

### 1.1.1 An Analogy from Music

In a traditional view of music, a *composer* writes music in a well-understood notation. This music is played by a *performer* who adds their own interpretation based on their insight and understanding of the composer's other work and general style. The performer (of non-vocal music) plays the music on an instrument created by an *instrument maker* who has assessed the available materials and used their knowledge of the construction of similar instruments to build a new one. The composer does not need to know about the construction process for the instrument or how to play music with it to write music for it. They only need a conception of the kind of sound that instruments make[1]. The performer does not need to know how to compose or about the internal structure within a composition be able to play it. Nor do they need to know how to construct the instrument they are playing.

The composer is creating recipes for interaction between performer and instruments from their conceptualisation of the interaction process. This process is prescribed to a certain extent by the musical notation that is used to communicate between the composer and the performer. The performer executes this interaction following the recipe to a certain degree and with an expertise on how to use their instrument within a musical framework, such as where to place their fingers on the string of a violin to achieve a particular note. Many instruments allow the user to play notes that cannot be recorded in conventional musical notation. A violinist can choose to place their finger half way in between two notes that can be scored and produce a quarter tone. This ability of the performer to play beyond anticipated limits is explored in music of the twentieth century, where composers have invented new notations for the communication of sound concepts such as quarter tones and notations that give the performer more freedom for improvisation[2].

[1] The British composer Vaughan-Williams wrote at least one concerto for every instrument of the orchestra, while only being able to play a few of them himself.

[2] A seminal composition that demonstrates the use of new notations for music is Penderecki's

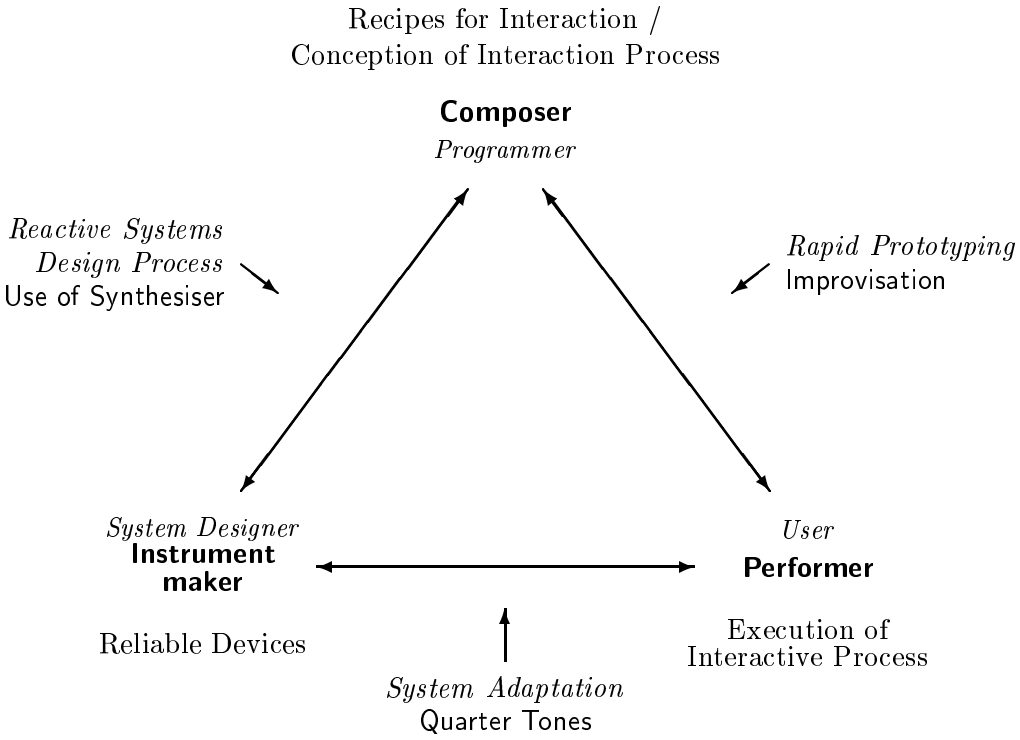


Figure 1.1: Roles in musical and *computer-based* interaction.

Technical developments have supported the conflation of the three musical roles, with significant benefits to all agents involved. The electronic synthesiser is a new musical instrument that allows a composer to experiment with new sounds that are beyond the scope of existing instruments, or can assist a performer to compose music by recording their improvisations and translating them into a musical notation via a computer system. Similar conflations of roles can benefit concurrent engineering, where there is a need for exploratory contexts that support references, values and privileges. Empirical modelling can support the conflation of these roles by providing interactive, computer-based exploratory contexts.

### 1.1.2 Computer-Based Interaction

In computer-based interaction, there are three conventional roles for human agents. The computer *systems* expert knows about the physical aspects of constructing reliable computational devices. The *programmer* (consider in the broad sense as an agent who specifies, designs, codes and tests software) creates recipes for interaction with the computer system. The interface between the programmer and the systems expert is through programming languages, libraries of code and common abstractions. The computer *user* uses applications created by programmers for their own purposes and has no interest in source code for the applications. All three roles must exist for the general use of computers in this conventional framework.

Empirical modelling on computer systems supports the conflation of the roles of programmer, user and systems expert by allowing them to explore common areas between their roles. In the development of *reactive systems*, an agent needs to have a good understanding of the reliable devices in the system to create software to control the system. This is the same for the evolving process of design, where a designer needs to understand the materials and environment that situate their model

*Threnody for the Victims of Hiroshima.*

to create new and improved designs. Alternatively, computer users may decide to use their computer hardware in a way that was not anticipated by the hardware designers, such as the use of video-conferencing equipment as security surveillance devices.

Suchman introduces the concept of *situated actions* in [Suc87]. She considers all purposeful actions as situated actions that take account of particular, concrete circumstance. She argues that no matter how carefully you make a plan before an event, it cannot take account of unforeseen circumstances. When asked to describe a plan in retrospect, plans filter out the detail that characterises situated actions. Situated actions are similar to the common understanding of music by the composer and the performer. A performer may adapt their performance to suit an audience or current stylistic fashion. In the same way, a user of a software package for computer aided design (CAD) may wish to explore the possibilities of shape beyond those permitted by the user interface of their software package[3] in the same way that they might use a paper sketch-pad.

## 1.2 Motivations for this Thesis

This thesis examines geometric aspects of empirical modelling from two angles:

1. *Empirical modelling for geometry*, where empirical modelling tools and techniques support geometric shape modelling, conceptual design and concurrent engineering;

2. *Empirical modelling with geometry*, where geometry supports the empirical modelling process in a non-geometric design context.

[3]The incorporation of a *Visual Basic* interpreter in the Microsoft *Excel* spreadsheet [Jac97, KDS96] is an example of a computer program that allows a user to become a programmer, tailoring the spreadsheet functions to their own needs.

The possibility of supporting both these angles through implementation on computer systems is examined in detail. In relation to geometric modelling, this thesis concerns the creation of instruments for geometric modelling that can support the development of computer-based applications for shape modelling and an exploratory design environment for a designer using a computer. Much of the discussion in the thesis can be applied generally to the implementation of tools that support empirical modelling, although the results presented are based on research relating to how to improve existing tools to better support geometry. Most of the case-studies presented relate to geometry and shape modelling.

Existing work on empirical modelling, and its application in a number of different case-studies[4], has demonstrated potential to support open-ended development and experimental interaction in several different contexts. In relation to supporting the empirical modelling angle, the work in this thesis is a continuation of existing research into the combination of empirical modelling and geometric modelling [ABH86, BC89, Car94a]. In the previous work, both existing tools that support empirical modelling and interfaces to CAD packages have been investigated. The aim was to bring the support for empirical with and for geometric modelling closer to the standard of interactivity and graphical flair that is expected by users of modern computer software. The existing tools proved too slow and it was found that the coding of CAD packages restricts the freedom of expression required to support open development.

This work is motivated by the need to overcome the technical hurdles described above and to be able to demonstrate the benefits of uniting empirical and geometric modelling. Empirical modelling is *observation-oriented*: all variables in empirical models are considered as representing the current state of some observed quantity in a referent. Empirical modelling requires strong visual metaphors and

[4]For a background to empirical modelling, see section 2.2 of chapter 2.

good interfaces for direct manipulation to represent entities as they are observed and controlled. This requires integral support in tools for the specification and manipulation of geometry. Empirical modelling takes direct account of dependency between observables in a manner that cannot be achieved by circumscribed mathematical models. In many geometric structures, there is inherent dependency (such as characteristic patterns of incidence, and dimensional constraints). For this reason, empirical modelling cannot make direct use of traditional mathematical models of geometry, and other representation techniques are required.

Section 1.2.1 introduces the essential concepts that underlie empirical modelling and the terminology relating to empirical modelling that are required in later sections. This is followed in section 1.2.2 by an explanation of the aims of this research work.

### 1.2.1 Essential Empirical Modelling Concepts

At this early point in the thesis, it is important to define some key concepts relating to Empirical Modelling related to the discussion throughout the document (see also [Tea, Bey85, BY88a, Bey97, Bey98a]). *Empirical modelling* focusses on the construction of artefacts (models) for interaction and experiment in a doamin that is not yet well understood. Analyis of a domain or referent proceeds through the identification of the fundamental structure of our personal experience of that domain. The atomic elements of this structure are *observables* which consist of unique name and value pairs. Each observable represents a measurable or quantifiable element of the domain and its current value in the model corresponds to a state of the referent. Empirical modelling principles concern the identification of measurable observables in some real-world referent and, if appropriate, the observation of dependency between observables and the agents associated with these observables.

Observables can be given a unique identifier and their current value can be

expressed by the statement "*identifier* = *value*", in the manner of assignment to a variable in a conventional programming language. This statement is a *definition* of the value of the observable. An artefact constructed through empirical modelling will have a collection of many observables, which each have their own definition. Such a collection of definitions is known as a *definitive script*. A *definitive notation* consists of the data types and operators of the underlying algebra suitable for modelling the domain over which definitive scripts for models are constructed.

Analysis of the domain also involves the observation of the syncronised patterns of changes in observables. Observables that are seen to be dependent on other observables can also be expressed as definitions by describing the relationships between their values. These definitions are of the form "*identifier* **is** *expression*", for example "`width is 2*height`". If a value that forms the right-hand side of a definition is changed, the value of a dependent observable is updated consistent with its definition. The process is similar to interaction with a spreadsheet application, where a user identifies the relationships between cells on the spreadsheet and changing the value in one cell causes other dependent cells to update automatically.

The process of constructing computer-based artefacts using definitive scripts is called *definitive programming*. Support for definitive programming on computer systems is provided by the EDEN interpreter [YY88, Yun90], a generic evaluator for definitive notations. Some EDEN definitions are shown in Table 1.1, where the value of observable `a` is defined to be equal to the sum of the values of observables `b` and `c` (`9` in the example). Subsequent change to the value of `b` or `c` during interaction with the interpreter will cause the value of `a` to update. Interaction with EDEN models takes place through an ongoing process of typing definitions line-by-line on-the-fly. These definitions represent the introduction of new observables and the *redefinition* of existing ones.

The definition of observable `d` in Table 1.1 to depend on the values of `b`

```
b = 4;
c = 5;
a is b + c;
d is f(b, c);
e = a;
```

EDEN definitions

```
func f {
  return $1*$1 - $2;
}
proc p : c, d {
  writeln(c, d);
}
```

EDEN functions and procedures (actions)

Table 1.1: Example definitions, functions and procedures for EDEN.

and `c` by a user defined *function* `f`. An EDEN specification of the function `f`, where `f`$(x, y) = x^2 - y$ is shown in the table. If the code describing a function is updated, then all values defined using the function are updated. EDEN also includes procedural triggered `actions`, like `proc p` in Table 1.1. These are segments of code that are executed every time the values of particular observables are updated. In the example, every time the value of `c` or `d` changes, the current values of both `c` and `d` are written out by the EDEN interpreter program[5].

### 1.2.2 Aims

Empirical modelling principles have not yet been applied to the specification of data types for underlying algebras for observables and their operators. In an ideal general definitive programming tool, it would be possible to introduce new data types on-the-fly to correspond with new observations. Using existing tools, a domain to be modelled must be understood in terms of the underlying algebra for a new definitive model in that domain prior to model construction. The efficiency of implementation of an ideal tool should be sufficient to provide realistic interaction corresponding to a user's experience of the domain, including domains containing geometric models

[5]Note that there is a distinction that exists *only* in the EDEN notation between the tokens "`is`", for a definition where a value is maintained automatically to be consistent with its definition, and "`=`", for a value that remains exactly the same until it is redefined. In the example, observable `e` remains equal to the value of observable `a` at the exact moment of definition.

with a large volumne of associated data. The existing tools provide no support for abstract data types, are not suitable for large volumes of data and execute slowly as they are based in interpreters. A central aim of this thesis is to investigate and experiment with other ways of implementing empirical modelling instruments to support generalised data representations and improve efficiency, so as to enhance the quality of stimulus-response interaction with tools.

This thesis also aims to address issues that are not tackled by existing tools such as EDEN, or at best are handled rather clumsily. These include:

- the representation and support for the privileges of interacting agents;
- unifying representations for references, values and privileges to support applications such as concurrent engineering;
- the potential for a greater degree of flexibility, efficiency and portability;
- the dynamic instantiation/elimination of definitions;
- support for higher-order definitions;
- integrated extension to general data types and operators beyond those currently supported.

The integration of support for general data structure and geometry will lead to richer computer-based models that correspond more closely to their real world counterparts. It will be possible to imitate geometric characteristics of the referent in the metaphor for interaction with an artefact. By improving the efficiency of implementation, it should be possible in the longer-term to produce convincing *real-time* animations and simulations of real-world reactive systems. A central goal of the work is to demonstrate the potential for the use of empirical modelling principles to support open-ended geometric modelling.

## 1.3 Contents of Thesis

In this section, a brief outline of the thesis is presented as a guide to the structure and dependency between the chapters and sections. The thesis describes a model for formulating and reasoning about dependency maintenance and includes information about three new tools developed as part of the research work. The technical issues raised by existing tools are identified, and a method for overcoming these is proposed. A new Java class library that allows a programmer to implement new applications for shape modelling and integrate a wide range of shape representations is presented. A method of integrating these geometric techniques into spreadsheet-like environments that are well-adapted to support empirical modelling is proposed. In section 1.3.2, the overall contribution of the thesis is described.

### 1.3.1 Thesis Layout

This thesis is divided into nine chapters, of which this is the first introductory chapter. Chapter 2 provides the background to this work, highlighting the relevant literature from geometric modelling as well as empirical modelling. This chapter also includes a discussion of the underlying concepts for empirical modelling with geometry to support the construction of computer-based artefacts, and empirical modelling to support the abstract, open development of geometric models. Case-studies to illustrate these concepts address modelling of timepieces and physical tables.

Chapter 3 examines the technical challenges of representing geometric data and dependency in definitive scripts, by examining existing definitive notations and other small case-studies. At the end of the chapter, the use of serialised data types to represent all types of data in a definitive notation is proposed and justified. Chapter 4 presents a method of reasoning about dependency maintenance free of concerns

of data types and structure, called the *Dependency Maintainer Model* (DM Model). The new *block redefinition algorithm* that improves efficiency in dependency maintenance by considering several redefinitions in a block simultaneously is described with the DM Model. The chapter ends by examining the relationship between dependency maintenance and conventional algorithms, such as sorting.

Chapters 5 and 6 describe two programming toolkits based on the DM Model of Chapter 4 that both implement the block redefinition algorithm. Chapter 5 describes the design of a novel machine concept called the *Definitive Assembly Maintainer Machine* (DAM Machine), contrived to maintain dependencies between words of computer RAM store. This machine has been implemented over the ARM architecture [Fur96] and programmed to support the DoNaLD definitive notation for line drawing [ABH86]. This use of the toolkit is described and the possibility for efficient animation is demonstrated with a case-study of an engine model.

Chapter 6 presents the *Java Maintainer Machine* (JaM Machine) application programming interface (API), which maintains dependency between Java objects [CH96]. The toolkit offers support for multi-user collaborative working and distribution of scripts to several computer systems simultaneously over a TCP/IP network [Har97]. Chapter 7 and 8 describe *empirical worlds* — a case-study in the use of the JaM Machine API for shape modelling. Chapter 8 includes a description of the *empirical world builder* application that integrates the empirical world class library and supports shape modelling within a World-Wide-Web browser application. Virtual reality worlds (cf. VRML [ANM96]) are used as the display mechanism for geometry described in a definitive script.

The thesis ends in Chapter 9 by drawing conclusions from the research described. Further research work is proposed, including the possibility of a new style of application to support shape modelling that is based on spreadsheet ideas.

### 1.3.2 Contribution of Thesis

In respect of empirical modelling, the work in this thesis contributes a new method for handling data structure with dependency to better support observation-oriented modelling. The new *block redefinition algorithm* provides a more efficient means to propagate updates through the values of observables by considering more than one redefinition simultaneously. The JaM Machine API brings many of the features available in object-oriented programming to empirical modelling. The use of compiled code in JaM and assembly language directly in the DAM Machine allows for empirical modelling that supports the efficient and smooth animation of models. This also enhances the scope for interactive experimentation with a model that plays an essential role in its construction and interpretation. The use of the Java programming language provides a platform-independent way of integrating existing (not definitive) libraries of objects for graphical user interfaces, device control, networking and databases, into applications that support empirical modelling.

In respect of geometric modelling, the thesis presents a new approach to representing dependency in geometric modelling that allows for open development and supports conceptual design "*what-if?*" experimentation. This approach is different in character from variational and parametric modelling techniques in that it does not require the solution of several constraints. The use of implicit function representation for shape allows designers to explore many different representations for shape in a unified design environment. This is demonstrated by the proof-of-concept *empirical world builder* tool. Scripts of definitions that represent a geometric model can be shared between more than one workstation and there is support provision (by the JaM Machine API) for collaborative working and concurrent engineering.

For more general computer science, this thesis provides an insight into a new approach to the use of objects in object-oriented programming, where the commu-

nication between objects is through dependency-maintenance mechanisms. Insight into how the integration of general data structure and dependency can be achieved in the same definitive script provides the possibility to use empirical modelling in a broader range of applications. These definitive scripts can support the incremental construction and open-ended development of models over a diverse and extendable range of data types. This could provide a basis for new powerful spreadsheets where the values in cells can be richer data types than text, numbers and dates alone. It also provides support for software development and modelling using computer systems in which the roles of the user, programmer and computer systems designer are conflated.

During the research work associated with this thesis, a number of other publications were made jointly by the author and other members of the empirical modelling group. These are [BC95, GYC+96, BC97, BCCY97, ABCY98].