THE UNIVERSITY OF WARWICK

Computers for learning: An Empirical Modelling perspective

by

Chris Roe

Thesis

Submitted in partial fulfilment of

the requirements for the degree of

Doctor of Philosophy in Computer Science

Department of Computer Science

University of Warwick

November 2003

Contents

List of Figures

List of Tables

List of Listings

Acknowledgements

I am indebted to a number of people who have helped me in various ways during the writing of this thesis.

First and foremost, I would like to thank my supervisor and friend Meurig Beynon for his unselfish support throughout the preparation of this thesis. I am very grateful for all the comments, discussions, encouragement and feedback. This thesis would not have been completed without his assistance.

Thanks also to the other members of the Empirical Modelling research group for providing a stimulating and interesting environment within which to conduct research. In particular, thanks to Steve Russ for useful guidance whenever it was requested, and a huge thank you to Ashley Ward for his friendship, discussions and seemingly unbounded technical knowledge. Further thanks to Dave Pratt from Warwick Institute of Education for stimulating discussions and constructive feedback.

Last, but certainly by no means least, I would like to thank my parents, Andy and Margaret, my sister Nicky, and girlfriend Sophie for all the support that they have given me during the course of this research. It means more to me than I could ever write down.

Declarations

This thesis is presented in accordance with the regulations for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. The work in this thesis has been undertaken by myself, except where otherwise stated.

The perspective of Empirical Modelling for educational technology is discussed in connection with constructionism in [RB02]. The restaurant manager model described in section 3.5 was discussed in [RRR00] in connection with strategic decision support systems. The digital watch model described in section 4.7 was discussed in [RBF00] in connection with engineering education. The use of Empirical Modelling to simulate LEGO Mindstorms robots described in section 6.4 was discussed in [EJR$^+$02].

Abstract

In this thesis, we explore the extent to which computers can provide support for domain learning. Computer support for domain learning is prominent in two main areas: in education, through model building and the use of educational software; and in the workplace, where models such as spreadsheets and prototypes are constructed. We shall argue that computer-based learning has only realised a fraction of its full potential due to the limited scope for combining domain learning with conventional computer programming. In this thesis, we identify some of the limitations in the current support that computers offer for learning, and propose Empirical Modelling (EM) as a way of overcoming them.

We shall argue that, if computers are to be successfully used for learning, they must support the widest possible range of learning activities. We introduce an Experiential Framework for Learning (EFL) within which to characterise learning activities that range from the private to the public, from the empirical to the theoretical, and from the concrete to the abstract. The term 'experiential' reflects a view of knowledge as rooted in personal experience. We discuss the merits of computer-based modelling methods with reference to a broad constructionist perspective on learning that encompasses bricolage and situated learning. We conclude that traditional programming practice is not well-suited to supporting bricolage and situated learning since the principles of program development inhibit the essential cognitive model building activity that informs domain learning. In contrast, the EM approach to model construction directly targets the semantic relation between the computer model and its domain referent and exploits principles that are closely related to the modeller's emerging understanding or construal. In this way, EM serves as a uniform modelling approach to support and integrate learning activities across the entire spectrum of the EFL. This quality makes EM a particularly suitable approach for computer-based model construction to support domain learning.

In the concluding chapters of the thesis, we demonstrate the qualities of EM for educational technology with reference to practical case studies. These include: a range of EM models that have advantages over conventional educational software due to their particularly open-ended and adaptable nature and that serve to illustrate a variety of ways in which learning activities across the EFL can be supported and scaffolded.

Abbreviations

AOP - Agent-oriented parser

BBC - British Broadcasting Corporation

CAI - Computer Assisted Instruction

CPL maps - Combinatorially Piecewise Linear maps

EDEN - Evaluator of DEfinitive Notations

DMT - Dependency Modelling Toolkit

DoNaLD - Definitive Notation for Line Drawing

DTkEden - Distributed TkEden

EDDI - Eden Definitive Database Interpreter

EM - Empirical Modelling

EFL - Experiential Framework for Learning

FDL - Free Distributive Lattice

ILE - Interactive Learning Environments

IPPE - Instructive Portable Programming Environment

ISBL - Information Systems Base Language

ITS - Interactive Tutoring Systems

MIT - Massachusetts Institute of Technology

OO - Object-Oriented

OpenGL - Open Graphics Library

OXO - Noughts-and-crosses style games

P4 - The lattice of subsets of {1,2,3,4}

PENGUIMS - Programmable ENvironment for Graphical User Interface Management and Specification

RAT - Relational Algebra Tutor

RCX - Robotic Command eXplorer

RSE - Robotic Simulation Environment

S4 - The symmetric group on 4 symbols

SASAMI - Solids Animation Simulator And Modelling Interface

SCOUT - SCreen LayOUT notation

SI - Spreadsheet for Images

SIN - Situation, Ignorance, Nonsense

SIV - Spreadsheet for Information Visualisation

SQL - Structured Query Language

SQLTE - SQL to EDDI translator

SQLZERO - a variant of SQL whose semantics is consistent with relational theory

TILT - Tools / Interfaces / Learner's needs / Tasks

TkEden - Tk/Tcl EDEN interpreter

UML - Unified Modelling Language

VAT - Visual Agent Talk

VBA - Visual Basic for Applications

VCCS - Vehicle Cruise Control Simulator

ZPD - Zone of Proximal Development

Chapter 1 – Introduction

The recent explosion in computer usage is transforming every aspect of our society. The rapidly increasing presence of computer technology in the workplace, the home and the school is almost certain to have impacts on the ways in which we learn. In education, the importance placed on using computers in schools is evident from governmental policy and the consequent levels of funding being deployed. For instance, in its latest policy document, the UK government has proposed that, by 2005-6, there should be an annual funding of £700 million on computer-based technology for the education sector [DFE03]. The same policy document states that the number of computers in schools has virtually doubled in the four years from 1998-2002 – an indication of the amount of money that has already been spent on computers for education. This dramatic increase in computer technology in schools is not limited to the UK. In the USA, for instance, funding for computers in schools in the year 2000 was estimated to be in the region of $8 billion [AFC00].

Governmental reports paint a rosy picture of the positive influence of computers on students and the commensurate increases in achievement that the use of computers effects. For instance, in the UK, the government claims that research shows that:

> '[Information and Communication Technology] can have a direct positive relationship to pupil performance – equivalent in some subjects to half a GCSE grade.' [DFE03]

Academic researchers are more ambivalent about whether computers enhance the quality or standard of learning. There are those who champion the use of computers by children (see e.g. [MO94, Tap98, Ben99, BR99]). However, a significant number of researchers claim that there is no positive impact on standards in children using computers, and further that the use of computers may indeed be harmful to their educational development (see e.g. [Tur95, Kay96, Opp97, KC98, Hea99, AFC00, Opp03]).

The increased interest in using 'computers for learning', and the conflicting viewpoints emerging from pedagogical research, give rise to questions that need to be addressed in order to guide future practical developments. To understand how we can use computers fruitfully in learning for both the construction and use of models, we must recognise what learning we wish to support on computers. This thesis addresses this theme with reference to two main categories of computer use for learning: the construction of models by children, and the construction of models by professional adults. The broader context for this research is conveniently framed using a similar bipartite classification in the agenda of the Human-Centric Computing symposium of 2001 [HCC01]. In respect of children realising domain learning through programming:

> 'The first part of the Symposium will focus on educational issues and end-user programming for beginners. How can kids build their own games? How can education be enriched with computational literacy allowing people to express complex ideas with interactive media? What kinds of programming approaches are particularly well suited to computer users with no programming background? What are the cognitive road blocks in programming for beginners?'

In respect of professional adults realising domain learning through programming:

> 'The second part of the Symposium will focus on professional users. How can users gain more control over their high functionality applications, such as word processors, browsers, and spreadsheets, through end-user programming? How well do end-user programming languages scale? How can end-user programming be integrated into high functionality applications? How can the reuse of end-user programs be stimulated? What are the trade offs between domain-specific and generic end-user programming languages?'

This thesis is concerned with investigating an alternative approach to providing computer support for learning that bears on many of the questions raised in the above agenda.

1.1 Construction of computer models by children

In 1980, Seymour Papert published a seminal work on computers for children, called 'Mindstorms' [Pap80]. He described a vision for how children could 'unleash the power of computer programming' to aid their learning in a personally meaningful domain. 'Mindstorms' contains many anecdotes of practical programming undertaken by children together with comments on the perceived effects it had on them. Papert's vehicle for practical programming was the computer language Logo. Logo is a simple procedural programming language originally targeted at the domain of geometry. It has a small number of commands that can be combined to produce complex geometrical patterns. In early versions of Logo, the language could be used to control a floor device called a turtle. There have been many subsequent programming languages inspired by Logo and Papert's vision. Development environments such as Microworlds Project Builder [LCS03] and Imagine [KB00] are examples of Logo-like languages in a more general-purpose context. Logo also spawned research languages aimed at specific contexts, such as massively parallel Logo (as represented in *Logo [Res94] and NetLogo [Wil99]) and Logo interfaced to physical devices such as vehicles and robots [ROP88].

In some respects, Logo has been a spectacular success, but in others a failure. This is apparent when we contrast Papert's visions for Logo with its practical applications in education. Papert saw Logo as pioneering a new wave of technology aimed at children that would liberate the child from the 'oppressive nature of school-based instruction' [Pap80]. In his vision, children would be able to use computer technology in a free way to learn personally important subjects at a pace and style that suited them. His 'Mindstorms' book was targeted at teachers and imparted his message that teaching would become a radically different profession under the influence of computer technology. Logo has been successful in terms of its widespread use – programming a floor turtle is mentioned in the United Kingdom National Curriculum [UNC03] as a way of teaching aspects of geometry. Due to this, it is almost certainly the programming language that has had the greatest exposure in UK classrooms.

However, Papert would not necessarily interpret this wide adoption of Logo as an indication of success. His second book, 'The Children's Machine', published in 1993, was already expressing concern about the way in which Logo was being used. For him, the classroom use of Logo – and computer technology in general – was moving away from his vision to reinforce traditional school-based instruction [Pap93].

Papert's design rationale for Logo was that of taking what he perceived as the best ideas from computer science and 'child engineering' them (as reported by Kahn [Kah01] in notes from a Logo project meeting in 1977). There have been a number of attempts at providing programming languages that are either designed for children, or are accessible to children. They include:

i) Boxer – a programming language that was spawned from the Logo project [diSA86]. It is a general purpose language that adopted a mixture of programming paradigms.

ii) Agentsheets – a rule-based visual programming language and development environment that is related to spreadsheets (see section 2.3.3) [Rep93].

iii) Toontalk – an animation-based programming language in which programs are created by manipulating animated tools and training robots by demonstration [Too03].

Each of these products takes its inspiration from a particular programming paradigm and attempts to develop a simplified programming language in this idiom. As Kahn mentions in [Kah01], these more recent languages are again attempting to 'child engineer' what are seen by their proponents to be the best ideas of computer science. The motivation for the educational use of the programming languages listed above is a belief that children constructing their own programs will concurrently be engaged in meaningful domain learning. In this thesis, we shall argue that the support that program construction can offer to domain learning is heavily influenced by the choice of programming paradigm – a theme we shall return to in chapter 4.

1.2 Construction of computer models by adults

In 1993, Bonnie Nardi wrote a seminal book on end-user programming called ‘A small matter of programming’ [Nar93]. Her investigations were centred on how non-programmers can utilise computers to create their own models through programming, or more generally through model construction. Nardi’s basic argument [Nar93] is that:

> ‘the problem with programming is not programming; it is the languages in which people are asked to program’.

Her research suggested that end-users can build their own models if the conditions are right. She highlighted several successful end-user languages:

i) Logo – widely used in the domain of geometry by a large number of children.

ii) Mathematica – a general software system for mathematical applications [GG00].

iii) Computer-Aided Design languages – used extensively in architecture and product design [Dug99].

iv) Spreadsheets – a ubiquitous desktop application used in many domains by an enormous number of end-users, ranging from children to accountants.

Nardi noted that these languages had been successful in part because of their domain specific primitives and in part because their design characteristics made it easier for non-programmers to learn through model building. She observed that spreadsheet development was a particularly successful end-user programming environment because [Nar93]:

> ‘Managing dependency relationships is a particularly good example of the way in which an end user programming system can allow users to focus on their domain-related problems at a very high level’.

Spreadsheets have been successful in the end-user programming domain because, in specific learning domains, there is a close relationship between constructing a model and learning about the domain. In the light of this observation, we use spreadsheets as our starting point for investigating ‘computers for learning’ (see chapter 2) and go on to connect spreadsheets with an approach to computer-based model construction that we have developed – called *Empirical Modelling*.

1.3 Introducing Empirical Modelling

In this section, we give a brief overview of the fundamental concepts of the Empirical Modelling (EM) approach. Dr Meurig Beynon initiated EM research in 1983 at the University of Warwick with the design of the definitive notation ARCA [Bey83]. Over the subsequent 20 years, the project has encompassed a wide range of interests including definitive notations [Yun90, Yun93, Run02], geometry [Car94, Car99], computer graphics [ABC$^+$98], business [BRR00, CRB00, RRR00], artificial intelligence [Bey99] and educational technology [Bey97].

Empirical Modelling is an approach to constructing models – typically computer-based – that can assist our understanding of a phenomenon. The term ‘empirical’ is used to reflect the emphasis on experiment, observation and interaction during the construction of a model. In contrast to typical computer models – which are formal mathematical models – the development of an EM model more closely resembles the development and use of a spreadsheet than that of a traditional computer program, in that the model is incrementally constructed through interaction with a partially completed model.

In EM, the primary emphasis is on modelling state-as-experienced rather than behaviour-as-abstracted, as respectively represented by spreadsheets and traditional computer programs. Whereas the construction of a traditional program relies upon the prior specification of its abstract behaviour, the development of a spreadsheet model evolves through the representation of state as currently perceived. The crucial distinction between spreadsheet development and conventional programming is that there is no circumscription of the possible future states that the system may enter. The identification of sensible behaviours is found through experimentation and interaction with the spreadsheet. An EM model exhibits similar qualities to a spreadsheet, in that the modeller has complete discretion over the interactions they perform.

Within the framework of observation-oriented state-based modelling, EM identifies three key concepts: observables, dependency and agency. Each of these concepts has a part to play in understanding and exploring a phenomenon. An observable is a perceived element of the state to which we can ascribe a value. Dependencies are indivisible relationships that exist between observables. Agency is concerned with attribution and realisation of state change.

In this thesis, we discuss and illustrate how the principles of EM can enable computer-based models to be constructed in a way that is intimately linked with domain learning. We shall discuss the concepts and orientation of EM in much greater depth in chapter 3.

1.4 Motivations for the thesis

The research in this thesis exploring connections between EM and learning is motivated by research from fields such as experimental science, education and psychology that has direct relevance to our exploration of computer-based modelling as a learning aid. Here we outline key ideas that have guided this thesis and which are discussed in detail in later chapters:

- **Constructionism**. In [Pap80, Pap93], Papert describes how children can use computers as exploratory tools to further their own private active learning. His theory of constructionism asserts that learning is most beneficial when learners are actively building their own knowledge structures in a domain of personal interest. In this thesis, we explore how

EM as a modelling approach can enable learners to construct computer-based models in a constructionist framework.

- **End-user programming**. In [Nar93], Nardi describes how successful end-user programming environments, such as spreadsheets, have allowed non-specialists to harness the power of the computational medium to construct their own artefacts to help them in solving their personal tasks. In this thesis, we explain the potential of EM as a way in which computer users can construct artefacts that builds on the principles embodied in spreadsheets.
- **Open development**. In [Brö95], Brödner describes two cultures in engineering that he calls 'closed-world' and 'open-development'. In a closed-world approach, the assumption is that all properties and relationships between objects can be stated as objectified, explicit, propositional knowledge. An open-development approach does not contest our ability to form objectified, explicit, propositional knowledge, but assumes the primary existence of practical experience that has been gained through an individual's interaction in the world. In this thesis, we explore how EM can be viewed as an open-development approach that emphasises the primacy of personal experience in constructing computer-based models that embody our emerging understanding of a phenomena.
- **Construals**. In [Goo90], Gooding describes how the physicist Michael Faraday constructed physical artefacts to enable him to understand electromagnetic phenomena. These concrete artefacts are termed 'construals' by Gooding. In this thesis, we argue that EM models should be considered as construals rather than preconceived programs and discuss the impact of this change in interpretation with reference to learning.
- **Bricolage**. In [Lev68], Levi-Strauss describes the idea of bricolage. Bricolage refers to an approach to construction that is hands-on, negotiational, exploratory, interactive and experimental. In domains where knowledge is provisional, a bricolage approach allows learners to construct physical models concurrently with their emerging understanding

of that domain. In this thesis, we explore how EM can be used as a computer-based modelling approach that embodies bricolage and consider the resulting benefits from a learning perspective.

- **Situated learning**. In [Lav88], Lave emphasises the need for learning to be situated in realistic situations, especially when the subject matter concerns human interaction or is hard to grasp in abstract terms. In this thesis, we explore how EM can act as a situated modelling approach (cf. Suchman [Suc87], Goguen [Gog96]) that allows personal viewpoints and conflicting interests to be represented.

Constructionism, originally introduced by Papert, was primarily concerned with children learning through writing computer programs. End-user programming is more generally concerned with domain learning through the construction of computer-based models. In respect of both these agendas, Brödner draws attention to the significance of practical experience explored through open development as a complement to propositional knowledge of the closed world. Gooding considers the role that the construction of artefacts can play in embodying our understanding. Brödner and Gooding's views endorse our use of artefacts in support of experiential learning activities. Levi-Strauss and Lave describe the key qualities that modelling approaches require to support the open-development approach to constructing artefacts.

Previous research has considered the relationship between EM and conventional methods in different application domains. Two common themes in this comparative research are: the emphasis placed on the human element in the modelling process; and the distinctive characteristics of the EM approach when compared to conventional programming and software development. Generally, these advantages are concerned with the knowledge that can be gained more easily using EM than with conventional methods in the particular application domain. However, none of them has considered in detail why this should be possible from a learning perspective.

Broadly speaking research has been directed at two main areas: software system development and business applications.

With reference to software system development, Paul Ness [Nes97], Patrick Sun [Sun99] and Allan Wong [Won03] have all compared development in EM with conventional software system development. Ness was concerned with identifying how computer technology could offer support for the construction of *creative artefacts*, which promote exploratory representation of unfamiliar subjects, rather than *analytical artefacts*, which promote methodological representation of familiar subjects. He argued that conventional software development focuses on analytical artefacts, whereas EM focuses on creative artefacts. Ness argued that the support for creative model building in EM stems from properties of creative artefacts that are not to be found in analytical artefacts. These properties, namely novelty, ambiguity, implicit meaningfulness, emergence, incongruity and divergence were identified in the work of Finke [FWS92]. These properties are ideally suited for supporting exploratory model building for learning, where there are misunderstandings, inconsistencies and digressions. Sun was concerned with identifying how computer technology could support distributed modelling. He argued the case for EM as an amethodical approach to software development for distributed applications in which knowledge was 'cultivated' through situated modelling. Situated modelling for learning is a prominent theme in this thesis. Wong identified EM as supplying 'a suitable setting for both the cognitive and collaborative aspects of system development in which the emphasis is on heuristic human problem solving and maintaining conceptual integrity in a system design' [Won03]. These characteristics of EM are significant in respect of learning because they relate to the negotiation of meaning in both the private and the public domains. Wong also considered how EM can be used to construct environments in which the user takes responsibility for the circumscription and customisation of a system. This vision is well matched to the needs of a teacher who needs to develop and customise resources to suit their educational context.

In terms of applications to business, Suwanna Rasmequan [Ras01], Soha Maad [Maa02] and Yih-Chang Chen [Che01] have compared an EM approach with currently used systems in various business areas. Rasmequan was concerned with the integration of human cognitive processes and computing processes in business software development. She argued that EM supports cognitive processes by promoting rich representations of situations that offer direct experience, encourage active engagement and spontaneous involvement. Maad was concerned with identifying a framework within which to conduct software system development in the domain of finance. She argued that EM offers support for an alternative culture in finance through an approach to software development that integrates experiential and situated aspects of finance together with close human involvement. Chen was concerned with how EM could be viewed as an approach to Business Process Re-engineering (BPR) where businesses revise their practices to adapt to new computer technology. He argued that the failures of BPR systems were attributable to the inability to preconceive or predict all the causalities when modelling real-world situations with high levels of human involvement. He argued that these problems could be alleviated by using EM for requirements engineering to gather knowledge prior to the construction of a business software system. All three of these authors stress the role that EM can play in acquiring and applying domain understanding throughout the development of a business system. This suggests that EM is well suited to active knowledge construction in many different domains.

This thesis establishes a connection between EM and learning that accounts for its fitness for active knowledge construction, and that can also offer a unifying perspective on the previous work described above. The potential of EM in relation to learning was first outlined by Beynon in his 1997 paper entitled ‘Empirical Modelling for Educational Technology’ [Bey97]. In the paper, Beynon outlined issues for technology in education from the perspectives of IT management, teachers and pupils. The relationship between EM and learning is at the core of the EM for the educational technology agenda. Beynon framed his discussion with reference to a perspective on learning that he termed the ‘Empiricist Perspective on Learning’. This perspective highlights the role that the experiential activities that inform pre-articulate understanding play in learning.

Beynon's paper did not discuss the Empiricist Perspective on Learning with reference to any received learning theories such as constructionism and situated learning. In 1999, I undertook a taught Masters project investigating the potential of EM for educational technology [Roe99]. This preliminary study comprised a limited literature review that concentrated on the work of Seymour Papert [Pap80, Pap93] on LOGO and the theory of constructionism. The limited scope of my study did not enable a full investigation of the potential of EM with respect to educational technology. For instance, it made no reference to Beynon's Empiricist Perspective on Learning or to other learning theories.

The past work on EM and educational technology, my MSc thesis, and the ideas of other authors outlined in this section have together motivated the research in this thesis.

1.5 Research Contributions

The major contention of this thesis is that computer-based model building, as generally practised, does not give adequate support for the experiential aspects of pre-articulate learning. We believe that the principles and practice of Empirical Modelling offer fuller support for pre-articulate learning in respect of both model building and model use. With regard to the construction of models, this thesis develops the 'Empiricist Perspective on Learning' introduced by Beynon in 1997 [Bey97]. Beynon's perspective has been revised in the light of the research in this thesis, and is now referred to as an 'Experiential Framework for Learning' (EFL). This thesis contains the first comprehensive account of the relationships between EM, the EFL and the use of computers for learning. This research has involved a careful investigation, synthesis and analysis of ideas from many learning theories. Relating learning theories and EM has required studying the literature from both fields to identify common ground. With regard to the use of EM models in an educational context, this thesis also gives a full account of principles and techniques that allow the creation of flexible and extensible learning environments.

In the course of this research, I have been the primary author of two papers [RB02, RBF00], and contributed to several other publications [BCH$^+$01, BRW$^+$01, EJR$^+$01, RRR00, BBC$^+$01] that relate to the ideas presented in this thesis. A central claim of the thesis – that EM better supports pre-articulate learning and its integration with formal learning – is discussed in [RB02] and developed in chapter 4. The digital watch artefact described in section 4.7 of the thesis was discussed in [RBF00] in connection with engineering education. The model has also been used as a case study for investigating cognitive aspects of user-artefact interaction [BRW$^+$01] and was also demonstrated at a workshop on cognitive dimensions (see e.g. [Gre89]) to which I contributed (see [BBC$^+$01]). The extent to which EM allows the computer to be used as an instrument rather than a tool is discussed in [BCH$^+$01]. The restaurant case study, discussed in chapter 3, featured as an illustrative example in a paper on Strategic Decision Support Systems [RRR00] and was also adopted as a case study by Rasmequan [Ras02]. The application of EM principles to simulate LEGO Mindstorms robots (developed in conjunction with the researchers at a children's technology club in Finland) was discussed in [EJR$^+$01]. This is described in section 6.4 of the thesis.

For this thesis, I have also constructed several example models of differing levels of sophistication. These include:

i) the spreadsheet model [EMRep, spreadsheetRoe2002], discussed in section 2.5.

ii) the restaurant manager model [EMRep, restaurantRoe2000], discussed in section 3.5.

iii) the digital watch model [EMRep, digitalwatchRoe2001], discussed in section 4.7.

iv) the variations on the OXO model, discussed in section 5.3.3.

v) the clown-and-maze model [EMRep, krustyRoe2002], discussed in section 5.4.2.

vi) the relational algebra tutor [EMRep, ratRoe2003], discussed in section 5.4.3.

vii) the robotic simulation environment [EMRep, rseRoe2003], discussed in section 6.4.

1.6 Contents of the thesis

This thesis is organised into seven chapters, of which this is the introductory one.

In Chapter 2, we consider the potential of exploratory modelling for learning. We first consider the spreadsheet as a tool for exploratory modelling. Three considerations motivate our choice of the spreadsheet as a starting point:

- it is a popular programming paradigm for the end-user.
- it is widely used in education for creating models and exploring phenomena through the use of ‘what-if?’ style queries.
- it is a ubiquitous application.

We use the spreadsheet concept to draw out two key aspects of exploratory modelling: namely, the *negotiation* and *elaboration* of the relationship between a computer model and its referent (designated as ‘the semantic relation β’ by Cantwell-Smith [Smi97]). We will argue that spreadsheets are well suited to negotiation but are limited in respect of elaboration. We identify research products based on spreadsheet ideas that have attempted to overcome some of the limitations of spreadsheets and consider their qualities in respect of negotiation and elaboration. In the final part of the chapter, we introduce practical Empirical Modelling and conclude that it offers support for negotiating and elaborating the semantic relation.

In Chapter 3, we outline the major challenges for the use of computers for learning. These challenges are concerned with bridging the gap between how the computer scientist and the educationalist view the use of computers for learning. A computer scientist is primarily interested in issues of usability, requirements specification and the choice of programming paradigm. The educationalist focuses on the qualities of the learning that is taking place, the actual computer implementation being of only secondary concern. Marrying these two viewpoints requires an approach to computer-based model construction which is such that:

- there is a close connection between domain learning and model construction.
- educational software that is developed can be easily and flexibly adapted in response to different learning situations and competencies.

With reference to many examples of learning activities, we describe a view of learning that presumes that knowledge is rooted in our personal experience. These examples motivate our 'Experiential Framework for Learning' (EFL). The EFL comprises many learning activities ranging from the private to the public, from the empirical to the theoretical, and from the concrete to the abstract. We introduce the key principles of EM: the development of construals; the primary emphasis on representing state-as-experienced; and the concepts of observable, dependency and agency. We argue that EM model construction that is based on these principles respects the relationships between activities in the EFL, so that EM can support the integration of the empirical and the theoretical within a single modelling environment. We illustrate these ideas with reference to the construction of a restaurant manager model.

In Chapter 4, we consider computers for learning from an educational perspective. We begin by discussing the educational theories of instructionism and constructionism. We adopt a broad perspective on constructionism that encompasses both bricolage and situated learning. Bricolage is a style of construction that puts emphasis on close personal engagement with a task where the evolving construction goes hand in hand with increasing comprehension of the task. Situated learning holds that the surrounding domain context of a problem is not incidental to its solution but provides the necessary social handles for learners to grapple successfully with the problem. We argue that principles and techniques for computer-based model construction must support bricolage and situated learning if they are not to inhibit domain learning. In the remainder of the chapter, we consider three practical techniques that can be applied in domain learning: concept mapping, conventional programming and EM. We discuss the extent to which each of these techniques supports the broad perspective on constructionism and how it is related to the EFL. We argue that concept mapping is only useful in the very early stages of learning and that knowledge gained using it is typically set aside when constructing models. We argue that conventional programming is not well oriented to the constructionist agenda because it emphasises planning, abstraction and circumscription and this detracts from its usefulness as an approach to model construction that promotes domain learning. Finally, we argue that an EM approach to model construction can support our broad perspective on constructionism and learning activities across the EFL, enabling effective domain learning to proceed in tandem with model construction. We illustrate this claim with reference to the construction of a digital watch model.

In Chapter 5, we consider the advantages of using EM to construct learning environments that support many different types of learning objective. We identify three types of learning that can be scaffolded in EM: comprehension of a fixed referent; exploration of possibilities and invention; and learning languages. We illustrate each of these types of learning by case studies in the form of EM models in which learning is scaffolded through gradual embellishment of the model. A different style of presentation to the learner is characteristic of each type of learning. In the Racing Cars model, the referent is fixed from the outset and each layer of the model adds a greater subset of the functionality to the interface so that the learner can explore more complex ideas. In the OXO case study, the model is built up incrementally, and – although a specific learning path is mapped out – the model is flexibly adaptable to different teaching requirements. We illustrate this adaptability by creating a family of games related to noughts-and-crosses. In respect of learning languages, we introduce an EM parsing utility that can allow languages to be incrementally extended or refined as a learner is interacting with a model. We use case studies based on a simple LOGO-like language and a more complex database query language to illustrate language learning in conjunction with emerging domain understanding.

In Chapter 6, we discuss and illustrate the links between EM and the EFL with reference to more elaborate case studies. Conventionally, there are two ways in which computers support learning: through personal model building and through the use of pre-constructed models that cannot be revised by the user. We argue that it is possible to have a third category, namely models that are partially built and that can be extended by a learner in response to their particular learning needs. We present three EM case studies that exhibit different degrees of model building and model use: the Free Distributive Lattice model; the Heapsort model; and the Robotic Simulation Environment. Each model places different demands on the learner and this is reflected in the specific learning activities that it supports within the EFL. These case studies give practical evidence in justification of our claim that EM can support learning activities from across the whole of the EFL.

Chapter 7 summarises the research undertaken for the thesis, drawing some conclusions, considering its limitations and outlining possible future work.

Chapter 2 – Paradigms for exploratory modelling

2.0 Overview of the chapter

In this chapter, we consider computer-based support for exploratory modelling. We firstly discuss modelling with spreadsheets and identify two key aspects of exploratory modelling with reference to a specific example. These two key aspects are negotiating and elaborating the semantic relation between the model and its referent. We argue that spreadsheets are suitable for negotiating the semantic relation in certain domains but have limitations in respect of elaborating the semantic relation. We discuss research products that extend the spreadsheet concept and the implications for their support of the key aspects of exploratory modelling. We consider how practical Empirical Modelling supports the key aspects of exploratory modelling and conclude that it offers better support than spreadsheets and Agentsheets for negotiating and elaborating the semantic relation.

2.1 Empirical Modelling and spreadsheets: In principle and practice

In this section, we consider the relationship between spreadsheets and EM. We explore this relationship through the examination of practical spreadsheet applications (e.g. Microsoft Excel™) and the academic literature in the field of spreadsheets [Lew90, WL90, Nar93, Lev94, VK96, CRB$^+$98, BAD$^+$01, Gro02]. It is apparent that spreadsheet construction is markedly different from conventional program construction, although both aspire to allow users to exploit computational power in problem solving. Unlike programming, constructing spreadsheets has become a common skill that can be used in education for a variety of purposes [New01, UNC03]. For instance spreadsheets can be used for data capture, exploratory modelling and graph plotting. This suggests that there are aspects of spreadsheet use that are particularly significant from a learning perspective.

Previous research in Empirical Modelling (see e.g. [Bey87a, Geh96, RRB00]) has identified Empirical Modelling as based on a “radical generalisation of spreadsheets” (cf. [RRB00]). In this thesis we are led to look more critically at this informal claim, and conclude that EM generalises those features of spreadsheet use that are intimately connected with learning. There are two complementary relationships to be understood: the relationship between Empirical Modelling principles and principles of spreadsheet use; and the relationship between Empirical Modelling tools and practical spreadsheet applications. This chapter is organised around these two comparisons.

The structure of the chapter is as follows: firstly in section 2.2 we identify the features of potential spreadsheet use that are particularly significant in a learning context. In section 2.3 we outline research that has extended the spreadsheet idea and its impact on potential applications of spreadsheet principles. Section 2.4 introduces Empirical Modelling from a practical perspective, describes the TkEden modelling tool, and (2.4.6) explores the relationship between Empirical Modelling principles and principles of spreadsheet use. In section 2.5, we demonstrate the relationship between Empirical Modelling tools and spreadsheets by building a spreadsheet program using the Empirical Modelling tool TkEden (introduced in section 2.4.2). This provides further practical evidence that Empirical Modelling offers better support for exploiting spreadsheet principles in learning since the TkEden spreadsheet allows models to be constructed that would be hard to replicate in a conventional spreadsheet program. Figure 2.1 depicts the relationship between the subsections of this chapter.

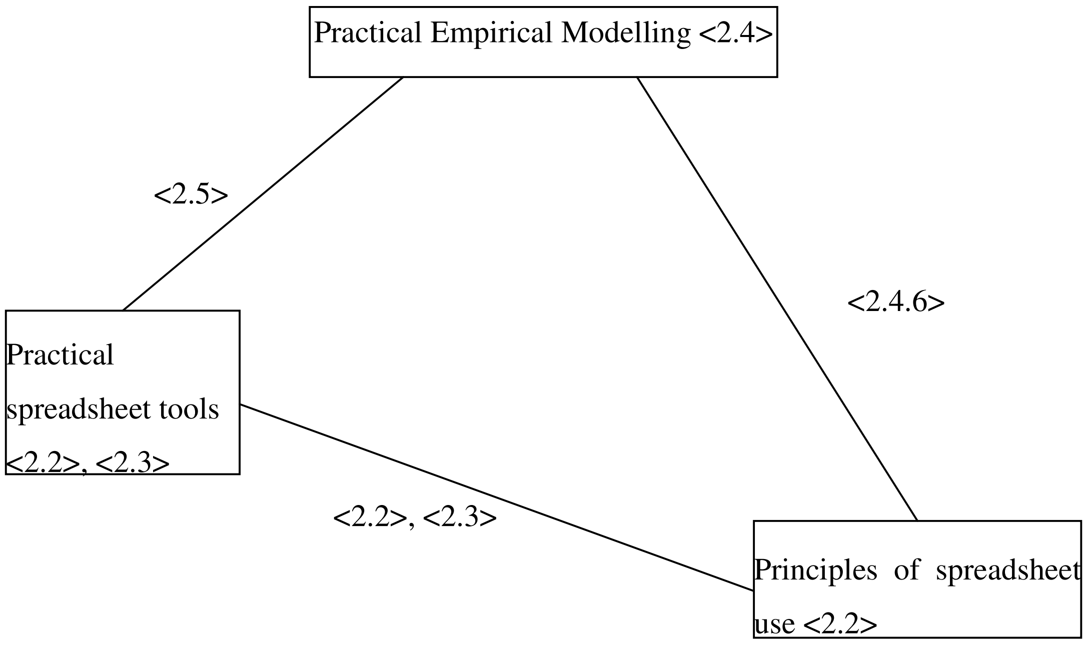

Practical Empirical Modelling <2.4>
<2.5>
<2.4.6>
Practical spreadsheet tools <2.2>, <2.3>
<2.2>, <2.3>
Principles of spreadsheet use <2.2>

Figure 2.1 – Connecting Empirical Modelling, practical spreadsheet tools and principles of spreadsheet use

2.2 Spreadsheets

The principal purpose of this section is to identify principles of spreadsheet use that are significant in a learning context. To this end, we firstly review the essential characteristics of a spreadsheet as commonly identified. This review serves two purposes: it highlights the distinction between the routine use of spreadsheets and their applications in exploratory and creative model construction; it also supplies a convenient base from which to introduce EM.

2.2.1 Introducing spreadsheets

Since the introduction of the spreadsheet into the commercial software world with VisiCalc in 1979 the spreadsheet has been one of the most widely used application packages on computers. Indeed it has been described as the ‘killer app’ that helped launch the personal computer market [CK95]. The spreadsheet has remained at the forefront of the application market and is used by millions of people every day, from home users to large corporations [MKT93]. Spreadsheets allow users to build their own programs, and have developed into the most popular 'end-user' programming paradigm [Nar93]. Users construct their own spreadsheets to perform tasks that they would find impossible using general purpose programming languages such as C or Java.

The spreadsheet features reviewed in this section (2.2.1) are typical of Microsoft Excel, the most widely used spreadsheet on the market today [Lan03]. A spreadsheet is a rectangular arrangement of cells, organised into a collection of columns, (usually identified by letters) and rows (usually identified by numbers). Each cell therefore has a unique reference (e.g. A3, B12) that is identified by the column and row headers. Each cell can contain one of a number of different elements. The basic type that a cell can have is a *value*. This can be either numeric (e.g. 12, 3.14) or textual (e.g. "VAT Rate"). There are a limited range of other types that are supported, including times and dates. Value cells can be combined through cells that contain *formulas.* A formula is a function composed of *operators* and values in the spreadsheet. Operators can be applied to individual cells or to groups of cells, such as columns, rows or two-dimensional regions. Examples of formulae are '`=MAX(B1,B2);`' and '`IF (B12=0), 100, 0;`'. Spreadsheet applications usually provide a large number of built-in functions that users can deploy in their spreadsheets. These functions cover a wide range of domains. In general, a spreadsheet will provide mathematical operators (e.g. sum, max, min, +, -, *, /), statistical operators (e.g. variance), financial operators (e.g. term, rate), time-based operators (e.g. month, year), logical operators (e.g. and, or, not) and textual operators (e.g. substr, findstr). Sophisticated calculations can be achieved using multiple cells or ranges of cells in multi-stage computational processes.

The essential feature of spreadsheet calculations is that when a value is changed, any formula that references that value, directly or indirectly, is automatically recomputed.

In this way, a single alteration to a spreadsheet can lead to widespread change. The major restriction for formulae is that there can be no cyclic dependencies. In the example below (Figure 2.2), the arrows show how the update of one cell propagates to the other cells, leading to an infinite cycle. The following cell definitions would not be permitted:

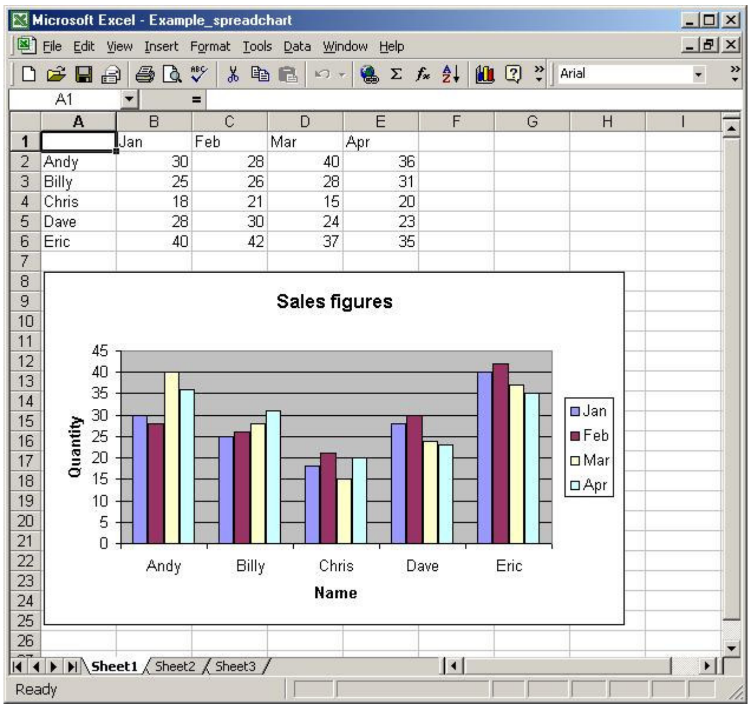
```
A1 = A3+3
A2 = A1+1
A3 = A2*4
```

$$\begin{tikzcd}
 & \mathtt{A1} \arrow[dr] & \\
\mathtt{A3} \arrow[ur] & & \mathtt{A2} \arrow[ll]
\end{tikzcd}$$

Figure 2.2 – Cyclic dependency.

Spreadsheets allow users to create charts based upon data in the spreadsheet. A wide variety of chart types are usually supported, including pie charts, bar graphs, line graphs, and scatter plots. Charts are dependent on the spreadsheet and an update to a relevant part of the spreadsheet will cause an update to the graph. Charts overlay a region of cells in the spreadsheet (as shown in Figure 2.3).

Formulae are the main way to describe relationships between cells. To provide the spreadsheet user with more control, procedural add-on languages are often supplied. Procedures can be written in a high-level language to create effects that would be impossible with the sole use of formulas. In Excel, this procedural language is Visual Basic for Applications (VBA). One common use of VBA is to provide front-end interfaces to spreadsheets so that users can click on buttons in the spreadsheet to perform actions.


Microsoft Excel - Example_spreadchart
File Edit View Insert Format Tools Data Window Help
Arial
A1
Sheet1
Sheet2
Sheet3
Ready
Sales figures
Quantity
Name
Jan
Feb
Mar
Apr

	A	B	C	D	E
1		Jan	Feb	Mar	Apr
2	Andy	30	28	40	36
3	Billy	25	26	28	31
4	Chris	18	21	15	20
5	Dave	28	30	24	23
6	Eric	40	42	37	35

Figure 2.3 – An example spreadsheet and chart in Excel.

Following [CRB+98], we can summarise the characteristic features of a conventional spreadsheet as follows:

F 1: There is an automatic mechanism in charge of **dependency maintenance**. Cells are automatically updated whenever one of the values or formulas that affects it has been changed.

F 2: **Operators** are used to construct relationships between cells through the definition of formulas. Examples of operators include arithmetical and statistical operators.

F 3: There is a **tabular layout**. Cells are organised into a two-dimensional grid. Users can exploit this regular structure to simplify their computations.

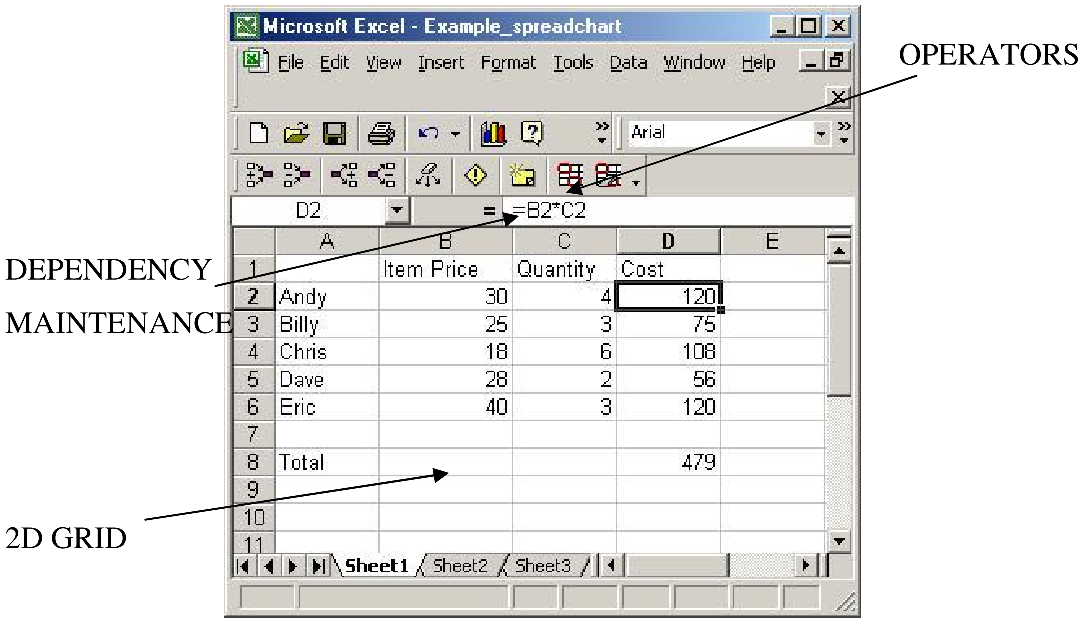
The three features, F1, F2 and F3 are shown in Figure 2.4. A further feature was originally identified by Allan Kay in 1984 [Kay84]. This is the **value rule**:

F 4': A cell's value is defined solely by the formula explicitly given to it by the user.

Burnett et al [BAD+01] interpret the value rule as '[disallowing] devices such as multi-way constraints, state modification, or other non-applicative mechanisms'. In this thesis, we prefer to work with a more relaxed version of the value rule, which admits the possibility of procedural extensions provided that they respect the relationship between the value of a cell and its defining formula. By this criterion, the automatic updating of the value of an explicitly defined spreadsheet cell would not violate the value rule, neither would the automatic assignment of a new formula to a cell provided that this was indivisibly associated with its re-evaluation. This relaxed version of the value rule can be defined as follows:

F 4: A cell's value is always consistent with the formula currently assigned to it from the perspective of the spreadsheet user.

This legitimises the principled use of procedural extensions to the spreadsheet (e.g. through the use of VBA).


Microsoft Excel - Example_spreadchart
File Edit View Insert Format Tools Data Window Help
Arial
OPERATORS
DEPENDENCY
MAINTENANCE
2D GRID
Sheet1
Sheet2
Sheet3

D2 = =B2*C2

	A	B	C	D	E
1		Item Price	Quantity	Cost	
2	Andy	30	4	120	
3	Billy	25	3	75	
4	Chris	18	6	108	
5	Dave	28	2	56	
6	Eric	40	3	120	
7					
8	Total			479	
9					
10					
11					

Figure 2.4 – A diagram of an Excel spreadsheet with the key characteristics of the paradigm being highlighted

As illustrated in Figure 2.4, conventional numerical spreadsheets exhibit the four features F1, F2, F3 and F4.

2.2.2 Key aspects of exploratory modelling

Spreadsheets are considered to be useful tools for learning through exploratory modelling [Nar93, New01, Gro02]. We aim to articulate some general principles as to why spreadsheets are suitable for exploratory modelling. To motivate this discussion we give a small example of a learning situation in which spreadsheets can be beneficially used and then abstract from this discussion some key aspects of exploratory modelling.

Brian Cantwell Smith [Smi97] identifies three aspects of a computer system: the *program*, the *process* and the *subject matter* (see Figure 2.5). The program is the source code, the process is the behaviour associated with the executing program, and the subject matter is the task domain to which the system refers. Conventionally, computer science is primarily concerned with understanding the relationship α between the program and the executable process.

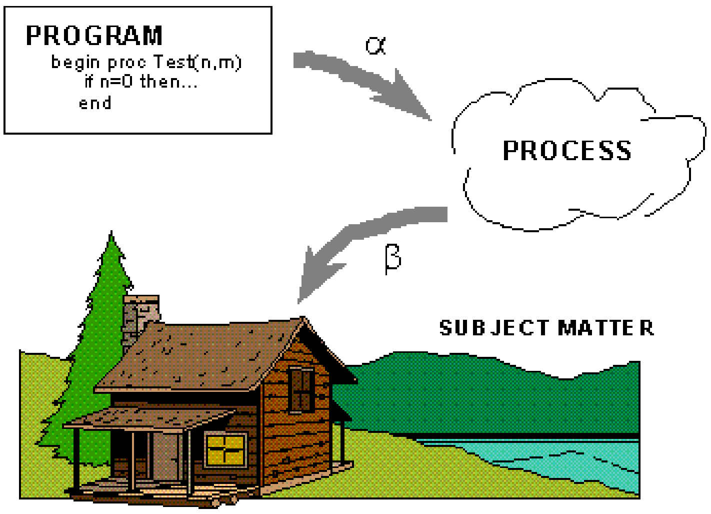
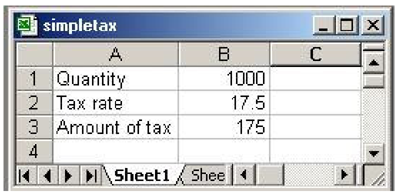


Figure 2.5 - Cantwell-Smith's program, process and subject matter [Smi97]

The relationship β is the semantic relation between the computer model and its real-world referent. In exploratory modelling, this is the important relation: understanding how to correlate the computer model with its subject matter. For instance, in spreadsheets the learner is not concerned with the relation between program and process because – through dependency maintenance – the spreadsheet abstracts away the details of the α relation (cf. [Nar93]). The question is then how does the spreadsheet support the β relation? We can demonstrate some of the activities that are involved by considering an example of learning about tax. This is in a similar spirit to Noss and Hoyles's investigations into helping bankers to explore the financial mathematics underlying the tools they had been using without full understanding [NH96].

Taxation is a concept that can be explored through the construction of small example spreadsheets. A spreadsheet can be easily constructed to investigate how a basic tax is calculated. We simply set up three cells A1, A2, A3 that respectively represent a taxable monetary quantity, a fixed rate of tax, and the tax payable:

```
B1 = <Quantity>
B2 = <Tax rate>
B3 = B1 * (A2/100)
```



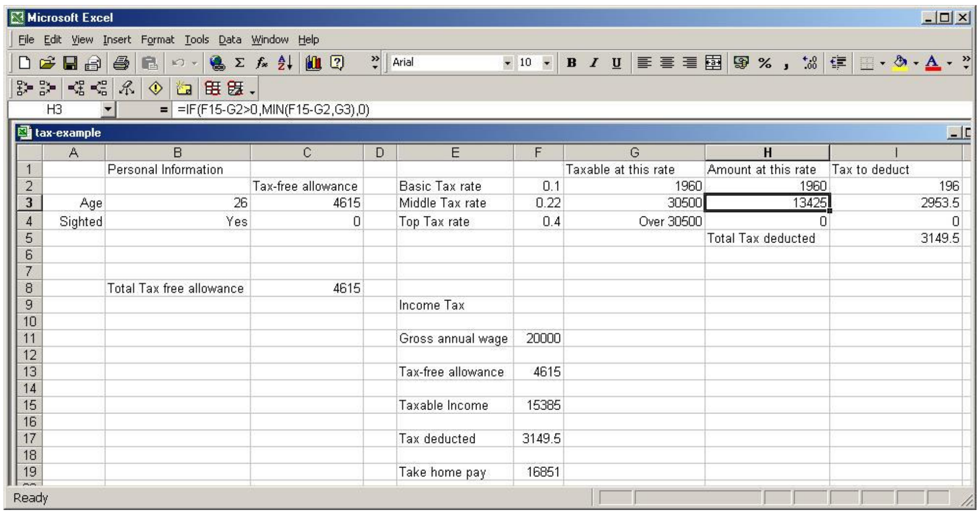
simpletax

	A	B	C
1	Quantity	1000	
2	Tax rate	17.5	
3	Amount of tax	175	
4			

Figure 2.6 - A simple tax spreadsheet

This simple spreadsheet is sufficient for comprehending a basic tax such as Value Added Tax.

Exploratory modelling can give a fuller understanding of how to calculate the price before tax was added or the effect of the fixed percentage on the total price. Embellishments can take the model into more sophisticated taxes such as income tax where tax rates are dependent on the amount of money earned. The original spreadsheet can be extended or refined to explore many different types of taxation.



H3 = =IF(F15-G2>0,MIN(F15-G2,G3),0)

tax-example

	A	B	C	D	E	F	G	H	I
1		Personal Information					Taxable at this rate	Amount at this rate	Tax to deduct
2			Tax-free allowance		Basic Tax rate	0.1	1960	1960	196
3	Age	26	4615		Middle Tax rate	0.22	30500	13425	2953.5
4	Sighted	Yes	0		Top Tax rate	0.4	Over 30500	0	0
5								Total Tax deducted	3149.5
6									
7									
8		Total Tax free allowance	4615						
9					Income Tax				
10									
11					Gross annual wage	20000			
12									
13					Tax-free allowance	4615			
14									
15					Taxable Income	15385			
16									
17					Tax deducted	3149.5			
18									
19					Take home pay	16851			

Figure 2.7 - A spreadsheet to explore income tax

There are two aspects to the exploratory understanding of taxation. The first aspect is concerned with 'knowing that the formulae correctly characterise income tax'. The spreadsheet supports several relevant types of learning activity. For instance, the tax model can be embellished by adding new data to the spreadsheet or refined by experimenting with its dependencies. This allows us to correlate the experience gained from the spreadsheet with our prior experience of the concept of tax. In the tax example, the regular grid structure is advantageous because it makes it so convenient to make appropriate changes to the model interactively. The types of activities that are important here are probing our current understanding and experimenting to further our understanding. This probing and experimenting is valuable from a learning perspective not only because it helps us to appreciate the implications of established theory and rules but also because it enables us to deal with pre-theory situations where the right answer (if one exists) is unknown and can emerge through having the freedom to experiment and the license to make mistakes.

The second aspect is concerned with 'exploring the broad personal and social implications of income tax'. The spreadsheet supports this aspect of exploratory modelling by enabling us to conveniently search and explore possible solutions to problems, to survey entire state spaces and to generate relevant patterns of behaviour.

For instance, we can use the spreadsheet to find out how to minimise tax or explore the consequences of life-changes. We can also use it to survey and present the implications of tax regimes on different social groups. For instance, the spreadsheet could be set up to compute the tax for a particular group (e.g. characterised by income, age, area of residence), under a particular tax regime (e.g. as determined by income tax, petrol tax, cigarette tax). It can also be used to explore the effects of varying taxation levels and to predict future strategies based on current data. In exploration of this nature, we exploit the facility offered by the spreadsheet to link data from diverse real-world domains through dependency. For instance, in calculating the tax payable on a project, a company might link a spreadsheet associated with the specific project to a general spreadsheet embodying tax regulations.

Both aspects of exploratory modelling discussed above relate to understanding Brian Cantwell-Smith's semantic relation β. The first aspect is concerned with the essential nature of the association between process and subject matter. For instance, in understanding income tax it is important to appreciate the different roles played by capital savings and interest on savings. The second aspect is concerned with experiencing the implications of this relation in its domain context to its fullest possible extent. The breadth of this activity is reflected in the myriad ways in which spreadsheets are used both personally and by businesses to explore 'what-if?' scenarios.

In the above discussion, we have identified two key aspects of exploratory modelling in relation to understanding a concept X:

A1 - negotiation of the semantic relation β. This involves satisfying ourselves that we understand the essential nature of the concept X (as apprehended through the relation between the model and the subject matter).

A2 - elaboration of the semantic relation β in its domain context. This involves satisfying ourselves that our understanding of concept X is consistent with the ways in which it can be applied in a domain context.

These two aspects are intrinsically intertwined. Neither A1 nor A2 can be carried out to completion - our understanding of a concept is always potentially open to future revision in the light of new insights and discoveries. In the negotiation of the semantic relation, elaboration has an essential role to play in confirming that our understanding is coherent. In the elaboration of the semantic relation, it may be necessary to renegotiate the semantic relation.

2.2.3 Spreadsheets for exploratory modelling

The above discussion has identified the qualities of spreadsheets in supporting the key aspects of exploratory modelling. They stem from three features:

- being able to record dependencies.
- making it convenient to explore state and generate behaviours that are meaningful to the modeller.
- being able to extend models easily through dependency.

The merits of spreadsheets for exploratory modelling and learning are endorsed by Grossman in his discussion of 'spreadsheet engineering' [Gro02]:

> 'When performing exploratory modeling in a spreadsheet, the spreadsheet serves as a modeling tool to structure, explore, and understand a problem; it becomes a means for expressing one's ideas'.
>
> 'During the modeling process, exploratory modelers learn much and benefit greatly. When they are done with their exploratory modeling, they find themselves in possession of an *artifact*: a spreadsheet. This spreadsheet artifact is the residue of their inchoate modeling process. This spreadsheet artifact is intimately connected to the powerful learning the user acquired during its creation'.

The spreadsheet is ideally suited to exploratory understanding of taxation because the data is numerical, can be suitably formatted into the spreadsheet grid and there are numerical relationships between the various components. In more general applications, spreadsheets have limitations that do not allow them to fully support A1 and A2. In respect of A1, the limited number of types and reliance on the grid leads to problems. For instance, imagine trying to construct a spreadsheet to investigate the concept of a vehicle cruise controller. In this situation, we would ideally require an exploratory construction tool that could support a wide range of graphical types and a display that did not impose a grid organisation upon values and dependencies. Furthermore, a spreadsheet obliges the modeller to display the entire state of the model in the grid interface. This is inappropriate for a model of such complexity as the vehicle cruise controller. Complexity in spreadsheets can lead to errors in construction and comprehension (see e.g. Panko's discussion of the prevalence of spreadsheet errors [Pan00]).

In respect of A2, the spreadsheet allows 'what-if?' style modelling by being able to set up templates of dependencies and change specific parameters. However, in a spreadsheet this is still constrained to a limited range of applications by the grid. Spreadsheets also give limited conceptual support for using procedural actions in combination with dependency (cf. the discussion of the relaxed version of the value rule F4 in section 2.2.1). Though procedural extensions to spreadsheets (such as VBA with Excel) in principle offer arbitrary computational power, it is hard in practice to integrate this within a spreadsheet without comprising intelligibility (cf. the discussion of the value rule in section 2.2.1). It is well recognised that the state-transitions in an application such as a vehicle cruise controller are derived from very diverse and subtle stimulus-response patterns [Deu88, Deu89]. For instance, the transitions to be modelled stem from time-based dynamic behaviour, human interventions, automatic responses by components and the influence of environmental factors. Though it may be possible to model such activity in a spreadsheet with procedural extensions there is very limited conceptual support for dealing with the complex issues of interaction and synchronisation that arise.

In this section, we have identified key aspects of exploratory modelling and shown that spreadsheets can only support these aspects to a limited extent. In the following section, we discuss research inspired by the spreadsheet principle and consider the impact for supporting exploratory modelling.

2.3 Extensions of the spreadsheet concept

This section describes existing research influenced by spreadsheet principles. Each system to be described exhibits some of the characteristic features of spreadsheets. The first feature, namely dependency maintenance, is the fundamental distinctive idea of the spreadsheet. To support the construction of dependencies between variables, formulae use operators to construct relationships between variables. Some systems have a broader range of operators than can be found in a conventional spreadsheet. These two features (F1 and F2) are crucial to the spreadsheet approach. In this section, we describe research that:

i) relaxes the need for a grid (F3). This is exemplified in the Forms/3 system.
ii) relaxes the value rule (F4). This is exemplified in the Spreadsheets for Images system.
iii) takes the form of a programming system based on generalising spreadsheet principles using agents. This is exemplified in the Agentsheets programming environment.

2.3.1 Forms/3

Forms/3 is a research-oriented declarative visual language that has been developed at Oregon State University. The Forms/3 language exhibits three of the four features (F1, F2, F4) of a spreadsheet because it does not require the use of a grid. The aim of the Forms/3 system was to provide end-users with powerful programming capabilities, and to equip professional programmers with tools that are as easy to use as a spreadsheet [BAD$^+$01].

The basic building block of a Forms/3 'program' is a collection of cells and formulae similar to that underlying a spreadsheet. Users are free to place cells wherever they like on a form (the basic structure) and give each cell a name (since they are not identified by grid position). Cells can be connected through formulas given by the user. These formulae are not restricted to the simple types permitted in conventional spreadsheets. Formulae can include graphical types and complex conditional dependencies, as shown in the following table adapted from Burnett et al [BAD+01].

Type	Format	Examples and Explanation
Algebraic Expressions	Integers and floating point numbers	Operators such as +, -, *, /. Example: A2*(100-53)
Logical Expressions	not (A and B). A and B can be numbers, cell references, Boolean expressions.	Example: not (A2=100 and B3>10)
Box	box Width Height	Draws a box of given dimensions, which can be references to other cells, or the results of expressions. Example: box 20 A3
Graphics	glyph filename	Loads an external graphics file into a cell. Example:glyph "disc.bmp"
Compose	compose X at (x1,y1) with Y at (x2,y2)	Composes graphical objects together. Example: compose (box 10,10) at (50,50) with (circle 8) at (25,25)
if/then/else	if condition then E1 else E2	if (B2<0) then 0 else B2

Table 2.1 – Some example Forms/3 commands, from Appendix B of [BAD+01].

A form is the basic structure in Forms/3, akin to a module or subprogram in a conventional programming language. It can consist of many kinds of elements, such as grids of cells, single free-floating cells, or a definition of a type, such as a circle. Forms allow abstract patterns of cells and formulae ('abstractions') to be collected together so that they can be reused. The example below (Figure 2.8) shows a form to define a circle with attributes defined in the cells.

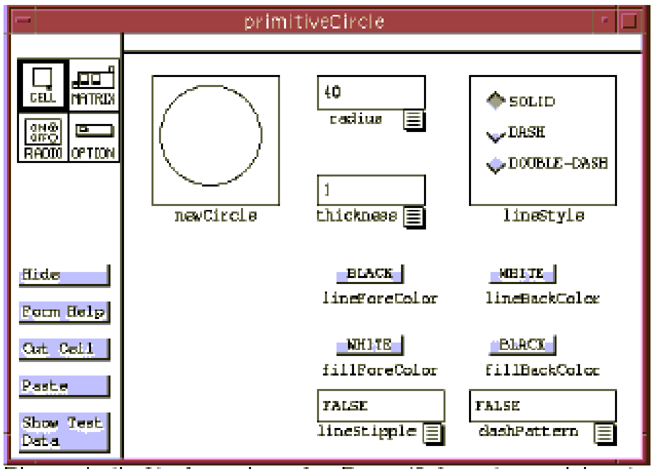


Figure 2.8 – Example of a Forms/3 form with a set of cells to define a circle and its attributes.

The Forms/3 system has advantages over spreadsheets in respect of the number of available types. For example, dependencies in Forms/3 can encompass graphical types and exploit data abstractions. Forms/3 allows the animation of values where dependencies are changing over time. It achieves this through an in-built model of time, where a cell's value changes over time. Forms/3 combines the ease of use of the spreadsheet with some of the powerful aspects of conventional programming languages. Removing the necessity for a grid allows programmers to only use a grid in situations where it is appropriate.

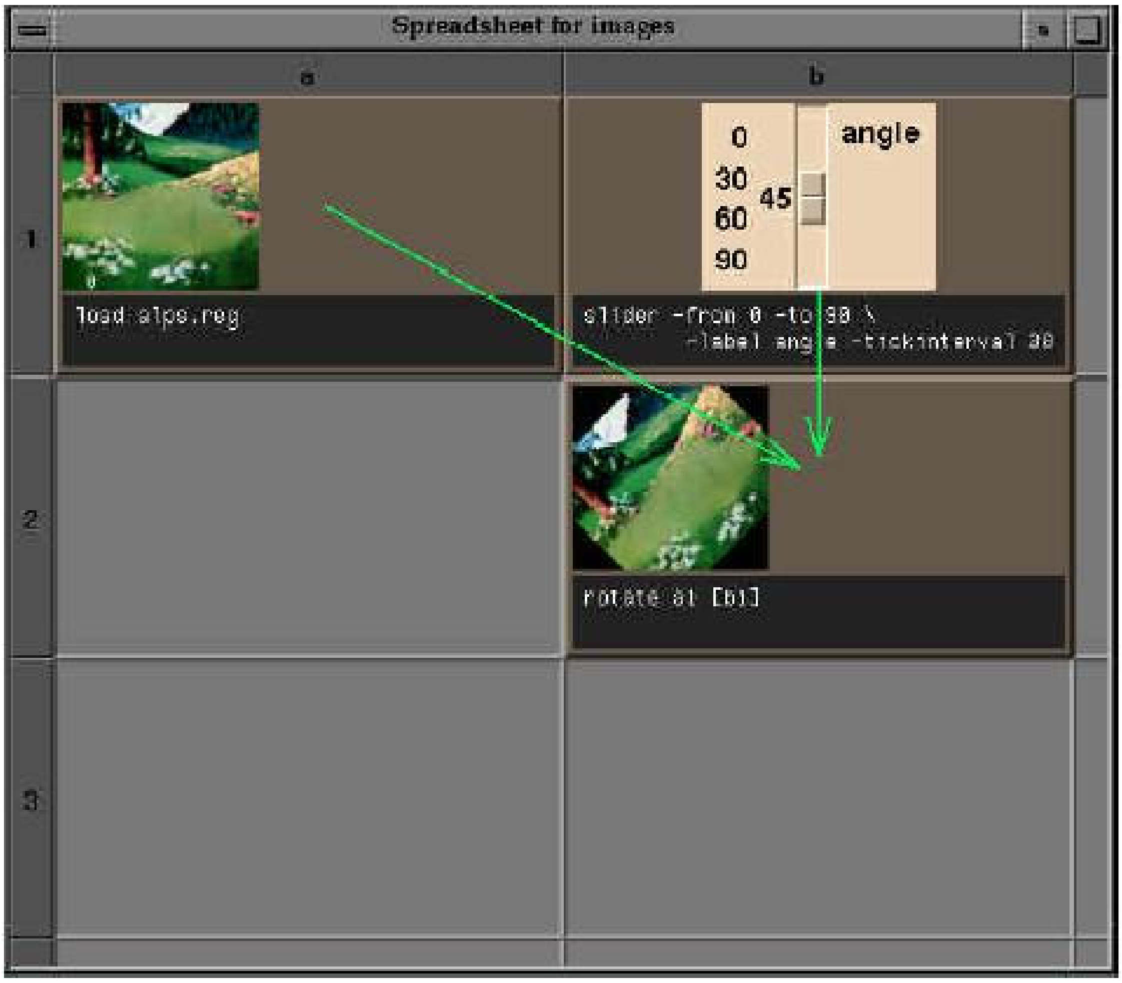
However, there is no way of specifying agent actions in Forms/3. This limits the extent to which Forms/3 can be used to support state exploration (A2). In his PhD thesis, Wong illustrates this by contrasting the construction of a business deal model using Forms/3 and using EM tools [Won03]. He concludes that Forms/3 cannot capture the semantics of the business deal model faithfully because it does not support agent actions.

NoPumpG is an early example of a spreadsheet-related system that attempts to extend the power of spreadsheets through relaxing the grid rule (F3) [Lew90, WL90]. NoPumpG uses free floating cells and allows the manipulation of graphical types, but it contains no facilities for grouping elements or implementing abstractions. Like Forms/3, NoPumpG has only limited support for agency. The Penguims (Programmable ENvironment for Graphical User Interface Management and Specification) environment is another spreadsheet style development environment. Penguims is specifically targeted at creating user interfaces [Hud94]. Components of an interface can be linked through dependency across a wide range of arithmetical and graphical types. Penguims gives support for exploratory modelling solely within the domain of user interface design.

2.3.2 Spreadsheets for Images

Levoy's spreadsheet for images (SI) [Lev94] enables users to visualise complex graphical data in a spreadsheet. Cells can contain 2D images, 3D volumes, movies or various interface widgets. The defining formula for a cell is written into the cell and takes the form of a fragment of the Tcl language [Tcl03] that can range in size from one line to a large program. An example spreadsheet, taken from [Lev94], is specified as:

```
a1: load alps.rgb
b1 : slider -from 0 -to 90 \ -label angle -tickinterval
30
b2: rotate a1 [b1]
```



```
load alps.reg
```

```
slider -from 0 -to 90 \
      -label angle -tickinterval 30
```

```
rotate a1 [b1]
```

Figure 2.9 – An example of a spreadsheet for images, taken from [Lev94]

Complex interactive visualisations of graphical data can be built using the extended range of types controlled by interactive widgets within a spreadsheet grid. The SI system relaxes the value rule (F4) by allowing a cell to modify values in other cells.

SI is a special-purpose system that is best suited to the manipulation of graphical data. In its particular domain, it has advantages over spreadsheets in the presentation and investigation of graphical data. Where aspects A1 and A2 are concerned, SI is effective within its restricted domain of application. Its use of a grid layout constrains the organisation of visual images, however (cf. the free organisation of the visual components and textual annotations in the vehicle cruise controller model discussed in section 2.2.3).

There are other systems that are similar to SI described above. The Spreadsheet for Information Visualisation (SIV) system, developed by Chi [CRB$^+$98], allows the definition of a wide variety of graphical primitives in a spreadsheet grid. SIV does not adhere to the value rule (F4). The Finesse system [VK96] (now known as ACUMEN [Acu03]) was developed to visualise real-time financial data. This system permits a wider variety of types than conventional spreadsheets. It uses acyclic relationships for formulae, and cyclic relationships to define the presentation of the cells. Both SIV and Finesse offer only application-specific support for aspect A2.

2.3.3 Agentsheets

Spreadsheet style grids have influenced the design of programming environments that are based on different programming paradigms. One such environment is Agentsheets, an interactive programming environment aimed at a wide range of users, that was developed by Alexander Repenning in 1993 [Rep93, Age03]. It exploits the simplicity of the grid environment in combination with a visual programming language. Agentsheets can be seen as an 'end-user programming' system as defined by Nardi [Nar93].

Agentsheets is targeted at creating interactive and exploratory simulations through locating agents in the cells of a grid and specifying their behaviour through a set of rules. Agentsheets has been used to build many hundreds of different simulations ranging from simple models constructed by children to models being used for serious research purposes at universities [Rep00, MPG$^+$02]. An Agentsheet incorporates two layers of abstraction: a graphical depiction of an agent and a set of rules that govern its behaviour as controlled by sensors and effectors [Rep93] (see Figure 2.10). The visible environment shows a graphical depiction of agents who can move and interact within the grid. Each agent has sensors to obtain information about the environment. These are fed into rules that affect, and are affected by, the current state of the environment.

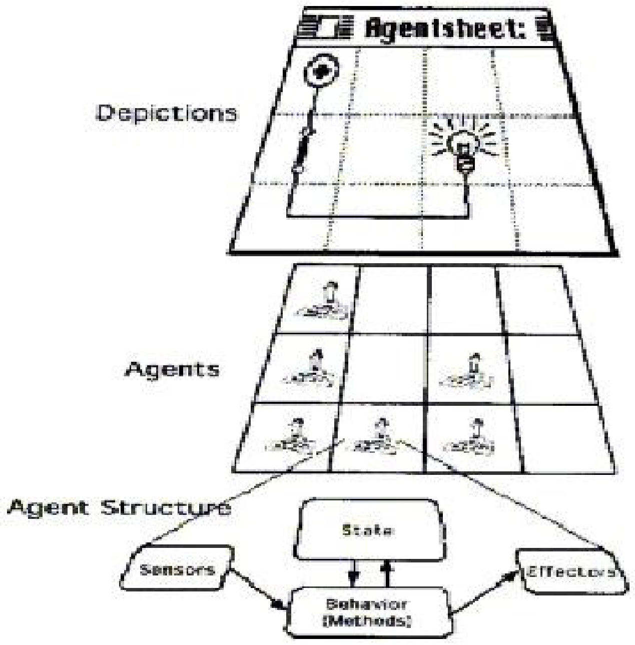


Figure 2.10 – Structure of an Agentsheet (taken from [Rep93])

Relationships between agents are defined in terms of rules that are bound to each type of agent. Each rule takes the form of an `if-then` clause and rules are executed in discrete time steps. Each agent can have many rules, although only the satisfied rule with the highest priority is executed in each time step. Agentsheets can be readily programmed to maintain dependencies between states of neighbouring agents; for example in Figure 2.10 the state of the light will depend on the state of the switch (cf. [Run03, p27] for a more sophisticated example of a similar nature). In other models, such as the epidemic model described in the next section, rules are time-dependent as agents move around their environment and change their physical characteristics. We shall use the epidemic example to illustrate in more detail how an Agentsheets model is constructed (see [RI01] for more details).

An Agentsheets example

The epidemic model illustrates how a contagious disease spreads throughout a population. It can be used to explore many questions; for example to investigate how quickly treatment needs to be available to control or extinguish the disease. As shown in Figure 2.11, a rectangular grid of squares represents the environment. There are two types of agent in the model: doctors and people. People can either be healthy or sick, as shown in the facial expressions in Figure 2.11.

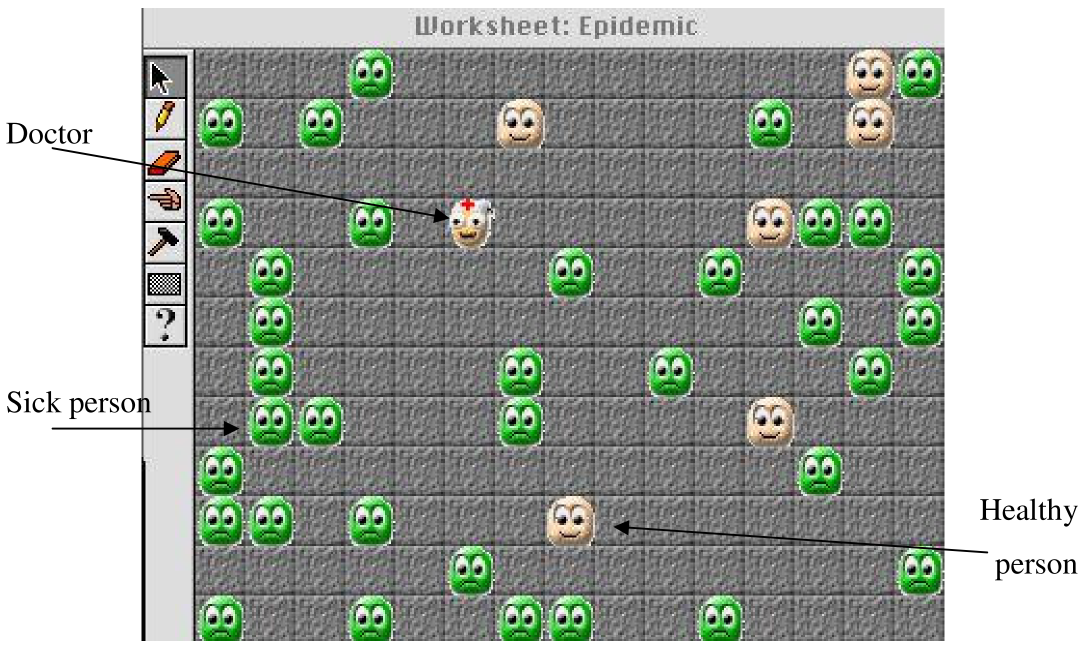


Figure 2.11 – A screenshot of the Agentsheets epidemic model

Each type of agent has rules that govern its behaviour and interactions with other agents in the model. Agents are programmed in Agentsheets using a visual programming language called VisualAgenTalk (VAT) [RA97]. Rules are composed through drag-and-drop construction from a menu of possible commands. Each rule has an identical structure and is expressed as an `IF-THEN` clause. By way of illustration, the rules for the behaviour of a person in the epidemic model are shown in Figure 2.12.

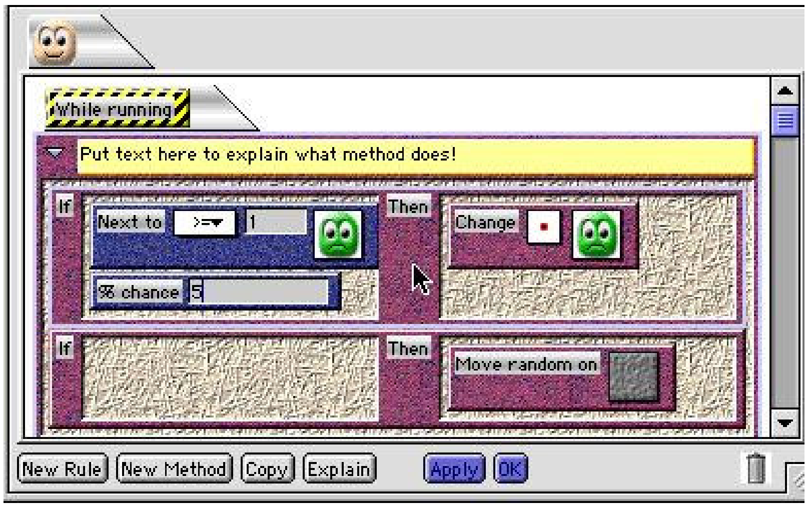


Figure 2.12 – A VisualAgenTalk rule for a person in the epidemic model.

Rules can be used to specify interactions with other agents and the environment. The top rule in Figure 2.12 can be read as: 'If I see a sick person next to me (i.e 1 square away in any direction) then, with a 5% chance, I become a sick person (i.e I get affected by the disease)'. Each agent can have multiple rules. Rules are checked in turn; if one has a conditional expression that evaluates to true, its `THEN` clause is executed and the other rules are ignored. In the spirit of investigative exploration, rules can be dropped onto agents to see their effects in the model [RIA98]. The following rules define the epidemic model:

i) person agent	– if I am next to a sick person, then with a 5% chance, I become sick.
	– I otherwise move around randomly in the world
ii) doctor agent	– if I see a sick person to my left, make them healthy.
	– I otherwise move around randomly in the world.

Experimentation can be used to investigate the conditions required for epidemics to spread by varying the number of people and doctors, their movement rules and the doctor's treatment rule.

The Agentsheets environment allows users to turn their Agentsheets simulations into Java applets for public demonstration on the Internet. Agents' behaviours are turned directly into class files and the agents' pictorial representations are turned into graphic files [RIA98]. After this compilation, no exploratory investigation of the simulation can be performed. We now discuss the support that Agentsheets offers for the key aspects of exploratory modelling A1 and A2.

Agentsheets and the key aspects of exploratory modelling

To support A1 and A2 in a computer-based modelling tool we need convenient metaphors and techniques to represent both dependency and agency. As the taxation example illustrated, the negotiation of the semantic relation is intimately linked with the identification of dependencies (cf. Figure 2.6, 2.7) and its elaboration is assisted by multiple types of agency. Although Agentsheets and spreadsheets share common characteristics, they also have significant differences. Agentsheets provides powerful mechanisms to implement both spreadsheet-like dependency and agency but lacks the explicit representations for dependencies (cf. [Her02]) that feature in spreadsheets. We now discuss the implications of this design and implementation strategy for exploratory modelling in more detail.

Both Agentsheets and spreadsheets use grids, but in a spreadsheet the grid is not usually a metaphor for space. By this we mean that the position of values and formulas in the spreadsheet are not representative of geometry in the referent. In contrast, in an Agentsheet, the grid is typically used as a metaphor for space in the referent being modelled (cf. [Rep93]). Agents' rules are defined with reference to the regular structure of the grid, for example using conditions that identify adjacent agents in a particular direction. Agentsheets is therefore especially suited to modelling situations where a rectangular geometry can be imposed on the referent. The visual language leverages this regular grid structure in the design of its primitives. As in spreadsheets, the use of a grid imposes some restriction on the range of modelling applications that can be conveniently supported by Agentsheets.

It may appear superficially that Agentsheets subsume spreadsheets in so far as they can be readily programmed to exhibit dependencies between cells (cf. [Run02, p27]). There is nonetheless a fundamental distinction between the use of spreadsheet-like definitions and the use of triggered actions to maintain dependencies between cells. Though triggered action can give the appearance of indivisibility in change, there is no counterpart in a rule-based system for the explicit identification of an indivisible relation in a spreadsheet definition. In rule-based programming, it is the modeller's responsibility to maintain the coherence of the state, and this is achieved by exploiting knowledge of the evaluation mechanisms. This is particularly relevant to supporting key aspect A1 of exploratory modelling.

Where the use of spreadsheets to support aspect A1 is concerned, the emphasis is on understanding the state of a referent and investigating 'atomic actions' from that state through manual state-transitions. It is this emphasis that motivates us to describe the development of a spreadsheet as concentrating on the representation of *state-as-experienced* (cf. section 3.4.2). A spreadsheet allows the user to identify the important observables in a situation and the relationships between them without any presumption of how they might change in the future. State-transitions are completely at the discretion of the human modeller, but the focus is on identifying - rather than automating - reliable behaviours.

In contrast, in Agentsheets the emphasis is on representing agents' behaviour through the construction of rules, not on the identification of important observables and relationships. Agentsheets concentrates on how a situation is changed through the overt behaviours of autonomous agents. Due to this, Agentsheets is not as suitable as a spreadsheet for investigating referents where there is limited knowledge at the outset of construction. It is in this respect that spreadsheets surpass Agentsheets in their capacity to support aspect A1.

Agentsheets has more power with respect to A2 than spreadsheets due to its use of agents that can perform autonomous actions in the grid. Whereas a spreadsheet offers no features specifically designed to model concurrent interaction, Agentsheets overcomes this through the creation of agents and rules to specify their behaviours. Agent rules, defined in the VAT language, are a way of introducing autonomous action that does not require the intervention of a human modeller.

Unlike the spreadsheet, where provision for autonomous action is of secondary importance, Agentsheets offers facilities for specifying action that are easy for the modeller to invoke. It has been shown that it is easier for the non-specialist to construct rules through the Agentsheets visual interface than to write them in textual form [RIZ00]. Syntax errors are eliminated through the dragging and dropping of pre-defined code primitives into each rule, which can be easily comprehended through the combination of graphical agent depictions and English language.

The support for specifying actions in Agentsheets extends to model design. The design philosophy of Agentsheets is that of *participatory theater*, an approach that combines direct manipulation of agents and delegation of roles to agents [RS94]. This builds on the interactive qualities of a spreadsheet system, where the user has complete discretion over the redefinitions that are made and the times at which they are made. Agentsheets draws on the metaphor of a theatre where the modeller is the director in charge of proceedings. The modeller has control over the roles that are given to agents, but once the simulation (or play) has started agents act according to their own ‘script’ of rules. Users can intervene in the play at any time and make alterations to agents’ rules on the fly. In this way, Agentsheets potentially supports the incremental and evolutionary construction that is observed in spreadsheets. The only problematic issue is that the absence of explicit dependency may make it difficult to link one model to another through dependency.

In this section, we have discussed three research products, Forms/3, Spreadsheet for Images, and Agentsheets, how they differ from spreadsheets and how this impacts on their support for exploratory modelling. We have concluded that none of these products succeeds entirely in supporting the key aspects of exploratory modelling. In the following section, we introduce modelling with definitive scripts, using the TkEden modelling tool developed by the Empirical Modelling research group. We shall argue that this tool offers better support for the key aspects of exploratory modelling.

2.4 Practical Empirical Modelling

The purpose of this section is to introduce practical Empirical Modelling and discuss the support that it offers for the key aspects of exploratory modelling discussed in section 2.2.2. An EM model consists of a definitive script together with a set of agent actions. Figure 2.13 is an abstract depiction of the way in which the significant concepts associated with EM are represented in practical model development.

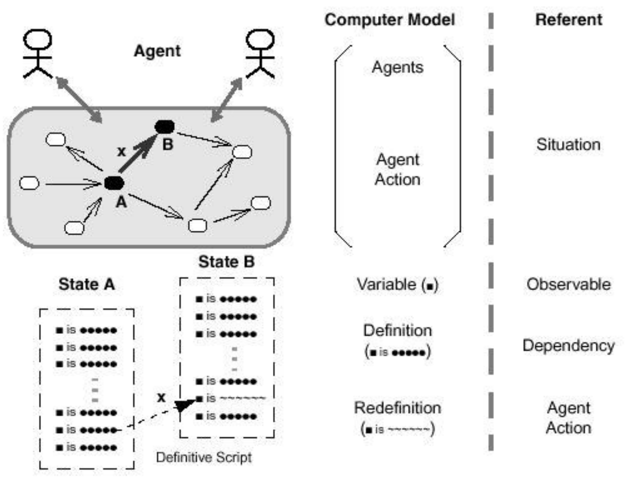


Figure 2.13 - The significant concepts associated with Empirical Modelling

The key concepts and relationships depicted in Figure 2.13 will be introduced in this section from a practical perspective (cf. section 3.4 for a complementary conceptual perspective). We firstly introduce the idea of a definitive script and then describe the TkEden modelling tool, showing how it can be used to support the key aspects of exploratory modelling A1 and A2.

2.4.1 Definitive scripts

The construction of computer-based models in EM is primarily achieved through the creation of definitive scripts. A definitive script is a set of definitions that represents *dependencies* between *observables* (represented as variables). A definition is of the form:

```
      q is f(a, b, …)
```

where `f` is a function and `a,b` are parameters passed to that function such as can be found in conventional programming languages. Definitions take the form of one-way dependencies that satisfy the value rule (cf. section 2.2.2). The value of `q` will always reflect the value of the function `f` applied to the parameters `a` and `b`. This is the literal meaning of the keyword '*is*'. Redefining either `a`, `b`, or the function `f` will mean that `q` is automatically re-evaluated. A collection of definitions forms a definitive script, which is used to represent the current state of a model. In Listing 2.1, the values of `E`, `F` and `G` will indivisibly change if `A`, `B`, `C` or `D` change. As in spreadsheets, there is no circumscription of the future states of the model. The user is free to make whatever redefinitions they desire.

```
A = 3;
E is A+B;
B = 4;
F is max(C,E);
C = 5;
G is D*pow(A,F);
D = 6;
```

Listing 2.1 – An example of a simple definitive script

A definitive script can be thought of as extracting the values and formulae from a spreadsheet and discarding the information regarding their grid locations (such a conversion from spreadsheet to script can also be found in Jocelyn Paine's Model Master [Pai01]). The relationships between variables in Listing 2.1 could be replicated in the cells of a spreadsheet by mapping the variables used onto valid cell references. Listing 2.1 preserves spreadsheet features F1 and F2 (section 2.2.1) in that it uses operands to define formulae, and uses dependency maintenance to ensure that those definitions are always correctly maintained. The primary difference is that the definitive script does not use the grid structure of the spreadsheet to display its values. A definitive script supports one-way dependencies (multi-way constraints are not allowed) and therefore supports the value rule. However, as described in the next section, models developed using TkEden combine definitive scripts to represent the state of a referent with agent actions to specify transitions between states.

2.4.2 The TkEden modelling tool

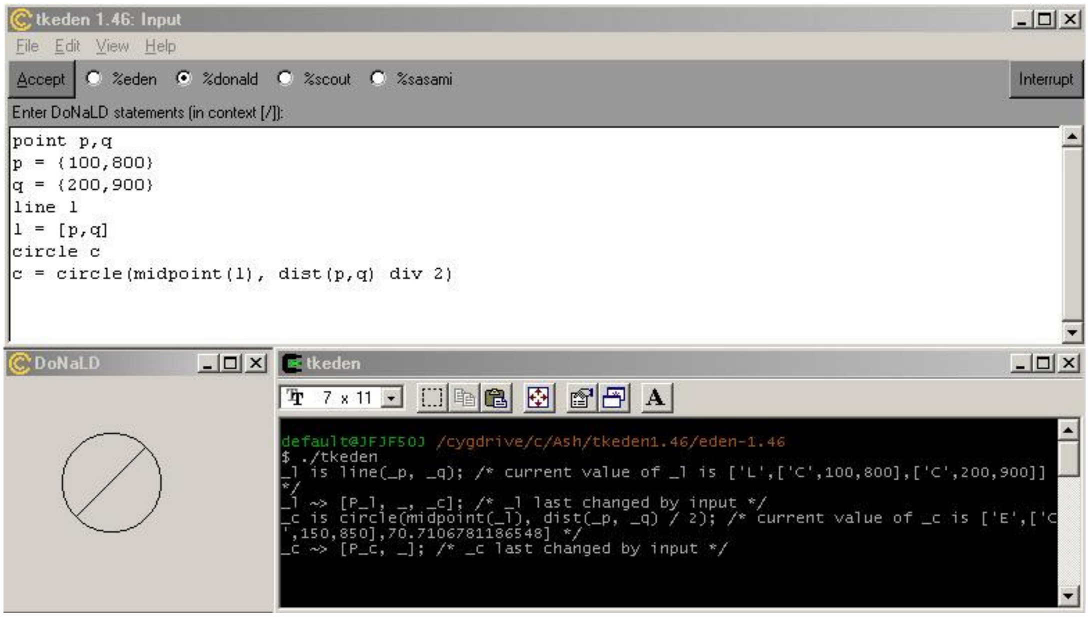
The Empirical Modelling research group has developed many computer-based tools to construct definitive scripts. The Eden tool (Evaluator of DEfinitive Notations) has been, and is currently, the most commonly used. It was originally written in 1989 by Y.W.Yung [Yun90]. It was updated by Y.P.Yung to run under a Tcl/Tk interface and acquired the name TkEden [Yun93]. Current versions run on Unix, Windows and Macintosh platforms. Several hundred student projects and academic case studies have been produced using it. A significant number of these can be accessed through a web repository of models that contains descriptions, screenshots and download facilities [EMRep].

The TkEden modelling environment comprises three windows, as shown in Figure 2.14 [BWM+00]:

i) The *input* window (top): This allows the modeller to add new definitions to the model, redefine existing definitions, introduce new scripts of definitions or interrogate the values of variables and formulae in the model.

ii) The *interface* window (bottom left). This shows the interface to the model that has been constructed by the modeller. In contrast to a spreadsheet (cf. section 2.2.1), not all the values are permanently displayed; the modeller chooses which values should be on screen and the form in which they are displayed.

iii) The *commentary* window (bottom right). This can be used to give information about the current state of the model when the modeller requests it.



```
point p,q
p = {100,800}
q = {200,900}
line l
l = [p,q]
circle c
c = circle(midpoint(l), dist(p,q) div 2)
```

```
default@JFJF50J /cygdrive/c/Ash/tkeden1.46/eden-1.46
$ ./tkeden
_l is line(_p, _q); /* current value of _l is ['L',['C',100,800],['C',200,900]]
*/
_l ~> [P_l, _, _c]; /* _l last changed by input */
_c is circle(midpoint(_l), dist(_p, _q) / 2); /* current value of _c is ['E',['C
',150,850],70.7106781186548] */
_c ~> [P_c, _]; /* _c last changed by input */
```

Figure 2.14 – The three windows in the TkEden modelling environment.

TkEden is an interactive modelling environment in which (in typical use) a modeller can create, modify and interrogate a definitive script. Firstly, new variables or definitions can be introduced to the script (cf. adding new cells to a spreadsheet). Secondly, definitions can be created or modified to update the network of dependencies in the script (cf. modifying the formulae in a spreadsheet). Thirdly, there are facilities for the modeller to query the values and defining formulae of variables in the script (cf. viewing the spreadsheet).

The main modelling notation used in TkEden is EDEN. The EDEN notation allows the modeller to create dependencies between observables using the keyword '*is*', as shown in Listing 2.1. TkEden uses dynamic typing to determine the type for a variable when it is defined. The TkEden tool contains a number of built-in notations that allow definitive scripts to be created to fulfil different specific functions. EDEN is a general-purpose notation. The special-purpose notations included in TkEden are DoNaLD (for 2-D line drawing), SCOUT (for screen layout), Sasami (for 3-D modelling) and Eddi (for database handling).

DoNaLD is a strongly typed notation for defining two-dimensional line drawings [BAB+96]. Graphical objects are created that can be dependent on any variable in the script. For instance, the length of a line can depend on a scalar value recorded in EDEN. A number of geometrical constructions and transformation functions are included as standard. When any element is changed, the consequent dependencies are indivisibly updated. For example, if the point p is moved in the example below then the line and circle will both be automatically updated.

```
%donald
viewport example
point p,q
line l
circle c
l = [p,q]
p = {50,50}
q = {100,100}
c = circle(q, dist(p,q) )
```

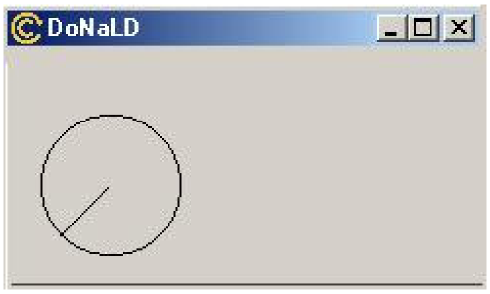


Figure 2.15 – An example DoNaLD fragment to define a circle

SCOUT is a definitive notation that describes screen layout [Yun93]. Its functionality is similar to the Penguims system in which interface objects can be related through the use of dependency [Hud94]. It can be used to define windows on the screen into which text, images and DoNaLD drawings can be placed. Each window has a number of attributes that can be dependent on EDEN variables in the script. The picture in the SCOUT window (Figure 2.16) is the DoNaLD drawing defined in Figure 2.15.

```
%scout
window donpic = {
  type: DONALD
  box: [{10, 10}, {200, 200}]
  pict: "example"
  border: 2
  xmax : 500
  ymax : 500
};
screen = <donpic>;
```

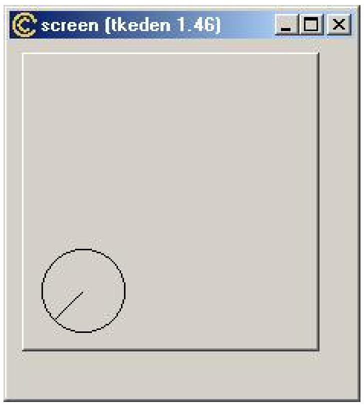
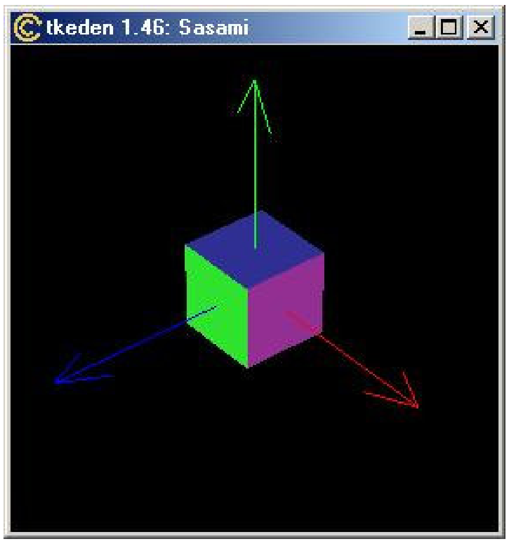


Figure 2.16 – An example of a SCOUT fragment to display the drawing in Figure 2.15 in a window

The Sasami notation [Car00] defines three-dimensional shapes, using the OpenGL library to render the graphics [OGL03]. It contains primitives to describe polygons together with their colour, material and lighting attributes. These can be linked through dependency to any part of an EDEN model. Files in OpenGL format can be loaded as Sasami data types. The listing below defines a 3D cube with distinct coloured faces.



```
%sasami
`size = 0.2;
viewport 280 280
open_display

vertex blf -size -size  size
vertex brf  size -size  size
. . .

polygon frontp
polygon backp
. . .

poly_geom_vertex frontp blf brf trf tlf
poly_geom_vertex backp blb tlb trb brb
. . .

poly_colour frontp  0 1 0 1
poly_colour backp   1 0 0 1
. . .

object cube
object_poly cube frontp backp topp bottomp leftp rightp
```

Figure 2.17 – An example of a Sasami fragment to display a coloured cube

The TkEden interpreter also includes a special purpose parser generator that enables user-defined definitive notations to be created interactively. This allows domain-specific notations for interaction to be created – a key consideration for end-user programming [Nar93]. New notations are implemented by using a novel observation-oriented parsing approach discussed in detail in section 5.4.1. This will be illustrated with reference to the EDDI (Eden Definitive Database Interpreter) notation for relational algebra.

2.4.3 State-transitions: Implementing agency in TkEden

The definitive notations described in the previous section supply rich metaphors for representing state-as-experienced in a computer model. TkEden has more expressive power than a conventional spreadsheet due to the larger number of types that can be used. The features of EM models as described thus far do not include facilities for moving between states automatically, though they support state-change through manual redefinition by the modeller. However, TkEden also allows the specification of automatic state transitions implemented using triggered actions. A triggered action is a procedure that is run every time any one of a set of variables is changed (cf. activation-oriented programming in Boxer [diS97a]). Triggered actions are usually incorporated into a model in order to automate reliable patterns of behaviour observed in the referent.

In exploratory modelling, a human modeller makes changes to one or more values or definitions to test whether they are sensible interactions. For example, redefining the capacity of a jug may be sensible, but defining the capacity to depend on the content clearly is not. A group of one or more redefinitions in a triggered action can represent a stimulus-response action based upon a well-understood change in an observable. For example, in the restaurant model (to be discussed in detail in chapter 3), I – as the modeller – experimented as if in the role of a restaurant manager allocating tables in response to queries. Initially I performed these actions manually, but when I gained enough understanding of the situation, I could introduce an automatic agent to carry out the routine. The crucial difference between EM and conventional programming is that such reliable behaviours are the end result of experience gained through interaction with the model, and not the starting point for writing a program.

We now give a small example to illustrate the EM model building approach.

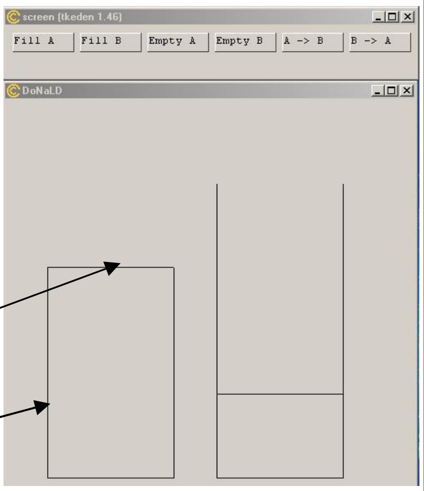
2.4.4 Building an example model

The easiest way to appreciate how a model is constructed is to build one yourself. With this in view, I will briefly outline the construction of an example model (see [Bey01] for a more extended account of the experience of constructing a simple clock model). Our example is a model of the game of Jugs, based on a program first developed for the BBC microcomputer by Ruth Townsend of the Chiltern Advisory Unit. The full listing for the model can be found in Appendix A. We recommend the reader consult Appendix A in conjunction with this section.

The basic game of Jugs is formulated as follows: There are two jugs of specified capacities that have no intermediate markings on them. The aim of the game is to collect a specific quantity of water in one of the jugs. There are three permissible operations: emptying a jug completely; filling a jug completely; or pouring water from one jug to the other until the destination jug becomes full or the source jug becomes empty.

The development of the model is interactive and evolutionary, beginning from identifying important aspects of the Jugs game and moving towards the automation of reliable behaviours. Construction starts by recognising the essential observables of the jugs game, such as the capacities and contents of the jugs. Development of the graphical interface is interspersed with the development of the underlying model. Throughout development the input window can be used to perform experiments or explore possibilities. Reliable behaviours can be automated through the use of actions to represent filling, emptying and pouring. At this point, the model merely represents two jugs and operations that can be performed on them. The model is open to any purpose to which we wish to put it. In particular, we can fix the functionality of the jugs model as a game in which the player manipulates the jugs to obtain a particular target. This is realised by introducing a target variable and a message that is dependent on the jugs contents, which tells us whether we have achieved the target.

The Jugs model can be built interactively in one modelling session by entering the definitions and actions into the input window and executing them. Figure 2.18 shows the Jugs model from Appendix A.



```
This listing contains some small
fragments of the Jugs model. The full
listing can be seen in Appendix A.

%eden
capA = 5;
capB = 7;
scalefactor = 100;
Afull is (capA == contentA);
Aempty is (contentA == 0);
%donald
Asurface = [Abotleft + {0,contentA! * scalefactor},
            Abotright + {0,contentA! * scalefactor} ]
Aleft = [ Abotleft,
          Abotleft + {0, capA!*scalefactor}]
```

Figure 2.18 – The Jugs model from Appendix A

A model is always open to extension or refinement (cf. how a spreadsheet is always extensible). For instance, we can add a feature to identify whether a particular instance of the jugs problem is solvable by introducing the following definition:
`targetachievable is gcd(capA,capB,target) == gcd(capA,capB);`
Personal appreciation of this criterion emerged through interaction with the model. A further extension of the model involves adding a feature to give advice on which operation is best performed next to achieve the specified target. This requires an intimate understanding of the Jugs game and the underlying mathematical concepts (see [Roe99, Appendix B]).

The purpose of highlighting opportunities for future development is to illustrate how EM supports exploratory modelling without preconception of intended use. The model is akin to a laboratory for investigating Jugs-based activities, because the emphasis is on open and flexible interactive construction. We now discuss an enhancement of the TkEden modelling tool that enables a group of modellers to collaboratively construct and/or interact with a model.

2.4.5 Distributed modelling: The DTkEden tool

The key aspects of exploratory modelling identified in section 2.2.2 can also be supported in situations where many distributed modellers interact to construct a joint model, or where many users play roles within a pre-constructed model. The DTkEden (Distributed TkEden) tool, built by Patrick Sun [Sun99], enables many modellers to construct a model collaboratively across a local area network. DTkEden exhibits a client-server architecture. To facilitate collaboration, communication between participants is mediated by sending definitions between clients, or to the central server. Different communication modes can simulate different types of inter-personal communication [BS99]. For instance, to simulate group conversations, communicated definitions are sent to every client.

An example of a distributed model is the Clayton Tunnel model, developed by Patrick Sun. It allows the enactment of railway operation in the vicinity of the Clayton Tunnel near Brighton in 1861 and can be used to illustrate how the use of a telegraph device contributed to a historic railway accident [Rol82]. For a fuller account of this model and the accident scenario see [Sun99, chapter 6]. The participants in the situation were the two signalmen at the ends of the tunnel (Killick and Brown), and three train drivers. The perspective of each participant is modelled at one of the client workstations. For example, the signalman Killick can operate the telegraph device, wave flags and reset an alarm that rings if the signal fails. Killick can only see trains when they are within a certain distance of his signalbox. The server shows the unfolding situation from the perspective of an external observer with exceptional state-changing privileges. The interfaces for the server and clients can be seen in Figure 2.19.

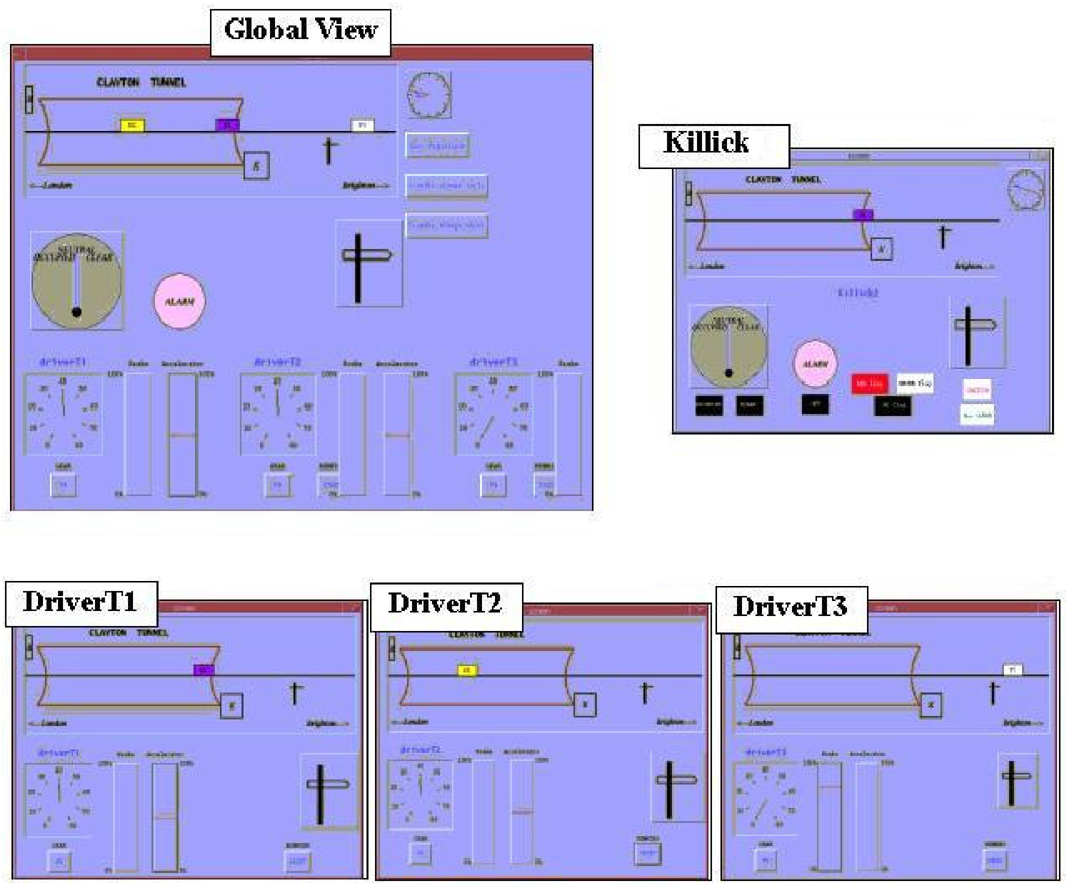


Figure 2.19 – Screenshots of the Clayton Tunnel simulation from the perspectives of each of the participants.

The Clayton Tunnel model has been used in a role-playing exercise in which users take the part of the participants and act out their roles to investigate how and why the crash occurred, and to explore how it could have been averted. In general, distributed simulations give a good idea of how situations that involve many interacting agents can be faithfully represented and explored [BS99]. Other applications of DTkEden in understanding situations involving concurrent interactions have been considered in different fields, such as business process reengineering [Che01], decision support systems [Ras01] and financial applications [Maa02]).

2.4.6 Empirical Modelling for exploratory modelling

In this section we describe how Empirical Modelling offers support for the key aspects of exploratory modelling introduced in section 2.2.2.

To the extent that EM generalises the spreadsheet concept, it inherits many of the spreadsheet's qualities in respect of the key aspect of exploratory modelling A1. Because EM is centrally concerned with modelling with definitive scripts, the construction of a model using TkEden has the same evolutionary and incremental character as the construction of spreadsheets. At any time, new definitions can be added to a model or existing definitions redefined in the light of new understanding of the situation.

The relaxing of the grid restriction in the TkEden modelling tool has implications that extend beyond simple practical matters. These relate directly to the observation in section 2.2.3 that the limited data types and the grid in the spreadsheet restrict the support for A1. Only certain types of data can be conveniently displayed in a grid. Moreover, in a spreadsheet, all the values are displayed on the interface. In an EM model, the interface contains only the features that the modeller requires to apprehend and interact with the current state of the model. This hiding of information enhances the model's usefulness as a metaphorical representation of its referent. In a spreadsheet, the grid constrains the way in which information can be visualised and dependency relationships can be conveyed. In a script, there is no grid and the interface can be organised to present the model in a way that is most suitable for the referent. The grid, although useful in certain applications, can in general lead to a comprehension problem because it detracts from the experiential, metaphorical role of the spreadsheet. One further practical advantage of the free format script is that, although spreadsheets do allow the modeller to add new variables at any time, the locations of new cells are potentially constrained by the information already in the grid. For example, to add a block of data cells to a spreadsheet in a sensible place may require some large scale editing of the spreadsheet. This in turn can lead to comprehension problems and errors in the spreadsheet. Spreadsheet errors that arise in this way are well documented in the literature [NM91, PH96, GP96, PH97] and numerous methods have been proposed for overcoming them. These include: visualisation of dataflow [IMC$^+$98]; and the concept of 'tested cells' [BSR99]. However, Panko doubts whether any of these methods will eliminate errors in sizable spreadsheets [Pan00]. In model building with TkEden, such errors in dependency structures that stem from the use of a grid are eliminated. The use of a grid also means that the organisation of an interface within a spreadsheet is relatively tightly constrained.

Computer-based support for key aspect A2 of exploratory modelling hinges on being able to complement dependency with powerful and appropriate means to represent procedural action. As a general purpose programming language, TkEden offers the same unrestricted functionality as procedural extensions to spreadsheets, but unlike these, it privileges the procedural actions that entail redefinition in a definitive script. Through the use of triggered procedures, TkEden supports an agent metaphor for action similar in character to that afforded by Agentsheets. Although TkEden does not have a visual agent language, agent communication and interaction is more general than in Agentsheets since agents are not defined with reference to grid locations. Although TkEden does not offer the end-user the facility to manage dependency and agency on the same scale as Excel and Agentsheets respectively, it makes more general provision for dependency and agency and gives greater support to their integrated use.

The relative merits of TkEden in supporting key aspect A2 of exploratory modelling are exemplified by revisiting the problem of modelling the concept of a vehicle cruise controller discussed in section 2.2.3. A TkEden model of a vehicle cruise controller can be found in [EMRep, cruisecontrolPavelin2002] and more details of its development are described in [BBY92].

2.5 Empirical Modelling and spreadsheets

In the previous sections of this chapter, we have identified principles for exploratory modelling and considered the extent to which the spreadsheet supports them. We looked at research efforts that have been based on the spreadsheet concept and discussed their support for these principles. We then discussed the extent to which practical EM overcomes some of the limitations of other approaches.

In this final section, we explore the relationship between EM and practical spreadsheets by considering a case study that implements a spreadsheet using TkEden. The primary motivation for this case study is to 'close the loop' in Figure 2.1 by investigating the links between EM and spreadsheets. A secondary motivation for this case study is to give practical evidence in support of claims made in past EM research concerning the connection between definitive scripts and spreadsheets. The following quotes, taken from past EM papers, illustrate these claims:

> 'A spreadsheet – stripped of its tabular interface – provides the simplest example of a definitive notation in which the underlying algebra is traditional arithmetic' [Bey87a]

> 'Definitive notations are a more general way of modelling than spreadsheets because they are not constrained by the grid interface and data type' [Geh96]

> 'The key significant idea of spreadsheets – state change through dependency and agency – has not really been taken up seriously in conventional software' [RRB00]

Spreadsheets are by far the most commonly referenced standard computing topic in EM research publications. I have used TkEden to construct a model that both replicates the essential features of a conventional spreadsheet and also allows significant extensions [EMRep, spreadsheetRoe2002]. This model:

i) exhibits all the essential features of a spreadsheet identified in section 2.2.1.

ii) illustrates the dependency over a wider range of types than a conventional spreadsheet (cf. Forms/3 in section 2.3.1) that can be leveraged in support of aspect A1 of exploratory modelling.

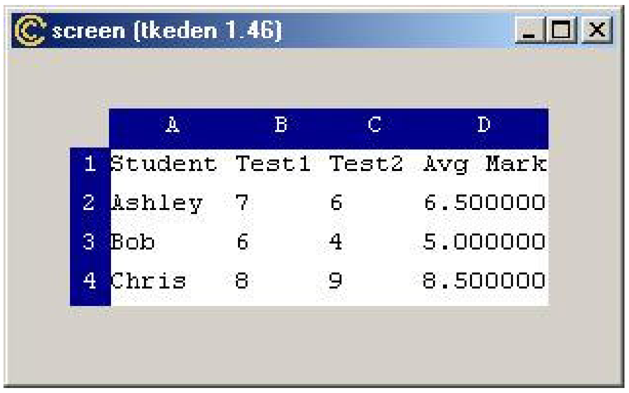
iii) enables manual and automatic execution of many varieties of procedural action consistent with the relaxed value rule (F4) that can be leveraged in support of aspect A2 of exploratory modelling.

The spreadsheet model comprises an underlying definitive script that defines the contents of the spreadsheet and its visual layout. An example spreadsheet can be defined by specifying values and definitions of cells, as shown in Figure 2.20. It is also possible to specify the attributes of cells as values or definitions. For instance, the definition:

```
D2_bgcolor is (D2 > 8) ? "green" : "red";
```

determines the background colour of the cell `D2` in such a way that it is green if its value is above a certain threshold.

```
A1 = "Student"; B1 = "Test1";
C1 = "Test2"; D1 = "Avg Mark";
A2 = "Ashley"; A3 = "Bob";
A4 = "Chris";
B2 = 7;
B3 = 6;
B4 = 8;
C2 = 6;
C3 = 4;
C4 = 9;
D2 is average(B2,C2);
D3 is average(B3,C3);
D4 is average(B4,C4);
```



	A	B	C	D
1	Student	Test1	Test2	Avg Mark
2	Ashley	7	6	6.500000
3	Bob	6	4	5.000000
4	Chris	8	9	8.500000

Figure 2.20 – A small example TkEden spreadsheet

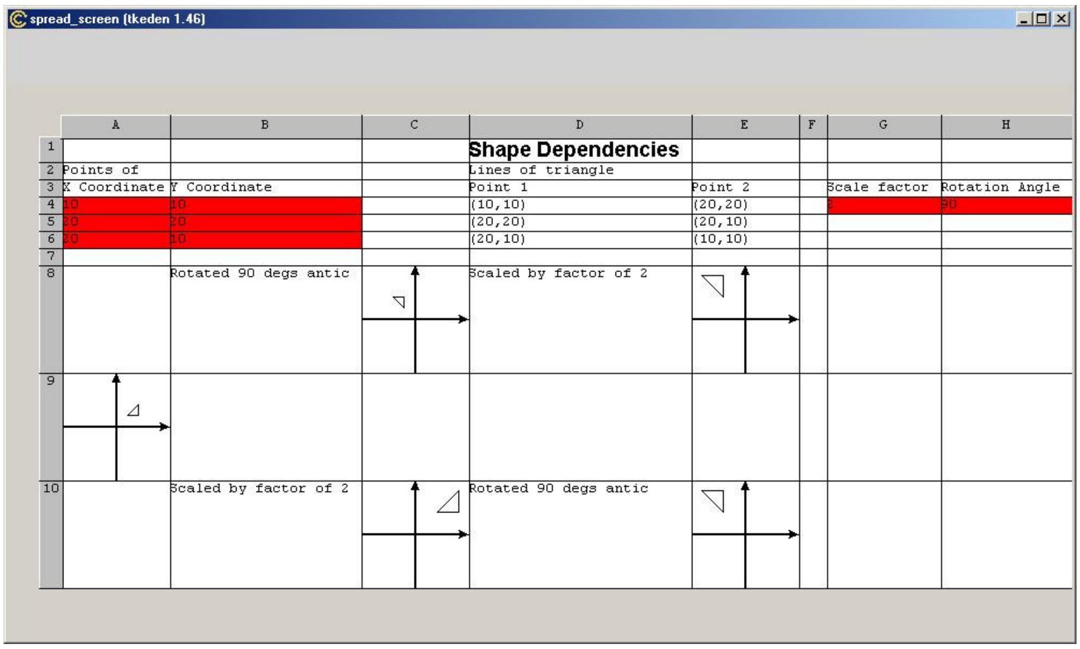
The TkEden spreadsheet readily illustrates the four characteristic features of a spreadsheet identified in section 2.2.1. Dependency maintenance is handled via the underlying Eden interpreter that automatically maintains definitions. Formulae can utilise any of the operators or functions available in the EDEN interpreter. These include the usual arithmetic operators, predefined functions and user-defined functions. The spreadsheet grid is defined as a set of SCOUT windows whose widths and heights depend on their contents, and whose positions depend on the other cells in the spreadsheet. A pure definitive script necessarily satisfies the relaxed version of the value rule. These four features show that a definitive script can be used to replicate a conventional spreadsheet.

In accordance with the demands of aspect A1 of exploratory modelling, the TkEden spreadsheet can generalise the notion of dependency by supporting a wider range of types and dependencies between types. To illustrate this, Figure 2.21 shows how the EM spreadsheet can be used to maintain dependencies based on transformations of geometrical shapes. In Figure 2.21, the rectangular block of cells (A4..B6) define the coordinates of three points in 2-dimensional space. The triangle defined by these three points is displayed on coordinate axes in cell A9. Cells G2 and H2 contain values relating to geometrical transformations, namely an angle of rotation and a degree of scaling, that are performed on the triangle in cell A9. Rows 8 and 10 contain the results of applying these transformations, and this incidentally shows that their order does not affect the final result. The shaded cells show the data values with which a user is expected to experiment.

In accordance with aspect A2 of exploratory modelling, we illustrate how the dependency in a spreadsheet script can be used to make the preconditions for agent actions visible to the user. For instance, in the restaurant manager example in Figure 2.22 cell B8 represents a menu option that is available provided that the cells C4, C5 and C6 have valid data. A definition of the general form:

```
B8_bgcolor is (C4!="")&&(C5!="")&&(C6!="")?"green" :"red";
```

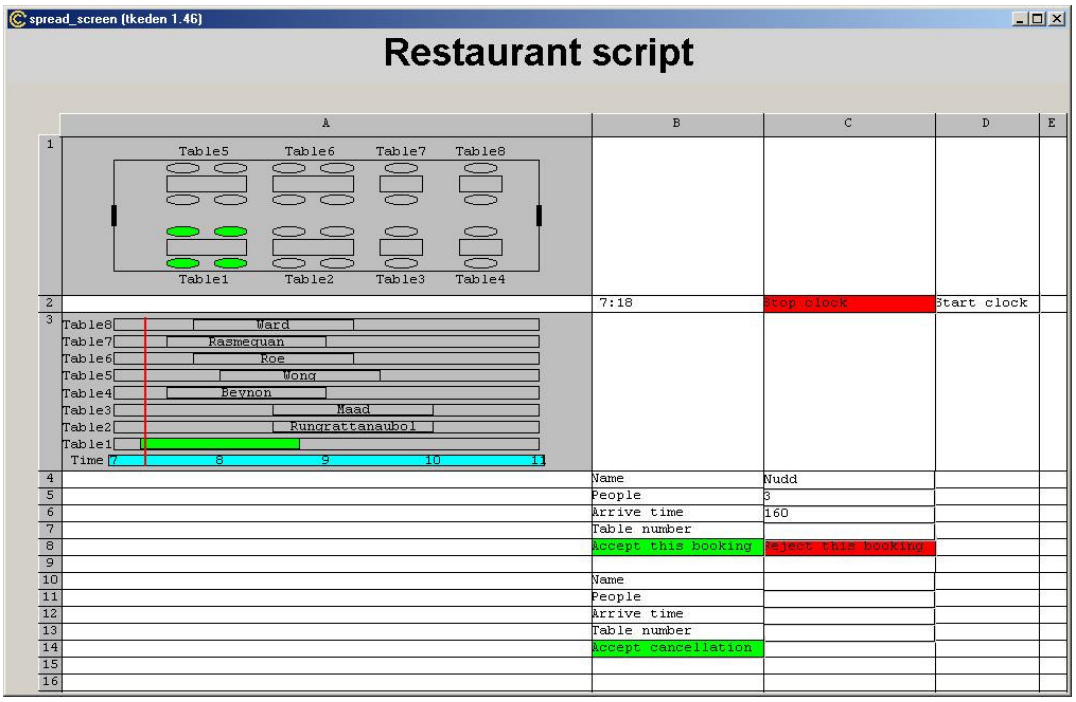
guarantees that the colour of the menu option always faithfully reflects its availability.


spread_screen (tkeden 1.46)

	A	B	C	D	E	F	G	H
1				**Shape Dependencies**				
2	Points of			Lines of triangle				
3	X Coordinate	Y Coordinate		Point 1	Point 2		Scale factor	Rotation Angle
4	10	10		(10,10)	(20,20)		2	90
5	20	20		(20,20)	(20,10)			
6	20	10		(20,10)	(10,10)			
7								
8		Rotated 90 degs antic		Scaled by factor of 2				
9								
10		Scaled by factor of 2		Rotated 90 degs antic				

Figure 2.21 – The TkEden spreadsheet illustrating geometrical shapes in a spreadsheet

Figure 2.22 is a spreadsheet that has been derived from an EM restaurant manager model that was originally developed without the use of a grid. The EM restaurant manager model is discussed in detail in chapter 3, and can also be found in [EMRep,restaurantRoe2000] and [RRR00, Ras01]. The interface of the restaurant model has been adapted to be displayed in the spreadsheet. Clicking on cells in the spreadsheet performs various actions in the restaurant model. For example, clicking on cell D2 will start a clock running that continues until a customer event is generated. Each event is one of three types: telephone enquiries, off-the-street enquiries and cancellations. By clicking on cell B8, C8 or B14 an appropriate action is undertaken, such as allocating an appropriate table. These actions resemble the roles and choices a restaurant manager is faced with when allocating tables in restaurants.



Restaurant script

	A	B	C	D	E
1					
2		7:18	Stop clock	Start clock	
3					
4		Name	Nudd		
5		People	3		
6		Arrive time	160		
7		Table number			
8		Accept this booking	Reject this booking		
9					
10		Name			
11		People			
12		Arrive time			
13		Table number			
14		Accept cancellation			
15					
16					

Figure 2.22 – The restaurant model in a spreadsheet

The original restaurant model (see Figure 3.4) does not use a spreadsheet grid interface and is not constrained by its geometry. It was advantageous to develop the model without the grid interface because this allowed freedom to organise interface objects at will and to refer to observables hidden from view. For instance, the graphical depictions of the restaurant utilise many observables that are dependent on the occupancy of the restaurant. It is instructive that once a useful functionality for the model has been identified, the use of a spreadsheet grid to display the model can assist the user in both comprehending and manipulating the model. This is because the key observables of experimental interest in the underlying data model can be added to the restaurant visualisation in the spreadsheet. Users can then use the familiar cell names to reference and change these observables directly.

The above examples illustrate some aspects of the practical relationship between Empirical Modelling and spreadsheets. There are many more examples in [EMRep, spreadsheetRoe2002]; these include features such as dependencies in images, presentational dependencies and the visualisation of other pre-existing EM models in spreadsheet grids.

2.6 Summary of the chapter

In this chapter, we have shown that an EM approach to model construction builds on the support that spreadsheets offer for negotiation and elaboration of the semantic relation β. In the following chapters, we shall consider the role of negotiation and elaboration of the semantic relation β from a learning perspective, with specific reference to the relationship between domain learning and computer-based model construction.

Chapter 3 – An experiential perspective on learning

3.0 Overview of the chapter

In this chapter, we set out the main challenges that are faced in using computers for learning and discuss the broad framework on learning that informs the research in this thesis. In the field of computers for learning, there are two perspectives that need to be considered: that of the educationalist and that of the computer specialist. Each typically has different concerns, and it is the successful marriage of these concerns that will yield positive results. In this chapter, we shall set out a major claim of this thesis: that in respect of learning through building computer models, EM is in general more suitable than other approaches because model construction and development of domain understanding are intimately linked. The latter part of the chapter discusses the theme of EM and learning in detail to conclude that the principles of EM model construction support a wide variety of learning activities. We introduce an Experiential Framework for Learning (EFL) that describes how learning activities are broadly related and are rooted in our personal and private experience. We shall argue that the EFL supports a general view of learning and that the principles of EM are well aligned to supporting the range of activities described in the EFL.

3.1 Challenges for computers for learning

In this thesis, we consider the challenges that are faced in successfully using computers for learning. There are a range of important issues that span computer science and education. The perspectives of the specialists in these two fields differ to such an extent that it is hard to identify a common agenda. Achieving a closer collaboration between all the participants in using computers for learning, namely educationalists, computer specialists and learners, is necessary for successful computer-based learning [diS97b]. In examining the issues from the perspectives of the educationalist and the computer scientist, we consider two rather different ways in which computers are used to support learning through:

- **the use of educational software**. In traditional educational software, a learner interacts with the software but does not modify it. This way of using computers for learning embraces different learning paradigms, including both instructionist and constructionist approaches (these terms are introduced in section 4.1).
- **building of computer-based models**. There are a number of areas in which computers are currently being used for building models to aid learning, including: financial models built in spreadsheets; scale models built in engineering; and prototypes built for software engineering. The primary objective in building these types of models is to acquire domain related knowledge. These models may be constructed using a variety of different programming languages and development environments. Tools for computer-based model building – often developed with children in mind – include programming languages and development environments.

Where the use of educational software is concerned, there are many issues and challenges relating to its development. The perspectives of the educationalist and the computer specialist are traditionally different. When constructing software, the computer scientist is typically concerned with issues such as the usability of the software, requirements specification, and the choice of programming paradigm. The educationalist’s focus is on the quality of the learning activities that are supported, and the actual computer implementation is a secondary concern. In the field of educational software – where educationalists and computer specialists have a common interest – the primary concern is typically that the software is ‘as easy as possible’ to adapt for use in different learning contexts. The demand for adaptable educational software stems from the fact that learners often have different needs, abilities and approaches to learning. Learners’ different requirements arise dynamically in the learning context and teachers would ideally like to respond to individual situations as they occur. The particular features of the culture within which learning takes place may also require software to be adaptable [RB02]. The concept of being 'as easy as possible' to adapt is not well defined – we shall take this to mean that a non-computer specialist (such as a teacher) can adapt the software themselves, or that a computer specialist can adapt the software quickly with very little effort. We acknowledge that this requirement is particularly difficult to satisfy given the demands on teachers' time. There may also be a conflict between the expectations of the educationalist and the computer specialist where the qualities of the software are concerned. The focus in software development is on providing polished software products to users [Bey01]. In contrast, a teacher may appreciate the benefits of 'do-it-yourself' software that may lack the sophistication of a commercial product but allows a higher degree of ownership, engagement and adaptability. Such qualities were in evidence in the 'cottage' educational software industry of the 1980's [Ker92]. Given the current focus on object-oriented principles in software development, the favoured way of trying to develop such adaptable software is through object re-use. The merits of EM as an approach to the development of educational software are discussed and illustrated in detail in chapters 5 and 6.

The issues and challenges in using computers for learning through model building are considered in chapters 3 and 4. In such use of computers, we are not concerned with the incidental learning of programming knowledge that is a necessary part of the process, but rather the way in which characteristics of the model construction approach assist or hinder the learning of domain knowledge. In chapter 2 we concluded that the synergy between learning domain knowledge and constructing a computer model depends on the paradigm that is used in the model construction. For instance, in building a spreadsheet – and in EM – the modeller's attention is focused on the semantic relation β that is intimately linked to understanding domain knowledge.

The above discussion motivates the two central claims of this thesis:

1) **EM for computer-based model building**. In respect of learning through the construction of computer-based models, EM is in general more suitable than other approaches because model construction is more intimately linked to the development of domain understanding. This claim is developed in chapters 3 and 4.
2) **EM for the development and use of educational software**. In respect of learning through the use of computer-based models, if EM is used for the development of educational software (following the principles of software development outlined in previous theses [Nes97, Ras01, Won03]) then this software will have qualities that are well-suited to the educational needs of learners and teachers. This claim is discussed in detail in chapters 5 and 6.

In justifying the first claim, we first address the question of how learning and EM are connected (in sections 3.2 – 3.6), and then review learning through model building in relation to established educational theories (in chapter 4).

In section 3.2, we discuss what we mean by 'learning' and delineate the scope of the learning activities that we aspire to support in computer-based exploratory modelling. We informally describe some types of learning that we undertake in the world, including learning skills, learning about situations and learning about artefacts. We use these examples to motivate a broad framework on learning encompassing a variety of learning activities. We have termed this framework an Experiential Framework on Learning (EFL) because it reflects the way in which learning is rooted in our personal and private experience. Within the EFL, we can analyse how computer-based modelling tools can support a wide-ranging view of learning. From section 3.3 onwards, we introduce the principles underlying EM and describe the extent to which it supports the learning activities described in the EFL.

3.2 A perspective on learning

In this section, we discuss learning in detail to explore various types and applications of learning that motivate the learning framework we use in the remainder of the thesis. It is evident that learning on computers cannot replicate the enormous diversity of learning that can take place ‘in the world’ at the present time – but a major aspiration of using computers for learning is to apply computers in as wide a range as possible of learning situations. In education, critics of children’s use of computers for learning stress that the virtual nature of computer reality leads children to less rich and engaging learning experiences (see e.g. [Tal95]). This has been one justification for arguing that young children should not use computers in learning [Hea99, AFC00], and that educational systems based on personal engagement with the world and other people are more beneficial (see e.g. [Opp97], [Opp03], Waldorf education [Aep86], Alliance For Childhood [AFC00]). It cannot be denied, however, that the interactive nature of the computer offers advantages over building real-world artefacts, in that experimentation can often be more easily performed.

It would be impossible to give an authoritative view on exactly what learning is: major debates in psychology centre on how we learn, what knowledge is, and what are the best ways to learn [HO96]. It is the difficulty of giving an objective definition of learning and how it occurs that motivates us to discuss the perspective on learning that informs this thesis. To this end, we shall first describe some examples of learning in abstract terms, then complement this with longer discussions of the learning of skills, learning about artefacts and learning about situations. Our framework for learning, the EFL, is intended to provide a general setting within which diverse learning activities can be discussed.

The definition of the verb to ‘learn’ from the Oxford English Dictionary is:

‘gain knowledge of or skill in by study, experience, or being taught’ [OED97].

This definition is so broad as to encompass self-directed learning and teacher-directed learning. These two activities are representative of constructionism and instructionism respectively (see section 4.1). In this thesis, the term ‘learning’ is used in a broad sense to embrace any kind of activity that enables us to adapt our future behaviour. One particular difficulty with this broad definition is in identifying separate learning activities that may be taking place concurrently within one and the same situation. For instance, in constructing a computer model, I may be learning new insights about the model’s domain, peculiarities of the modelling tool and better ways of organising my model. A wide variety of different types of learning may also be concurrently represented:

i) **learning as equipping us to respond to questions of fact** – e.g. what is the capital of England?

ii) **learning to understand a concept** – e.g. what is taxation and how is it applied?

iii) **learning about social situations** – e.g. learning the roles and responsibilities of signalmen and drivers in the safe passage of trains through a tunnel.

iv) **learning a physical skill** – e.g. playing the piano, or learning to row.

v) **learning about a real-world artefact** – e.g. learning about the controls of a new watch.

The distinction between these different types of learning is manifest in the different ways in which we would assess whether learning has taken place. From one perspective, the knowledge that two times three is six is a matter of fact. One kind of activity that informs such knowledge is rote learning of multiplication tables. From another perspective, knowing that two times three is six entails knowing the meaning of ‘two’, ‘three’ and ‘times’ and being familiar with many concrete examples of how ‘times’ occurs in practical situations. Such illustrations show the subtlety of the distinctions between different notions of learning and the difficulty of expressing them formally. To address this subtlety, we shall introduce a framework within which to organise the activities associated with different types of learning.

To motivate this framework, we now describe some informal examples of learning. We shall do this with reference to contexts in which the learning process is complex, such as learning a new skill (e.g. a new sport or a musical instrument), learning about an artefact (e.g. a digital watch) or learning about a situation (e.g. being a restaurant manager). In the discussion that follows, it is useful to refer to a learning situation you have been in and reflect on the learning activities that you undertook. I will use two examples of learning skills: firstly that of learning how to row in a boat (an activity that I first attempted to learn two years ago); and secondly that of learning to play the piano (as we discussed in a previous paper [RB02]).

3.2.1 Learning skills, learning about artefacts and learning about situations

Watching an expert performing a skill, honed to near-perfection through innumerable learning experiences belies the difficulty of undertaking it for yourself. Any task can seem easy when performed by somebody who has been through an extensive learning process to reach their advanced standard. The first time you sit in a boat and try and row, or sit at a piano to learn how to play is a daunting experience. You have none of the necessary skills; you have acquired none of the language of the domain. Where conscious learning of a skill is involved, your primary source of knowledge is an initial idea of how you think the skill is executed derived from previous experience of observing others. Howell claims that learning a skill passes through a succession of four stages [How82]:

1) Unconscious incompetence – we don't know that we don't know how to do something.
2) Conscious incompetence – we know we want to do something but we don't know how to do it.

3) Conscious competence – we can do something but only by concentrating fully on it and by focusing on individual parts of the task.
4) Unconscious competence – we know how to do something and can do it automatically whilst concentrating on other things.

A fifth stage has been proposed by Pike [Pik89], namely conscious unconscious competence, which is taken to mean an ability to do a task without thinking about it, yet retain a level of awareness of how it is done that enables you to teach the skill. Performing a skill initially requires a commitment of time and energy to learn the 'basic' skills of the discipline, such as the rudiments of the rowing stroke or performing scales on a piano (cf. moving through conscious incompetence). Through our interactions we move from having to consciously think about each and every element of the stroke to a level where it is a natural, ingrained movement that we can perform without conscious thought, (cf. progressing from stage 1 to stage 4 of Howell's stages). This leaves our minds free to engage in more advanced thoughts, such as 'are we rowing at an appropriate speed and stroke rate to win this race', or 'have we started to play this piece of music at a tempo that is feasible for the most difficult passages'.

In learning, we gradually build up experience of important features of a situation and how they are dependent on each other. For example, the balance of a rowing boat depends on the positions of the oars in such a way that if the heights are the same then the boat is balanced and 'runs' along the water more smoothly. Understanding the 'feel' of the boat running requires experience and experimentation in order for it to become repeatable (cf. Howell's stage 3). Each individual rower (through personal experience) will learn to 'feel' a good stroke and be aware of relationships between the position of the hands on the oar and the feel of the blade in the water. This requires observation of factors such as 'is the blade in the water?', 'is it at the right angle?' and 'am I putting an appropriate amount of force on the oar for its position in the water?'. Over time, experienced oarsmen gain a comprehensive understanding of how their individual movements affect the run of the boat and the performance of the crew as a whole.

During the entire learning process the learner is engaged in non-verbal communication through the use of artefacts and physical hands-on demonstrations. The coach of a rowing crew will demonstrate the particular stroke pattern they are looking for and (on a land-based simulator) will physically control the oarsmen, isolating each part of the stroke to perfect it. The learning of skills often involves a coach who will demonstrate how to perform an aspect of the skill, communicating through physical manipulation, pictorial representations and the use of domain-oriented language.

The novice rower meets language that they have not encountered before that refers to either directly observable features of the skill or skill domain, or more complex culturally situated features of the environment that are meaningful only with reference to that skill. For example, in rowing, there are simple concepts such as 'bow side' and 'stroke side' that refer to sides of the boat that are accessible to a beginner. These are directly observable and have a definite meaning. There are then terms such as 'drive', 'recovery', 'quarter slide' and 'backstops' that are particular positions within the rowing stroke, or particular phases of the stroke. There may be some disagreement over the precise meaning of these terms, but their meaning is unambiguous enough to enable rowers with a modest amount of experience to communicate. However, other terms are imbued with meaning that is more difficult to directly apprehend. For example, the terminology of different oar pressures is a purely individual matter. A coach may ask for a particular training interval to be completed at 'half pressure' but it is almost certain that individual rowers (and the coach) will interpret this term in different ways. It is also likely that an individual's interpretation of such a term will change with their experience. As a rower gains confidence and can apply more pressure to the oar, the concept of half pressure will change.

The specialist language of a domain can also have exceptionally broad and rich cultural connotations. For example, in musical performance – as in rowing – there are different types of language that are appropriate at different competency levels and are directly correlated with the experience of the performer. Over and above this, the intelligent interpretation of music can draw upon diverse kinds of knowledge (e.g. of history and of musical forms) and experience (e.g. of emotions and of symbolic pattern recognition) as explained in [RB02]:

> 'For each level of attainment and genre of piano-playing, there is a pianistic competence and an appropriate level of sophistication in musical language (cf. "Play Middle C", "Play the harmonic scale of C sharp minor", "Play the octave passages in the coda of the Rondo in Beethoven's Waldstein sonata as glissandi"). It is significant that at its most sophisticated the language associated with a culture draws on such extensive experience and so many different sources of knowledge (e.g. in the above instance: music theory – *octave*; classical musical forms – *coda*, *Rondo*; musical history – *Beethoven, Waldstein*; and instrumental techniques – *glissando*) that it is only intelligible to the musical specialist.'

In summary, it is through extensive experience of gaining the necessary skills, identifying patterns of interaction and stimulus-response mechanisms and the acquisition and understanding of the relevant language of the domain that a learner progresses from a complete beginner to an expert in a domain.

Learning a skill is one aspect of learning about situations or artefacts. In learning to face a new situation or to use an unfamiliar artefact, learners are required to correlate the acquisition of new skills with the identification of important features in their context. Learning about artefacts need not commence with a user digesting formal instructions from a manual. The experimental psychologist John Carroll's theory of minimalism suggests that learning is more successful if learners are involved in hands-on tasks and not on reading obstructive instructional materials [Car90].

There is some evidence for Carroll's claim in the empirical observation of typical users of a new product [Nor98]. Exploratory interaction provides an initial understanding of how to use an artefact. We prefer to experiment, noting important observations and building up experience through interaction, without relying on objective prescriptions for how to use the artefact. We create mental models of the artefact under study that inform us throughout our learning [Joh83, Nor83]. This everyday, hands-on, empirical approach to learning contrasts with traditional educational approaches where problems and skills are mediated to the learner through the use of language, and particular emphasis is placed on logical and mathematical thinking as the most important aspect of intelligence [Gar93]. As the concrete examples described in this section illustrate, learning is much more than can be described through formal representable knowledge. Exploratory interaction is a key feature of complex learning situations. We now outline a framework for learning that is informed by the above discussion and that will be used in the remainder of the thesis.

3.2.2 An experiential framework for learning (EFL)

This section introduces an Experiential Framework for Learning (EFL) as a way of classifying learning activities on a spectrum between the private and the public domain. An earlier version of the EFL appeared in [Bey97], and has been adapted from a previous paper on educational technology [RB02] for this thesis. Figure 3.1 shows different categories of learning activity within the EFL. These categories range from concrete to formal learning and are concerned with issues that span empirical and theoretical knowledge. Activities towards the formal end have their foundation in experience-driven activities at the concrete end. This view is consistent with Noss and Hoyles's perspective on learning, as expressed in [NH96]:

> 'Although knowledge is constantly constructed and reconstructed through experience, this same experience also shapes and reforms a global and theoretical perspective'.

The purpose of the EFL is not to portray learning as a simple linear transition from private experience to public knowledge, but rather to express the way in which different learning activities depend upon each other. For instance, the learner can only progress to using symbolic representations meaningfully when they have a degree of experience gained through interaction in the domain. The interdependency between learning activities does not prescribe the learning pattern completely, but it imposes some loose constraints on the order in which they can occur. For instance, the focus of attention typically moves gradually from private experience to public knowledge as we learn about a domain.

private experience / empirical / concrete

interaction with artefacts: identification of persistent features and contexts
practical knowledge: correlations between artefacts, acquisition of skills
identification of dependencies and postulation of independent agency
identification of generic patterns of interaction and stimulus-response mechanisms
non-verbal communication through interaction in a common environment
directly situated uses of language
identification of common experience and objective knowledge
symbolic representations and formal languages: public conventions for interpretation

public knowledge / theoretical / formal

Figure 3.1 – An experiential framework for learning

As our examples of learning about rowing and piano-playing have illustrated, learning begins from private experience. Our preliminary interactions are informed by our previous experience. We begin to understand the persistent and important features of the domain and acquire the practical skills to manipulate them. Our interactions can lead us to understand the dependencies between our actions and events and understand how other agencies can affect the situation. With experience we come to understand that particular patterns of interaction are common and stable and we can communicate within the domain through non-verbal means. We are continually extending and refining our understanding of the situated language of the domain. Learning can eventually lead us to be able to establish the empirical basis for common experience and objective knowledge, which can in turn be representable as formal languages and have public conventions for interpretation.

In learning, there are identifiable ways in which we move from one category of learning activity to another within the EFL. Practising to develop a skill, experimenting to frame a theory or hypothesis and identifying new concepts in deriving new words are characteristic of moving from the empirical to the theoretical within the EFL. Practising to refine and debug skills, experimenting to test theories and hypotheses and devising situations in which to test the integrity of new vocabulary are characteristic of moving from the theoretical to the empirical within the EFL. These characteristic aspects of learning can be regarded as metaphorically 'moving down and up between levels' within the EFL in a way that may tend to stability. We may understand a concept and its application so thoroughly that exploratory interaction with it is unnecessary – but it is unnecessary precisely because we possess the experience of interaction with it that informs its use. When we are learning about a new concept, it then becomes important to support the learning activities that enable us to gain the broad base of experience required to interact with it in the fullest possible way.

In understanding the EFL, it is important to consider how it can be applied to support learning in practice.

The EFL is to be viewed as a generic template for learning. The specific character of learning activities in the EFL can be entirely different depending on the context in which learning takes place. Relevant considerations are: the subject domain (e.g. learning to row, to count, to write); the nature of the learning task (e.g. learning the concept of number, learning to use a calculator, learning times tables); the character of the learning environment (e.g. teacher-supported, self study); and the technology available (e.g. physical artefact, computer, virtual reality environment). There are nevertheless general patterns according to which learning activities are organised, as has been explained above.

'Moving up and down between levels within the EFL in a way that tends to stability' can be interpreted as negotiation of the semantic relation β (cf. section 2.2.2). Negotiation can be associated with genuine creation and novel discovery (as e.g. in Newton's discovery of the refraction of light [CW95]). In this context, the learning activities in the EFL are emergent rather than previously understood. Negotiation can also be associated with coming to a common understanding through personal experiment and communication (as e.g. in learning to generate a spectrum using a prism). In this context, the learning activities in the EFL are familiar to the knowledgeable observer.

'Moving from the empirical towards the theoretical within the EFL' is a process of *abstraction*. Abstraction is concerned with formalising learning. 'Moving from the theoretical towards the empirical within the EFL' is a process of *concretisation*. Concretisation is concerned with gaining familiarity with underpinning activities and experience. For instance, this concretisation may take the form of testing abstract relationships or refining primitive skills.

Concretisation is one aspect of elaboration of the semantic relation β (cf. section 2.2.2). It is associated with enriching the specific experiences that inform a particular learning objective. For instance, in learning to row, diagnosing the difficulties in achieving a smooth stroke may involve working on particular basic elements of the stroke in isolation. A further aspect of elaboration is associated with setting a learning activity in a richer domain context. For instance, a novice may be introduced to rowing on a static machine, and progress via rowing a machine on slides to rowing in a boat on the water. In this example, the learning activity changes from one context to another – the skills become more complex (e.g. balance becomes important) and the terminology is necessarily embellished (e.g. concepts regarding the oar become relevant). In elaboration of this nature, the mapping from the EFL to specific learning activities is hard to formalise as the learning activities in themselves evolve.

As stated earlier, the 'computers for learning' agenda must aspire to support the widest possible range of different types of learning. This aspiration cannot be fully realised with existing computer technology: computer-supported interaction and visualisation is limited in comparison with activity in the real world (cf. the accounts of learning to row and play the piano in section 3.2.1). Developments in computing are already introducing richer interaction metaphors that potentially offer support to a wider range of learning activities (see e.g. [RJM$^+$98]). The principles of EM to be introduced and discussed in this thesis are conceived as potentially general enough to embrace computer-related technology as it may develop in the future (cf. the discussion of ubiquitous computing in [Won03]).

In considering EM for learning, we aspire to provide computer support for the whole range of learning activities described in the EFL. In the world, learning often begins from tentative hypotheses, a type of interaction we aspire to support in EM model construction. A computer-based approach to model construction that reflects the EFL must be able to support fluid movement between many different types of learning activities. In the remainder of the chapter, we discuss how the principles of EM model construction (section 3.3, 3.4) match up with the EFL (section 3.6). We shall illustrate EM principles with reference to the construction of a restaurant manager model (section 3.5).

3.3 Learning by experience

Within the EFL, the most primitive learning activities originate from private experience. In this section, we expand on the role of experience in learning by considering Kolb's theory of experiential learning and relating the EFL to an underlying philosophical attitude of Radical Empiricism first promoted by William James [Jam96].

3.3.1 Experiential learning

The dictionary definition of learning (as cited in section 3.2) is: to 'gain knowledge of or skill in by study, experience, or being taught' [OED97]. Many scholars have emphasised that experience is fundamental to learning. The seminal American educationalist, John Dewey, made the claim that learning has to be grounded in experience [Dew38]. Jean Piaget, in research on children's learning, proposed that children have different stages of learning, from sensori-motor, through concrete learning to abstract learning [Bra78]. Piaget stressed the important experience gained through interaction between the learner and their environment. Kurt Lewin's research in organisational behaviour also emphasised the importance of experience in learning, particularly stressing the active nature of the learner [Lew51]. The ideas of Dewey, Piaget and Lewin underpin David Kolb's well-known experiential learning cycle [Kol84].

Kolb's experiential learning cycle is based on an iterative cycle of four activities, namely concrete experience, reflective observation, abstract conceptualisation and active experimentation. Experience initiates the cycle; as Kolb says [Kol84]:

> 'Immediate personal experience is the focal point for learning, giving life, texture, and subjective personal meaning to abstract concepts and at the same time providing a concrete, publicly shared reference point for testing the implications and validity of ideas created during the learning process'.

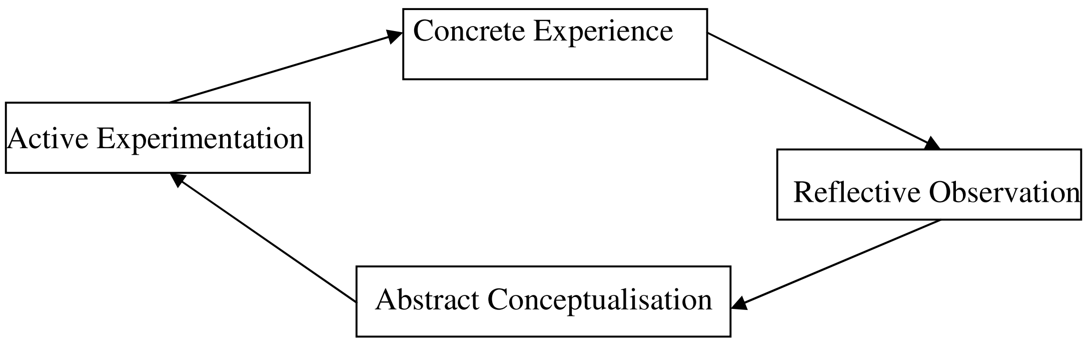


Figure 3.2 – Kolb's experiential learning cycle

In Kolb's cycle (see Figure 3.2), reflection on our personal experience gives rise to new concepts or ideas. These ideas in turn stimulate experiments that typically lead to new experience or new perspectives on our previous experience.

The four activities in Kolb's experiential learning cycle can refer either to private or public activities. Atherton's interpretation of Kolb's cycle makes this private/public distinction [Ath02]. Atherton classes concrete experience and reflective observation as internal activities, and abstract conceptualisation and active experimentation as external activities. This classification is appropriate in certain circumstances. Concrete experience and reflective observation are surely private activities. The nature of abstract conceptualisation and active experimentation depends on the learning context. Abstract conceptualisation may or may not involve concepts that belong to the public domain. Active experimentation may or may not be publicly interpretable or accessible. However, Atherton's interpretation makes it apparent that experiential learning can involve both private and public learning activities.

Kolb's experiential learning cycle is reflected in the learning activities in the EFL. As we pointed out in section 3.2.2, the learning activities associated with the EFL are not necessarily addressed in a rigid sequence; learners will move between activities in a fluid fashion. Concrete experience and reflective observation are closely related to activities at the private end of the EFL, whilst – depending on context – active experimentation and abstract conceptualisation are more closely related to the learning activities at the public end of the EFL.

As Kolb's cycle illustrates, learning can consist of many different types of activities, which draw on our experience, and change our experience. In many ways, learning can be considered to be reclassification of experience. We can understand this with reference to different categories to which experience may belong. Some of our experience is stable and revisitable ("I know how to do this now and I know I can do it again"), whereas parts might be unstable and tentative ("I have done that, but I am not sure how I did it or if I could do it again"). With reference to Howell's learning stages [How82] (cf. section 3.2.1), the process by which our experience migrates from being tentative to reliable is mirrored in the move from stage 2 to stage 4. In terms of Kolb's cycle, the reclassification of experience is mediated by a succession of activities that sees us experiment, reflect and form new ideas. All of our experience is open to reclassification in the light of new insights or of new circumstances to be taken into account. In learning, we always have the possibility of being surprised ("I didn't know that that could happen") and this can lead to new classification for our experience. When we are entirely sure that our experience of some phenomena is reliable, it is in some circumstances appropriate to explore the possibility of 'sharing the experience'. This notion of sharing experience depends on observing the social interaction that underpins inter-subjectivity. Part of the communication difficulty in establishing inter-subjectivity stems from the fact that it is hard for you to relate my account of my experience with your newly forming experience which you do not yet understand ("You are telling me this is true and I am not sure if I believe you until I try it for myself"). Our personal experience of a phenomenon in the social context can be said to be public knowledge when we can share it and others agree that it is true ("I know this and you agree with me").

3.3.2 Radical Empiricism

In this section, we outline William James's 'philosophic attitude' of Radical Empiricism, as first described in his "Essays in Radical Empiricism", first published in 1912 [Jam96]. According to Naur [Nau95], James was the pioneer of the experiential view of knowledge. Radical Empiricism has been considered in connection with EM in previous papers [Bey97, Bey99, Bey03]. The relevance of James's thinking to emphasising the concrete above the abstract in education can be seen in the following quote [Jam96]:

> '… the one thing that is sure is the inadequacy of the extant school solutions. The dissatisfaction with these seems due for the most part to a feeling that they are too abstract and academic. Life is confused and superabundant, and

> what the younger generation appears to crave is more of the temperament of life in its philosophy, even though it were at some cost of logical rigor and of formal purity'.

Radical Empiricism draws on James's descriptive account of human mental activity in his "Principles of Psychology" [Jam90]. As emphasised by Naur (cf. the entry for **stream of thought** in [Nau01), James's philosophic attitude is distinguished by his readiness to talk about such issues as thought, feeling, association and knowledge by placing them in clear relation to *every person's experience of his or her thoughts and feelings*. Central to James's thinking is the capacity of the mind to make associations between experiences in the stream of thought. James identifies the roots of knowledge in how 'one experience knows another' in the stream of thought [Jam96]. He characterises the relationship between two experiences, one of which knows the other, as being given in experience and not rationally apprehended with reference to some explicit preconceived account.

Beynon [Bey97, Bey03] has made the connection between William James's outlook on experience and the philosophical issues raised by focusing on Cantwell-Smith's semantic relation β as it relates to spreadsheet design and use (cf. section 2.2.3). The key observation is that the evolution of a spreadsheet in design and use is similar in character to that of states of mind in the stream of thought. Changes to the spreadsheet are not to be interpreted as specifying a new spreadsheet, but as reflecting some change in our experience of its referent.

By generalising spreadsheet principles (cf. chapter 2), EM aims to account for the semantic relation β between a computer-based model and its referent in terms of James's notion of one experience knowing another. Experience of the computer model stimulates us to understand it in terms of our experiences of corresponding interaction with its referent. This correspondence leads to a conflation of the external and computer-based experiences, resembling what Turner has characterised as *blending* [Tur96]. The negotiative process of blending gives rise to new insights and directions in which to take the computer-based model, much in the spirit of Levi-

Strauss's bricolage [Lev68], a theme we return to in chapter 4. Negotiational blending of this nature is the vehicle for learning through the reclassification of experience discussed in the previous section. This is consistent with James's view that [Jam96]:

> 'subjectivity and objectivity are affairs not of what an experience is aboriginally made of, but of its classification. Classifications depend on our temporary purposes'.

EM endorses a view of knowledge that is consonant with James's idea that knowledge is rooted in personal experience. The EFL can be viewed as mapping out the activities that account for public knowledge with reference to private experience (cf. [Bey99]). It also represents a particular perspective on learning as 'gaining knowledge'. In his essay 'The experience of activity' [Jam96] James advocates Radical Empiricism as an appropriate philosophical stance from which:

> '… to try and solve the concrete questions of where effectuation in this world is located, of which things are the true causal agents there, and of what the more remote effects consist'.

EM could be seen as bringing computer support to this agenda. As will be discussed in detail in section 3.4.3, negotiational blending in EM traces the progression of model building through the elements identified in James's quote above: identifying agency, attributing state-changes to those agents, and interpreting agent interaction in global state-based terms (cf. [Bey03]). As will be discussed in the next section, this is reflected in the way that, in EM, the model and the modeller's construal of the referent evolve together.

3.4 Principles of Empirical Modelling

In this section, we describe how the private experience that forms the basis of learning can be utilised in computer-based model construction at the empirical end of the EFL. Our understanding when we begin to construct a model is tentative and requires computer support that does not commit us to build on experience that is at present unstable. As Russ remarks [Rus97]:

> 'where there is no adequate theory we may wish to build models simply in order to aid our understanding; any specific purpose may be unknown, or provisional, and it is then only an impediment to make early commitments to certain properties we wish to preserve in the model'.

EM models that support the agenda above can be regarded as *construals* in the sense of Gooding [Goo90], rather than as conventional programs with preconceived functionality. This shift in perspective stems from focusing on *state-as-experienced* as being prior to *behaviour-as-abstracted* (cf. the distinction between EM and conventional programs described in chapter 4). Furthermore, this shift in perspective requires a different set of key concepts that underlie the modelling process; in EM these are *observables*, *dependency* and *agency*.

3.4.1 Construals

Real-world learning can often involve the making of models to supplement current understanding of a situation and give the opportunity to experimentally comprehend how changes affect it. For example: an engineer creates a prototype to gain fundamental knowledge about an artefact before the construction of the final system; a financial analyst constructs a spreadsheet to understand and explore potential changes to a situation. David Gooding introduces the term *construal* to refer to a concrete artefact that is used to embody evolving understanding of a phenomenon [Goo90]. He developed the idea of a construal from studying the experimental practices that Michael Faraday used in investigating electro-magnetic phenomena. Faraday used physical objects to convey his evolving understanding of electromagnetism. Gooding [Goo90, p22] characterises construals as:

> '… a means of interpreting unfamiliar experience and communicating one's trial interpretations. Construals are practical, situational and often concrete. They belong to the pre-verbal context of ostensive practices. … A construal

> cannot be grasped independently of the exploratory behaviour that produces it or the ostensive practices whereby an observer tries to convey it'.

Gooding emphasises the close connection between the evolving understanding of a referent and the exploratory interactions that are used in developing the construal.

In EM, we observe an external referent, and concurrently build a computer model that metaphorically exhibits similar patterns of observables, dependency and agency [BS98]. 'What-if?' style modelling enables the interrogation of personal construals in testing beliefs about a referent. If experiments return expected results then a modeller's construal is reinforced (cf. stabilising our experience). Unexpected results in experiments serve to change a modeller's construal, because either the construal is mistaken or the referent exhibits some previously unknown characteristic (cf. new insights). A construal in EM is a voyage of discovery, a creative process quite unlike conventional modelling where the emphasis is on the representation of well-understood behaviours. The key features of a construal are that (cf. [Bey99]):

i) it is empirically established. It is informed by past experience and subject to modification in the light of future experience.

ii) it is experimentally mediated. Our experience with it guides its evolution.

iii) the choice of agents is pragmatic (what is deemed to be an agent may be shaped by the context for our investigation of the system); it only accounts for changes of state in the system to a limited degree (the future states of the system are not circumscribed).

A construal must be testable beyond the limits of the expected range of interactions with it [BRS99]. In specifying a conventional program, the modeller has to preconceive its behaviour thereby restricting the exploratory interactions that can be undertaken. In contrast, EM model construction privileges experimental interaction. Interactions can take account of the changing real-world situation; can probe unknown aspects of a referent; and may even be nonsensical in the world. Beynon has described these interactions, which reflect Situation, Ignorance and Nonsense (SIN) respectively, as exhibiting the SIN modelling principle [Bey01]. He claims that this principle is not well supported in classical computer programming, which requires the abstraction of well-understood problems. The SIN principle can also be seen in spreadsheets: the spreadsheet refers to an external situation; there is incomplete understanding; and we can test our understanding by making experimental changes that may be nonsensical.

Building construals using EM is closely associated with learning. The process of model construction is a private learning experience and our construal represents our evolving understanding of a situation [Bey97]. Experiments performed during the early stages of modelling an artefact are tentative and exploratory; they are a reflection of our provisional construal. Modelling dependencies is a prominent aspect of the early stages of EM. Rungrattanaubol [Run02] highlights the significance of modelling of this nature when knowledge is pre-articulate, informal, situational and takes account of personal viewpoints. Such modelling is intimately concerned with state-as-experienced rather than behaviour-as-abstracted, as discussed in the following section.

3.4.2 State-as-experienced and behaviour-as-abstracted

Formal computer science encourages the view that the only significant semantics of a computer program resides in the abstract patterns of behaviour and interaction that it supports. This is consistent with what Brödner has characterised as the 'closed world' paradigm:

> ' …, 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" .' [Brö95]

To support this 'closed world' paradigm, the key requirement is to be able to develop programs which support planned user interactions and preconceived interpretations.

The users of such a program have no choice but to adapt themselves to the features of the program and its interaction style. In contexts where a domain is well understood, this viewpoint is satisfactory – both the designer and the user conceive of the program in a similar way. In situations where knowledge is uncertain, the programmer faces problems because they cannot conceive the abstract behaviour of the referent in its entirety. Beynon [Bey99] suggests that classical computer science has limitations that stem from concentrating on a 'closed world' paradigm.

EM is attempting to supply principles that can support what Brödner identifies as a counterposition – the 'open development' paradigm:

> '…, the "open development" paradigm, does not deny the fundamental human ability to form explicit, conceptual, and propositional knowledge, but it 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.' [Brö95]

The emphasis on practical experience and on growing knowledge in 'open development' requires an approach to modelling that enables unconstrained interaction with a computer model. This cannot be achieved if explicit behaviours are the primary concern of the modelling process. EM emphasises modelling that is state-based, where the term 'state' is to be understood as referring to 'state-as-experienced' rather than abstract computational state. The term 'state-as-experienced' necessarily refers to the experience of an individual, which may not be objective because our interpretation of the world may well be different from that of another person. State-as-experienced may confound us by changing in unpredictable and uncircumscribed ways, for example through events occurring that are beyond our expectations. State-transitions in EM are constrained only by the modeller's imagination. An open development approach requires a close correlation between the state of the computer model and the state of its external referent that reflects Cantwell-Smith's semantic relation β [Smi97]. Open development in EM has close connections with spreadsheet development that was discussed in section 2.2. For example, a spreadsheet user always interprets the spreadsheet with reference to its current state and in relation to the external situational state to which it refers. Construction of a spreadsheet goes hand in hand with its use; changes can be made on-the-fly as insights occur. Although spreadsheets can be used in a rigid predefined way, circumscribed use occurs only after significant evolutionary development.

The distinction between the concept of state in EM and in traditional computer-based modelling is depicted in Figure 3.3.

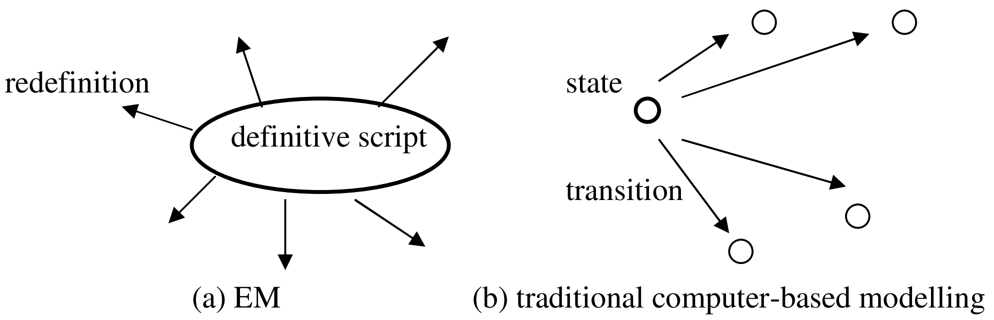


Figure 3.3 – State-based and Behavioural-based views on development processes

The concept of state in Figure 3.3(b) relies upon a circumscription of system behaviour that is characteristic of the closed world paradigm. Each circle depicts an abstract computational state and each edge a valid state-transition consistent with the abstract system behaviour.

In contrast, Figure 3.3(a) depicts the concept of state as it applies to modelling in an open development paradigm. The characteristics of such a state are not defined with reference to preconceived neighbouring states or an abstract behaviour. In keeping with the notion of state-as-experienced, the semantics of the state is implicitly defined by exploring plausible atomic state-transitions in an experimental fashion. It is for this reason that the state is represented by a spreadsheet-like definitive script and possible redefinitions (cf. section 2.4.1), rather than by a configuration of abstract states.

The identification of the plausible atomic state-transitions in Figure 3.3(a) depends on the modeller's viewpoint (who's making the observations?) and notion of indivisibility (when are observations deemed to be simultaneous?) [BC95]. In EM, these issues – which are crucial to changing our perspective on developing models from behaviour-as-abstracted to state-as-experienced – are addressed by introducing special concepts. In the following section we outline the three key concepts that underpin this shift in perspective, namely: *observables*; *dependency*; and *agency*.

3.4.3 Observation, dependency and agency

In this section we describe the EM concepts of observables, dependency and agency.

An **observable** is a feature of the situation or domain that we are modelling to which we can attach an identity (cf. a cell in a spreadsheet). The main requirement of an observable is that it has a current value or status (cf. a value in a spreadsheet). An observable can refer to a physical entity, an abstract entity or a conceptual entity. Examples of observables could be the mass of an object, the status of my bank account, whether I own a car, and the quality of the television reception. Observables can be of different kinds. These include: events (my train has arrived); quantities that are directly or indirectly measurable (the amount of petrol in the tank); booleans (my tank is half-full of petrol). What we deem to be an observable will in general depend on the context and the observer. An observable is understood to be something that an agent can apprehend instantly but such apprehension may be dependent on experience (e.g. this knot is a reef knot).

Observables in the domain are represented in EM by variables in a definitive script. The meaning that the modeller attaches to a variable in the script is negotiated in relation to the referent for the model [Nes97]. The plausible redefinitions for such a variable are those that have counterparts in interaction with the referent (cf. the way in which states acquire their semantics in Figure 3.3(a)). As discussed in [Run02], this gives definitive variables in the script the characteristic qualities of observables rather than abstract programming variables.

A **dependency** is a relationship between observables that expresses how they are indivisibly linked in change. A change to the value of an observable will cause changes to the observables that are dependent on it. For instance: the amount of tax payable is dependent on the current tax rate, personal income and tax-free allowances; the quality of the television reception depends on the weather conditions, distance from transmitter and strength of the signal. Unlike constraints between observables, which express persistent relationships between values in a closed-world, dependencies express the modeller's current expectation about how a change to one variable will affect the value of another. In open development, such expectations are subject to change. In EM, dependencies between observables are represented by definitions in a definitive script.

Observables and dependencies together are used to represent the current state of an EM model. The concept of agency is used to express state-transitions.

An **agent** is an entity in the domain being modelled that is perceived as capable of initiating state-change. The agents identified by the modeller will depend upon their construal and the purpose of the modelling. On this basis, the notion of agency encompasses such diverse possibilities as the manager of a restaurant, the battery of a digital watch or the modeller in the role of experimenter.

In an EM model, an agent is conceived as an entity that is capable of changing the values of observables or dependencies. Such an agent is itself typically associated with a set of observables. In practical EM using TkEden, the actions of agents are represented by redefinitions that may be manual (performed by the modeller via the input window) or automated (through the use of triggered actions as described in section 2.4.3).

In EM, there is a special purpose notation called LSD for describing observables, dependency and agency that can help in classifying agents and their capabilities. The LSD notation was initially motivated by research into the CCITT Specification and Description Language, and offers a way of describing systems at an abstract level [Bey86]. The LSD notation can be used in an Empirical Modelling framework:

i) to guide our evolving understanding of a situation by elaborating the observable elements of a situation and how they are viewed and controlled by agents.

ii) as a documentation tool to be used after model construction to guide others to the important features of the model.

Constructing an LSD account involves the identification of agents and the classification of observables with respect to agents. Each observable belongs to categories that reflect its status with respect to an agent. Observables can be classified into the following categories:

State observables – these are the observables that are associated with the presence of the agent. If the agent ceased to exist, such an observable would disappear. Examples include: the speed of a car (bound to the car) or the number of tables in a restaurant (bound to the restaurant).

Oracle observables – these are the observables to which an agent can respond in the current state of the environment. An agent does not necessarily have privileges to change the values of such observables. Examples include: the colour of a traffic light (an oracle to a car driver) or the time at which a customer phones the restaurant (an oracle to the manager). In certain contexts, a very significant oracle is the ‘absolute time’, which all agents can be presumed to know but none has the privilege to change [Bey86].

Handle observables – these are the observables that an agent has the privilege to conditionally change. Examples include: the speed of the car (a handle for the driver) or the table allocated to a customer in a restaurant (a handle for the manager). An agent does not necessarily need to have an observable as an oracle in order to change it.

Observables that are related through dependencies in the view of the agent are termed *derivates*. This term reflects the fact that the value of an observable can be derived from the values of other observables. An example of a derivate is: whether there is a table free in a restaurant depends on the occupancy of the tables.

An observable can be classified in different ways with respect to different agents (e.g. I can see the speed you are driving but have no control over it) and may appear in many different categories for a single agent (e.g. to you, the speed of your car is both a handle and an oracle).

The privileges that the agent has to make changes to observables are recorded as a set of guarded sequences of redefinitions. This set of redefinitions is referred to as the agent's *protocol*. Examples of protocols include: if a customer has requested a booking, the manager will allocate an appropriate table; and when a customer leaves the manager will take payment.

The protocols of the agents in an LSD account specify possible state changes that the system of agents can perform. They are not in general construed as circumscribing the behaviour of the system for a variety of reasons that are discussed in detail in [BNO$^+$90]. For instance, an LSD account does not take matters of synchronisation, speed and reliability of response into consideration. A more typical use of an LSD account is in documenting our evolving understanding of a situation so that for instance other privileges could be added to the protocol of an agent.

In developing an EM model, our perspective on agency within the domain evolves with our construal. The way that we perceive agency is related to our experience of the situation and our current purpose for studying it. In [Bey97], Beynon has identified three viewpoints on agency that are representative of how the modeller's perception of agency may develop during model building in EM:

View 1: **The modeller identifies primitive entities as agents**. In this view, every observable or object-like set of observables is potentially an agent, as is the external observer. Any entity that is a cause, cue or trigger for some action on the part of another agent is identified as being an agent.
View 2: **The modeller attributes state-changes to agents**. In this view, the modeller construes specific observables and objects as responsible for particular state changes. This corresponds to understanding how state changes can be correlated with the presence (and action) of a particular agent (e.g. a flag moves only if there is a wind).
View 3: **The modeller identifies a system behaviour**. In this view, the modeller understands the system so comprehensively that it is possible to circumscribe its behaviour; agents act through reliable and objective stimulus-response patterns and have no capacity for surprise. This corresponds to virtual agency in the closed world.

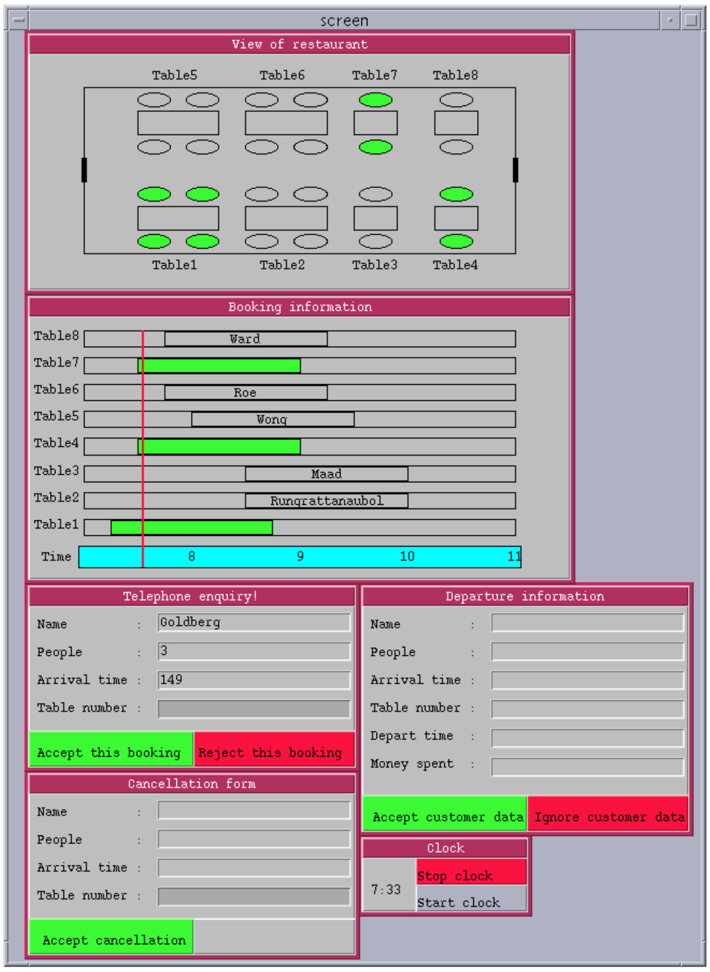
Each of the above viewpoints can be correlated with different learning activities within the EFL. In View 1, we are concerned with identifying entities that can potentially cause state change in a domain, whereas in View 3 we have identified a specific systematic behaviour within the domain and are interested in whether we have identified all the relevant agents together with their actions. Moving from a View 1 to a View 3 perspective is like moving from a broad unfocused view of a domain to a narrow specialised application within the domain. Making this transition requires correlating agents with state changes as described in View 2. EM is concerned with facilitating the transition between a View 1 and View 3 perspective by providing ways of describing agents and actions in an exploratory way so as to embrace the learning that occurs in this process of correlation. In learning about a specific system, the View 1 – View 2 – View 3 transition is enabled in EM by a development environment in which agency in all three viewpoints can co-exist. This means that, even as we aspire to understand a system completely enough to represent it from a View 3 perspective, we can always override automatic operation to take advantage of new insights or experiments. One of the central problems in developing learning environments is that we need to accommodate incomplete and imperfect understanding. This demands support for modelling that is broader than mere specialisation to a View 3 perspective on a system. This is particularly problematic for conventional system development, which focuses on a View 3 perspective.

The model building discussed in this thesis makes use of the TkEden modelling tool (see section 2.4.2). Because this tool does not give full support for modelling agency, we shall not normally make explicit use of object-like abstractions and LSD accounts of situations. As discussed in detail in [Run02, Chapter 3], modelling with TkEden can be construed as taking place within an 'abstract definitive modelling' framework in which there is more comprehensive conceptual support for agency and entities. The way in which agency is implicitly represented in modelling with TkEden will be illustrated in the following section.

3.5 Modelling restaurant management

To illustrate the concepts of model construction introduced in this chapter we describe a case study of a restaurant management model. The restaurant manager model was originally developed in order to investigate decisions that a manager might make regarding the allocation of customers to tables in a restaurant. It is a model that could be used in order to inform decision support within a business context [RRR00]. Decisions on table allocations are an important and imprecise task for a restaurant manager. There is a need to accommodate the particular needs of each client, but both the requirements (numbers of tables, available time) and the resources available (tables, waiting staff, chefs) are changing dynamically. The customers (or potential customers) are agents who can act in unforeseen ways and so make the job of allocating tables more than a merely quantitative exercise in profit maximisation.

Figure 3.4 shows the model in use. The model contains two main display windows, a clock window and three forms which are used to generate customer events. The top window shows a plan view of a fictional restaurant with a total of eight tables of two different sizes. Below this is a window that contains a representation of a booking timetable for the restaurant for an evening. The booking timetable has a record of the current bookings for the evening and can be interpreted to establish when particular tables will be occupied or empty. The vertical red line shows the current time in the evening as displayed on the clock window in the bottom right. The other three windows are used to generate customer events. Potential customers can walk in off the street and request a booking immediately or may telephone the restaurant requiring a booking for a time in the future. It is also possible for customers with bookings to ring up and cancel their booking. Furthermore, when a customer departs the restaurant manager may record customer information for later analysis.



Telephone enquiry!

Name :	Goldberg
People :	3
Arrival time :	149
Table number :	

Departure information

Name :	
People :	
Arrival time :	
Table number :	
Depart time :	
Money spent :	

Cancellation form

Name :	
People :	
Arrival time :	
Table number :	

Figure 3.4 – The restaurant manager model

My motivation in constructing the restaurant manager model was to gain an appreciation of the issues involved in restaurant management. For example, during the construction of the model, I – as the modeller – initially allocated tables. Identifying the factors involved in human judgment was important in understanding how to add an automatic table allocation routine (cf. the changing viewpoints of agency discussed in the previous section). This involved understanding complex issues such as the relationship between unused capacity and unused time (cf. section 3.5.2).

The restaurant model had no prior specification, and in the model building I had no specific features in mind – these emerged during the construction of the model as a direct result of interaction with the partial model. This is possible in EM because of the emphasis placed on the representation of state rather than the recognition of abstract patterns of behaviour (see section 3.4.2). Initial model construction focused on identifying the important qualities of a restaurant and building on these basic ideas to shape the more advanced concepts such as automation of table booking (see section 3.5.2). In constructing a conventional program to perform restaurant management we would primarily consider the important actions that the restaurant manager has to perform and set out to automate these (cf. software development based on the analysis of use cases in the spirit of Jacobson [Jac92]). In the construction of the EM model, there is no circumscription of the possible uses to which the model can be put and these remain open throughout the development. For example, we might wish to add waiters to the restaurant model and animate them on the restaurant window, or consider the effect that changing the number of waiters has on customers and fellow staff. In a conventional program, where use is preconceived, it can be much more difficult to alter the interpretation of the partially constructed program flexibly since it is already optimised to serve a particular function.

3.5.1 Experiential learning and the restaurant manager model

The initial construction of the restaurant manager model was guided by my experience of visiting restaurants. Each visit to a restaurant has built up the background knowledge of how restaurants look and function on which I drew in building the model. Some aspects of my experience of restaurants are stable and represent objective knowledge. For instance, customers are allocated to a table, eat, and then depart after a period of time. Other knowledge acquired through experience such as my conception of the thought processes behind how tables are allocated to customers take the form of subjective hypotheses. The initial phase of model construction is concerned with building a computer model that embodies my objective and stable knowledge of restaurants.

Interaction with the model, through experiments and observations, is the source of new insights into restaurant management. New experience is acquired as a direct result of intimate engagement with the restaurant model. As Kolb's cycle indicates, new experience results from reflection, conceptualisation and experimentation. This cycle works repeatedly in our ongoing and increasing experience of restaurant management. Experience matures from tentative hypotheses about aspects of restaurant management to stable experience that we can reliably revisit and reproduce as requested. My stable experience – for instance that there are tables to which customers are allocated – is embodied as part of the restaurant model that I am assured is valid. In summary, stable experience of actual restaurants underpins our initial construction of the model, and our emerging experience gained from the computer-based restaurant model guides its future construction.

3.5.2 Empirical Modelling principles and the restaurant manager model

The EM restaurant model is a construal of restaurant management. Throughout the model building, the computer-based model is always provisional and reflects my current understanding of restaurant management. It is always open to new insight and new exploration based on my emerging understanding of the situation. In developing the restaurant construal, there were three broad phases in the model construction: I initially concentrated on the representation of state; then investigated sensible behaviours through experimental manual redefinitions; and finally automated appropriate behaviours.

There are a number of observables that are important in the restaurant model. The most primitive observables concern the physical characteristics of the restaurant itself. These include: the room layout; the number of tables; and the number of people that can sit at each table. As illustrated in Figure 3.4, the model includes a visualisation of the restaurant with different size tables and a table-booking sheet for an evening. Other relevant observables include: the time of each booking; the number of bookings; the number of people in each booking; and customer preferences (such as smoking or window seats). A possible classification of these observables with respect to the restaurant manager is shown in the LSD account in Listing 3.1.

The modeller initially defines customer events using the forms on the interface (see Figure 3.4). Through creating sample reservations and cancellations, a preliminary appreciation of the booking experience and knowledge that a manager possesses can be gained. Choosing a sensible table for a customer is not a trivial matter. There are many issues to consider, including customer satisfaction, staff morale and past occupancy patterns. Each of these factors will consciously (or subconsciously) influence the decision made by the manager. In justifying his or her decision, a manager may appeal to their tacit knowledge, which may be exceedingly difficult to articulate ("I feel that's the right decision"). Experience gained through the experimental generation of enquiries can lead to insights into the manager's job, even if their task is viewed simply in terms of maximising profit without considering other resources such as numbers of waiters and chefs.

```
agent restaurant_manager {

  oracle
           table1_position, table2_position, ...
           table1_occupancy, ...
           future_bookings
           telephone_ringing
           customer_preferences
           size_of_customer_party
           restaurant_full

  handle
           table1_occupancy, ...
           future_bookings
           table1_position, ...

  derivate
           restaurant_full is (table1_occupancy>0) && ...

  protocol
           table1_occupancy==0
                 → table1_occupancy = size_of_customer_party,
                 ...
           size_of_customer_party > maxtablesize
                 →    table1/2 = table1+table2;
                      table1/2_occupancy  =  size_of_customer_party,
                      ...
 ...
}
```

Listing 3.1 – Part of an LSD specification for the restaurant manager model

The eventual construction of an automated routine to simulate a restaurant manager in allocating tables to potential customers entails deeper insights into the subtle nature of the job. There are obvious considerations, such as ensuring that there is enough time to fit in a particular booking on a table and that there are enough seats for the number of people in a party. The results of applying these simple conditions leave a set of possible tables on which to place a party. As the model is developed, more sophisticated questions emerge. For instance, what is the relationship between unused capacity (such as results from seating a party of 2 at a table for 4) and unused time (as when a table has no people on it for a period of time)? The answer to this question could guide the manager in deciding whether to accept a party of 2 on an empty table for 4. Other factors relevant to the above question may include: the type of restaurant; the type of cuisine; the night of the week; and the geographical area. These factors influence the manager but are very difficult to quantify. The decision to refuse a booking for a couple when a table for four is free might indicate that the restaurant manager believes that a larger party is likely to arrive soon enough to be more profitable. The elements of the model are linked through dependencies so that we can easily perform 'what-if' style queries to see the effects of changing features such as the length of booking slots or the opening hours of the restaurant. Queries of this nature can be made 'on-the-fly', and can support other interactions, such as changing the layout of the restaurant to simulate bookings from large groups.

From a personal viewpoint, experimentation with the model led to personal insights that have enabled me to understand some of the difficulties of a restaurant manager's job, although it would of course be impossible to fully replace a manager's decision-making by an automated routine. The construction of a faithful model of a restaurant, where human agents can play roles as they would in the real world, allows the judgments and insights of the individual to be expressed through interaction. The process of model construction in EM is intimately associated with domain learning.

3.6 Chapter Summary: Empirical Modelling and the EFL

In the final section of this chapter, we discuss connections between EM and the EFL and argue that they are intimately linked. In section 3.2.2, we proposed the EFL as a generic learning framework that can be interpreted with reference to any subject domain. We shall now argue that 'EM supports the EFL', in the sense that it supports learning activities from across the whole of the EFL and enables fluid movement between them.

EM endorses a view of knowledge similar to that proposed in Radical Empiricism (cf. section 3.3). In this view, all knowledge is rooted in the primitive notion of ‘one experience knows another’ in personal experience. In EM, the building of an artefact offers experience that ‘knows’ experience of its referent. In [Bey99], Beynon considers the way in which EM allows objective and theoretical knowledge to be traced to its experiential roots. In [Bey03], Beynon discusses the similarities between the perspectives on knowing represented in EM and in Radical Empiricism. The EFL is also motivated by the idea that learning is rooted in private experience and that abstract activities are grounded in our sound understanding of concrete examples.

EM emphasises interaction and experimentation with artefacts in order to generate and test our construal of a referent (see section 3.4.1). During construction, the modeller is always able to interact with the evolving artefact. As discussed in section 3.4.2, our early interactions are primarily concerned with exploring the current state of the artefact. This is analogous to the experimental changes a spreadsheet modeller might make in order to work out sensible future states. In EM model construction, the emphasis is on interactive exploration of plausible state transitions to increase our understanding: the modeller is exploring the ‘space of sense’ (cf. [Bey01] and section 3.4.2). It is through the occasional verifiably mistaken experiment that appropriate stimulus-response mechanisms and generic patterns of interaction are identified [Bey01]. The account of the construction of a clock model in Appendix D gives a practical illustration of how model construction in EM can be experimental, reflecting our evolving understanding of the referent and the differing purposes to which we may want to put the model. With reference to the EFL, activities of this nature are concerned with the identification of dependencies, generic patterns of interaction and stimulus-response mechanisms at the concrete end.

In section 3.4.3, we discussed how agency in EM can be used to automate reliable behaviours of an artefact and hence move towards the abstract end of the EFL. Because the modeller has the discretion to perform experimental interactions at all times, they can always step back from an abstract behaviour into a concrete situation. The activities described in the previous two sentences are forms of abstraction and concretisation respectively (cf. section 3.2.2).

The above discussion of EM and the EFL leads to two conclusions:

i) EM is well suited to support learning activities at the concrete end of the EFL because of its primary emphasis on experimental interactions with artefacts and the representation of state-as-experienced.

ii) EM can support the fluid movement between learning activities in the EFL due to its ability to integrate the abstract and the concrete within a single modelling environment.

In later sections of the thesis, we will discuss the connections between EM and the EFL from different perspectives. In section 4.6, we consider the links between EM and the EFL with respect to the educational theory of constructionism. In section 6.5, we consider the links between EM and the EFL with reference to sizable EM case studies.

Chapter 4 – Constructionism and computers for learning

4.0 Overview of the Chapter

In this chapter, we introduce the theories of constructionism and instructionism and discuss them with reference to the EFL. All approaches to domain learning that involve programming satisfy Seymour Papert's basic definition of constructionism. It is impossible for a learner to construct computer models passively; there has to be a degree of engagement with the task. In this thesis, we take a broad view of constructionism that embraces bricolage and situated learning. We observe that constructionism can be broadly identified with activities at the concrete end of the EFL, and that instructionism can be broadly identified with activities at the formal end of the EFL. We argue that conventional programming is typically concerned with learning activities that are found at the formal end of the EFL, and hence that it is not well suited for supporting domain learning through constructionist model building. In contrast, EM – which gives support to learning activities at the concrete end of the EFL – is better placed than conventional programming to support domain learning through constructionist model building. The construction and use of a digital watch model is used to illustrate the ideas presented in this chapter.

4.1 Constructionism and instructionism

To motivate the theory of constructionism defined by Seymour Papert [Pap93] we shall first briefly review the theories of objectivism and constructivism.

4.1.1 Objectivism, Cognitivism and Constructivism

Psychology is full of debates about how learning occurs [HO96]. Theories of learning and the nature of understanding have a profound influence on the design of instructional materials, teaching philosophies and learning. The epistemological debate between objectivist and constructivist positions can be traced back to issues of ontology debated by the Greeks [Sae67].

Objectivism contends that there is a given reality which learners are expected to reproduce in their minds. There is no personal reconstruction of reality from an individual's viewpoint [Jon91]. The work of Skinner is characterised in the theory of *behaviourism* [Ski74]. Skinner developed a theory from experiments with animals placed in boxes. When an animal discovered the secret to escaping from a box the likelihood of it repeating that behaviour in the future was increased. Skinner proposed a theory of human learning called *operant conditioning*, which claims that learning can be totally characterised in terms of changes in overt behaviour [Dri00]. Learning occurs in response to environmental stimuli where particular stimulus-response patterns are reinforced through reward, and are thereby more likely to occur in the future. Skinner was motivated by his experiments to introduce the behaviourist approach to learning, in which the mind is treated as a black box. Behaviourism contends that the internal processes of the mind are not important in studying learning; it is sufficient to concentrate on the overt behaviour of the learner.

Cognitivism came to prominence in the 1950's [BGA56]. The assumption behind cognitivism is that the brain acts as an information processor. It takes input from the world and processes this to produce overt behaviours. Cognitivism places importance on the internal processes of the mind but is still objective in its approach – it assumes that there is a given reality that the mind processes and a learner's role is to passively acquire knowledge transmitted by an instructor [May99]. Newell and Simon's work on human problem solving was a major research project that took a cognitivist approach; it viewed problem solving as an information processing system [NS72].

Constructivism is founded on Kantian beliefs: it claims that the knower constructs reality based upon their mental activity [Jon91]:

> '… What the mind produces are mental models that explain to the knower what he or she has perceived … We all conceive of the external reality somewhat differently, based on our unique set of experiences with the world and our beliefs about them.' [Jon91]

The origins of constructivism may be found in John Dewey's view of learning as a constant reorganisation or reconstructing of experience [Dew16]. The research of Piaget [Bra78] and Vygotsky [Vyg62] provides the cognitive development theories that underpin the constructivist position. In their view, all meaning is rooted in personal interpretation of the world. The educational researcher Cooper [Coo93] compares constructivism with cognitivism and behaviourism in the following terms:

> 'The constructivist… sees reality as determined by the experiences of the knower. The move from behaviourism through cognitivism to constructivism represents shifts in emphasis away from an external view to an internal view. To the behaviourist, the internal processing is of no interest; to the cognitivist, the internal processing is only of importance to the extent to which it explains how external reality is understood. In contrast, the constructivist view the mind as a builder of symbols – the tools used to represent the knower's reality. External phenomena are meaningless except as the mind perceives them… Constructivists view reality as personally constructed, and state that personal experiences determine reality, not the other way around' [Coo93, p16].

Piaget stated that constructivism requires the learner to actively build their own knowledge structures, based on their own mental activity [Bra78]. Learning builds on existing knowledge, and each learner creates an individual representation of the subject being studied. It is inevitable that our initial constructions are naive and contain misconceptions. Personal constructions become more realistic as our experience grows. This approach is more natural for learners because it directly addresses the process of knowledge construction and is sensitive to mistakes in the learning process [Ben01]. The construction of viable mental models is also an important part of the learning process, since these are the containers within which the knowledge can be organised [Ben01] (cf. the discussion of construals in section 3.4.1). In 1991, Merrill collated ideas from a variety of sources to identify the assumptions of constructivism [Mer91]. These assumptions are that:

i) knowledge is constructed from experience;
ii) learning is an active process;
iii) learning is collaborative with meaning negotiated from multiple perspectives;
iv) learning should be situated in realistic settings;
v) testing should be integrated with the task, not a separate activity;
vi) interpretation of reality is personal – there is no shared reality.

These assumptions are consistent with Kolb's model of experiential learning, as described in section 3.3.1.

Vygotsky stresses the importance of social and cultural contexts within the learning environment in supporting a discovery-based learning model [Vyg62]. His *Zone of Proximal Development* (ZPD) is an important concept with regard to learning because it defines the potential of a learner, in contrast with traditional tests that give only an accurate measure of current performance. In learning situations, the ZPD is an area within which a learner can interact given suitable assistance from other people and supporting technologies.

In the next section, we consider how the objectivist and constructivist movements have influenced the design of educational technology.

4.1.2 Instructionism and Constructionism

The dominant educational approach in the 20th century assumed an objectivist viewpoint, and was termed *instructionism*. In an instructionist approach, knowledge is transmitted to passive learners, for example through the use of lecturing and whole class teaching. Friere has described this as a 'banking method of education', where a student's mind is an empty receptacle to be filled up with knowledge, in much the same way that you might top up your bank account [Fri70]. The instructionist approach is epitomised by John Carroll's notion of the Nurnberg funnel [Car90], a mythological device that allows teachers to pour facts directly into a learner's head. Instructionism is characterised by Jonassen in the following terms:

> 'Learning consists of assimilating objective reality. The role of education is to help students learn about the real world. The goal of designers or teachers is to interpret events for them. Learners are told about the world and are expected to replicate its content and structure in their thinking'. [Jon92]

Skinner's operant conditioning, when applied to instructional design, organises material into graded problems to which the student must correctly respond. This model was adopted in early software for computer-based instruction [Dri00]. The software followed the pattern of traditional teaching, where teachers reward students who do well with praise – a form of extrinsic motivation [Cov98]. Students are conditioned to achieve good results by linking the stimuli of good test results with the response of good marks and praise. The influence of the behaviourist approach can be seen in the proliferation of computer-assisted instruction (CAI) and 'drill-and-kill' educational software. This type of software aims to teach students by presenting a topic together with a selection of questions that they have to answer. Feedback is then given on their answers. CAI is simply an extension of the student-teacher transmission model of learning that has come in for strong criticism from many authors [Fri70, Ill71, Pap80, Pos92, Tal95, Opp97].

Constructionism has its basis in the theory of constructivism [Pap80]. In addition to the active building of knowledge structures, Papert claims that construction that takes place in the head often happens especially felicitously when it is supported by construction of a more public sort 'in the world' [PH91]. By 'in the world', Papert means that the public nature of a constructed artefact enables discussion, examination, probing and admiration [Pap93]. Although the theory of constructionism was originally introduced with reference to children, it is relevant across all age groups, as Papert demonstrated by describing examples from his own learning processes [Pap80]. Knowles's andragogical model of adult learning is related to constructionism and places emphasis on self-direction, positive use of previous experience and internal motivation [Kno70, Kno90]. The important connection between andragogy and constructionism is the emphasis placed on the active role of the learner. The constructionist emphasis in learning is on the process, not on the product [KR96]. Jonassen supports the idea that constructionist educational software emphasises the process of knowledge construction as being more important than the resulting product:

> 'if meaning is determined by the mental processes of the individual, and these processes are grounded in perception and grow out of experience, then those are the things we should evaluate – not the extant behaviour or the product of that behaviour' [Jon92].

Ostwald endorses this emphasis on process over product in learning in his study of knowledge construction in software development [Ost96]. In his view, the building of artefacts enables us to learn through their construction, through experimentation (to see how they work) and through modification (to make them 'better'). He claims that the construction of an artefact and its understanding proceed in tandem:

> 'Experiential artefacts allow us to interact with the world. They provide information that enables us to interpret a situation through our perceptions. The danger is that they don't provide us with the knowledge – they provide us with information that is tacitly interpreted. When what we perceive is different from what we tacitly expect, a breakdown occurs, and the cause of this breakdown is brought to the surface where it can be interpreted and knowledge can be constructed' [Ost96].

The personal construction of artefacts is quite a different kind of activity from the assimilation of material designed by others. A constructionist perspective is well aligned with the learning activities at the concrete, empirical end of the EFL and an instructionist perspective is well aligned with the activities at the formal, theoretical end (cf. Figure 4.1).

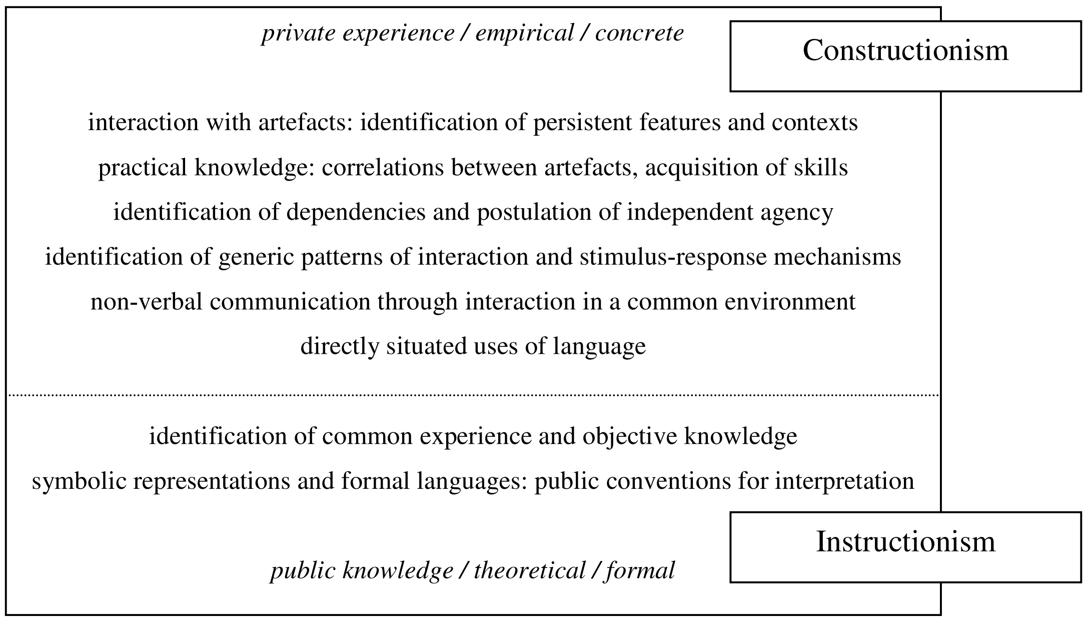

private experience / empirical / concrete
Constructionism
interaction with artefacts: identification of persistent features and contexts
practical knowledge: correlations between artefacts, acquisition of skills
identification of dependencies and postulation of independent agency
identification of generic patterns of interaction and stimulus-response mechanisms
non-verbal communication through interaction in a common environment
directly situated uses of language
identification of common experience and objective knowledge
symbolic representations and formal languages: public conventions for interpretation
Instructionism
public knowledge / theoretical / formal

Figure 4.1 – Relating constructionism and instructionism to the EFL

Constructionism is closely linked with the learning activities at the concrete end of the EFL. Active knowledge construction plays a prominent part in interaction with artefacts and practical knowledge. These activities rely on our personal interpretation of the world, based on our private experience. At the formal end of the EFL, an important learning activity is the identification of common experience and objective knowledge. In an instructionist approach this is not an end-goal, but a prerequisite – the goal of education is to transmit objective knowledge. This is reflected in the quote from Jonassen cited above:

> 'Learners are told about the world and are expected to replicate its content and structure in their thinking' [Jon92].

Ever since Papert defined the theory of constructionism, and used the Logo programming language as its vehicle for delivery, constructionism has been closely associated with Logo. There are good reasons to expect computers to play a major role in the future of constructionism in the classroom. On the other hand, it is not obvious that the use of computers is allowing constructionist principles to be fully expressed in educational practice. For instance, not all educationalists are convinced that programming in Logo promotes domain learning [Sol93, Ste94, Tal95]. To address this concern, we need a better understanding of the relationship between computers and constructionism. To this end, we consider a broader view of constructionism that encompasses model-building approaches such as *bricolage* and *situated learning*, which are not necessarily computer-based.

Bricolage [Lev68] is a style of construction that places emphasis on concrete experimentation and negotiation with artefacts. Bricolage is situated in the realm of primitive knowledge that concerns the acquisition of practical knowledge and the identification of persistent features and contexts. The style and manner of construction is important as well as the finished product. Situated learning [Lav88] advocates learning in the context of interaction in real-world situations, and is a prominent feature of a constructionist approach to learning. This can be seen from Papert's view of the public nature of the artefact constructed in learning and the role it plays in the 'real world' [Pap93]. It is very difficult to create our own model of reality without interacting in the real-world situation we are seeking to understand. We discuss the notions of bricolage and situated learning in more detail in sections 4.2 and 4.3 respectively. This supplies the context for the subsequent discussion of concept mapping (section 4.4), traditional computer programming (section 4.5) and EM (section 4.6) in support of constructionist approaches to learning.

4.2 Bricolage

The concept of bricolage originates in the work of Claude Levi-Strauss, an anthropologist who studied people working in primitive societies [Lev68]. Levi-Strauss was interested in contrasting approaches to task solving in what he characterised as 'primitive' and 'western' societies. In western societies, the most advanced form of thought is generally believed to be abstract and scientific. Jean Piaget's well-known theory of the stages of learning identifies abstract thinking as evidence of maturity [Bra78]. The importance attached to abstract thought can be seen in the style of instruction and evaluation in the traditional Western schooling system. Levi-Strauss argued that primitive societies view ‘concrete’ thinking processes as more important than abstract thought [Lev68]. He defined bricolage as a way of performing work that emphasises human involvement and engagement where subjective interaction with the artefact guides solving of a task. A person who is involved in bricolage style activity is called a bricoleur. This French word is best translated as ‘a handyman’; it emphasises a working style that takes advantage of whatever tools are at hand to perform tasks for which these tools were not specifically designed.

We can illustrate bricolage by considering an example of a situation in which it is used. Imagine that we are going to build a chair. One approach would be to design the chair and create a plan specifying how the chair should be built before production begins. The end result is a chair that matches our original plans which has been shaped entirely away from the situation in which the construction takes place. There are advantages to this ‘planning’ approach; we can be confident that the plan will realise a functional chair. However, if requirements for the chair change, or new insights become available during construction then this approach cannot take advantage of them. The planning approach to chair building has parallels in the ‘waterfall’ stereotype for software development [Boe76] where the knowledge-gathering phase freezes requirements, which are then rigidly implemented according to the specification. In contrast, the bricolage approach emphasises minimal forward planning and continual negotiation with the referent throughout the process of construction. Levi-Strauss characterises this negotiation in the following terms [Lev68, p18]:

> ‘Consider the bricoleur at work and excited by his project. His first practical step is retrospective. He has to turn back to an already existent set made up of tools and materials, to consider or reconsider what it contains and engage in a sort of dialogue with it and, before choosing between them, to index the possible answers which the whole set can offer to his problem’.

With reference to chair building, the process of construction is characteristic of a craftsman at work – changes are made to the current prototype chair, and construction is guided by the current state of the chair:

> ‘bricolage involves an informal subjective interaction between a craftworker and the artefact he/she is creating that more closely resembles discovery than organised construction. The model-building activity has an experimental and creative quality: if it is successful, the character of the artefact itself changes in the mind of the discoverer as it develops – it is continuously being newly conceived and reinterpreted in stimulating ways’ [RB02].

Turkle and Papert have taken up the idea of bricolage and applied it in different fields as a way of validating individual approaches to problem solving [TP91]. Even in mathematics and science, there is problem solving activity that is not centrally concerned with the manipulation of formal symbols. Mathematicians and scientists stumble across discoveries through concrete model-building activities resembling bricolage, and only later do they refine these into scientifically acceptable formal abstractions. In this connection, Turkle and Papert [TP91] stress the importance of the computer as a tool that has ‘revalued concrete thought’. They claim that the computer can be used in a way that privileges thinking with concrete artefacts rather than abstract thought and suggest that:

> ‘the diversity of approaches to programming suggests that equal access to even the most basic elements of computation requires accepting the validity of multiple ways of knowing and thinking’.

Turkle and Papert refer to this ‘validity of multiple ways of knowing and thinking’ as an *epistemological pluralism* [TP91].

Although Turkle and Papert recognise that classically computer science promotes a structured planning approach, they argue that the distinction between planners and bricoleurs is manifest in the process of construction, not in the end result [TP91]. Papert identified two styles of programming that he called ‘hard-edged’ and ‘smoky’ and observed that to move from a hard-edged to a smoky style requires moving from an abstract formal approach to a concrete bricolage approach [Pap93]. Turkle and Papert call the users of these two styles 'planners' and 'bricoleurs' respectively, and claim that:

> 'observation[s] of programmers at work calls into question deeply entrenched assumptions about the classification and value of different ways of knowing. It provides examples of the validity and power of concrete thinking in situations that are traditionally assumed to demand the abstract' [TP91].

The importance of concrete thinking in programming has been endorsed over the subsequent decade through the emergence, and growing popularity, of programming paradigms that emphasise non-traditional development cycles (e.g. eXtreme Programming [Bec00] and Rapid Application Development [Mar92]). Planners value hierarchy and abstraction; bricoleurs prefer negotiation and concrete experiments.

Many differences can be identified between the bricoleur's and planner's approaches to programming. Ownership of a program is one important difference. Planners typically want to be able to ignore the detail of individual components in their program and treat each component as a black box. Bricoleurs want the internal workings of their model to be exposed because they are personally involved with their program and want to maintain their engagement with it. If we consider the learning style of bricoleurs and planners, we find another distinction:

> '…the bricoleurs are happy to get to know a new object by interacting with it, learning about it through its behaviour the way you would learn about a person, while the planners usually find this intolerable. Their more analytic approach demands knowing how the program works with a kind of assurance that can only come from transparent understanding, from dissection and demonstration' [TP91, p173].

The above discussion shows that there are fundamental differences between bricolage and planning approaches. Turkle and Papert claim that an epistemological pluralism is a necessary condition for a computer culture that encompasses every individual [TP91]. This view is endorsed in psychology by Gardner's work on multiple intelligences [Gar93], work on different learning styles [DD93] and approaches to problem solving [Pol57]. It is difficult for current computer programming cultures to accept an epistemological pluralism, because this:

> 'requires calling into question, not simply computational practices, but dominant models of intellectual development and the rarely challenged assumption that rules and logic are the highest form of reason' [TP91, p185].

Turkle and Papert propose to achieve epistemological pluralism through the development of computer programming languages that embrace both planning and bricolage styles. However, their critics claim that what Turkle and Papert identify as bricolage in computer programming is not a case of 'trial-and-error vs planning' but of '*aimless* trial-and-error vs planning' [YB01]. For instance, Ben-Ari is concerned that the development of explicit mental models must take place in tandem with trial-and-error, or learning will be hindered. In respect of programming, Ben-Ari claims that:

> 'premature attempts to write programs lead to bricolage and delay the development of viable models … There is nothing wrong with experimentation and bricolage-style debugging, as long as it supplements, rather than supplants, planning and formal methods' [Ben01].

Although Ben-Ari believes that we all practise some bricolage thinking, he does not see it as a substitute for the conventional programming discipline:

> 'The manifestation of bricolage in computer science is endless debugging: try it and see what happens. While we all practice a certain amount of bricolage and while concrete thinking can be especially helpful – if not essential – for students in introductory courses, bricolage is not an effective methodology for professional programming, nor an effective epistemology for dealing with the massive amount of detailed knowledge (that) must be constructed and organised in levels of abstraction (cf. object-oriented programming). The normative planning style that we call software engineering must eventually be learned and practiced' [Ben01].

In due course, we shall argue for a radically different view of the place of bricolage in computer programming from that represented by Ben-Ari. We attribute the failure of current programming and software engineering practices to support bricolage fully to their lack of maturity and contend that – by adopting different practices – it is possible to integrate the construction of computer-based artefacts and mental models. The possibility of better computer support for bricolage notwithstanding, we endorse Ben-Ari's claim that planning has an essential role in large scale software development. However, in considering the wider agenda of 'learning through creating computer models' – where process is more important than product – bricolage is profitable if it leads to a more valuable learning experience on the part of the student. In this context, what we wish to recognise as bricolage in conventional programming is the province of experts – who shape their programs skilfully in response to emerging requirements, rather than that of novices – who hack their program in an undirected manner. Without better support for bricolage, learners who are not expert programmers are forced to write programs in a style that may not be appropriate for them (cf. [TP91, Gar93]). With reference to Ben-Ari's quotes above, we see bricolage as an important aid in learning through model-construction, and in the prototyping – rather than the production – of a final product.

A further criticism of bricolage in computer programming has been made by Steve Talbott. He claims that – although an approach to programming may be conceived in isolation from the computer as bricolage – at the level of implementation, no matter how the student may think about the problem, it has to be coded in an abstract algorithmic method:

> 'While it may be legitimate to speak of the hard-edged and smoky *effects* the programmer aims at, the programming itself – which is the child's immediate activity – possesses a fundamental character that remains the same regardless of the style of the effects. The programmer may start with an interest in some aspect of the world, but the act of programming *forces* him to begin filtering that interest through a mesh of almost pure abstraction. To draw a figure with Logo, the child must derive a step-by-step procedure (algorithm) by which he can construct the desired result.' [Tal95, p157].

It would be hard to refute Talbott's claim that all computer programming involves some aspects of learning (typically abstract in character) that are not related to domain learning. For instance, in the use of Logo, a child requires some rudimentary knowledge of abstract computational ideas such as parameters and procedures. However, as Nardi has argued, a programming activity such as spreadsheet construction is more closely linked to domain learning than conventional programming [Nar93]. More generally, computer model-building that enables the modeller to focus primarily on the semantic relation β between the model and the real-world (cf. section 2.2.2) can alleviate the emphasis on abstraction. Current computer programming practice arguably obscures the possibility of a bricolage-based approach that does not force the learner to 'filter their interest through a mesh of almost pure abstraction'. As will be discussed in section 4.6, we believe that there is scope for alternative practices that give more access to a concrete, experiential learning style and move away from computer programming as an abstract preconceived activity.

Who is making it?	**Bricoleur**	**Planner**
What is being made?	Concrete artefact	Abstract program
Usage of tools	Uses whatever tools are already available	Uses standard tools with preconceived modes of use
Type of thought	Concrete	Abstract
Level of preplanning	Minimal	Entirely preplanned
Priorities	Negotiation, Engagement	Hierarchy, Abstraction
Knowledge Construction	Through open-ended interaction	Through analysis
Most suitable for	Non-programmers	Software engineers
Most similar software development style	RAD, XP	Conventional software development
Scale of application	Suitable for prototyping and small scale production	Suitable for mass production and large systems
Cognitive implications	Delays the construction of viable mental models	Demands mental models as a prerequisite

Table 4.1 – Differences between bricoleurs and planners, as identified in [TP91, Ben01]

The differences between the approaches to computer programming of the bricoleur and the planner that have been discussed in this section lie in the process and not the product. These are summarised in Table 4.1.

4.3 Situated learning

The proponents of situated learning claim that, for meaningful learning to take place, a learner must be placed within a realistic cultural and situational context. They also argue that the abstraction of problems from their real-world origins does not remove the complexity of the problem – it removes their essence [Lav88, Gog96]. In this section, we outline what situated learning is and how it relates to bricolage and planning approaches to model construction.

Situated learning is linked with John Dewey's claim that learning develops from experience and through social interaction [Dew38]. Learning in a situation is promoted in the idea of *cognitive apprenticeship*:

> ‘Cognitive apprenticeship supports learning in a domain by enabling students to acquire, develop and use cognitive tools in authentic domain activity’ [BCD89].

Apprentices, in trades such as mechanics and medicine, learn through authentic hands-on activities. Apprenticeship highlights how learning is an active process that is context-dependent, situated and cultural [BCD89]. Apprenticeship skills often arise out of what Lave and Wenger term *legitimate peripheral participation* [LW91]. Learners come to understand a skill by initially watching others perform the activity and gradually they take on some aspects of the role. Learning occurs through hands-on interaction. The full role is learnt legitimately through peripheral participation. This approach is in evidence in job shadowing for new employees.

Problems that are encountered in apprenticeship situations are different in character from problems met in abstract school-like settings. This is illustrated by Table 4.2, which has been extracted from Table 1 in [BCD89]. Table 4.2 can be interpreted as showing how problem solving in a situation (i.e. by practitioners) differs from problem solving in an abstract setting (i.e. by students).

	Students	Practitioners
Reasoning with	Laws	Casual models
Acting on	Symbols	Conceptual situations
Resolving	Well-defined problems	Ill-defined problems
Producing	Fixed meaning and immutable concepts	Negotiable meaning and socially constructed understanding

Table 4.2 – Comparing problem solving in situation and in abstract settings (adapted from Table 1 in [BCD89])

In the spirit of learning through cognitive apprenticeship, many researchers have called for a revolution in the school environment, where problems set for students are often devoid of real-world relevance (see e.g. [Dew38, Fri70, Ill71, Pap80, Pos92, Tal95, Opp97]). In [Res87], Lauren Resnick compares learning in a school environment with what she terms 'everyday learning', namely learning that takes place out in the world rather than in a classroom setting. She reasons that school learning often consists of individuals thinking abstractly about ways of solving problems where the emphasis is on the manipulation of formal symbols and the generation of general routines for solving classes of problems. In contrast, everyday learning often involves solving particular concrete problems where the structure of the task or the tools available can guide the solution to a problem, as in bricolage

[Lev68]. The differences between school and everyday learning are summarised in Table 4.3.

	In school	Everyday learning
Type of learning	Primarily mental ‘THINKING’	Tool manipulation ‘DOING’
Personnel	Individualised	Shared cognition
Applicability	General	Situation-specific
Objects of Thought	Concentrated on manipulating symbols	Contextualised reasoning

Table 4.3 – Comparing school learning and everyday learning [Res87].

CAI provides support for delivering and assessing traditional school problems. In CAI, the computer is used as a rigid device for asking students questions and eliciting responses from them, as in an instructionist approach. Providing support for Resnick’s everyday learning requires understanding the part situational elements play in the learning process and their implications on the design of learning environments. Jean Lave promotes the idea of situated learning, claiming that the situation within which a problem is posed is not incidental to its successful solution, but often provides the enabling factor through which learners can solve the problem [Lav88]. Lave gave examples of adult shoppers and tailors who could perform mathematical calculations in a real-world situation, but were unable to solve the same problems posed as abstract mathematical questions.

The psychologist, John Anderson has raised two concerns about situated learning approaches. He argues firstly that abstract knowledge can be transferred between tasks; and secondly that studying parts of an activity in isolation before considering them in combination can be more effective than instruction in complex, social environments [ARS96]. Anderson also observes that learning that is situated in the world is not necessarily to be preferred. In many respects, learning to solve a particular instance of a problem in a specific real-world context is limited in comparison with understanding a general abstract solution to that particular class of problems. However, the knowledge that leads to comprehension of the general is often gained from the experience of interacting with particular concrete problems.

Anderson's critique of situated learning motivates domain learning techniques that make it possible to combine situational and abstract approaches to learning. With reference to the EFL, such domain learning techniques need to support a fluid migration between concrete and formal learning activities.

Having introduced our broad perspective on constructionism, we now consider the extent to which this is supported by three domain learning techniques: concept mapping (section 4.4), conventional programming (section 4.5) and EM (4.6).

4.4 Concept mapping for domain learning

In this section, we consider concept mapping and discuss how this technique of representing emerging domain knowledge is related to the EFL. We shall argue that concept mapping is particularly closely connected with private and experiential learning activities in the EFL. We shall also argue that concept maps cannot support learning activities across the whole of the EFL, particularly the integration of experiential and formal learning activities.

4.4.1 Reviewing Concept Maps

Concept maps were introduced by Joseph Novak, an educational psyhcologist at Cornell University in the 1960's. Concept maps draw on the psychologist Ausubel's idea that the most important factor in learning is what the learner already knows [Aus68]. Concept maps are an example of a cognitive tool: a mental or computational device that can support, guide or extend the thinking process of its user [KJM92]. Cognitive tools support a constructionist approach because they actively engage learners in organising their knowledge to reflect their comprehension and conception of a domain [GKL$^+$01].

Concept maps are a way of representing knowledge in a visual graph structure, as seen in the concept map of this chapter in Figure 4.2. Nodes in a graph represent concepts. Connections between nodes define relationships between concepts. Novak states that concept mapping comprises three important elements: concepts; propositions; and learning [NG84]. A *concept* in the context of a concept map is 'a perceived regularity, designated by a label'. Concepts can be well understood or only partially understood; the concept map does not require knowledge to be complete or formalised. A *proposition* in the context of a concept map is defined as 'a link between concepts'. Propositions can be labelled with arrows and, if appropriate, annotated with a description of the link that should be concise, as complete as possible, and understandable to another person. A set of concepts linked by propositions constitutes a *concept map*. Novak's third element of concept mapping, namely *learning*, is 'the active construction of new propositions'. Creating a concept map is an active learning process – the actual process of constructing it furthers our knowledge and affects the structure of the map itself.

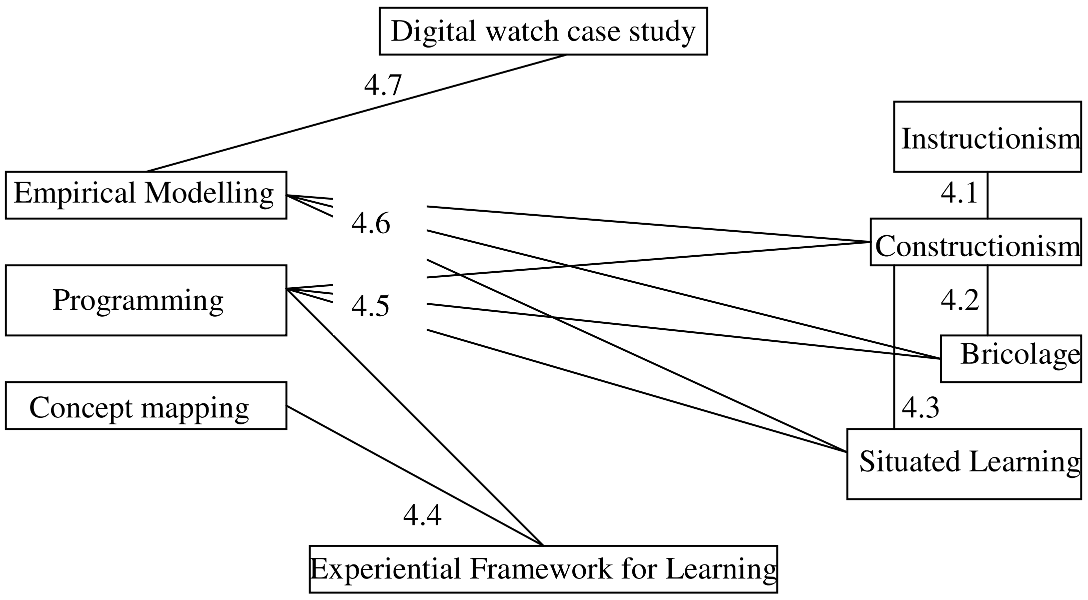


Figure 4.2 – An example concept map of this chapter

Mind maps, developed at the British Broadcasting Corporation (BBC) by Tony Buzan is an idea related to concept maps. They consist of one central word or concept, around which you draw the 5 to 10 main ideas that relate to that word [BB95]. The essential difference between mind maps and concept maps is that a mind map has one central concept but a concept map may contain several.

Learning through linking together concepts is endorsed by Marvin Minsky [Min86]:

> 'The secret of what anything means to us depends on how we've connected it to all the other things we know. That's why it's almost always wrong to seek the 'real meaning' of anything. A thing with just one meaning has scarcely any meaning at all'.

Uri Wilenski has also argued that establishing connections between concepts is a powerful learning objective. He describes a concept as being 'concrete' if it is well connected to other concepts, we have multiple representations of it, and we know how to interact with it in many modalities [Wil93]. In Wilenski's view, a concept becomes concrete through connecting it to other concepts, through many modes of interaction and through engaging in activities followed by reflection (a process he calls 'concretion'). Wilenski's use of the term 'concrete' differs from our use of the term in this thesis where it refers to *existing in a material form; real* [OED97], as opposed to being abstract. His notion of 'concretion' closely resembles the notion of the elaboration of a concept introduced in section 2.2.2.

The general nature of concept mapping means that it is widely applicable. Concept maps can be used, for example:

i) to see connections between current ideas. This is helpful in establishing our current state of knowledge.
ii) to connect new ideas to knowledge that we already possess. This is helpful in organising and understanding the place of new ideas. The assimilation of new knowledge is associated with adding new concepts and connections between concepts.

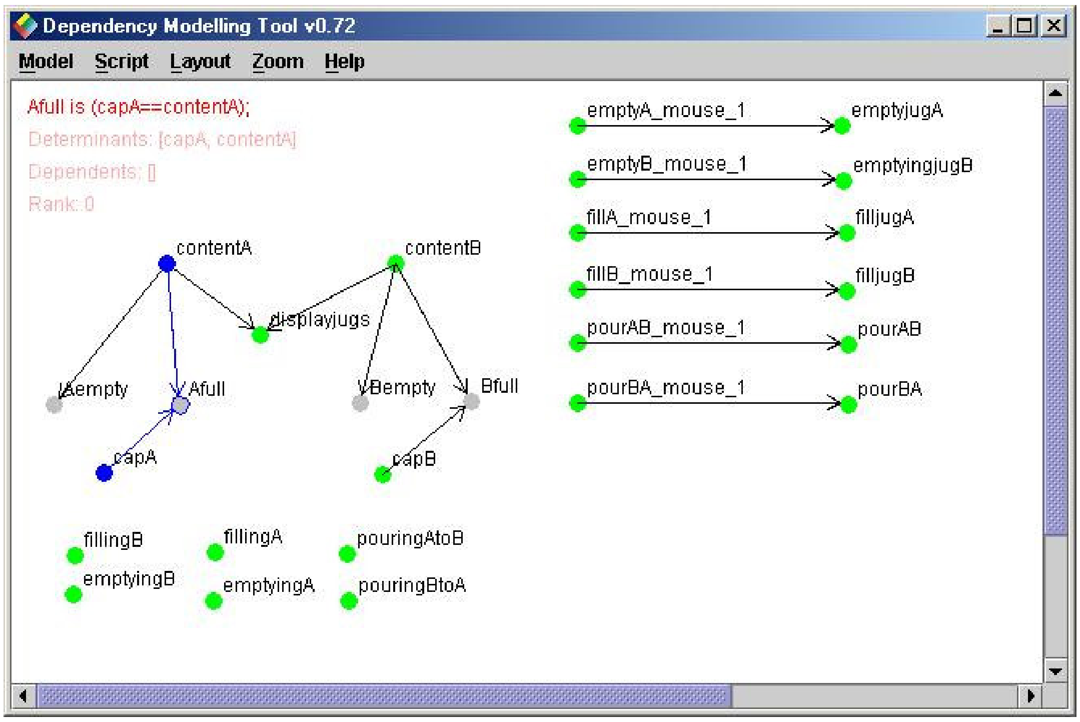
Concept maps can be constructed using pencil and paper, but there are also many software programs that support their construction (e.g. Inspiration [Ins93]). The major advantage of computer-supported concept mapping over pencil and paper approaches is the forgiving nature of the medium – links and concepts can be edited or removed easily. Anderson-Inman and Zeitz [AZ93] found that the computer-based medium encouraged learners to revise their maps as their understanding changes. In their research, Heeren and Kommers [HK92] concluded that concept mapping software should allow expressive flexibility so that students with different learning styles and techniques can demonstrate and develop their knowledge and understanding.

In summary, concept mapping is a constructionist approach to articulating current knowledge, organising current ideas and establishing links between current and emerging ideas. We now consider how concept maps can be viewed from the perspective of the EFL.

4.4.2 Concept maps and the EFL

Concept mapping, as an activity to explore and classify personal knowledge, has many characteristics in common with learning activities at the private end of the EFL. Concept mapping involves surveying a domain with a view to identifying important features and contexts. The construction of a concept map is not an objectively defined process; it is an iterative process that encourages reflection on construction as active learning to stimulate new knowledge [GKL$^+$01]. The process of constructing a concept map to articulate our current knowledge is related to Gooding's notion of a construal (cf. section 3.4.1) [Goo90]. A concept map can be viewed as a construal because it is a concrete artefact being used to understand a phenomenon. Gooding views construals as concrete and situational interpretations of unfamiliar experience and trial interpretations – this characterisation is consistent with the role that concept maps play in representing the known and connecting the known to our emerging understanding.

In other respects, concept mapping does not necessarily resemble EM at the private end of the EFL. Like the jugs visualisation in Figure 2.19, the concept map in Figure 4.2 serves as an artefact, but its nodes depict concepts that are much more sophisticated than the primitive observables that represent the current contents of a jug. Likewise the relationships between the nodes in the concept map in Figure 4.2 are less precisely prescribed than the dependencies between primitive observables. This loose analogy between observables and nodes, and dependencies and propositions was exploited by Wong in his design of the Dependency Modelling Toolkit (DMT) [Won03]. Figure 4.3 shows the DMT representation of the jugs model shown in Figure 2.19.



Afull is (capA==contentA);
Determinants: [capA, contentA]
Dependents: []
Rank: 0

Figure 4.3 – The simple jugs model from Figure 2.19 in the DMT

In [Won03], Wong envisages the DMT as the basis of a visual environment for EM. It is evident that such an environment could be used to construct complex definitive scripts that support EM at the concrete end of the EFL. In effect, Wong has identified a particular kind of concept map that can be used to migrate from primitive learning activities towards objective knowledge (cf. the discussion of heapsort in section 6.3

and in [Run02]). Though concept maps are in general suitable for expressing our current level of understanding and assist as a learning device in assimilating new knowledge neither computer-based nor pencil-and-paper concept maps can integrate experiential and formal learning activities. In effect, the concept map is set aside when constructing a computer model based on the knowledge that it embodies.

4.5 Programming for domain learning

Computer-based modelling has had a crucial role in the development of constructionist approaches to learning. The future prospects for constructionist model-building approaches will depend on the quality of the paradigm being used to build models. For this reason, we need to consider conventional approaches to programming and the implications for their support of constructionism. In this section, we first review the main tenets of the Object-Oriented (OO) approaches to model construction that are so prominent in modern approaches to software development (e.g. [Jac92]). We then describe how programming can be viewed from within the EFL, and conclude that programming as classically conceived has deficiencies with respect to constructionism because of its emphasis on preconception, abstraction and generality over flexibility and experimentation. The work reported in this section draws on material from three previous EM theses, namely Allan Wong's research on software system development in EM [Won03], and comparative studies of OO and EM in Timothy Heron's MSc thesis [Her02] and Ruyuan Wang's MSc thesis [Wan03].

4.5.1 Programming from a learning perspective

If computer-based model building is to support the educational objectives of constructionism, then the programming activity must be well-aligned to domain learning. In general, a program is primarily conceived as a piece of software that fulfils some purpose based on a set of requirements. Programming involves writing computer code to satisfy the intended specification of the program. In general, the programmer's objective is not to learn from the construction of the program, but to deliver a final product. In the context of the constructionist agenda, the main question is whether the programming process is beneficial from a learning perspective.

If we consider the programming process as having two aspects – the design and analysis of the domain, and the writing of code – then learning is predominantly associated with the design and analysis activities. In the constructionist approach to learning, the evolving artefact is an embodiment of ignorance, and interactions with it serve to shape the artefact and its interpretation. Knowledge of the domain is acquired as the artefact is constructed. The roles that artefacts and domain knowledge play in programming are in contrast quite different. While knowledge of the domain plays a fundamental role in programming, the only essential knowledge is concerned with the intended interaction and interpretation of the program that is prerequisite for conventional programming. There *is* a role for interaction with artefacts (as represented by use cases, UML diagrams [JCJ+92], and prototypes of various kinds) but these artefacts embody the knowledge prerequisite for programming and are typically discarded once programming commences.

In previous EM theses, particular case studies have been used for comparative studies of model building in EM and program construction using OO principles. We will discuss these briefly to highlight their key conclusions. Heron's MSc thesis compares EM and OO development with reference to two case studies: a game of draughts [EMRep, draughtsRawles1997]; and a vehicle cruise control simulator (VCCS) [EMRep, cruisecontrolPavelin2002]. From these comparisons, he concludes that changes in the OO model are as easy to make as in EM as long as the change has been preconceived in advance. In his study of the VCCS model, he observes that the OO programmer has more to do; for example, all the method calls used to propagate changes of state have to be worked out and ordered before coding. Wang's MSc thesis discusses the VCCS models developed in OO and EM in much greater depth. She concludes that the functionality of the object-oriented VCCS is abstractly and precisely prescribed, whereas in EM the VCCS model can take account of issues that are outside the normal scope of correct operation and that might be useful for learning. Her key observation is that use-cases [Jac92] only give an account of typical interaction between a system and external actors. However, from a learning perspective this requires all interactions to be conceived before any programming takes place – there is no room for evolution of ideas in tandem with the developing program.

Wong's PhD thesis contains a comparative study of EM and OO development of a dishwasher model. Through implementing models using both approaches, he identifies a number of characteristics from which to compare the two development styles [Won03]:

i) Modelling focus – In EM, the focus is initially on the subjective interpretations of the modeller. Observations recorded in the EM model are based on the imagined interactions in the system, but there is no circumscription of the system boundary. In UML, the focus is on modelling the structure and behaviour of the system. Once the system boundary has been established, the system is constructed in isolation from its operating environment.

ii) Interactiveness – In EM, the modeller can always get feedback on any part of the model, because there is always a working model. Experimental interactions to test new insights or consolidate current understanding are part of the modelling approach. In UML, diagrams are abstract representations that are not primarily to be interpreted with reference to a particular state of a system. The main role of the UML is to specify system behaviour; experimental interactions are limited in their extent and changes must be compiled into an executable program for each change.

iii) Comprehension – In EM, comprehension is typically gained from experimental interactions with the model. Exploration in this sense is not part of the UML approach; the diagrams purely represent the predetermined relationships between components rather than allowing the exploration of possible alternatives.

iv) Openness – In EM, there is no particular viewpoint from which a model should be built, and the model emerges from experimental interactions rather than through prior circumscription. There is no system boundary within which construction must take place and the modeller is always in the position of being able to step in and make experimental changes. In UML, a fixed set of diagrams guides the modelling process. Use-cases and class diagrams constrain the interactions between components until implementation is clear. The emphasis is on circumscription as the guiding principle of system construction.

v) Interfaces – In EM, model building and user interface construction occur within the same framework. In UML, there is no support for specifying interfaces of the target system. Interfaces must therefore be constructed and tested separately.

The comparative studies of EM and OO approaches to software development highlight the following distinctions:

- in conventional programming, the artefacts generated to represent the program requirement reflect richer knowledge of the application domain than the program itself. In EM, in contrast, the knowledge prerequisite for realising an abstract behaviour is cultivated throughout the modelling process and is never discarded.
- in conventional programming, domain knowledge and identification of purpose are essential prerequisites for writing a program, and optimisation to purpose is a measure of program quality. In EM, in contrast, the identification of purpose emerges during the process of model construction and this identification does not compromise the character of the model as a construal.

In the next section, we interpret the conclusions from the comparative studies of EM and programming with reference to the EFL and our broad perspective on constructionism.

4.5.2 Conventional programming, constructionism and the EFL

Conventionally, programs are built by defining an external system boundary and circumscribing the possible interactions and interpretations. With reference to the EFL, these principles for programming are targeted at the identification of common experience and objective knowledge together with symbolic representations and formal languages. In other words, programming is associated with activities at the theoretical end of the EFL. What is more – though computer programs may be able to support learning activities at the end of the EFL – conventional programming activity itself is very different in character from the learning activities at the empirical end of the EFL.

The discussion above indicates that conventional programming is only well aligned to a small subset of the learning activities that make up the EFL. It is obvious that model construction using a conventional programming language satisfies Papert's basic definition of constructionism in so far as the act of programming involves actively creating a personally meaningful entity. However, as we shall argue below, conventional programming lacks the characteristics needed to give full support to the broad perspective on constructionism that we adopt in this thesis (cf. Figure 4.1).

The aspiration in bricolage and situated learning is to support concrete learning through interaction and exploration. However, this aspiration is not being well served by the current emphasis on computers in learning. Our experience of EM has led us to believe that the paradigm used for model building has a significant bearing on the quality of support for the constructionist approach that computers can provide. As we have argued in a previous paper:

> 'there are profound conceptual issues to be addressed before such a shift in emphasis [from 'planning' to 'bricolage' in computer model construction] can be achieved: the fundamental preconceptions about computation that inform classical computer science are ill-oriented for this purpose' [RB02].

We agree with Ben-Ari (cf. section 4.2) that computer programming – as practised – contains elements of bricolage [Ben01] and with Brooks's view [Bro95] that programmers see their work as a craft where they wrestle with incompletely understood meaning. Software development identifies two general phases, of knowledge gathering and knowledge deployment. The first phase is one of engagement with the world, and of preliminary knowledge gathering, whilst the second is a complementary phase where knowledge is deployed in program specification and design. Interaction with artefacts is common in gathering knowledge (through the building of prototypes, the creation of use-cases and UML diagrams [JCJ$^+$92]), but this knowledge is targeted at achieving a specific functional goal. The obligation to frame knowledge in this goal-directed fashion reflects the conception of programs within the classical theory of computation. This means that in practice, even though the phases of knowledge gathering and deployment are interleaved, the intimate relationship between the two is obstructed by the way in which knowledge is deployed.

The relationship between knowledge gathering and knowledge deployment in computer programming reflects a commonly accepted view of the relationship between experiment and theory in science [LJ98]. Our concern about the classical separation between the open-ended experience that informs requirements and the circumscribed behaviour of a program mirrors Gooding's concern about the bifurcation of the scientist's world into the empirical and the literary:

> "Scientists' descriptions of nature result from two sorts of encounter: they interact with each other and with nature. Philosophy of science has, by and large, failed to give an account of either sort of interaction. Philosophers typically imagine that scientists observe, theorize and experiment in order to produce general knowledge of natural laws, knowledge which can be applied to generate new theories and technologies. This view bifurcates the scientist's world into an empirical world of pre-articulate experience and know-how and another world of talk, thought and argument. Most received philosophies of science focus so exclusively on the literary world of representations that they cannot begin to address the philosophical problems arising from the interaction of these worlds: empirical access as a source of knowledge, meaning and reference, and of course, realism." [Goo90, p. xi]

The alternation between requirements gathering and program specification testifies to the bifurcation of the computer scientist's world. This is evidenced by the problems encountered in conventional programming approaches when seeking to admit truly experimental interactions and achieve flexible adaptation of the program as it is being developed. In our view, software development to support constructionist use demands a conceptual integration of the pre-articulate exploration and formalisation of knowledge that are respectively associated with the phases of knowledge gathering and knowledge deployment [RB02].

The above discussion suggests that conventional programming is inadequate as a basis for a general constructionist approach to model building (cf. the observations by Soloway [Sol93] and by Steinberger [Ste94] that general purpose programming languages obstruct meaningful domain learning). This may be a factor in accounting for the relative lack of popularity of programming as a learning tool for the non-specialist (cf. [Nar93]), and the emergence and subsequent disappearance of Logo from the National Curriculum in the United Kingdom (cf. [NH96]). In the following section, we consider the merits of EM as an approach to model construction that enables the conceptual integration of concrete and formal learning required to support our broad notion of constructionism.

4.6 Empirical Modelling, constructionism and the EFL

In this section, we discuss the connections between EM, constructionism and the EFL. The strength of these connections determines the extent to which EM can support our broad notion of constructionism. Like conventional programming, EM evidently satisfies Papert's basic definition of constructionism. In section 3.6, we showed that EM can support a wide range of learning activities that are identified in the EFL. It remains to show that EM is well suited to supporting broader aspects of constructionism such as bricolage and situated learning.

4.6.1 Empirical Modelling and bricolage

In this section, we discuss how EM model construction supports bricolage. To do this we review the literature on both EM and bricolage, to identify their common points. Bricolage and EM originate from entirely different contexts. In developing the concept of bricolage, the anthropologist Levi-Strauss was concerned with the construction of physical artefacts within 'primitive' societies. In contrast, EM is concerned with computer-based model construction that has been observed in practice, primarily in the work of computer science students at the University of Warwick over the past fifteen years. This has involved the construction of several hundred models in connection with student projects and academic research (see [EMRep] for a representative sample).

Evidence that the EM modeller is a bricoleur rather than a planner can be seen in key phrases taken from Russ's comparison of EM and programming in [CRB00]:

- 'there is really no counterpart in EM to the 'planning' phase. ... conceptual modelling in EM can conveniently be directly put into a script with a visualisation and experimented with on the computer.'
- 'it is significant that testing occurs in advance of any commitment to a particular form of program.'
- 'in an EM development it is typical that the interface is left until an advanced stage of the development – when the purpose and requirement has been clarified through extensive use of the very open-ended phase of model construction and exploration.'

In EM, the purpose of model building may not be initially clear:

> 'The objective of a (student) project has often been uncertain at the early stages, and a theme has emerged as the model-building activity proceeds incrementally. The profile of work on the project is distinctively different from that practised in other paradigms, such as object-oriented software development. Students typically carry out significant model construction even at the early stages, and are guided by this in their strategic decisions' [Bey01].

Though the modeller may have a general problem in mind, the initial phase of work involves surveying tools and models both to identify useful resources and to shape the provisional direction of development. This preliminary activity typically influences the modeller's original conception of their project. This resonates with Levi-Strauss's account of bricolage in the preliminary stages of a project:

> 'Consider the bricoleur at work and excited by his project. His first practical step is retrospective. He has to turn back to an already existent set made up of tools and materials, to consider or reconsider what it contains and to engage in a sort of dialogue with it and, before choosing between them, to index the possible answers which the whole set can offer to his problem.' [Lev68, p18]

> 'Once it materialises the project will therefore inevitably be at a remove from the initial aim, a phenomenon which the surrealists have felicitously called "objective hazard".' [Lev68, p21]

The influence of the developing artefact over the modeller's conception of his or her project is prominent throughout the development of an EM model, and the final outcome of a project may differ significantly from the initial idea.

As highlighted above, the process of model construction in EM is one of negotiation; the modeller uses the partially constructed artefact (and its real world counterpart if it exists) to further refine their current understanding of it. This emphasis on understanding the artefact under construction is also seen in bricolage. Levi-Strauss says of model building:

> 'Now the model being an artefact, it is possible to understand how it is made and this understanding of the method of construction adds a supplementary dimension.' [Lev68, p24]

The products of EM and bricolage both relate to the embodiment of rich experience in a real-world artefact. In both, the emphasis is on human engagement in the model building and concrete rather than abstract representations of knowledge. Levi-Strauss refers to the products of bricolage as ‘miniatures’ and stresses the importance of real-world human engagement:

> ‘… miniatures have a further feature. They are ‘man made’ and, what is more, made by hand. They are therefore not just projections or passive homologues of the object: they constitute a real experiment with it.’ [Lev68,p24]

The emphasis in EM and bricolage is on the learning that occurs during construction of an artefact rather than on the finished product, as illustrated in the quotes below:

> ‘... the ‘bricoleur’ also, and indeed principally, derives his poetry from the fact that he does not confine himself to accomplishment and execution ... The ‘bricoleur’ may not ever complete his purpose but he always puts something of himself into it.’ [Lev68, p21]

> ‘Computer models constructed using Empirical Modelling principles are not to be viewed as implementing an abstract mathematical model. Their significance is instead similar to that of the physical model that an experimental scientist might build to account for a phenomena, or that an engineer constructs to prototype or test a design concept.’ [BS99]

The qualities of bricolage in relation to the EFL can be inferred from Table 4.1. The defining characteristic of bricolage – of intimate engagement through interaction with the artefact – is found in activities at the empirical end of the EFL. Planners – who preconceive modes of use and functionality of their product before programming – do not engage with the empirical learning activities during construction. This approach is only suitable if they have a good understanding of the situation they are modelling. Model building approaches that embrace bricolage must be capable of supporting learning activities at the experimental end of the EFL.

In summary, the discussion in this section has illustrated that there are close links between EM and bricolage, and that both offer support to the concrete learning activities at the empirical end of the EFL.

4.6.2 Empirical Modelling and situated learning

In situated learning, hands-on interaction with tangible artefacts guides the learning process. The technologist, John Seely Brown claims that educational practice has been dominated by a belief that conceptual representation (typically abstract and symbolic) is of the most importance, and argues that situated cognition, giving activity and perception a prior place over representation, could solve some of the problems in school learning:

> 'For centuries, the epistemology that has guided educational practice has concentrated primarily on conceptual representation and made its relation to objects in the world problematic by assuming that, cognitively, representation is prior to all else. A theory of situated cognition suggests that activity and perception are important and epistemologically prior -- at a non-conceptual level -- to conceptualisation and that it is on them that more attention needs to be focused. An epistemology that begins with activity and perception, which are first and foremost embedded in the world, may simply bypass the classical problem of reference -- of mediating conceptual representations' [BCD89].

In Seely Brown et al [BCD89], situated learning is associated with 'cognitive apprenticeship' in which learning progresses from activity embedded in a situation to general principles of the culture. Apprenticeship goes together with coaching, and students undertake modelling in situ that is scaffolded to get them started in authentic activity.

To give computer support to situated learning as characterised by Seely Brown requires a modelling approach that gives a high priority to activity and perception. As discussed in section 3.3.2, in EM, activity and perception are viewed as 'epistemologically prior to conceptualisation' and the idea of 'one experience knowing another' may be seen as 'bypassing the classical problem of reference' (cf. [Run02, chapter 2]). The fundamental concept of EM is not that 'activity and perception are first and foremost embedded in the world', but rather that activity and perception are first and foremost embedded *in personal experience*, that can then be classified as subjective or objective. In EM, activity and perception that is embedded in objective experience is nevertheless important. In such a context, EM is a form of situated modelling.

Situated modelling is an essential constituent of computer support for situated learning. The situated nature of EM has been discussed in detail in connection with its potential role in software development [Sun99]. In particular, EM can be seen as meeting Goguen's concern for situatedness in requirements analysis:

> 'EM activities are carried out with reference to an external situation, even though in practice this situation can be imaginary rather than concrete. Practical experience of EM confirms its status as a situated modelling method, and activities in EM exhibit Goguen's "qualities of situatedness": emergence, contingence, locality, openness and vagueness [Gog96]. The main reason why EM exhibits these qualities is that, because of the nature of the modeller's interaction, the process of formulating definitive scripts is never separated from the modelling context.' [BS98]

The qualities of EM as a situated modelling approach are that:

> 'the properties of openness and situatedness reduce the separation of model and world and offer the possibility of a user deriving qualitative knowledge of the world through interactive use of the model' [BRR00].

These qualities are significant for the computer support that EM can give to situated learning. They support the learner in interacting with artefacts and developing practical skills in particular concrete situations and also assist the learner in applying and understanding general abstract solutions.

The organisation of the constituent learning activities within the EFL is consonant with Seely Brown's claim for the prior place of activity and perception over conceptual representation. Modelling an artefact is situated; its construction takes place with and during consultation of a real-world referent. In situated modelling in EM, the important activities have direct counterparts in the real world: namely the identification of salient features (through observation); understanding the nature of indivisible changes in the referent (through dependency); and determining who, or what, is responsible for those changes (through agency). With reference to the EFL, situated learning is concerned with interaction and experimentation with particular concrete instances of a problem in context and this corresponds to learning activities at the concrete end of the EFL.

In summary, the discussion in this section has illustrated that there are close links between EM and situated learning, and that both offer support to the concrete learning activities at the empirical end of the EFL.

The above discussion has considered situated factors in model construction. The benefits of EM as an approach to situated modelling can also be seen in relation to learning environments where situational factors have an important role to play. This is illustrated by the Clayton Tunnel model (see section 2.4.5). In this model, each participant views the situation from his or her own perspective on a different computer. These perspectives only contain the elements that each participant can see and interact with. For example, train drivers can only interact with controls related to their train and can only see the signalman from particular parts of the track. Despite the limited nature of the visualisation in this model, the open-endedness of the interaction between participants and environmental factors in the situation offers elements of realism that would not necessarily be within the scope of an immersive environment. For instance, it is in principle quite straightforward to simulate failures of communication due to misunderstanding; mechanical breakdowns; and environmental variations that lie within the frame of the construal.

4.7 The digital watch case study

We conclude this chapter with a concrete example to illustrate how EM supports the broad view of constructionism taken in this thesis. The construction of a digital watch model is an example of situated modelling in EM since the referent is an actual digital watch. The digital watch model has been used to illustrate many aspects of EM; for more detailed information, see [BC95, RBF00, FB00, BRW^{+}01]. Relevant issues illustrated in this section include:

i) bricolage style development involving the re-use of previous models.

ii) knowledge gained through hands-on interaction rather than reference to a user manual.

iii) the combined use of formal and informal artefacts.

iv) multiple perspectives on digital watch design and use.

v) incorporating situated aspects of digital watch use in the model.

An important facet of bricolage is the unprescribed path that the development of the artefact follows. Bricolage involves subjective interaction of the modeller with the artefact, and the character of both the artefact and the modeller's relationship to it are subject to change through exploratory interaction. This can be illustrated with reference to the digital watch through the re-use and extensions made to the model by a group of modellers over an extended period of time (see Figure 4.4).

Four different people were involved in the development of the digital watch model over a period of eight years. The collaboration and communication between the modellers involved was limited; at each stage, the artefact itself embodied much of the knowledge that guided its future development. Beynon constructed the initial digital display and a basic statechart (as presented in [Har87a]) in 1992. Richard Cartwright added the analogue clock and the digital watch buttons in 1994 [BC95]; he also developed a chess clock variant of the model [BC95]. The functionality of the digital watch at this stage was restricted to that of Harel's original statechart, which describes the display functions in detail but omits the functionality of components such as the stopwatch and the mechanisms for setting the time. The full functionality for the watch, together with buttons for updating the date and time, was added by Carlos Fischer in 1999 [FB00]. In 2000, I altered the functionality to match the watch I was using at the time and added an alternative visualisation that aided comprehension of tasks that could be undertaken with the watch [RBF00]. My version of the model is shown in Figure 4.5.

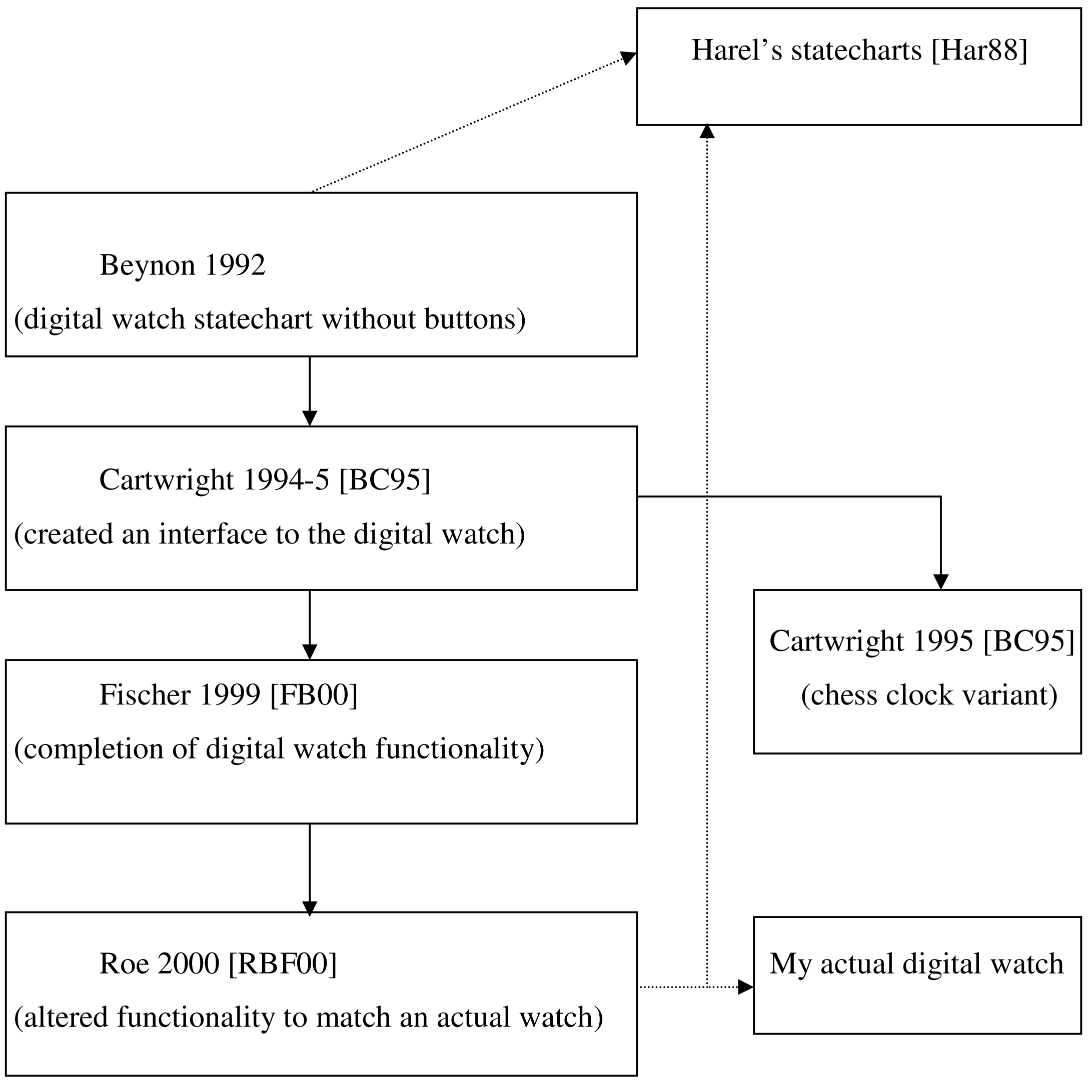
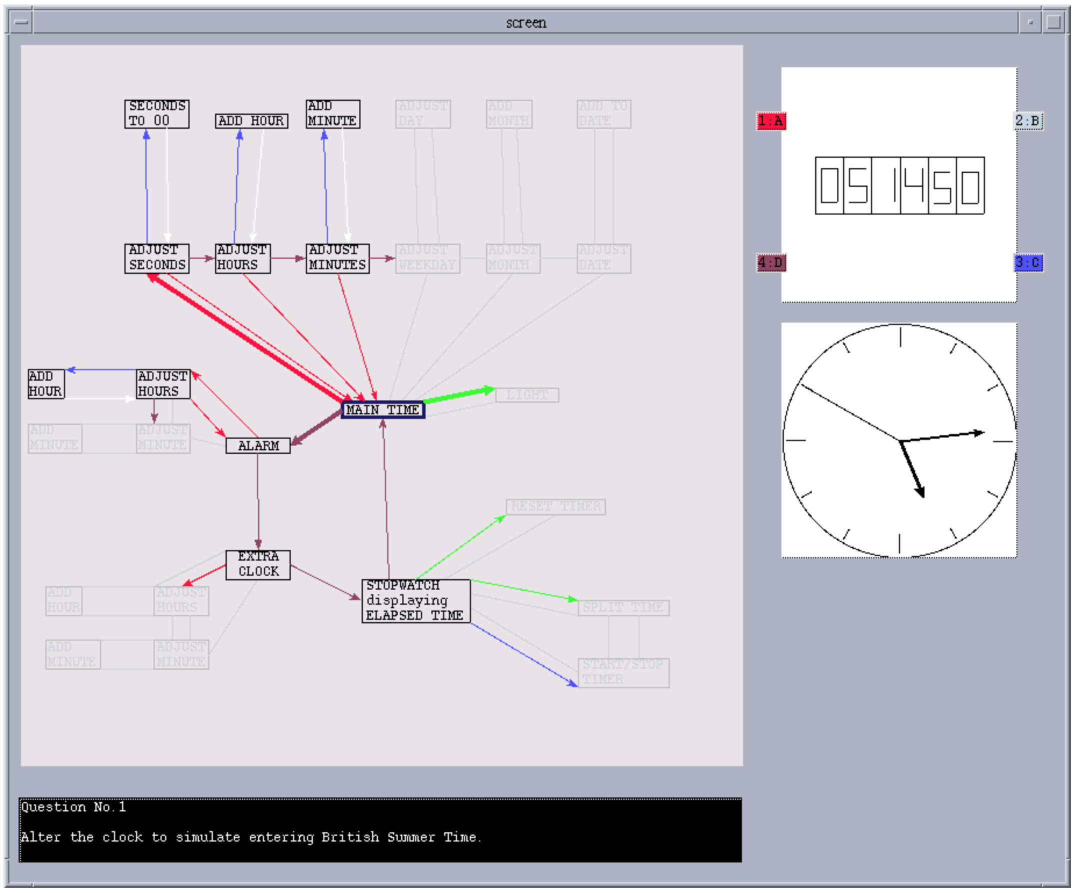

Harel's statecharts [Har88]
Beynon 1992
(digital watch statechart without buttons)
Cartwright 1994-5 [BC95]
(created an interface to the digital watch)
Cartwright 1995 [BC95]
(chess clock variant)
Fischer 1999 [FB00]
(completion of digital watch functionality)
Roe 2000 [RBF00]
(altered functionality to match an actual watch)
My actual digital watch

Figure 4.4 – The development history of the digital watch

Conceptually, the model was constructed in one continuous sequence of interactions as in one 'stream of thought' (cf. section 3.3.2). The introduction of new definitions to attach components to the model, and the addition of procedural actions to simulate agents, are examples of typical interactions involved in the development. As in bricolage, the impact of additions to the model guided the modeller in making future changes. There is no sense in which the model in its current state represents a finished product or the development process has reached a point of closure; within EM, the model always remains open to potential future revision and extension. For instance, the functionality of the digital watch could be extended to include the heart rate monitoring facilities on a sports watch.



Question No.1

Alter the clock to simulate entering British Summer Time.

Figure 4.5 – The digital watch artefact (top right), an analogue clock (middle right) and a mental stategraph (left)

The model consists of a digital watch artefact, a corresponding analogue timepiece and a 'mental stategraph' that indicates the user's current level of familiarity with the states of the digital watch. The buttons labelled A,B,C,D on the digital watch interface in Figure 4.5 depict four physical buttons that were similarly located on my personal watch. The functionalities of these buttons correspond to the physical watch operations. There are many dependencies present in the artefact. For instance, the visible elements of the watch are dependent on its internal state. Button pressing allows certain patterns of state change corresponding to watch operations to be performed through the interface. These patterns reflect the circumscribed functionality of the actual digital watch. They are expressed on the mental stategraph by using coloured arrows to represent the state transitions that will occur if a button is pressed. The current state is indicated with a bold border. For example (see Figure 4.5), if button A is pressed then the watch enters the time setting mode, and if button D is pressed then the watch enters the alarm mode.

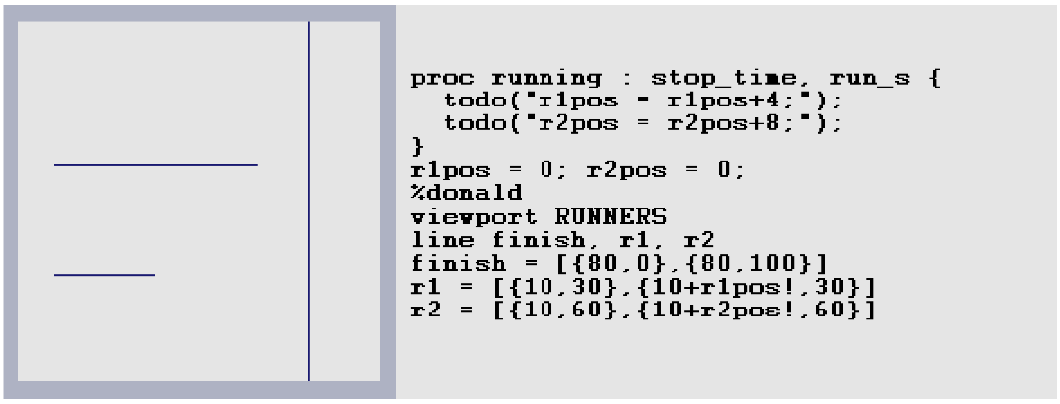
Beynon's initial model was originally conceived with Harel's agenda of using visualisation to support complex system development in mind [Har87b, Har88]. It features a statechart (a concept introduced in [Har87a]) that is used to specify state-transitions and events. Statecharts are much richer than traditional state transition diagrams because they exploit the notions of depth and orthogonality [Har88]. A statechart is most suitable for recording reliable and comprehensive system knowledge; it is not necessarily the most appropriate way to represent our emerging understanding of the system that it describes.

The interface to my digital watch model was designed to be as faithful as possible to my construal of the behaviour of the actual watch. In Figure 4.5, the visual organisation of states in the stategraph reflects my conception of the main and subsidiary functions of the watch. The changes of state are precisely correlated to the actions of pressing and releasing buttons. In the stopwatch component, the effect of a button press is dependent on the current state of the stopwatch: specifically the transition made in response to button press B is determined by whether or not the stopwatch is running. In all these respects, the stategraph differs from a statechart. Its primary role is to provide an experiential rather than an abstract representation of state.

The stategraph is better oriented than the statechart towards studying a learner's interaction with an artefact as it is conceived by Carroll (cf. section 4.1.2 and [Car90]). The learner does not have the comprehensive knowledge of the artefact that the user manual and the statechart abstractly supply: they develop understanding haphazardly through experiment with the artefact itself. The stategraph in the digital watch model is intended to be helpful in tracing a learner's emerging understanding. With this in mind, the stategraph in Figure 4.5 discloses states to the user as they encounter them. This is a preliminary step towards representing the current knowledge of the learner more accurately. For instance, from Figure 4.5 we can deduce that the user has explored changing the time of the main clock and altering the alarm, but has not yet encountered the other features. Visualisation of this kind is only a first step towards evaluating a learner's understanding; the fact that a learner has encountered a particular function is no guarantee of understanding. More insight can be gained from using the visualisation in conjunction with a worksheet that specifies activities to be undertaken by the learner, such as setting the clock to British Summer Time (see Figure 4.5).

It is evident that the digital watch model supplies a useful environment for situated learning. For instance, it can be used to learn about telling the time on digital and analogue clocks, to learn about the relationship between different time zones and to understand many issues concerning the design and use of clocks, stopwatches and digital watches (cf. Appendix D). Specific applications of the watch model in situated learning are targeted by the worksheet questions that are incorporated into the model.

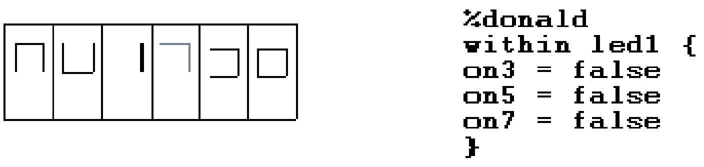
To illustrate situated use in a broader context, the modeller can introduce extra observables to embellish the current model. These observables – rather than referring to the digital watch itself – will be situational in nature. Situational observables become significant when specific user activities involving the watch are being studied, such as when the stopwatch feature is being used to record the finishing times for two runners in a race. Figure 4.6 is a simple line drawing to represent a race between two runners. The horizontal lines extend to the right towards the vertical finishing line when the stopwatch is started. The watch user can then record the finishing times of both the runners using the 'split time' feature. The TkEden definitions and action required to model the runners in this extension is also displayed in Figure 4.6. Note that there are two aspects to this extension, of the model: firstly, the actual script has been changed; and secondly, the way that the user interacts with the script changes to reflect the new situational emphases.



```
proc running : stop_time, run_s {
  todo("r1pos = r1pos+4;");
  todo("r2pos = r2pos+8;");
}
r1pos = 0; r2pos = 0;
%donald
viewport RUNNERS
line finish, r1, r2
finish = [{80,0},{80,100}]
r1  =  [{10,30},{10+r1pos!,30}]
r2  =  [{10,60},{10+r2pos!,60}]
```

Figure 4.6 – Situational observables – timing two runners.

The potential for extension of the digital watch model is such that we can take account of the exceptionally rich aspects of experience and knowledge that can inform everyday interaction. Figure 4.7 depicts the digital watch display as it might appear when observed when lying in bed, where it may be partially obscured.



```
%donald
within led1 {
on3  =  false
on5  =  false
on7  =  false
}
```

Figure 4.7 – A partially obscured digital display

In this situation, careful observation of the clock is required before we can establish the correct time. This will take into account such factors as the parts of the digits we can observe, our knowledge of the patterns that govern the digits changing and contextual knowledge, such as our estimation of the current time [BRW+01].

4.8 Summary of the chapter

In this chapter, we have considered how techniques for domain learning are related to the educational theories of constructionism and instructionism. We have shown that there is a paradoxical aspect to the way that conventional programming offers support for constructionism. It satisfies Papert's basic definition of constructionism, yet on deeper inspection it is not well aligned to the constructionist learning activities in the EFL, or to the ideas of bricolage and situated learning. Further, we have shown that model construction in EM can support the ideas of bricolage and situated learning, and the broad range of learning activities in the EFL. It is for this reason that we regard EM as establishing a more intimate link between domain learning and model construction than other approaches.

Chapter 5 – Scaffolding different types of learning

5.0 Overview of the Chapter

The focus in this chapter is on the merits of EM for the development of educational software. In previous chapters, we have argued that the EM approach to model construction supports a wide range of learning activities based on a broad constructionist view. We now consider the potential of EM for the development and use of learning environments. We shall argue that the use of EM in developing learning environments is advantageous because the highly flexible and adaptable nature of EM allows for relatively easy customisation of learning resources through its support for a very broad definition of scaffolding. We discuss scaffolding in relation to three different types of learning: of fixed referents; of exploration of possibilities; and of learning languages. We illustrate these ideas with reference to EM case studies of learning environments.

5.1 Model use vs Model building

5.1.1 Constructionist learning environments

Up to this point in the thesis we have been concerned with the support for learning that is afforded by EM model-building activity. We have concluded that EM offers better support for learning than conventional programming due to its ability to integrate pre-articulate and formal learning activities. However, it is not always possible for users to create their own models, and therefore in order to provide a rounded picture of learning and EM we also need to consider the benefits of EM models in use.

There are many reasons why learners may not be able to, be allowed to, or wish to, create their own models. This is especially true in the educational context, where, for example:

i) school children generally do not have enough computing expertise to be able to construct models to meet all their educational needs.

ii) model construction following personal interests cannot guarantee that learning relevant to the curriculum occurs.

iii) teachers may lack the necessary skills or the available time to be able to construct models for pedagogical use.

We elaborate each of these points in turn.

In respect of the first point, Nardi [Nar93] has observed that the construction of programs by end-users may not be a realistic aim. This is apparent in relation to EM model construction with our current tools. To date, all the authors of models built using EM tools have had prior knowledge of the fundamentals of computers and programming. For instance, understanding functions, variables and parameter passing are at present an essential prerequisite to EM model creation. Our own experiments with 17 – 18 year old college students have exposed this problem. We found that students without any previous programming experience could not use the TkEden tool to create models because they lacked essential computing knowledge. However, students with programming experience succeeded in extending previously created models. When students do not have a good understanding of basic programming concepts they cannot develop their own models and are reliant on others to produce learning environments for them.

In respect of the second point, even if students can create their own models, there needs to be a degree of accountability where learning through model creation is concerned. Students are usually following a prescribed curriculum and if they follow their own interests when creating models they may be learning subjects outside their curriculum. Further evidence of the difficulties of accountability in constructionist learning is evident in Noss and Hoyles's idea of the *play paradox* where time spent at the computer may not be being used for meaningful learning [NH92]. Even in the established practice of computer-based model building for learning, such as was introduced by Papert through Logo [Pap83], it could be argued that the need to learn computer programming skills detracts from domain learning. The disappearance of Logo from the United Kingdom National Curriculum in the 1990s has been cited as evidence of uncertainty about its educational merits [NH96].

In respect of the third point, although teachers have the educational knowledge required to develop useful learning environments they cannot put that into practise without the necessary programming skills. Ideally, teachers want to be able to customise educational resources to suit individual learning needs. This requires that small, but often unpredictable, changes to programs can be made with limited knowledge of their construction. Traditional approaches to programming favour development in which very high cognitive demands are placed upon the developer prior to programming, and do not lend themselves to unpredictable end-user customisation. As Nardi observes [Nar93]:

> 'While programmers can be called in to provide applications for minority areas, once the software is written, users are stuck with the applications given them by programmers, and the applications cannot easily be changed, extended, or tailored to meet the demands for local conditions.'

This is diSessa's motivation for proposing that teachers and software designers should work closely together with children to produce useful learning environments [diS97b].

Broadly speaking, educational software can be classified on a spectrum between instructionist and constructionist-based approaches (see Figure 5.1). This spectrum has historical significance in that Computer Assisted Instruction (CAI) preceded Intelligent Tutoring Systems (ITS), which in turn preceded Interactive Learning Environments (ILE). CAI uses computers to replicate the traditional school learning model that has been criticised by many [Fri70, Ill71, Pap93, Opp97]. CAI has often been called the ‘drill-and-kill’ approach, whereby students are presented with a set of textbook style questions to answer. ITS, introduced by Hartley and Sleeman in 1973 [HS73], are an extension of CAI. In addition to providing exercises for students, an ITS system assesses what a student knows and what they should know, and generates new exercises based on this assessment. However there is no scope for a learner to take control of their own learning experience because the system designer has preconceived the material for delivery and the mode of interaction. CAI and ITS are instructionist approaches aimed at imparting and testing objective knowledge.

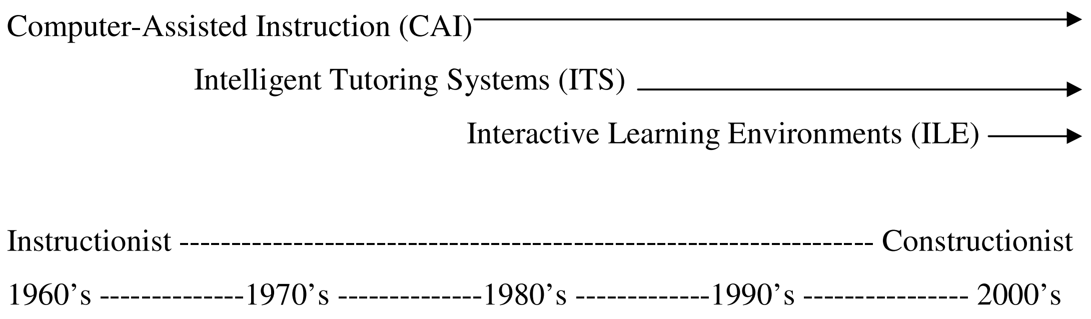

Computer-Assisted Instruction (CAI)
Intelligent Tutoring Systems (ITS)
Interactive Learning Environments (ILE)
Instructionist
Constructionist
1960’s
1970’s
1980’s
1990’s
2000’s

Figure 5.1 – A spectrum of learning perspectives

Constructionist software takes advantage of the medium of the computer to provide a qualitatively different learning experience. The essential difference between constructionist computer environments and CAI/ITS systems is that the learner has more control over their learning. As Soloway et al note, this necessitates a switch from user-centred design to learner-centred design [SGH94]. ILEs are constructionist because they emphasise the active role of the learner and are often called *microworlds*. A microworld is a small world within which students can understand concepts through active learning [Pap93]. For example, Cockburn’s microworld to support the learning of Newtonian physics allows students to manipulate the parameters in physical laws and observe the resulting behaviour of objects [CG95].

The important requirements for constructionist software are that it should provide a learning environment in which:

1) a learner can explore the consequences of hypotheses whether or not they are correct.
2) learning objectives are situated in realistic domain contexts.
3) the designer or the learner can adapt or extend the environment to shape the learning process.

In the remainder of this chapter, we shall consider EM techniques for creating constructionist models that can support many types of learning activities.

5.1.2 Supporting different types of learning

In the 1980s, teachers had a high level of software ownership (as witnessed by the proliferation of small educational software companies often set up by teachers such as 4mation [4mat03] and Sherston [She03]) through being able to create software for their own particular teaching requirements. In recent years, software has become more powerful and more complex, but also less easily understood and adapted by its users [Joh03]. This motivates educational software that: can embrace a wide range of competencies; that is easily adaptable by both developers and users; and can provide teachers with resources that can be tailored to their particular pedagogical needs and context.

A standard approach to developing educational software that can be targeted at different learning scenarios is to expose simple concepts before more complex ones. This is similar in spirit to the HCI principle of *progressive disclosure,* which states that software should initially provide only the most commonly used features to a user, keeping more complex choices hidden in order to not overwhelm new users [PRS$^+$94]. As a user becomes more competent, exploration leads them to find and explore the more complex features. In this way, the program is easy to learn for novices but still contains the powerful features that advanced users require. A simple example can be found in expanding menu systems. For instance, in the Microsoft Office range of products, only the most commonly selected options from the menu are visible – to use other choices the mouse can be positioned over a double arrow to reveal all the possibilities.

The principle of progressive disclosure is not in general suitable for educational software. The purpose of progressive disclosure in application software is to hide the complex *features* of the software from novice users. In educational software, the aim is not to learn how to use a particular set of features, but to learn the *concepts* embedded within a learning environment. The exploration of progressively more complex concepts is associated with the idea of *scaffolding*. Scaffolding is defined as a technique for providing support to learners whilst they are learning a new task [Rog90]. EM gives support for scaffolding many different types of learning. For example:

i) **Learning as comprehension of a fixed referent** – a simple model of a specific referent is initially presented to the learner. This model is then gradually refined and extended by introducing more advanced concepts associated with the referent. The focus is on providing a computer-based model that accurately reflects its referent and level-by-level is guided by more precise observation of the referent.

ii) **Learning as in exploring possibilities and invention** – a simple model is initially presented to the learner. Although specific learning paths can be mapped out, the learner has discretion over how the model is extended at each layer, and different paths are associated with different referents. At each layer a learner is encouraged to interact as if in an exploratory laboratory.

iii) **Learning languages** – as a learner becomes more competent in a domain, their knowledge of domain specific language is progressively enhanced (cf. the music and rowing examples discussed in chapter 2). This can be reflected in the language used for interaction with the model of the domain.

Scaffolding of learning is analogous to the scaffolding that is used in constructing a building, which is removed when the building can stand by itself. As Soloway et al note [SGH94], scaffolding is provided to help a learner with a task they do not know how to do, and it gradually becomes less important as the learner becomes more competent. In educational terms, scaffolding is operating in Vygotsky's ZPD. The ZPD is defined as an area of domain knowledge, beyond a students' current comprehension, but which they can successfully navigate their way through with the assistance of their peers or an expert (such as a teacher) [Vyg62]. Soloway's Tools / Interfaces / Learner's needs / Tools (TILT) model [SGH94] is a classification of different types of scaffolding and their roles in the learning environment (cf. Figure 5.2). To scaffold learning tasks, software can coach a learner by providing helpful advice at appropriate points in the learning process. To support a learner's growing competence the tools in a model must be adaptable to the task in question (cf. bricolage). To support the learner at a communication level, the interface must provide different means of expression appropriate to the learners' competency.

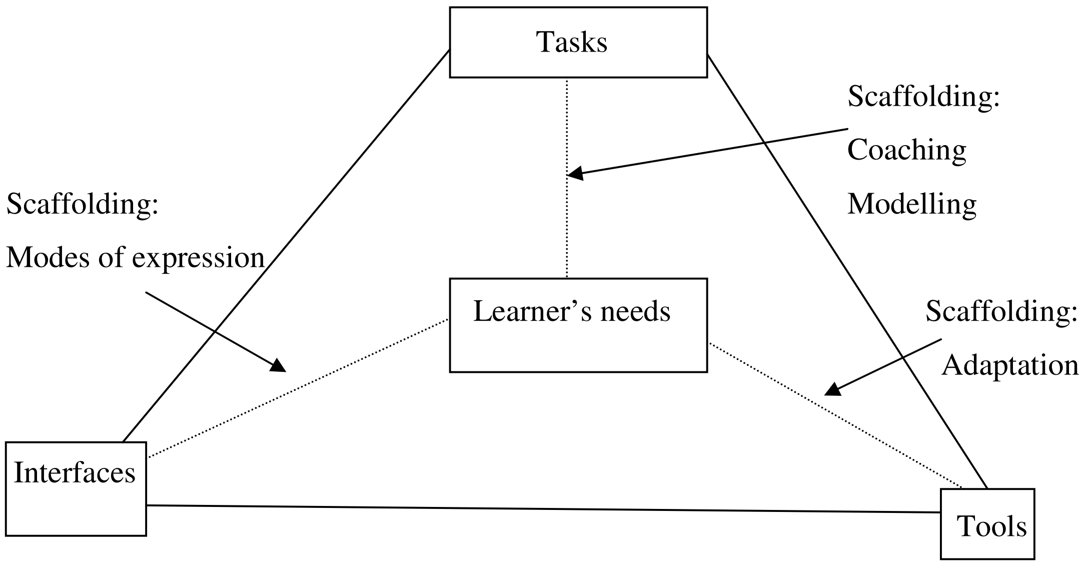


Figure 5.2 – Soloway's TILT model [SGH94]

The remaining sections of this chapter describe – and illustrate with reference to case studies – how EM learning environments can scaffold each type of learning identified in i), ii), iii) above. Our case studies also illustrate how EM could be exploited in different types of scaffolding similar to those identified in Soloway's TILT model, namely through scaffolding – for Tasks (section 5.2), cognitive layering – for Tools (section 5.3), and domain specific notations – for Interfaces (section 5.4).

5.2 Learning as comprehension of a fixed referent

In some learning environments, the major aim is to allow students to gain an exact understanding of a specific referent. The typical application is in modelling 'real-world' situations. In such a context, the domain being modelled is presumed to behave reliably according to some well-defined rules. For instance, balls on a snooker table will, after being struck, behave in a definite manner when bouncing off the cushions, colliding with other balls and slowing down through friction. In order to give the leaner an appropriate construal of the real-world situation the balls, as modelled, should ideally behave according to similar physical rules. Pratt associates this similarity between model and domain with the ideas of *surface* and *cultural familiarity* [Pra98]. Surface familiarity is concerned with whether objects in the computer environment look like their real-world counterparts. Cultural familiarity is concerned with whether objects in the computer environment behave like their real-world counterparts. Where models have both surface and cultural familiarity, learners can leverage prior experience of the real-world situation and can successfully transfer knowledge gained from the computer-based environment back into the world. The ideas of surface and cultural familiarity are similar to Green's notion of 'closeness of mapping' in the Cognitive Dimensions framework [GP96].

The scaffolding principle suggests that the concepts in a learning environment should be layered and introduced only when the learner has a solid understanding of simpler concepts. For example, the benefit that can be gained from a fully functioning snooker model in learning mechanics could be limited because the complexity of the complete model obscures the learning of simple concepts. Understanding the complete model requires comprehension of a number of inter-related concepts. In a snooker model for learning about mechanics, a first layer could consist of a single ball that bounces around a 2D table without ever slowing down. This could be used to explore how a ball bounces off a cushion. A second layer could introduce a concept of friction, so that the ball slows down over time. This could be used to explore forces acting on a ball. A third layer could introduce more balls and illustrate what happens when balls collide with each other. This could be used to investigate principles such as the conservation of momentum. A model constructed with these simple layers learners can serve as an exploratory environment for learning about mechanics (cf. [EMRep, billiardsMoissenkov1999] for a prototype implementation).

The notion of developing increasingly complex microworlds, where each microworld adds more complex ideas or tasks to perform, is a well-established educational strategy (cf. Burton et al [BBF84]). The various layers of the snooker model can be viewed as a series of graded microworld instances in the sense of Graci et al [GON92]. This means that each microworld builds on the concepts in the previous level to provide a more complete picture of the real-world situation portrayed in the model. As Graci et al note [GON92], these graded microworld instances are essentially increasing subsets of the functionality of the complete learning environment. These microworld instances provide the means for scaffolding the domain.

In the next section, we consider a case study that illustrates the idea of scaffolding in a practical EM learning environment, where the emphasis is on construing how the real-world domain of car racing works.

5.2.1 The racing cars case study

In this section, we describe an EM learning environment targeted at exploring factors that are important in car racing. Applying Pratt's principles to this real-world situation [Pra98], the computer-based learning environment must be recognisable as a car racing environment ('surface familiarity'), and the cars should be set up to behave like their real-world counterparts so that learners are able to draw on their prior experience of the domain ('cultural familiarity'). The racing cars model was constructed by Simon Gardner in 1999 [EMRep, racingGardner1999] and takes the form of a series of increasingly complex microworlds. The final microworld contains two customisable cars racing each other around a partially customisable track. The full functionality of the model is implicit in each microworld, but only a subset of that functionality is exposed to the learner. There are seven microworlds in Gardner's model, and we discuss four that give a flavour of the increasing complexity of the concepts being introduced. Figure 5.3 shows the main concepts in Gardner's seven microworlds and highlights the four discussed in this section.

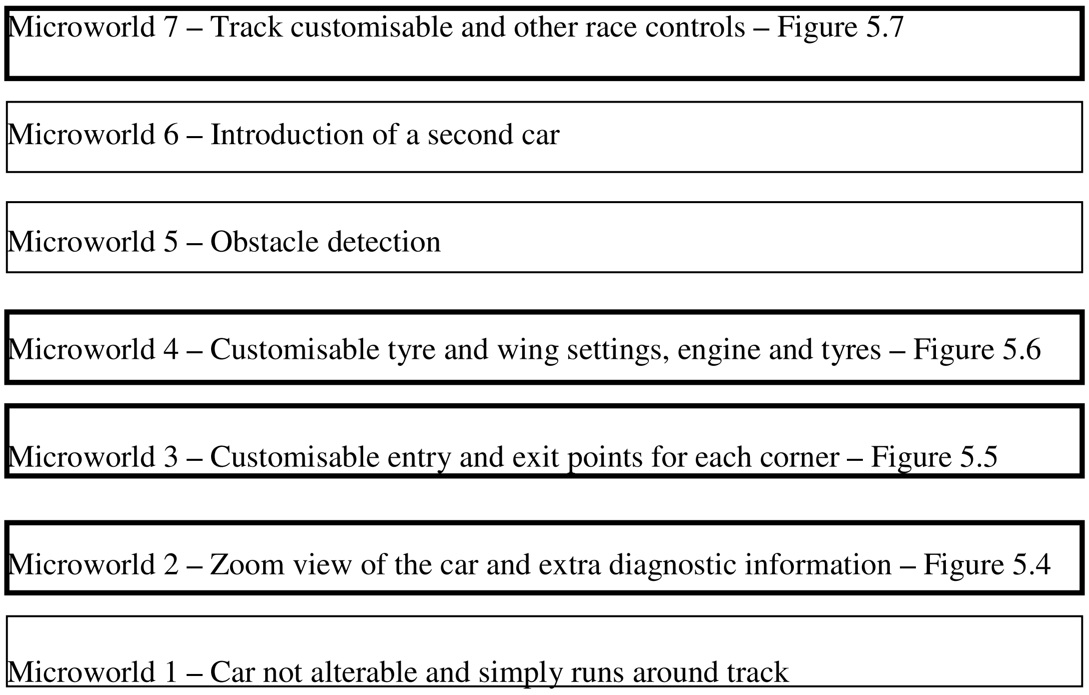
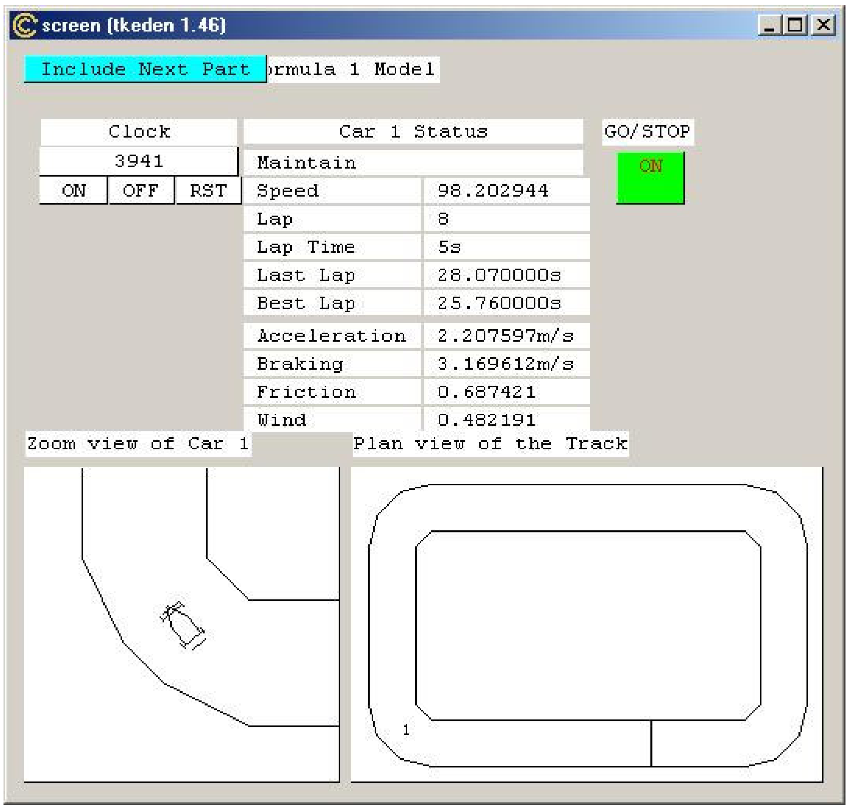

Microworld 7 – Track customisable and other race controls – Figure 5.7
Microworld 6 – Introduction of a second car
Microworld 5 – Obstacle detection
Microworld 4 – Customisable tyre and wing settings, engine and tyres – Figure 5.6
Microworld 3 – Customisable entry and exit points for each corner – Figure 5.5
Microworld 2 – Zoom view of the car and extra diagnostic information – Figure 5.4
Microworld 1 – Car not alterable and simply runs around track

Figure 5.3 – The microworlds in the racing car model

Microworld 2 (see Figure 5.4) shows a car that is moving around a track. The learner is able to observe many significant attributes of the car, but there are no controls for the learner to experiment with the car and its environment. The ‘Car 1 status’ table contains information about the car such as its acceleration, braking, friction and wind resistance. The acceleration and braking values refer to the change in speed that will occur in the next clock cycle if the car is accelerating or braking respectively. On the plan view of the track, the symbol ‘1’ will move around the track. The zoom view on the left depicts the car and its neighbourhood in more detail.

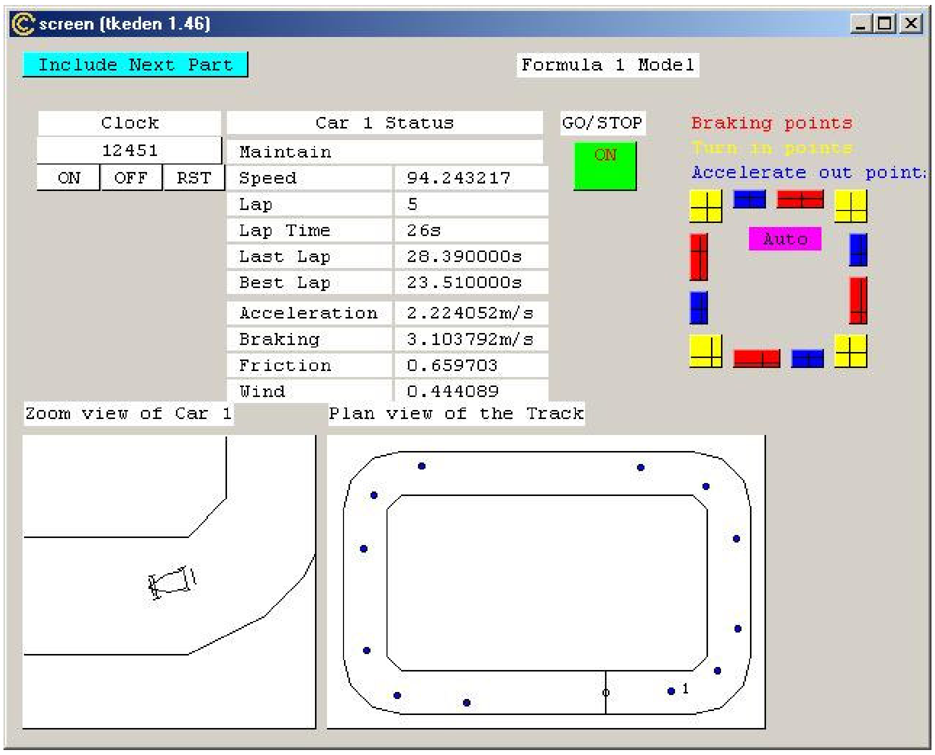


Car 1 Status	
Maintain	
Speed	98.202944
Lap	8
Lap Time	5s
Last Lap	28.070000s
Best Lap	25.760000s
Acceleration	2.207597m/s
Braking	3.169612m/s
Friction	0.687421
Wind	0.482191

Figure 5.4 – Microworld 2 of the racing cars model

In this elementary microworld, learners can observe how patterns of acceleration and braking are correlated with the motion of the car and its position on the track. This can be used to gain a basic understanding of how the car accelerates and brakes in cornering, and how concepts such as wind resistance and friction are related to car speed. These concepts form the necessary background for exploring how to move the car around the track faster in the more advanced microworlds.

Microworld 3 (see Figure 5.5) builds on the previous microworld and introduces the concept of braking, turn-in and accelerate-out points for corners. These respectively refer to the key control points on the track where the car will begin to brake for a corner, where it will start turning inwards to take the corner, and where it will start to accelerate away from the corner. The set of crosshairs in the top right of the interface can be used to alter the significant points for each corner. Clicking on a different position in each rectangle will move the corresponding point on the track. This selection method restricts the key control points to a sensible region of the track. Changes to the points can be made at any time, even whilst the car is approaching the point being moved.



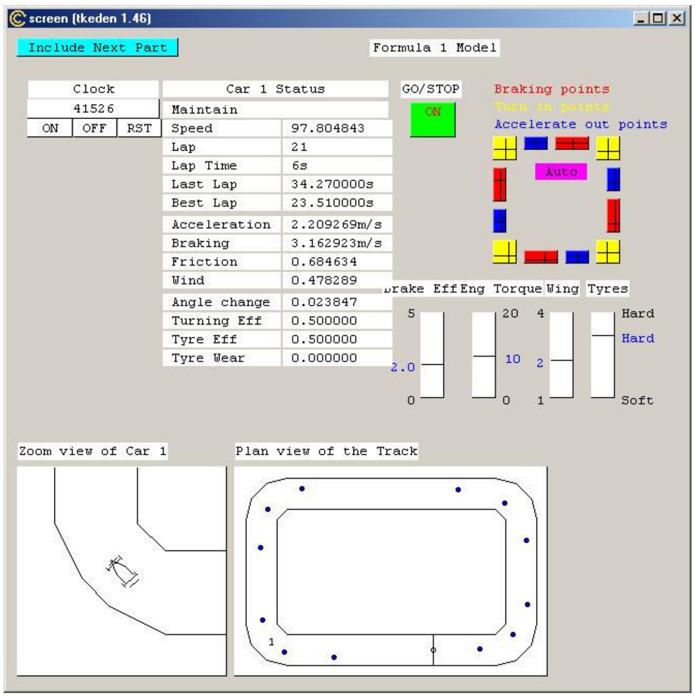
Clock
12451

Car 1 Status	
Maintain	
Speed	94.243217
Lap	5
Lap Time	26s
Last Lap	28.390000s
Best Lap	23.510000s
Acceleration	2.224052m/s
Braking	3.103792m/s
Friction	0.659703
Wind	0.444089

Figure 5.5 – Microworld 3 of the racing cars model

By manipulating the key control points and monitoring changing lap times, learners can explore the positioning of points required to achieve optimum car performance. If a car brakes too late for a corner then it will not stay on the track. In a stable situation, the speed at each position on each lap will be the same. If a car is going faster at the same track position on each successive lap then eventually the car will miss a corner and leave the track. These characteristics of car racing can be appreciated by interacting with the model. Additional insight could be obtained from the model by linking it to an auxiliary model to plot graphs of speed against lap position and so more easily observe how the car responds to the key track points being changed. In this microworld, the behaviour of the car on the circuit can be explored, but only for a given car set-up. In reality, the car could be set up in many different ways. This additional complexity is introduced in the next microworld.

Microworld 4 (see Figure 5.6) builds on the previous microworlds and allows exploration of how factors associated with the design of the car, such as braking efficiency, engine torque (power), tyres and wing settings affect the behaviour of the car on the track. For instance, an increase in braking efficiency means that the car can brake later for each corner. Likewise, an increase in engine torque means that the car will accelerate faster so that the braking point for each corner must occur further in advance of the corner.



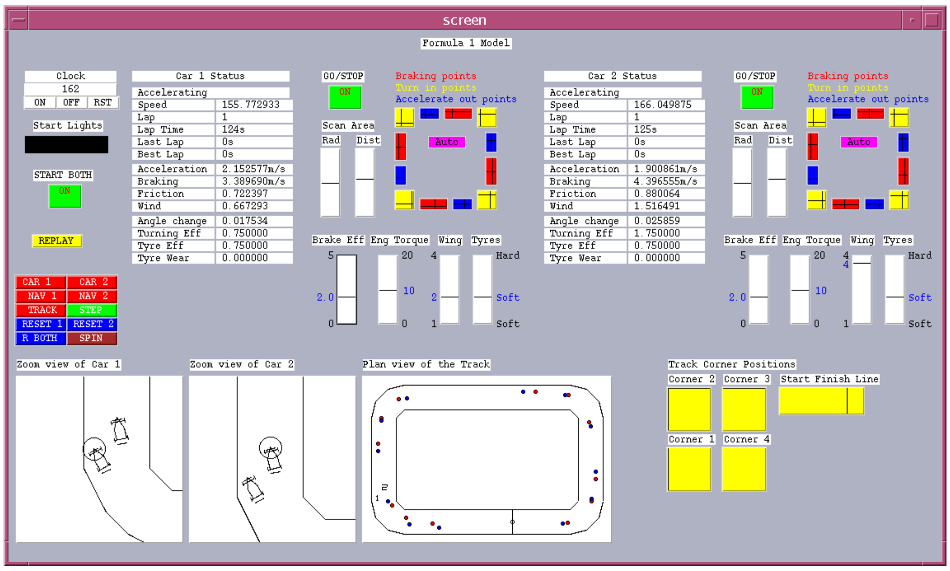
Clock
41526
ON · OFF · RST

Car 1 Status	
Maintain	
Speed	97.804843
Lap	21
Lap Time	6s
Last Lap	34.270000s
Best Lap	23.510000s
Acceleration	2.209269m/s
Braking	3.162923m/s
Friction	0.684634
Wind	0.478289
Angle change	0.023847
Turning Eff	0.500000
Tyre Eff	0.500000
Tyre Wear	0.000000

Brake Eff	Eng Torque	Wing	Tyres
5	20	4	Hard
2.0	10	2	Hard
0	0	1	Soft

Figure 5.6 – Microworld 4 of the racing cars model

There is a trade-off between the tyre setting and the wing setting. This trade-off can be explored through experimenting with different settings and observing the effect on lap times. Empirical investigation plays an essential role in learning about this trade-off and forms a large part of the experimental testing undertaken by Formula 1 teams. In Figure 5.6, tyre efficiency, tyre wear, turning efficiency and turning angle are also displayed in the 'Car 1 Status' table. Microworld 4 addresses learning objectives for which modelling is essential, such as finding the optimum route around the track for the current car specification, and the optimum set-up of the car for the current track.



Car 1 Status	
Accelerating	
Speed	155.772933
Lap	1
Lap Time	124s
Last Lap	0s
Best Lap	0s
Acceleration	2.152577m/s
Braking	3.389690m/s
Friction	0.722397
Wind	0.667293
Angle change	0.017534
Turning Eff	0.750000
Tyre Eff	0.750000
Tyre Wear	0.000000

Car 2 Status	
Accelerating	
Speed	166.049875
Lap	1
Lap Time	125s
Last Lap	0s
Best Lap	0s
Acceleration	1.900861m/s
Braking	4.396555m/s
Friction	0.880064
Wind	1.516491
Angle change	0.025859
Turning Eff	1.750000
Tyre Eff	0.750000
Tyre Wear	0.000000

Figure 5.7 – Microworld 7 of the racing cars model

Microworld 7 (see Figure 5.7) shows two fully customisable cars that can be raced around a partially customisable track. In this final microworld each car has a means for obstacle detection: each car avoids obstacles within a scan area of specified radius and distance from the front of the car. There are also facilities for editing the corners of the track and the position of the starting line. One interesting area for exploration is trying to change the set-up of one car to beat the other around the track. Microworld 7 reflects the construal an expert has when exploring car racing situations. If all the set-up options in Figure 5.7 were available in an initial microworld, a learner would be likely to be overwhelmed by the complexity of the model. The object of constructing the racing cars model is to enable a learner to come to appreciate this level of complexity.

The racing cars environment is constructionist because the learner is free to experiment and is not given a set of questions to answer. From a personal viewpoint, the experience gained through interaction with the model proved useful to me in developing my cornering technique the first time that I went go-karting. However, in such an environment, there is a limit to the level of creativity and invention that a learner can exhibit because the fixed referent constrains the construal that is being explored. In the next section, we consider more abstract microworlds where meaningful interaction is not constrained by a fixed referent and there is learning benefit in open exploration.

5.3 Learning as exploring possibilities and invention

Where the layering of microworlds is constrained by a fixed referent it is inappropriate to allow learners to interact in ways that subvert their emerging understanding. In other contexts, it can be beneficial for a learner to explore unrealistic scenarios and invent scenarios of their own. This is typically the case where the learning activity is directed towards design or invention rather than mere comprehension. For instance, in designing a new board game, learners can benefit from tinkering with the rules of an existing game in an *ad hoc* way that reflects the open-ended nature of experimentation in the world. Developing learning environments for this purpose requires a different style of scaffolding for which we have introduced the term *cognitive layering*.

5.3.1 Cognitive Layering

Scaffolding in educational software is analogous to scaffolding for building houses. This type of scaffolding is sensible if the learning environment is targeted at a learning objective that can be attained through a well understood progression of stages. In such a context, the form of the scaffolding is itself shaped by the prior knowledge of the overall structure of the learning task. Not all learning tasks can be so structured or have such clear objectives from the start. A different type of scaffolding is required for these tasks. To appreciate this, imagine that, during the construction of a building, its plans were to be changed. This might well mean that the building itself would become impossible to construct with the existing scaffolding.

The term 'cognitive layering' describes an approach to scaffolding microworlds that takes the fact that learners can benefit from open exploration into account [RB02]. To support this open exploration, it is essential to offer the learner less restricted access to the underlying data model than is afforded by closed interfaces such as can be found in the racing cars model. Such open interaction is supported in our principal EM tool TkEden through the specification of redefinitions in the input window. Open interaction is necessary because the designer cannot preconceive the possibilities that a learner may want to explore. With conventional scaffolding, the complete model is preconceived and the learner only has access to a partial subset of its functionality. In cognitive layering, future layers are not preconceived, and can be flexibly adapted in any direction. Figure 5.8 depicts this essential difference between scaffolding and cognitive layering.

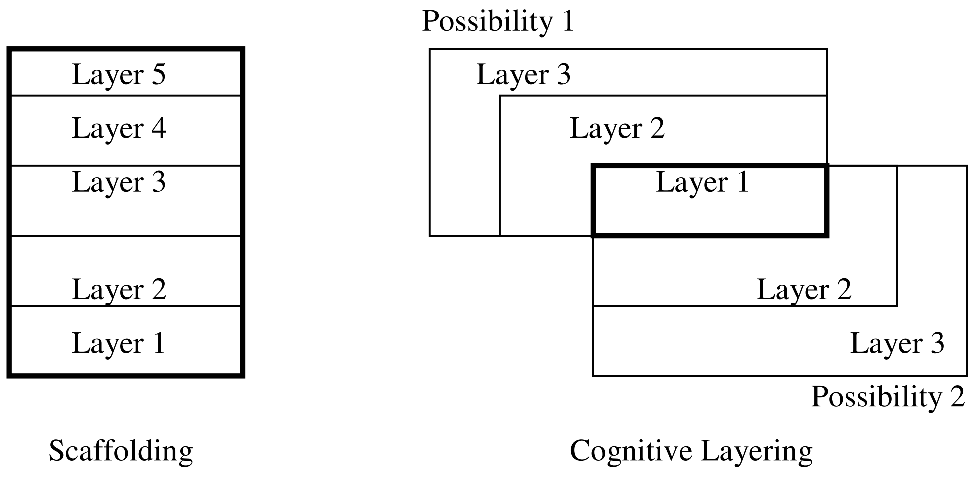


Figure 5.8 – Differences between scaffolding and cognitive layering

We now describe an EM case study in the form of a laboratory for investigating noughts-and-crosses style games.

5.3.2 The noughts-and-crosses case study

Noughts-and-crosses is a simple two-player game. Players take turns to place counters on a 3x3 grid aiming to make a straight line containing three of their own counters. From a strategy perspective, noughts-and-crosses is simple because of the limited number of different games that can be played. However, for children the game can be a challenge, as evidenced by Lawler's research into children's learning that used noughts-and-crosses as a case study [Law85]. In this section, we describe how a series of microworlds to investigate the game of noughts-and-crosses illustrates the idea of cognitive layering in practice. The scope of this investigation embraces a whole family of noughts-and-crosses style games to be referred to generically as OXO games.

The EM OXO model has been developed by a number of people over the past 10 years. The initial model ran in a textual interface and was developed by Meurig Beynon and Mike Joy in 1994 [BJ94]. Simon Gardner added a graphical interface in 1999 [EMRep, oxoGardner1999]. I adapted the model to create a 3D version of OXO using the Sasami notation in 2001 [EMRep, 3doxoRoe2001]. The OXO model that is illustrated in this section is Gardner's 1999 version.

The OXO model is a layered series of four microworlds, each of which introduces concepts not in the previous layer. This is in contrast to the racing cars model described in 5.2.2, where the concepts of tyre compound and wing settings were present in every layer of the model, but were initially inaccessible to the learner. In the OXO model, each microworld embellishes the situation by building upon the previously introduced concepts. Successive microworlds specialise the OXO model so that it more closely resembles the game of noughts-and-crosses [BJ94]. The layers of the OXO model are depicted in Figure 5.9. These layers reflect: the layout and geometry of the board; the placing of pieces on the board; rules governing the playing of pieces; and strategic play.

Microworld 4 – Issues of strategic play, the complete model – Figure 5.12

Microworld 3 – Defining how the pieces can be placed on the board

Microworld 2 – Defining the pieces that can be placed on the board – Figure 5.11

Microworld 1 – Defining the board that the game is to be played on – Figure 5.10

Figure 5.9 – The structure of the OXO model

At all times, the learner can alter aspects of the OXO model by redefining relevant parts of the model using the TkEden input window. In the OXO model, a specific learning path directed towards learning conventional noughts-and-crosses has been mapped out by the model designer. At every layer, redefinitions allow the learner to deviate from the mapped out path to explore variants in the OXO family. The discussion of the layers that follows is illustrated by examples of experimental redefinitions. It is important to note that the experiments that can be carried out are only limited by the learner's imagination.

Microworld 1 specifies the geometry of the board and the concept of lines upon it (see Figure 5.10). There is no presumption about the desired functionality, except that there is a regular geometric board with significant lines. When learning to play a new board game, our attention is initially directed to the geometry of the board and its important features. In our OXO model, the significant lines are highlighted through animation as displayed in Figure 5.10.

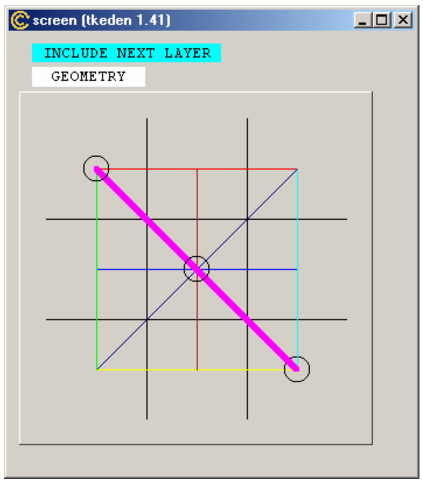


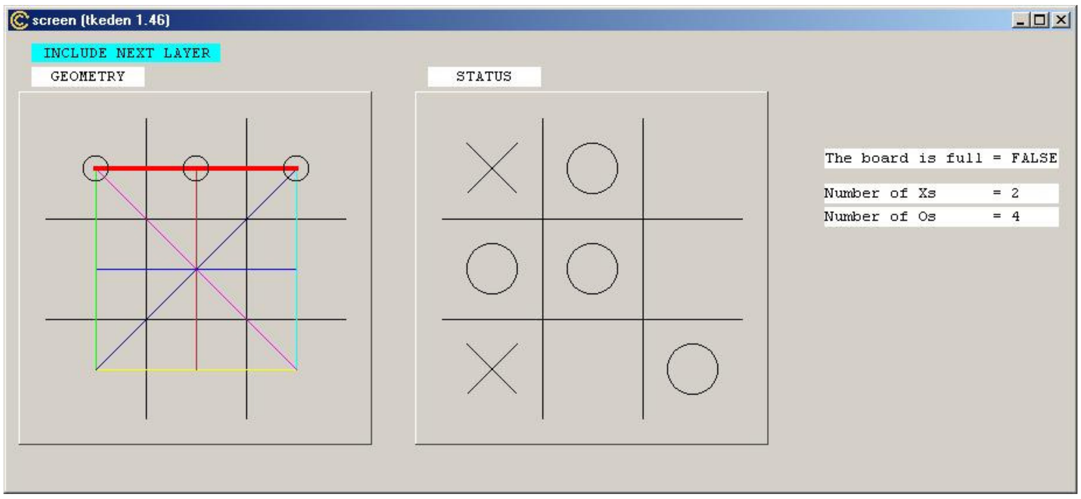
Figure 5.10 – Microworld 1 of the OXO model

In keeping with the theme of learning through exploring possibilities, changes can be made to the board. For example, the significant lines on the board can be redefined individually, or as a whole. The example redefinition:

```
    lines[1] = [1,8,3];
```

will replace the horizontal line across the top row of the board by a new 'line' that contains the top left square, the bottom middle square and the top right square. The significance of such a redefinition is not apparent in this microworld, but in later microworlds this would have an impact on winning conditions and good strategic play. In the EM OXO model, the scope for adaptation of this nature is not preconceived. In contrast, educational software designers usually preconceive the useful adaptations that a learner can make in order to guide the learner to explore the possibilities that are deemed important by the designer.

Microworld 2 introduces the concept of placing pieces on the board (see Figure 5.11) and the criteria for a winning position. There are no restrictions on where pieces can be placed or on the order in which they should be placed. This might reflect a situation where players have yet to decide upon the rules of play. The interface in Figure 5.11 displays information that can be ascertained about the state of the board from a static analysis of a position. This includes the number of pieces of each type and whether the board is full. These can be determined by observing the current state of the board.



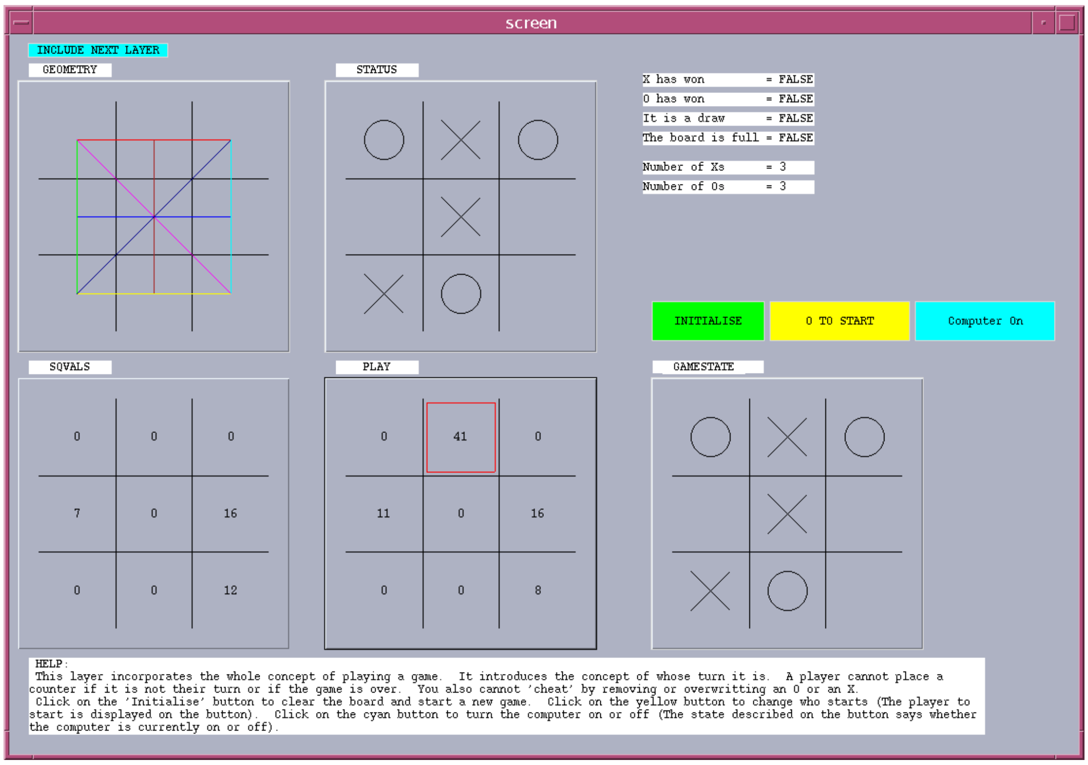
```
The board is full = FALSE
Number of Xs      = 2
Number of Os      = 4
```

Figure 5.11 – Microworld 2 of the OXO model

This microworld resembles a laboratory where learners can explore the placing of pieces without adhering to any rules of play. It can be used, for instance, to experiment with OXO strategies, or to devise new OXO-like games. For example, learners can experiment with different ways of placing pieces, since the model imposes no restriction on the placement of pieces. Such unrestricted interaction would be outside the scope of a conventional environment for learning about noughts-and-crosses.

Microworld 3 (figure not shown) introduces the concept of playing rules that are characteristic of an OXO-like game. For example, in noughts-and-crosses the rules are simple; players take turns to place pieces and a piece cannot be placed on an occupied square. This microworld is the first layer in which there are two players who are constrained to play according to the current rules. Note that, in keeping with our aspiration to develop an open learning environment, this microworld can reflect the extraordinary variety of ways in which playing rules can be enacted in the world. For instance, we might imagine that players take turns to throw a piece onto the board and forfeit their turn if their piece does not land on an empty square.

Microworld 4 introduces the issues of strategy required to construct an automatic OXO player (see Figure 5.12). This involves two aspects: evaluating the squares on the board and deciding on the 'best' move to make. The value of individual squares is dependent on the state of the board, the rules of the game being played and the evaluation function being used. For example, in noughts-and-crosses the value of a square is dependent on the number of lines that pass through it and the number of pieces already in each line. In this OXO microworld, the learner can investigate the factors that are important in the positional evaluation of OXO boards by tinkering with the scores for each type of line identified.



```
X has won          = FALSE
O has won          = FALSE
It is a draw       = FALSE
The board is full  = FALSE

Number of Xs       = 3
Number of Os       = 3
```

```
HELP:
 This layer incorporates the whole concept of playing a game.  It introduces the concept of whose turn it is.  A player cannot place a
counter if it is not their turn or if the game is over.  You also cannot 'cheat' by removing or overwritting an O or an X.
 Click on the 'Initialise' button to clear the board and start a new game.  Click on the yellow button to change who starts (The player to
start is displayed on the button).  Click on the cyan button to turn the computer on or off (The state described on the button says whether
the computer is currently on or off).
```

Figure 5.12 – Microworld 4 of the OXO model.

5.3.3 Case study – Adapting layers to form a family of models

The previous section considered the benefits of cognitive layering of microworlds from the perspective of the learner. There are also advantages in using cognitive layering for the developers of models, since microworlds can be extended in different directions to create a family of models. In the EM OXO model discussed above, each successive microworld constrains the model to more accurately resemble the game of noughts-and-crosses. Adding a different set of rules creates a variant of noughts-and-crosses. In this way, games in the OXO family can be created by reusing some of the original microworld layers. In this section, we give examples of variations that can be introduced at each layer. This leads to a tree of possible models, as depicted in Figure 5.13.

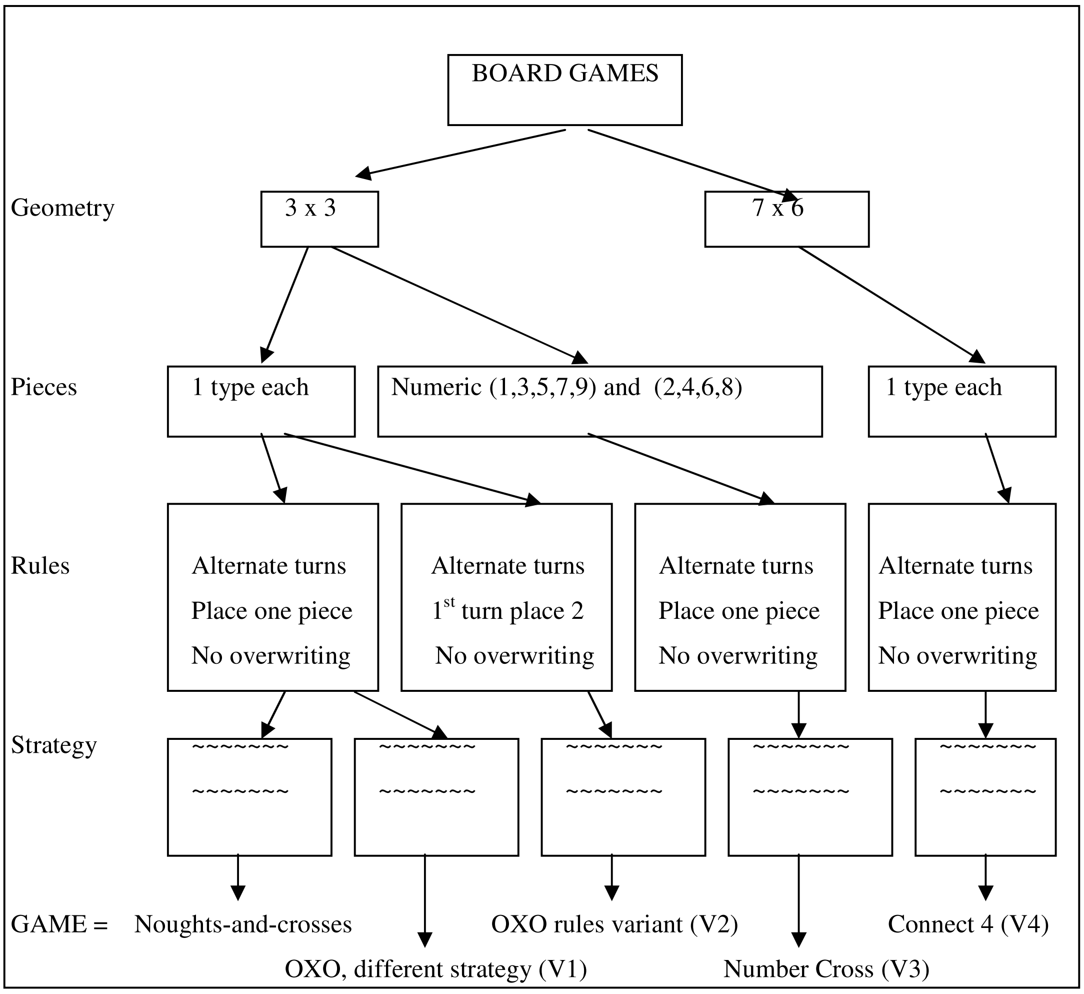


Fig 5.13 – A tree of possible models based on the cognitively layered OXO model

In Figure 5.13, four variants (V1, V2, V3, V4) are created by reusing some of the original OXO model. If any layer is altered, then the subsequent layers will, in general, be different. For example, changing the rules of noughts-and-crosses will probably mean that a different strategy is required. The variants outlined in Figure 5.13 are illustrative of the kind of adaptation that a teacher might want to carry out in order to customise learning resources.

V1 – Altering the computer strategy.

The computer OXO player described in section 5.2.2 contains a serious flaw because a particular pattern of opposition moves is guaranteed to lead to a win. In this variant, we adapt the computer player to eliminate this defect in its play. In the original model, the computer player does not use a minimax algorithm (see [BB96]) but simply analyses the set of lines incident with each square to compute its value. The value of a square is dependent on the number of pieces in the line, and these values are summed to give each square an overall value (see Table 5.1). In EM terms the values `weight1`, …, `weight5` can be regarded as observables for the computer player and can be changed to alter the way the computer plays.

Condition	Observable	Value
X X _, X _ X, _ X X	`weight1`	100
O O _, O _ O, _ O O	`weight2`	40
X _ _, _ X _, _ _ X	`weight3`	10
O _ _, _ O _, _ _ O	`weight4`	6
_ _ _	`weight5`	4

Table 5.1 – The evaluation strategy for player X in OXO

Using this evaluation routine, the computer player would respond to the game situation in Figure 5.14 by playing in the bottom left, as indicated by the highlighted square. The value of 22 attached to this square can be construed as the result of a particular 'mode of observation' on the part of the computer player. The threatened winning diagonal from bottom left to top right contributes 10 to the value of the square (cf. `weight3`). The blocking of the left and bottom lines each contributes 6 to the value of the square (cf. `weight4`). The opponent can respond by playing in the top right square thereby blocking the potential diagonal winning line and creating two winning squares for their next move.

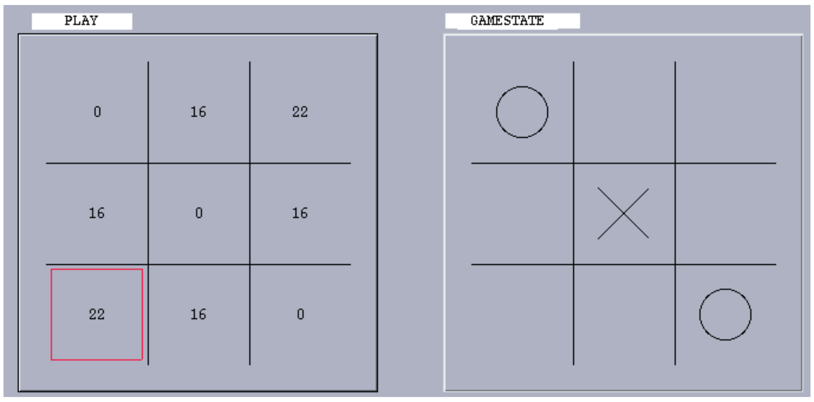


Figure 5.14 – A problem situation for the OXO computer player.

The problem with the existing evaluation routine is that the computer player does not observe situations where the opponent can introduce a *fork*: a situation where the opponent can make a move that sets up two independent ways to win. An extra condition to recognise fork situations, together with a change to the evaluation routine, changes the strategy of the computer player to avoid the trap in Figure 5.14.

This example is fairly trivial due to the simple nature of noughts-and-crosses. In more complex games such as chess, changes to the computer player could alter its strategy to play defensively, to attack, or to try and control particular important squares. The OXO variant described in this example uses the same board, the same pieces and the same rules as noughts-and-crosses. The only difference is in the strategy of the computer player. EM principles are well-suited for making changes of this nature, which involve changing the way in which the computer player is construed to observe the state of the game. The above example shows how our modelling principles enable the computer strategy to evolve through experimental interaction. Papert has observed that children use a similar style of development when writing computer programs to play noughts-and-crosses [Pap93]:

> 'rather than following strictly in the path of the so-called "knowledge engineers" who build expert systems, children followed in the path of psychologists who deliberately construct a series of "inexpert" systems that made the computer act like a "novice" and then pass through a progression of levels of increasing expertise'.

It would be of particular interest to adapt the computer player so as to model human strategies employed in learning to play noughts-and-crosses. Understanding of good strategic play emerges from experience of the game. Learners, especially children, cannot initially expect to fully understand how to play a good game. Lawler's extensive study of how an individual child learnt to play noughts and crosses supports this claim [Law85]. His study led him to recognise four stages of comprehension in playing noughts-and-crosses, namely [Law85]:

i) Naive comprehension – the learner's play is guided by individual inclinations but with no idea about how to achieve particular outcomes. They typically move anywhere for obscure reasons.

ii) Fragmentary comprehension – the learner acts on the basis of highly specific knowledge of one or two games. They typically respond in a rigid way, independent of the opponent's strategy.

iii) Procedural comprehension – the learner can recognise situations in which victory can be forced. They typically know when they are going to win before their opponent plays their last piece.

iv) Systematic comprehension – the learner is familiar with all the possible game situations and appropriate responses (cf. Lawler's comprehensive classification of noughts-and-crosses games – such classification is only possible for simple games). They typically make the optimum move at all times.

In applying EM principles to model these particular stages in learning we would adapt the computer player to reflect the observation and construal of the child at their current level of competency. This would also be a way of providing an appropriate opponent to scaffold the child's learning of noughts-and-crosses at each of Lawler's stages.

V2 – Altering the rules of the game

Variant 1 only differs from the standard OXO model of noughts-and-crosses at the strategy layer. Variant 2 differs from the standard model at the rules layer; the board and the pieces placed on it are the same as for the game of noughts-and-crosses. In variant 2, the standard rules of noughts-and-crosses have been changed so that, on their first turn only, each player can place two pieces. In the OXO model, there are definitions for whose turn it is to play. To make the simple change to the rules specified above, it suffices to replace these definitions. The new definition for player x is shown in Listing 5.1.

```
x_to_play is (!end_of_game) &&
  ( /* X is about to make their first move */
      ( (( nofx<2)&&(nofo==0)&&(startplayer==x)) ||
      ((nofo==2)&&(nofx<2)&&(startplayer==o))
  )
||
  ( /* X is about to make a subsequent move */
      ((nofo>nofx)&&(nofo>1)) || ((nofo==nofx)&&(startplayer==x))
  )
);
```

Listing 5.1 – The new definition that describes the state of the board for when player x should play.

In general, a change to the rules of a game also requires a change of playing strategy. The best playing strategy will depend on the specific features of the game, the board, the pieces and the rules.

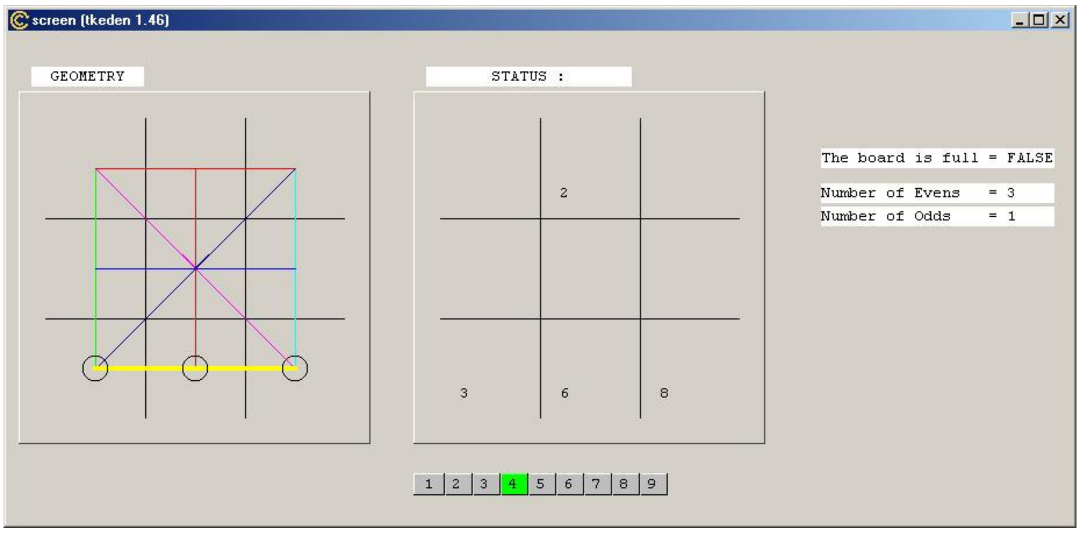
The sample rule change described above is one of a number of possibilities that is limited only by the imagination of the learner. Experimentation with different rules gives the OXO model the flavour of a 'rule laboratory' in which the implications of different rules for the game can be explored. Many changes can be made: rules can be added to an existing set; existing rules can be redefined; or rules can be removed.

Educational arguments have been made in favour of acquainting pupils with the notion of devising and adapting rules because of the significant part this plays in social behaviour [BFP+03]. On this basis, experimentation with rules is perceived as an important educational activity in programming games within the Pathways programming environment [GKN+01]. In Agentsheets (see section 2.3.3), the creation and manipulation of agents' rules is the main programming activity [RRP+98]. Both Agentsheets and Pathways are rule-based programming systems and the rules are obligations to agents to behave according to the specified rules. Rule-based programming is recognised to be a problematic way of specifying behaviour since changes to rules are liable to lead to instability and incoherence [Akm00]. For instance, the requirement to bind rules to agents can lead to difficult issues relating to object-orientation (cf. [GKN+01, section 6.1]).

From an educational perspective, it is important that the semantics of rules in programming should conform as closely as possible to that of rules in the world. In EM, rules are implemented in conjunction with dependencies in such a way as to guarantee that the integrity of state is preserved: the maintenance of dependency is not itself the product of user-specified rule-based action (cf. the discussion of Agentsheets in section 2.3.3). This makes it much easier to imitate the semantics of real-world rules inside EM models. For instance, the rules in the EM OXO model place a constraint on the moves that can be made and so shape the referent. This shifts the emphasis from using rules to maintain the semantic relation α to using rules to maintain the semantic relation β (cf. section 2.2.2).

V3 – Altering the pieces that are being played

This variant of the OXO model only re-uses the board layer. The second layer of the OXO model describes the pieces to be used. We will refer to the game in this section as *number cross*, a game based on noughts-and-crosses. In noughts-and-crosses, each player has just one type of piece. Number cross uses the same board as noughts-and-crosses but uses the numbers 1...9 as the pieces. The aim of the game is to complete a line of three numbers that sum to 15. Figure 5.15 shows the number cross model. At this layer, the rules of the game are as yet unspecified.

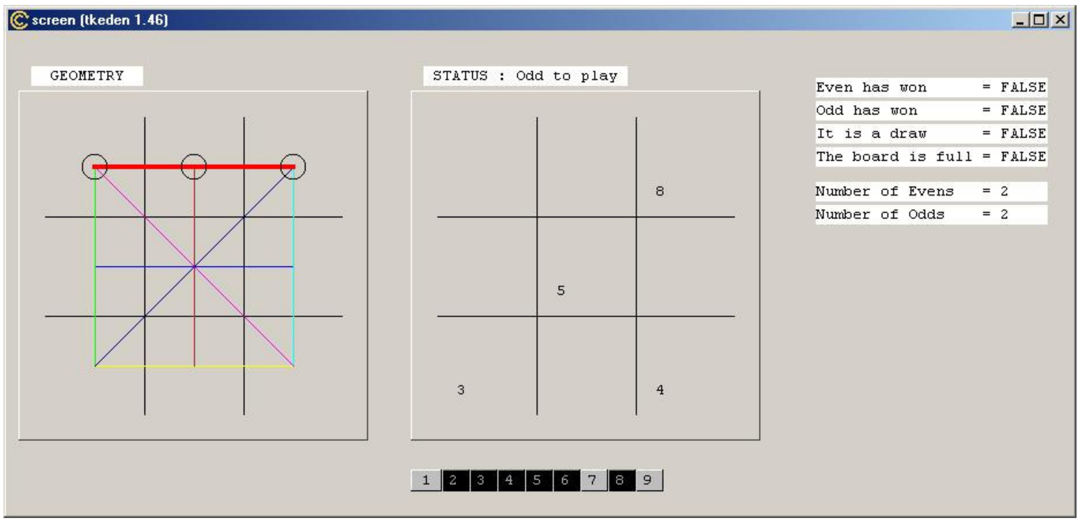


```
The board is full = FALSE
Number of Evens   = 3
Number of Odds    = 1
```

Figure 5.15 – The board and pieces of the number cross model

The interface in Figure 5.15 allows each player to select a piece to place on the board – a feature that is not required for noughts-and-crosses. There are no restrictions on where a number may be placed, or any rules governing who can play a particular number.

In number cross, the players take turns to place pieces. The player who starts ('odd') can only place odd numbers and his opponent ('even') can only play even numbers. Each number can only be used once and may only be placed in an empty square. These rules are introduced into the model at the next layer. Figure 5.16 shows the game with the rules layer added.



```
Even has won       = FALSE
Odd has won        = FALSE
It is a draw       = FALSE
The board is full  = FALSE

Number of Evens    = 2
Number of Odds     = 2
```

Figure 5.16 – The game of number cross with the rules present.

In order to model the game of number cross, the rules of noughts-and-crosses have been extended through adding game rules such as:

```
n1valid is isodd(player) && not_used(1);
```

to indicate that piece 1 can only be placed by the 'odd' player and it is not already on the board.

The strategy layer for the number cross model is quite different from that of the noughts-and-crosses model because winning lines can use opponent's pieces.

The example described in this section illustrates that different board games can be constructed with little revision. This ease of revision can be of educational benefit when teachers can adapt the model to take advantage of a particular learning situation. For instance, a related game to number cross is 'the game of 15' where there are no restrictions on which pieces can be played. There is an intimate relationship between the game of 15 and noughts-and-crosses. If the numbers 1, 2, ... , 9 are placed on a 3x3 grid so that they form a magic square, then the game of 15 is equivalent to noughts-and-crosses. Learners and teachers can use the game of 15 as a base from which to explore simple properties of odd and even numbers and the mathematics of magic squares.

V4 – Altering the board

The above sections have illustrated how the EM OXO model can be adapted through systematically replacing layers. The purpose of introducing this final variant is to illustrate that the OXO model can serve as a template from which to construct more general board games. This level of adaptation is typical of what might be required in educational contexts. Each of the variants discussed so far has used some of the original OXO model. Even when we change the board, it is still possible to maintain the same layered structure of the model. By way of illustration, in a game of Connect 4, the vertical ‘board’ has 7 columns and 6 rows, and a winning configuration is one in which four pieces of the same type lie on a line of contiguous squares. The Connect 4 model has the same layered structure as the OXO model, but differs at every layer. A layered model of Connect 4 can be found in the EM repository [EMRep, connect4Roe2003].

The model development discussed in this section illustrates principles that offer advantages to model developers, teachers and learners. These can be summarised as follows:

- For the model developer – the structured design exhibited in the OXO model is an aid to reuse (cf. design patterns in OO programming [FRK+01]).
- For the teacher – cognitively layered models allow easy customisation to take advantage of opportunities offered by learning situations.
- For the learner – cognitively layered models allow issues in the neighbourhood of the original model to be explored.

In the next section, we consider how EM models can address issues of learning to communicate and represent our emerging understanding of a referent using language.

5.4 Learning languages

Talking about our experience has a fundamental role in learning about a domain. With reference to the EFL, language is associated with moving from pre-articulate interactions to objective knowledge. As the discussions of rowing and piano-playing in chapter 3 illustrate, language skills develop alongside our experience of a domain. The role of language in learning often transcends the typical use of formal language – meanings are personal, determined by situation and negotiated through interaction. In the learning context, knowledge of domain specific language develops incrementally with the learner's competency, and builds on their evolving experience and understanding of the domain.

Conventionally, computer-based languages are not well suited for adaptation to their context in use. Interaction languages in computer-based learning environments typically have their functionality fixed by a designer. A learner must interact with the language of the domain as specified by the designer. This is appropriate when the designer understands a language well, and it is not subject to change. Enabling full engagement with the learning agenda involves adapting the language to the needs of the learner. For this purpose, the language must be opportunistically adaptable: to take account of new concepts as they are encountered; or to promote new ways of interacting with existing concepts. This level of adaptability is not a feature of formal languages, which stand in a preconceived relation to the domain (cf. Figure 5.17). It is more characteristic of natural language, as when we use the same word to describe rowing on a static machine, rowing on a machine with slides and rowing in a boat (cf. section 3.2.1). The key issue is that the semantics of a formal language is abstractly specified and independent of its context of use, whereas that of a natural language develops with experience of use in context.

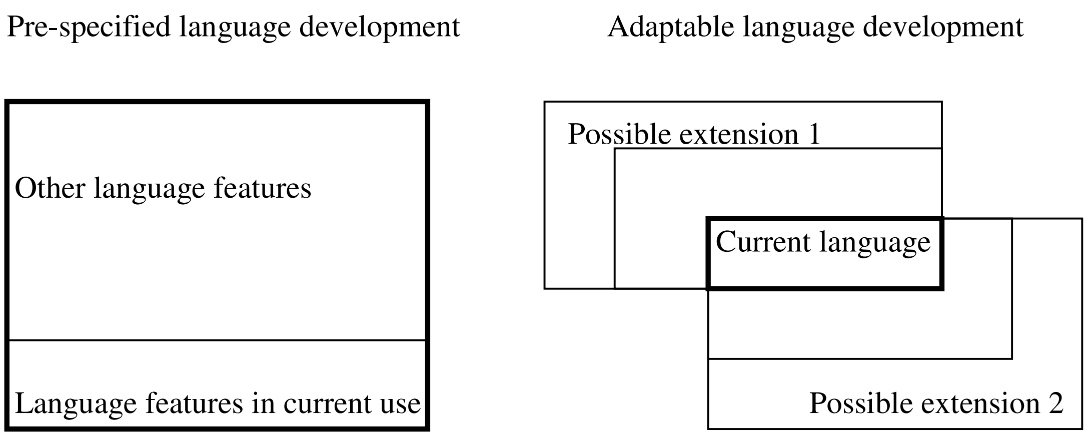

Pre-specified language development
Adaptable language development
Other language features
Language features in current use
Possible extension 1
Current language
Possible extension 2

Figure 5.17 – The relationship between pre-specified and adaptable language development

In this section, we argue that learning environments need to exploit languages that can be framed on-the-fly and opportunistically extended to match learners' competencies. This is useful for teachers in customising learning resources to take advantage of particular learning situations. In the next section, we describe an approach to interactive parsing of adaptable computer-based languages, and then illustrate it with two practical case studies.

5.4.1 The Agent-Oriented Parser

The Agent-Oriented Parser (AOP) is a utility that can be used in conjunction with the EM tool TkEden, for interactive parsing of adaptable languages. The AOP utility was respectively constructed and refined by two final year undergraduates, Chris Brown [Bro01] and Antony Harfield [Har03].

The AOP differs from a conventional parser in many respects. Instead of parsing blocks or lines from left to right, the parser searches for the most salient features of a statement, in the way that we might derive the meaning of a statement by inspection. For example, when we look at the string '`a=b+c;`' we recognise that it is an assignment by observing the = symbol, then expect to find a variable identifier on the left hand side and an expression on the right hand side. When parsing the string from left to right, symbols may have to be stored without knowing their semantic significance until the meaning of the entire statement becomes clear. The AOP also allows the parser itself to be modified on-the-fly, that is in such a way that the parsing conventions can be changed even whilst the interpreter is executing.

A full technical discussion of setting up a parser for a complete notation is beyond the scope of this thesis. Appendix B contains documentation from [Har03] that shows how an example calculator notation can be constructed. The following discussion assumes a basic level of familiarity with the parsing approach described in Appendix B.

Two key advantages of the AOP are its flexibility, and the way in which it generates parsers that can be adapted on-the-fly to suit particular learning circumstances, or to reflect a change in the language of interaction. Each AOP language contains a set of definitions that describes how the language should be parsed, together with the actions required to translate these statements into EDEN code for execution. For example, in the krusty notation (see section 5.4.2) the statement that recognises the `down` command translates this into a procedure call to move the clown in the maze (as shown in Listing 5.2).

```
krusty_statement3 =
      ["literal","down",
           ["action",
                 ["later","move_clown(3,1);"]],
                      ["fail","krusty_statement3_2"]];
```

Listing 5.2 – The example command for the down operator in the krusty language described in section 5.4.2

In building an EM learning environment, the languages that are developed with the AOP are not preconceived, and can be introduced, refined or extended on-the-fly to reflect emerging understanding or emerging requirements of a teaching/learning situation. Conventional parsing could easily be used to construct a language functionally equivalent to an AOP language, once all refinements and extensions have been specified. What is important to stress however, is that there is no restriction in how a language developed using the AOP can evolve and be reconfigured in interactive use. This is beyond the scope of conventional parsing approaches such as are described in [GBJ+00].

We now discuss two case studies that have used the AOP to define domain-specific languages. The first, the clown-and-maze environment, shows how a simple language can be defined for young children to navigate a maze and how this language can be subsequently adapted and extended to suit different learning requirements and abilities in a way that was not preconceived. The second, the SQL-EDDI environment, is targeted at learning about relational database query languages. In this example, the languages are much more complex and were developed incrementally to suit the evolving educational objectives of an undergraduate database module as it was being taught [BBR+03].

5.4.2 Case study – A clown-and-maze language

The clown-and-maze environment [EMRep, krustyRoe2002] shows how learners can be scaffolded towards understanding the Logo language [Pap93]. In this case study, we illustrate an extensible notation for young children that initially allows them to express geometric concepts in a simpler way than in Logo. In the Logo language, a turtle is controlled by giving it commands such as `forward 50` or `left 90`, which move or turn the turtle appropriately. Commands can be combined into repeating blocks, or grouped into a procedure that can be referenced by name. Papert intended young children to use Logo to explore geometrical concepts. However, the concept of ‘angle’, and even of ‘turning’, may be too sophisticated for young children. The clown-and-maze environment provides basic primitives – in the form of the krusty language – that can be used as a starting point for learning about directions and turning. The krusty language can be incrementally extended towards Logo, and in this way can provide scaffolding for understanding Logo. Figure 5.18 shows the relationship between the languages described in this section.

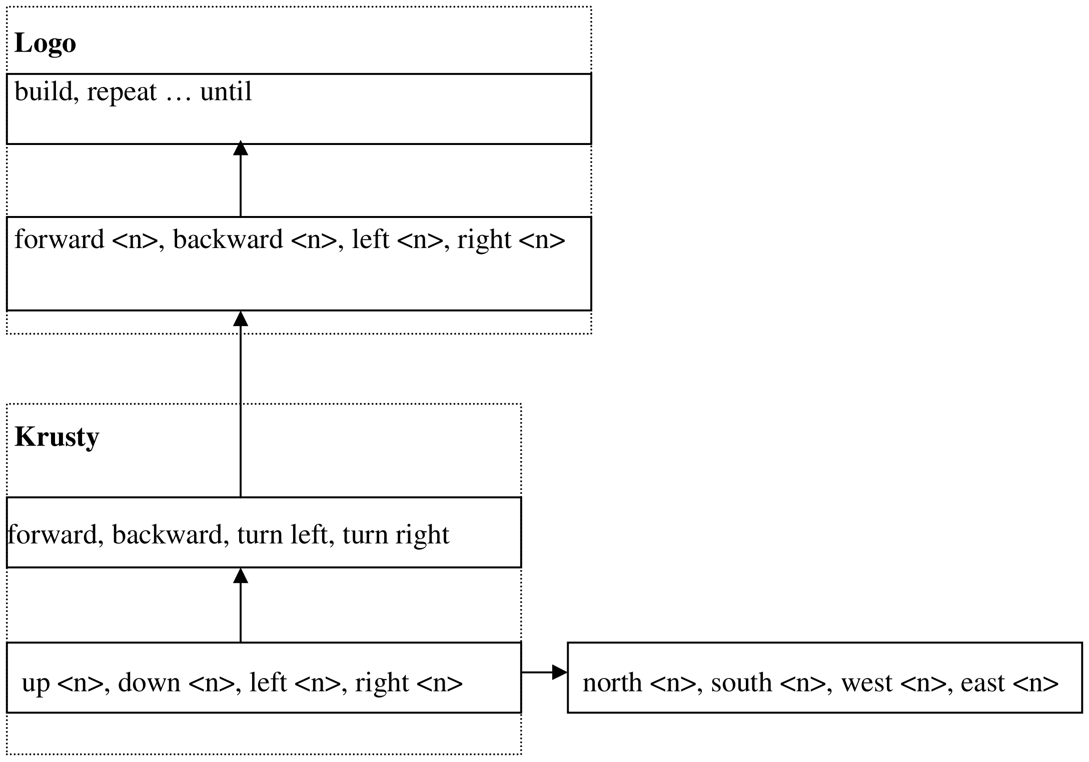

Logo
build, repeat … until
forward <n>, backward <n>, left <n>, right <n>
Krusty
forward, backward, turn left, turn right
up <n>, down <n>, left <n>, right <n>
north <n>, south <n>, west <n>, east <n>

Figure 5.18 – The structure of the clown-and-maze languages

Within the clown-and-maze environment, the learner’s task is to direct the clown to the treasure in the centre of a maze (see Figure 5.19). The maze is a 5x5 grid whose walls become visible as the clown visits the squares in the maze.

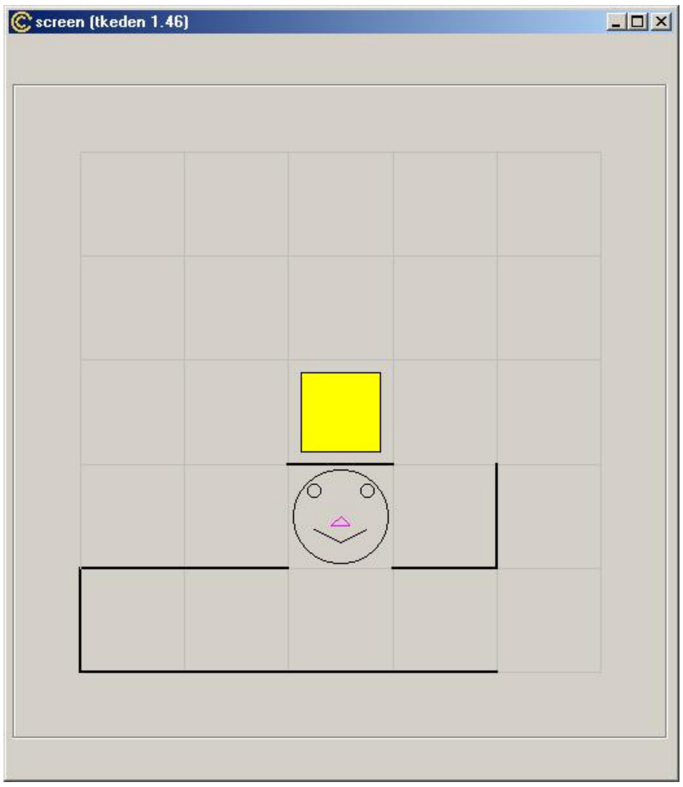


Figure 5.19 – The clown-and-maze environment.

Initially the clown can be controlled using the basic set of Krusty commands, `up`, `down`, `left` and `right`. Each of these directional commands can have an optional numeric parameter to move the clown that number of squares in the specified direction (e.g. `up 2`, `left 3`). This exposes the learner to the use of parameters, a concept required to use Logo. Krusty is a simpler interaction language than Logo because it is not necessary for the learner to take account of the way the turtle is facing.

Using the AOP, we can incrementally and interactively adapt or extend the basic krusty language to scaffold learning of more advanced manipulation languages. For example, compass directions could easily be substituted for the basic movement commands (substituting '`east`' for '`right`' etc), to satisfy different educational objectives. To scaffold Logo learning, an intermediate language that introduces the concept of turning can be introduced. Children who are in the process of learning to distinguish ‘right’ from ‘left’ could benefit from a control language where the commands are `forward`, `backward`, `turn left` and `turn right`. Success in controlling the clown now depends on understanding the concept of turning. In Figure 5.19, the point of reference for the direction the clown is facing is the tip of its nose.

The next language layer introduces the concepts of distance and angle that are found in Logo. The new commands available in this layer are `forward <d>`, `backward <d>`, `left <a>` and `right <a>`. Controlling the clown using Logo is one way of learning about angles. The clown’s nose is actually a Logo turtle, re-used from an earlier EM student project [EMRep, logoEdwards2000]. In moving the clown around the maze, the values for the angle `a` should be confined to multiples of ninety degrees and `d` to multiples of the square size, in order to keep the clown in alignment with the maze.

The clown-and-maze environment could be used to learn about more complex movements. For example, mazes could be irregular in shape, so that the learner would have to manoeuvre the clown through the maze using arbitrary angles and distances. This would refine the learner’s concepts of ‘angle of turn’ and ‘distance’. The clown-and-maze environment could also be used in significantly more advanced learning situations. For instance, notations and primitives could be designed to allow users to investigate and develop algorithms for maze solving.

The clown-and-maze case study illustrates how the learning of a domain-specific language for interaction can be scaffolded from a simple level through a number of competency levels. The AOP allows the construction of a flexible layered learning environment, where there is no restriction on how the interaction language at each layer can be extended or refined. It would not be difficult to construct a learning environment in which the interaction language would adapt dynamically to match the competency exhibited by the learner in moving the clown successfully around the maze. A more advanced illustration of the use of the AOP is described in the following section.

5.4.3 Case study – A learning environment for relational query languages

At the University of Warwick, a core 2nd year module is Introduction to Database Systems. The module aims to give students a basic understanding of relational database theory and practice. The practical component of the course comprises an introduction to SQL (Structured Query Language) and exposure to relational algebra, the mathematical language that underpins relational query languages. The objectives of the practical component of the course are to:

1) teach SQL as a relational database query language
2) get students to appreciate that relational query languages are based on relational algebra
3) get students to appreciate that SQL has a poor mathematical semantics because it is unfaithful to the relational model of query languages.

Objective 3 is the major focus of the learning environment discussed in this section.

In the past, the practical component of the course was taught exclusively using a commercial relational database system. Whilst students have undoubtedly benefited from this experience, it is not ideally suited for the learning agenda outlined above. In particular, commercial database systems are not designed for highlighting the flaws in the design of SQL and so give little support for learning objective 3. A special purpose environment targeted at this objective could show how the design of SQL deviates from the relational model it supposedly embraces [Dat00, DD00].

The SQL-EDDI environment [EMRep, sqleddiWard2003] was developed by Meurig Beynon from an original prototype developed by EM group members Chris Brown, Michael Evans and Ashley Ward. It allows learners to interact with tables and views using [BBR+03]:

- a pure relational algebra query language ("EDDI")
- a variant of SQL whose semantics is consistent with relational theory ("SQLZERO")
- a subset of standard SQL.

The main educational objective of the SQL-EDDI environment is to allow students to study the evaluation of relational algebra expressions, and relate these to the translation and interpretation of standard SQL queries.

The interpreter can be interactively changed so that SQLZERO is interpreted according to the evaluation conventions of relational algebra, or those of standard SQL. Figure 5.20 shows how the languages within the SQL-EDDI environment are related.

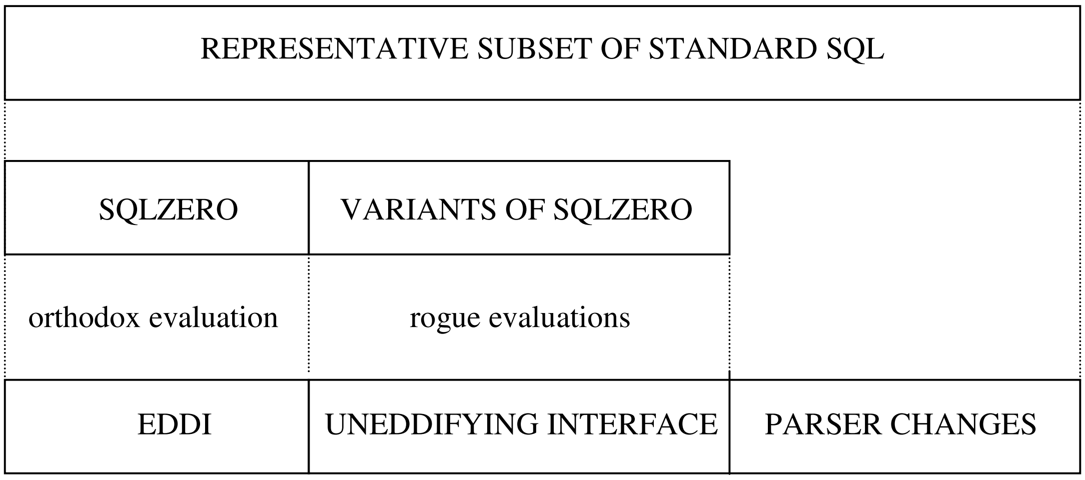


Figure 5.20 – The relationship between the query languages in SQL-EDDI

EDDI is a relational algebra language that allows users to create tables and interrogate them using the basic relational operators of union, difference, intersection, projection, selection and natural join. It is loosely based on Todd's Information Systems Base Language (ISBL) [Tod76], realising all of its functionality but adopting different syntactic conventions. The `eddi` interpreter is a front-end to the EDEN interpreter since commands are translated into EDEN for execution. Listing 5.3 shows some EDDI code to create a small example database. The line numbers are not part of each command and are only included for purposes of discussion. Lines 1-12 create the database by defining the tables and populating them with records. Lines 13 and 14 create views on the tables, whose current value is always kept up to date, and line 15 assigns the value of a relational algebra expression to a table.

```
1. %eddi
2. allfruits (name CHAR, begin INT, end INT);
3. allfruits << ["granny",8,10],["lemon",5,12],
                ["kiwi",5,6],["passion",5,7];
4. allfruits << ["orange",4,11],["grape",3,6],
                ["lime",4,7],["pear",4,8];
5. allfruits << ["cox",1,12],["red",4,8];
6. apple (name CHAR, price REAL, qnt INT);
7. apple << ["cox",0.20,8],["red",0.35,4],["granny",0.25,10];
8. citrus (name CHAR, price REAL, qnt INT);
9. citrus << ["lime",0.30,3],["orange",0.55,8],
             ["kiwi",0.75,5],["lemon",0.50,2];
10. soldfruit (name CHAR, unitsold INT);
11. soldfruit << ["cox",100],["granny",153],["red",70];
12. soldfruit << ["kiwi",23],["lime",15],
                 ["lemon",55],["orange",78];
13. fruits is allfruits % name;
14. popcitrus is (fruits.citrus % name).
                        (soldfruit : unitsold > 50 % name);
15. nonapplesncox = allfruits-
      (allfruits*apple%name,begin,end)+allfruits:name==”cox”;
```

Listing 5.3 – An EDDI extract illustrating the definition of the FRUITS database

Understanding of how standard SQL is related to relational algebra is scaffolded through the introduction of SQLZERO, an SQL-like notation. SQLZERO queries are translated into EDDI by using the `sqlte` interpreter, which displays the translation but does not carry out the evaluation. The evaluation strategy in EDDI can be interactively changed so that SQLZERO is interpreted according to the evaluation conventions of relational algebra, or in such a way that it mimics standard SQL. Changes to the evaluation strategy are made via the ‘Uneddifying Interface’ to be described below (see Figure 5.22).

When the orthodox evaluation conventions of relational algebra are adopted, SQLZERO is a variant of SQL that is faithful to Codd’s relational model [Cod70]. Through interaction with the SQL-EDDI environment in this orthodox evaluation mode, students can appreciate the intimate connection between SQLZERO and relational algebra. By interacting with the SQL-EDDI environment in other evaluation modes (cf. Figure 5.22), students become aware of the flaws in the design of standard SQL. Though changing the evaluation strategy readily makes it possible to mimic the interpretation of simple queries in standard SQL, it becomes evident that much more is required to support the interpretation of more complex standard SQL queries. It was at this point in developing the SQL-EDDI environment that the flexibility for exploratory language development proved to be most significant; it was only through experiment that a feasible strategy for implementing a more representative subset of standard SQL emerged.

As stated above, the main educational objective of the SQL-EDDI environment is to provide a way of exploring how the design of SQL has deviated from the relational model and the implications of this. SQLZERO with the orthodox evaluation conventions differs from standard SQL in that:

i) `SELECT` is treated as a synonym for `SELECT DISTINCT`,

ii) type checking on constructing union, intersection and difference of relations takes account of both domain types and attribute names,

iii) `SELECT * FROM X,Y` is interpreted as a natural join of relations.

Figure 5.21 shows SQL and EDDI statements that can be used to highlight the flaws in the design of SQL that respectively stem from i), ii) and iii) above. Query 1b) is the nearest equivalent in relational algebra terms of the SQL query 1a). In EDDI, query 1b) returns a set of *distinct* fruit names. In standard SQL, query 1a) returns duplicate rows for the cox, red and granny tuples. We should expect both queries 2a) and 2b) to be equivalent to the relational algebra expression underlying the EDDI query 2c). In standard SQL, queries 2a) and 2b) return tables with the same contents but with different attribute names (cf. the output tables in Figure 5.22). In EDDI, query 2c) causes a semantic error when type checked. Much syntactic complexity in standard SQL could be avoided if query 3a) generated the natural join that is specified in EDDI query 3c) but in practice query 3b) has to be used to achieve this result. In standard SQL, query 3a) returns a table with six columns (allfruits.name, begin, end, apple.name, price, qnt), two of which have identical contents whereas query 3c) returns the natural join of the two tables, namely a table with five distinct columns (name, begin, end, price, qnt). SQL query 3b) explicitly eliminates the duplicate column that is generated in query 3a).

Duplicate rows:

```
1a) SQL:  (SELECT name FROM apple) UNION (SELECT name FROM
      allfruits)
1b) EDDI: ?apple % name + allfruits % name;
```

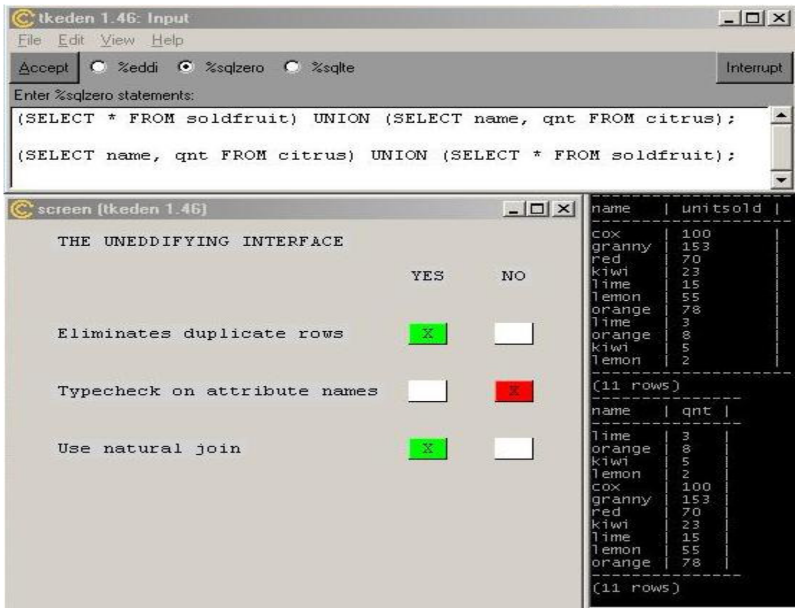
Loose type checking in creating unions:

```
2a) SQL:  (SELECT * FROM soldfruit) UNION (SELECT name, qnt FROM
      citrus)
2b) SQL:  (SELECT name, qnt FROM citrus) UNION (SELECT * FROM
      soldfruit)
2c) EDDI: ?soldfruit + citrus % name, qnt;
```

Indirect and clumsy representation of natural join:

```
3a) SQL:  SELECT * FROM allfruits, apple
3b) SQL:  SELECT allfruits.name, begin, end, price, qnt FROM
      allfruits, apple WHERE allfruits.name=apple.name
3c) EDDI: ?allfruits * apple;
```

Figure 5.21 – Some example SQL queries and their EDDI equivalents



%eddi [✓ %sqlzero] %sqlte

Enter %sqlzero statements:

```
(SELECT * FROM soldfruit) UNION (SELECT name, qnt FROM citrus);

(SELECT name, qnt FROM citrus) UNION (SELECT * FROM soldfruit);
```

THE UNEDDIFYING INTERFACE

	YES	NO
Eliminates duplicate rows	🟩 X	
Typecheck on attribute names		🟥 X
Use natural join	🟩 X	

```
--------------------------
name    | unitsold |
--------------------------
cox     | 100      |
granny  | 153      |
red     | 70       |
kiwi    | 23       |
lime    | 15       |
lemon   | 55       |
orange  | 78       |
lime    | 3        |
orange  | 8        |
kiwi    | 5        |
lemon   | 2        |
--------------------------
(11 rows)
----------------
name    | qnt |
----------------
lime    | 3   |
orange  | 8   |
kiwi    | 5   |
lemon   | 2   |
cox     | 100 |
granny  | 153 |
red     | 70  |
kiwi    | 23  |
lime    | 15  |
lemon   | 55  |
orange  | 78  |
----------------
(11 rows)
```

Figure 5.22 – The SQL-EDDI environment in use (cf. queries 2a and 2b in Figure 5.21)

EDDI queries obey the strict mathematical conventions of the relational model. In the SQL-EDDI environment, the interpretation of SQLZERO is changed via the 'Uneddifying Interface'; this adapts the evaluation so that (cf. the three logical flaws described above), it allows duplicate rows, typechecks on domains alone, and uses 'unnatural' join.

The design of the SQL-EDDI environment was not preconceived, and the pedagogical goals for the software emerged as the development was being carried out by Beynon on-the-fly in parallel with the teaching of the database module. The use of EM in the development of SQL-EDDI was significant in two respects:

- the flexible and organic nature of the development meant that it could proceed alongside the teaching.
- the adaptable language parsing offered by the AOP meant that incomplete languages could be developed and flexibly modified on-the-fly to support different teaching requirements.

By way of illustration, the eventual development of a parser for a more representative subset of standard SQL required changes to both the syntax and the evaluation strategy used in implementing SQLZERO – this could be effected by introducing small files comprising new definitions and redefinitions. This was not a conceptually simple process, free of error, or technically straightforward, but the entire activity of testing, modifying and debugging the environment revolved around interpretation through experimental interaction of the modification of small groups of definitions.

The EM development of SQL-EDDI was carried out within the same environment that the students were using for tutorial purposes. In principle, this process could be continued in extending the SQL-EDDI environment to address issues such as:

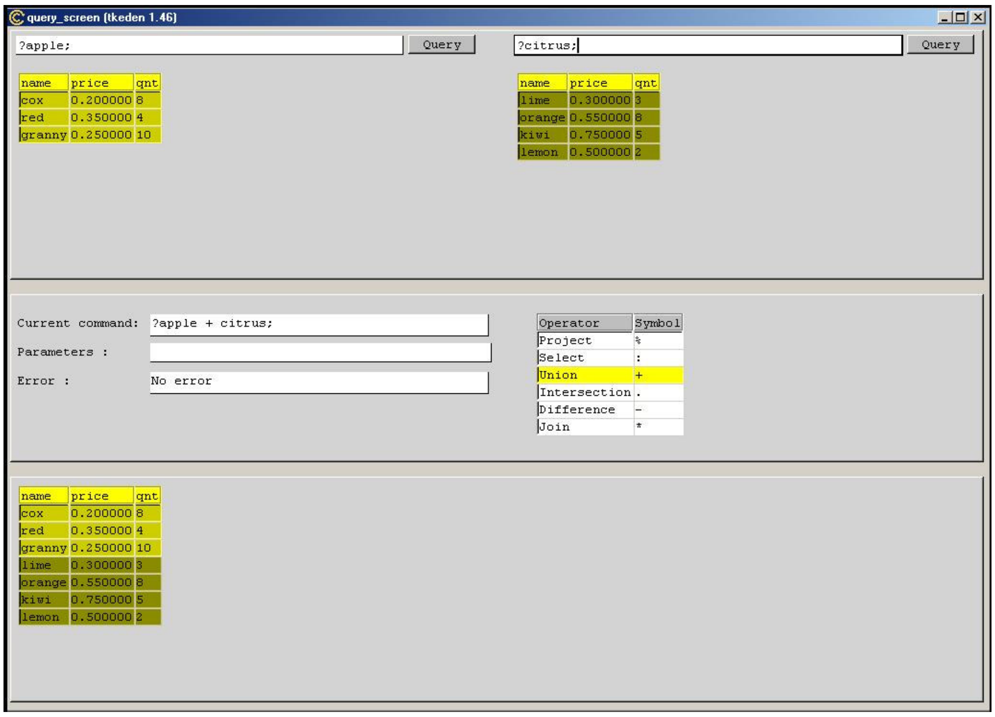
1. supporting a larger subset of SQL features, (e.g. more sophisticated data definition, integrity constraints and support for nulls).
2. implementing other relational query languages (e.g. QUEL [Dat87]).
3. incorporating an interface to study optimisation of relational database queries.

To further illustrate the concept of scaffolding we now show how the SQL-EDDI database environment can be tailored for use with younger age groups to introduce relational algebra operators as operators on tables.

The Relational Algebra Tutor (RAT) uses colour coding to suggest how the operators of relational algebra work. Each of the six relational operators (project, select, union, intersection, difference, join) has a different meaning and is applicable in different circumstances. Students will be unable to formulate queries in EDDI without a sound conceptual grasp of how these operators work.

Figure 5.23 shows the RAT in use. The interface is split into three sections:

- The top section shows the input tables that are generated from EDDI queries. These can either be individual tables or complex EDDI queries.
- The middle section contains a switching mechanism to change the currently selected operation, together with information about the currently selected operation. This information comprises the EDDI language statement that produces the output table from the input table(s), and the currently selected operator. The field for specifying parameters for a command is only required for the project operator (when it specifies the names of the columns to project) and the select operator (when it specifies the boolean condition used to select rows from the table). If an operation cannot be performed – for instance, if tables are not be compatible for union, intersection and difference – then this is reported in the error window.
- The bottom section shows the output table formed by the operator applied to the input tables. The rows and column headers of the output and input tables are colour coded to show how the result of the query is composed from the input tables. For example, in Figure 5.23, the current operation (union) is coloured yellow, the column headers are also highlighted in yellow, and the rows from the output table are coloured differently depending on the input table from which they have been derived.



?apple;

name	price	qnt
cox	0.200000	8
red	0.350000	4
granny	0.250000	10

?citrus;

name	price	qnt
lime	0.300000	3
orange	0.550000	8
kiwi	0.750000	5
lemon	0.500000	2

Current command:	?apple + citrus;
Parameters :	
Error :	No error

Operator	Symbol
Project	%
Select	:
Union	+
Intersection	.
Difference	-
Join	*

name	price	qnt
cox	0.200000	8
red	0.350000	4
granny	0.250000	10
lime	0.300000	3
orange	0.550000	8
kiwi	0.750000	5
lemon	0.500000	2

Figure 5.23 – Using the RAT to support understanding of operations on tables

The construction of the RAT environment illustrates a high degree of code re-use. The development time I required – as an EM expert – was about 2 days, but such development would be impossible for a non-computer specialist. The RAT uses spreadsheet grids (see section 2.2.1) to display the input table(s), the operators table and the output table, and uses EDDI to generate the output table by executing the command string built up in the ‘Current command’ window. The high level of re-use meant that the majority of the model was constructed from existing resources. The colour coding for the input and output tables is dependent on each individual operator and was implemented using simple search and matching routines.

With reference to the EFL, the purpose of the RAT is to allow learners to experiment with basic relational algebra operations on various tables to establish and reinforce their conceptual understanding of the operations on tables and the EDDI language. RAT provides the support for learners to gain the experience of interpreting symbolic relational algebra operators that is required to use EDDI successfully.

5.5 Chapter Summary: Scaffolding with Empirical Modelling

In this chapter, we have described EM case studies that have illustrated scaffolding operating in the zone of proximal development in a wide variety of contexts (cf. Soloway's TILT model, Figure 5.2). The analogy suggested by scaffolding – of rigid and predefined buildings – seems inappropriate to describe the rich ways in which the EM models described in this chapter have been flexibly developed and presented. In [NH96] Noss and Hoyles described three criticisms of the scaffolding metaphor in computer learning:

i) the notion of scaffolding suggests a structure being erected around the learner by an external agency. This may not take account of how learners structure their own learning.

ii) The idea of a 'zone' is a useful metaphor that suggests the idea of a bounded territory. It is important to leave open how it is defined and where its limits are.

iii) The idea of the scaffolding fading away with learning implies that, if the computer provides scaffolding, then it should be removed at some point. This is not always desirable.

Our case studies illustrate how EM can give more support for learning than the traditional scaffolding metaphor suggests. For instance they exhibit support systems for learning with characteristics that address Noss and Hoyles's criticisms outlined above:

i) Our case studies support the idea that the learner should control their own learning. For instance, in the racing cars model (see section 5.2.1), the learner is always in control over when they move on to the next microworld.

ii) The OXO family of games case study (see section 5.3.3) presumes no preconceived bounded territory within which learning is to take place, since the learner is always being encouraged to explore.

iii) The SQL-EDDI environment (see section 5.4.3) gives support to learners that remain accessible to the expert. For instance, the SQLTE translation interface can always be used to confirm relationships between SQL and relational algebra.

Noss and Hoyles propose an extension of scaffolding that they call *webbing*. This draws on the metaphor of the World Wide Web to convey that the learner accesses a support structure that they can draw upon and reconstruct as they learn. Webbing is distinctive because [NH96]:

i) It is under the learner's control.

ii) It is available to signal possible user paths rather than point towards a unique, directed goal.

iii) The local and global support structures are dependent on the learner's current level of understanding.

The support structures provided in the EM case studies described in this chapter give practical evidence of the use of webbing in learning environments. For instance, the specific OXO game is a possible path that a learner can follow, but there are many other games that can be explored 'in the neighbourhood of OXO'.

In chapter 2, we discussed how learning activity can be associated with the negotiation and elaboration of concepts (cf. section 2.2.2). The notion of scaffolding supports the negotiation of the semantic relation β but is limited in respect of elaboration. In the racing cars model, the concept of 'car racing' is gradually exposed to the learner. A learner understands the concept at a simple level before it is embellished. This leads the learner to embark on a process of negotiation of the concept through experimental interactions and making and testing hypotheses. When a learner is comfortable with the concept at a particular level of complexity they have the control to move on to the next level. However, the fixed nature of the referent limits the scope for investigative exploration around the subject. In this respect, scaffolding is limited with respect to elaborating the semantic relation β.

In contrast to scaffolding, webbing offers better support for learning through the elaboration of the semantic relation β. Since webbing is an extension of scaffolding it is natural to expect that it still supports negotiation of the semantic relation β. The analogy that underpins webbing – that of building connections in a flexible structure as in the Web – shows that elaboration of the semantic relation is represented in a webbing approach. The scope for using EM in 'building connections in a flexible structure' is illustrated in our OXO case study.

Learning is nevertheless much more than can be represented in terms of scaffolding or indeed webbing. Our previous discussions (cf. chapter 3) have shown how learning activities can be very diverse. This diversity cannot be represented within preconceived frameworks for presenting models to learners. Model use can be more varied than is represented in the case studies presented in this chapter. Model building can likewise take exceedingly diverse forms. In the following chapter, we discuss three EM case studies that illustrate a variety of different types of learning and ways of developing and interacting with models, and interpret them with reference to the EFL.

Chapter 6 – Exploratory learning and the EFL

6.0 Overview of the Chapter

In this chapter, we discuss three case studies that represent different ways of applying EM to educational technology. These case studies show: how EM can be used to support pre-articulate exploration in a private learning scenario; how pre-articulate and formal learning activities can be connected within a common exploratory learning environment; and how a learning environment can reinforce a learner's construal of a physical situation. Broadly speaking, the application of computers for learning can be classified into building models and using models. In EM, the distinction between model construction and model use is blurred. Many EM models exhibit qualities of both model construction and model use. This enables modellers to explore partial models and build on top of them. The case studies in this chapter have been selected to highlight how EM can be used in learning activities across the whole range of the EFL.

6.1 Integrating model use and model building

In this section, we consider the use of EM to support a range of learning activities within the EFL. We also show how EM modelling can blur the distinction between building and using models. As discussed in chapter 3, the use of computers for learning is broadly of two kinds. These are:

i) learning through model-building: this involves the construction of models to enhance our personal understanding of a domain. In chapters 2, 3 and 4 we discussed the EM approach to the construction of models from computational and educational perspectives. We concluded that model construction in EM has two advantages over writing conventional programs. Firstly, EM supports the pre-articulate learning activities situated at the experiential end of the EFL; and secondly, it allows model construction to integrate pre-articulate and formal activities in a single modelling approach.

ii) learning through model-use: this involves the use of a pre-constructed learning environment in the form of (e.g.) instructionist software or a constructionist microworld. In chapter 5, we discussed how EM learning environments can support many different types of learning. The emphasis was on describing techniques for model development that offer flexibility to the developer and facilitate an enhanced learning experience.

The two perspectives outlined above relate to traditional perspectives on the use of computers for learning – users either write their own programs or use programs written by other people. However, in EM, the learner has complete discretion over the interactions they undertake with a model, and is free to add new definitions to a model or refine existing definitions. Therefore, in EM, learning environments can combine model building with model exploration. The two traditional perspectives outlined above define the extreme ends of a spectrum ranging from model building to model use, as represented by the restaurant model and the relational algebra tutor respectively. In this chapter, we discuss three further EM case studies representative of learning environments within which model building and model-use for learning are conflated in different ways. These models are: the Monotone Boolean Functions in 4 variables (MBF4); Heapsort; and the Robotic Simulation Environment (RSE) (see Figure 6.1). We shall first informally describe the roles played in each case study by model building and model use, then discuss some of those aspects that are most significant for learning with reference to the EFL.

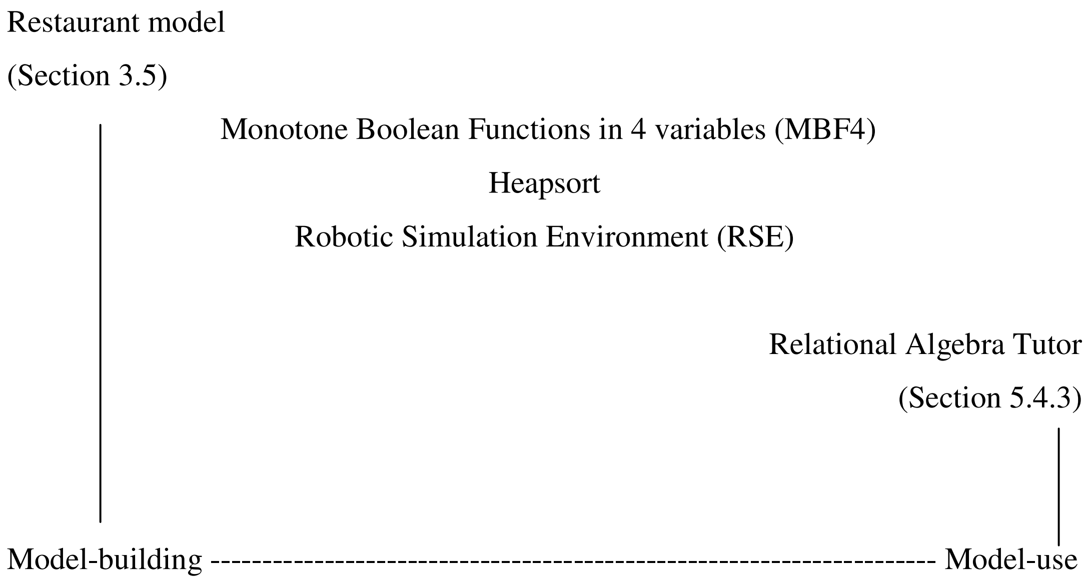

Restaurant model
(Section 3.5)
Monotone Boolean Functions in 4 variables (MBF4)
Heapsort
Robotic Simulation Environment (RSE)
Relational Algebra Tutor
(Section 5.4.3)
Model-building -- Model-use

Figure 6.1 – Conflating model building and model use

The Monotone Boolean Functions in 4 variables (MBF4) model [EMRep, mbf4Beynon2003], is primarily the work of Meurig Beynon with some contribution from myself. This model was originally developed by Beynon as a private artefact [EMRep, fdl4Beynon2002] through which to enhance his comprehension of the relationships between several different abstract realisations of FDL4 ("learning through model-building"). There have been two aspects to my engagement with the model as a learner: I have interacted with the model to gain experience of unfamiliar mathematical concepts ("learning through model-use") and have extended the model in ways that make it more easily intelligible to the learner ("learning through model-building").

The Heapsort model [EMRep, heapsortBeynon1998] was constructed by Meurig Beynon, Amanda Wright and Jaratsri Rungrattanaubol and has been discussed in [Bey98, BRS^{+}98, BRS00, Run02]. The model has a particular pedagogical aim, namely to support the exposition of the heapsort algorithm, but it has been constructed in stages in such a way that learners can carry out experiments and extensions. Learning therefore combines elements of model building and model use.

Rather than focusing exclusively on the heapsort algorithm, the model promotes exploration of the concepts around heapsort through interaction outside the bounds of heapsort. Unconstrained exploration can aid learning about heapsort by challenging and reinforcing the construal of the learner.

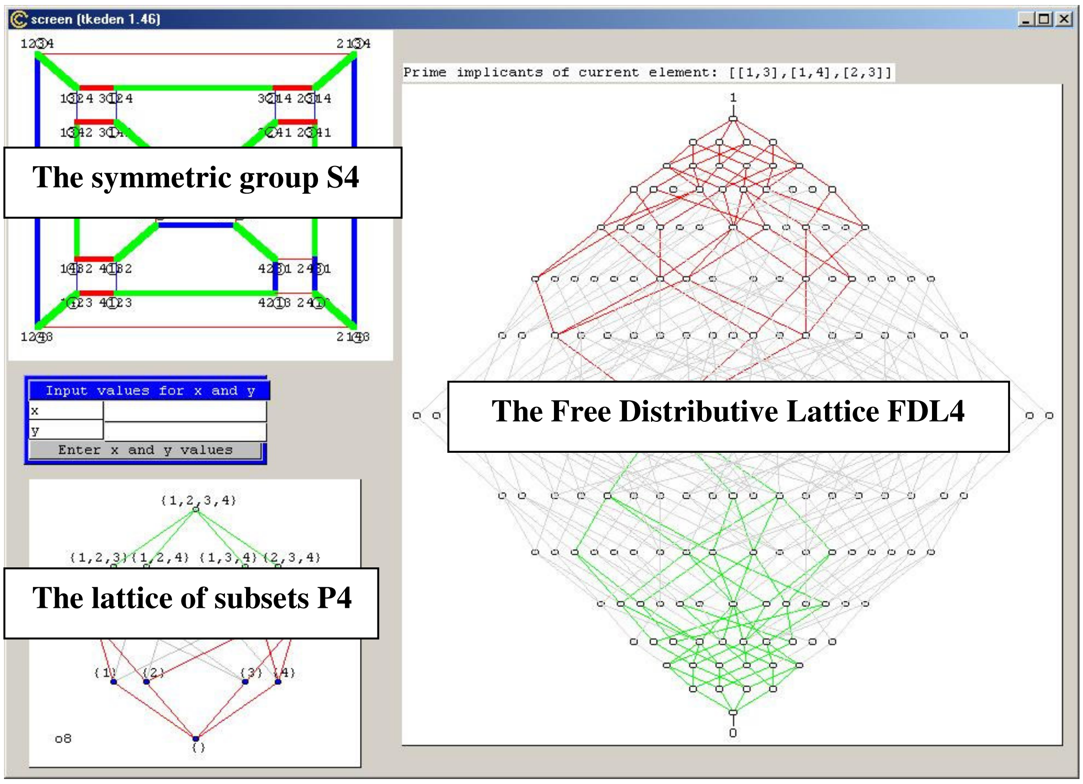
The Robotic Simulation Environment (RSE) [EMRep, rseRoe2003] is an example of a learning environment that aims to promote conceptual understanding of a real-world situation. I have constructed the model on the basis of a preliminary design developed in collaboration with researchers from the Kids' Club in Joensuu, Finland, as documented in [EJR$^+$02]. In the Kids' Club, the children use the LEGO Mindstorms robot programming environment to program robot behaviours. The RSE is intended to supplement the current system. It provides an exploratory environment for the investigation of robot behaviour through a computer model. The RSE is to be regarded as a case study in model use for learning because the environment contains all the necessary elements for exploratory investigation of the robots. In keeping with a constructionist approach, the environment supports many different styles of interaction and allows learners to engage with the problems of robot programming at many levels.

6.2 Monotone Boolean Functions in 4 variables

The EM model of Monotone Boolean Functions in 4 variables (MBF4) illustrates Wilenski's observation that:

> '[t]he more connections we make between an object and other objects the more concrete (familiar) it becomes for us. The richer the set of representations of the object, the more ways we have of interacting with it, the more concrete it is for us' [Wil93].

The MBF4 model [EMRep, mbf4Beynon2003] has been composed by combining resources created by a number of different people. These include Bibi Hussain's EM model of the free distributive lattice on 3 elements [EMRep, fdl3Hussain2001], Allan Wong's EM model of the group of symmetries of a cube [EMRep, symcubeWong2001] and John Buckle's diagrammatic representation of the Hasse diagram for the Free Distributive Lattice on 4 generators (FDL4) [Buc90]. The relevant parts of the EM models have been brought into a single environment and, using dependency, links have been created between them to establish the appropriate connections. The three main components of the MBF4 model are depicted in Figure 6.2.



Prime implicants of current element: [[1,3],[1,4],[2,3]]

Figure 6.2 – The diverse components of the MBF4 model

The original prototype for the MBF4 model (see [EMRep, fdl4Beynon2002]) was developed by Beynon as a personal visual aid to mathematical research, and its interface was not designed for use by novices. The visual representations of the lattices P4 and FDL4 in this model are Hasse diagrams [Bir95] with 16 and 166 nodes respectively, and S4 is represented by a Cayley diagram in which the edges correspond to the basic transpositions (12), (23) and (34).

The above description of the model components is appropriate for the mathematical expert – it presumes some advanced knowledge of mathematical concepts and terminology. Our purpose in this section is to describe how interaction with the original MBF4 model – involving both model use and model building – can assist the novice in gaining the understanding of the mathematical objects involved. In this section, writing as a novice lattice theorist, I give an account of the types of activities that contributed to my learning, and the partial comprehension of the abstract mathematics represented in the model that emerged during the interaction. In the process of learning, I have also enhanced the model as a learning artefact.

For a learner, the principal role of the MBF4 artefact is to provide concrete support for coming to understand the concepts behind the artefact. Interaction and experimentation with the artefact can provide a stable source of experience on which to base an understanding of the lattice theoretic concepts in the model. The primary objective of the learner is to utilise the experience gained from interacting with the MBF4 artefact in order to understand the formal mathematics it embodies. The central importance of concrete intuitions as a foundation for formal mathematics is something that I have observed in my experience of teaching mathematics to undergraduates (cf. the emphasis placed on concrete understanding by Papert and others [TP91, Pap93, Wil93]). For instance, it would be absurd to expect a learner to reason symbolically about decreasing subsets if they did not have some experiential support for grasping the concept. The role of the interactive artefact is to provide a concrete means by which to attain the appropriate construal. In the following discussion, we outline the abstract mathematics underlying the MBF4 model, and explain the role of the model in helping a learner gain concrete experience of abstract structures through active experimentation. Each of the mathematical components in Figure 6.2, (P4, FDL4 and S4) exists as an EM model. We shall discuss each of these in isolation before illustrating how their connections can be explored.

6.2.1 P4: The lattice of subsets of {1,2,3,4} ordered by inclusion

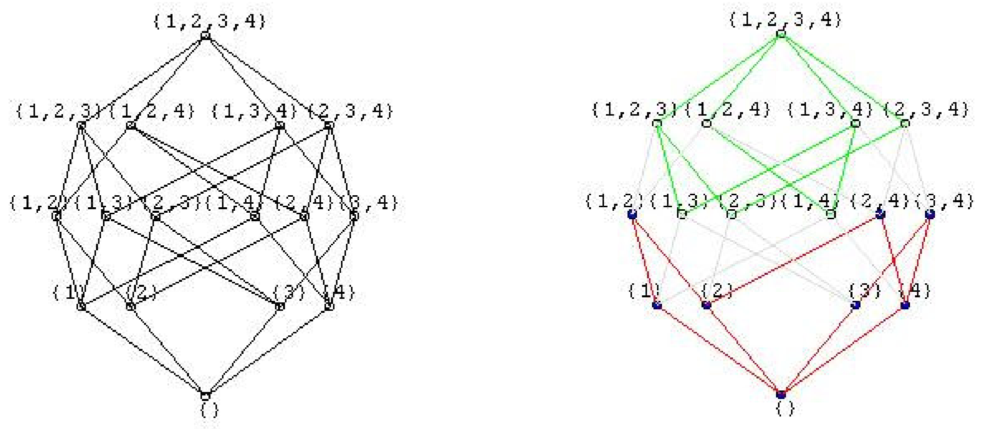
The EM model of P4 can be used to illustrate some basic concepts of lattice theory, including decreasing subsets, partially ordered sets and lattices. Figure 6.3(a), which has been extracted from a screenshot of the executing P4 model, shows all the subsets of the set {1,2,3,4}. These are depicted using the conventional representation of a partially ordered set as a Hasse diagram. The role of the Hasse diagram is to provide concrete support for the abstract notion of a partial order on the subsets in P4. For instance, one subset contains another if there is an upward path in the Hasse diagram between the two. A mathematician might describe this visual relationship more formally as: the subset X contains the subset Y if and only if there is an upward path in the Hasse diagram from the node x that represents X to the node y that represents Y. This attempt to give a more precise and abstract description of a visual experience does not convey the immediate and concrete way in which the Hasse diagram is apprehended by the learner.

In similar fashion, the set of subsets of {1,2,3,4} ordered by inclusion is an example of a partially ordered set since inclusion of sets is reflexive, antisymmetric and transitive. For instance, in the Hasse diagram, if there is an upward path from node x to node y and from node y to node z there is an upward path from node x to node z. We can also establish that the partially ordered set {1,2,3,4} with inclusion is a lattice because every pair of elements has a union and an intersection, where both are defined as being the minimal instances of such unions and intersections. For example, the pair of elements {1,2} and {1,4} have a union of {1,2,4} and an intersection of {1}, both of which exist and can easily be apprehended by following links on the Hasse diagram. By virtue of being an interactive artefact, the P4 model can provide additional visual support for identifying unions and intersections by allowing the learner to select a pair of subsets and displaying their union and intersection.

The mathematical concepts discussed above could be introduced to the learner through their formal representations, but this does not give a learner access to the stable experience that underpins this formal representation. The problems faced by the learner become more significant as the concepts become more complex. For instance, the notion of a decreasing subset in a partial order relies upon prior understanding of the notion of 'subset' and 'partial order'. Its mathematical formalisation is:

A subset Y of a partially ordered set P is decreasing if $\forall x,y \in P$: $y \in Y$ and $x \leq y \Rightarrow x \in Y$.

Figure 6.3(b) depicts a decreasing subset Z of the partially ordered set P4 as a set of blue nodes linked by red edges. The fact that Z is a decreasing subset can be appreciated by observing that all nodes beneath a blue node are blue. Reasoning about formal concepts can also gain experiential support from artefacts. For instance, a learner can observe that the decreasing subset Z is not a lattice as not all pairs of elements have a union in the subset (e.g. {2,4} and {3,4} are in the subset, but {2,3,4} is not).

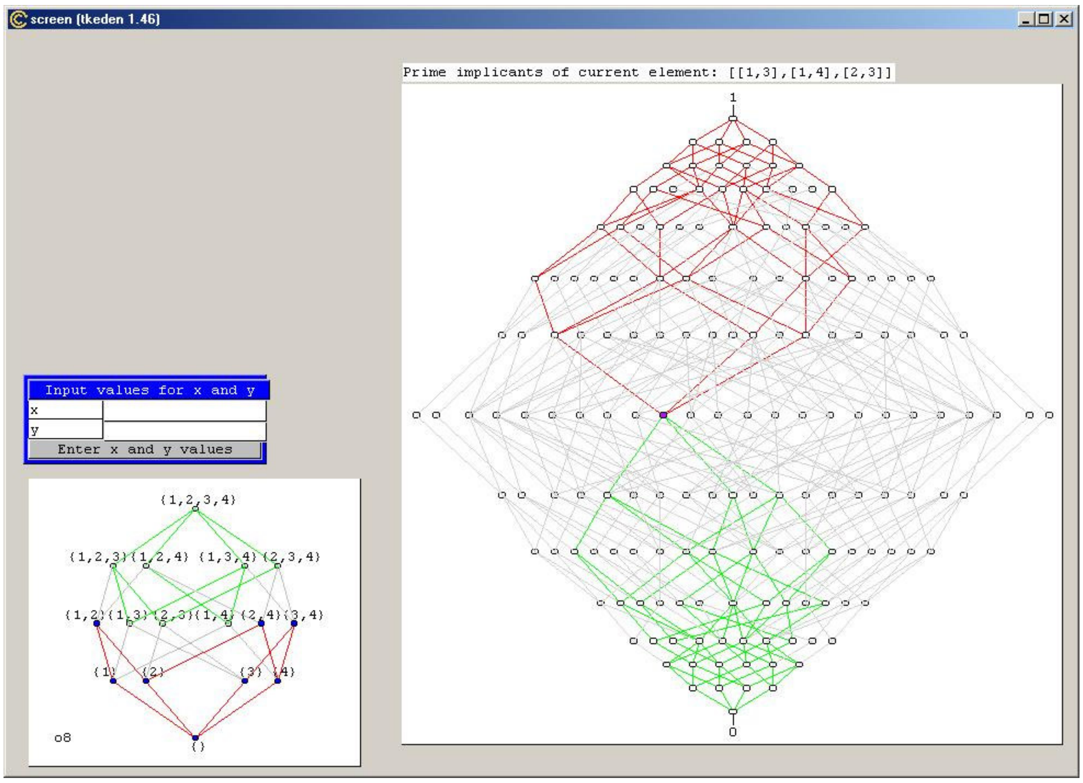


(a) (b)

Figure 6.3 – (a) The lattice of subsets P4; (b) an example of a decreasing subset of P4

6.2.2 FDL4 as the lattice of decreasing subsets of P4 ordered by inclusion

Each decreasing subset of {1,2,3,4} can be regarded as itself an element in a set. This set of decreasing subsets is itself a partial order when ordered by inclusion. This partial order is depicted as a Hasse diagram on the right of Figure 6.4 – it defines one representation of the free distributive lattice on 4 generators (FDL4). The Hasse diagrams of P4 and FDL4 are linked by dependency in the EM model MBF4 in such a way that the selection of a node in FDL4 leads to the display of the corresponding decreasing subset of P4. Through experimenting with the selection of nodes in FDL4, I could observe that there is a line between two nodes if one of the associated decreasing subsets can be obtained from the other by adding a single subset of P4. From this, we can infer that the horizontal row of a node in FDL4 is determined by the number of subsets of P4 in the corresponding decreasing subset.



```
Prime implicants of current element: [[1,3],[1,4],[2,3]]
```

Figure 6.4 – The Hasse diagram with 166 nodes corresponding to the set of decreasing subsets of {1,2,3,4}

Other experiments that can be performed with the Hasse diagram of FDL4 include picking pairs of nodes and establishing their point of union and intersection. For this purpose an interface to select two points and show their union and intersection is essential, given the size and complexity of the diagram. It is clearly infeasible to verify by hand that all pairs have a union and an intersection. However, through testing many examples we can become convinced that this is indeed true. As Beynon observes, such testing still leaves room for uncertainty:

> 'expectations developed through experiment are always subject to falsification, and are asserted subject to faith in prediction from past evidence. Expectations can be confirmed and confounded – but never justified – by experiment' [BAC$^+$94].

As it happened, the visualisation in the original model contained an error: the internal model of union and intersection of decreasing subsets was correct but one link between two nodes was wrongly placed. It was only through personally extending the FDL4 artefact that I encountered this error in the display of the lattice – it appeared that a pair of nodes had no intersection. The model user/developer's response to encountering anomalies of this nature in a model is heavily dependent on the context in which they occur. For instance, in model building at the frontiers of research such occurrences might dispose the modeller to discard a valid hypothesis. In our lattice example, had I encountered this error during my early interactions with the model I would have been less likely to believe that the structure was truly a lattice. As it was, I was already convinced that the structure was a lattice – and I construed the anomaly as an error in the graphical representation rather than an abnormality in the underlying mathematical structure.

6.2.3 FDL4 as monotone boolean functions in 4 variables ordered by implication

FDL4 (see Figure 6.4) admits another interpretation – as a lattice of monotone boolean functions in 4 variables. A monotone boolean function is a function $f:\{0,1\}^4 \rightarrow \{0,1\}$ such that $f(x_1,x_2,x_3,x_4)$ is defined by a logical expression in x_1,x_2,x_3,x_4 using the operators *or* and *and*. An example of such a function is: $f(x_1,x_2,x_3,x_4) = (x_1 \text{ and } x_3) \text{ or } (x_1 \text{ and } x_4) \text{ or } (x_2 \text{ and } x_3)$.
Such a function is commonly represented by a circuit containing *and* and *or* gates. For this reason, we often refer to the tuple (x_1,x_2,x_3,x_4) as an input value and to $f(x_1,x_2,x_3,x_4)$ as the output. Given a monotone boolean function, we can consider the input values for which the output is 0. For instance, for the function f above, if $x_1=1,\ x_2=1,\ x_3=0,\ x_4=0$ then $f(x_1,x_2,x_3,x_4) = 0$. We can identify this input value with the set {1,2}, the set of indices of x's that are assigned the value 1. With this convention, for the example function f above, the complete set of input sets for which $f(x_1,x_2,x_3,x_4) = 0$ is $\{\ \{\},\ \{1\},\ \{2\},\ \{3\},\ \{4\},\ \{1,2\},\ \{2,4\},\ \{3,4\}\ \}$. This set is exactly the decreasing subset that is depicted in Figure 6.3(b). By applying this general construction, the diagram of FDL4 in Figure 6.4 can be interpreted as the set of logically distinct monotone boolean function in 4 variables. Under this interpretation, the ordering of monotone boolean functions is by implication.

For me as a learner, the interactive artefacts described above played an important role in understanding lattices and related concepts. Drawing on my own personal experience, the interaction with the artefact served to both create and reinforce my understanding of the connections between the abstract mathematical objects P4 and FDL4, and monotone boolean functions in 4 variables. Interaction based on a formal symbolic approach would not have been as effective as I did not have the solid conceptual understanding needed to interpret the formal representations.

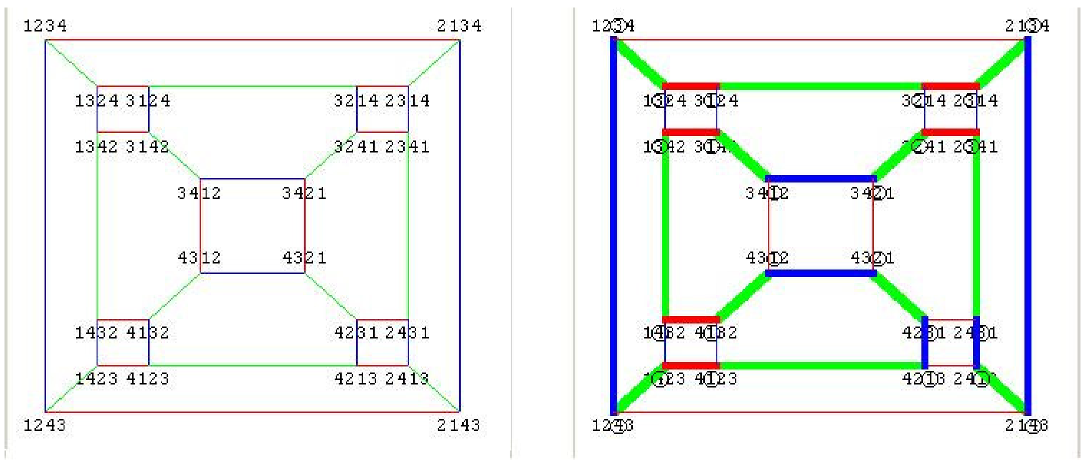
I was able to introduce the extra visualisations apparent in Figure 6.4 despite my incomplete understanding of the mathematics they represent. The following account of how I carried out this development is included to show that there is an intimate correspondence between analysing the dependencies in the model and understanding the relationship between different abstract representations of FDL4.

The red and green lines in Figure 6.4 are defined such that, if a line is above (respectively below) the currently selected point and it can be reached from the point by a strict upward (respectively downward) path then it is red (respectively green). With this convention, determining whether a particular line should be coloured red or green is equivalent to checking that the union of the selected point and the points at the ends of the line is either the selected point or one end of the line. Each point in Figure 6.4 is stored in an internal database and has a representation in EDDI (see section 5.4.3). This representation takes the form of a tuple that contains sixteen fields, each corresponding to a different subset of {1,2,3,4}, together with an identifier for each point. To discover the criterion for colouring lines above, I generated EDDI queries to construct tables of three example points, and observed a correlation between the existence of an upward/downward path in the Hasse diagram and a relationship between the internal representations of the three points. It was on the basis of this experimental evidence that I was able to understand the general relationship and implement this through introducing appropriate definitions (this was the activity that disclosed the visualisation error described earlier in this section).

The artefacts described in this section illustrate representations of particular concrete instances of free distributive lattices. The FDL4 artefact – which has only 166 nodes – can be used to support learners in gaining experience of the concepts underlying the MBF4 model in an exploratory fashion, but this approach cannot be extended to larger free distributive lattices. The visualisation of FDL5 would be impossible as it has several thousand nodes. The FDL4 artefact can nevertheless be used to gain the experience and concrete understanding necessary to be confident in manipulating the more general abstract concepts (cf. learning about geometry in 2 or 3 dimensions and moving to higher-order dimensional geometry).

6.2.4 S4: The symmetric group on 4 symbols

The third artefact that is part of the MBF4 model is a Cayley diagram for the symmetric group S4 of permutations of the 4 elements {1,2,3,4}. The structure of this Cayley diagram can be informally explained as follows. The numbers {1,2,3,4} can be arranged in 24 different permutations. Each permutation can have a pair of elements switched in three different ways – by transposing the first and second, second and third, and third and fourth elements respectively. In the Cayley diagram, two permutations are connected by an edge if each can be obtained from the other by such a transposition. This edge is coloured red, green or blue respectively, according to whether the transposition involves the first, second or third pair of elements. This is shown in Figure 6.5(a).



(a) (b)

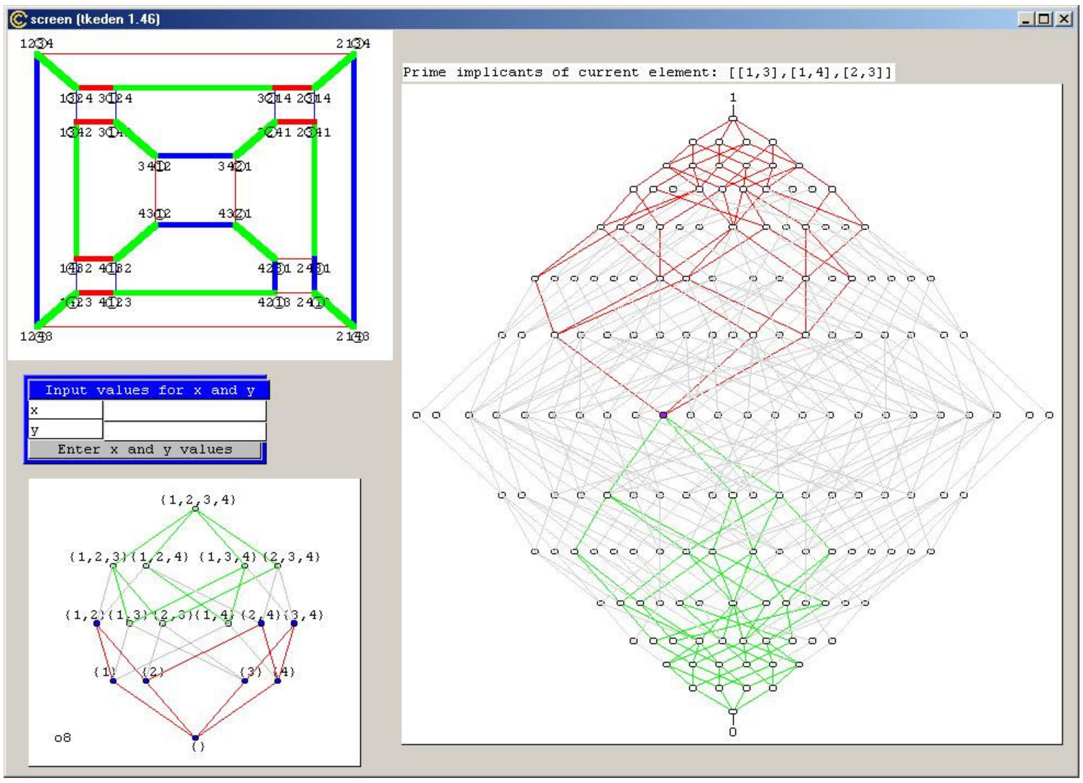
Figure 6.5 – (a) A Cayley diagram for S4; (b) an example of a CPL map

Beynon's purpose in constructing the MBF4 model was to investigate the relationship between monotone boolean functions and functions defined on the Cayley diagram. This correspondence is formally described in the paragraph that follows. It represents an aspect of the MBF4 model that is outside the scope of interest of the novice learner.

Each monotone boolean function g in 4 variables determines a function G from the symmetric group S4 into the set {1,2,3,4}. To determine the function G, we interpret each permutation of {1,2,3,4} as an ordering for switching on the inputs to the monotone boolean function g. If we then switch on the inputs to the monotone boolean function g in the order associated with permutation p then the value of G on the permutation p is the index of the input that switches the output from false to true. For instance, for the function f introduced above, the switching sequence {1,4,2,3} will make the function true for the first time when input 4 switches from false to true (since f(1,0,0,0)=0 and f(1,0,0,1)=1). On this basis, the function F: S4 → {1,2,3,4} associated with f assigns the value 4 to the permutation 1423. The functions F and G generated in this way are known as *combinatorially piecewise linear* maps (CPL maps) [Bey74]. There is a constraint on a CPL map H: if two permutations are connected by an edge in the Cayley diagram and the value of H at one of the permutations is not one of the pair of values being transposed across the edge then H must have the same value at the other permutation. This constraint is sufficient to characterise CPL maps [Bey74]. For a given CPL map H, an edge of the Cayley diagram is *non-singular* if the value of H at the endpoints of the edge is not one of the pair of values being transposed.

The characteristic features of a CPL map can be visually represented as shown in Figure 6.5(b). For a given switching sequence, the index of the input that switches the output from false to true is circled. The singular edges are thicker than the non-singular edges. As Figure 6.5(b) illustrates, for some of the nodes in FDL4 (Figure 6.4) the singular edges form cycles in the Cayley diagram. For instance, in the example displayed in Figure 6.5(b), there is a Hamiltonian cycle of length 24. The interactive nature of the artefact allows the researcher to record and explore the relationship between FDL4 and CPL maps far more effectively than any paper-based approach (cf. the way in which this relationship is documented in [Bey74, Bey87b]).

The MBF4 artefact comprising the FDL4, S4 and P4 models described above is shown in Figure 6.6. As explained above, each correspondence between a pair of components reflects a different perspective on lattice theoretic issues. Other relationships can easily be added to the model as they are encountered, by linking them through dependency to the relevant parts of the existing artefact.


screen (tkeden 1.46)

Prime implicants of current element: [[1,3],[1,4],[2,3]]

Input values for x and y
x
y
Enter x and y values

Figure 6.6 – The complete MBF4 model

The discussion in this section has focused on how the MBF4 model permits experimental interaction in order to learn about the concepts that inform its construction. This experiential approach is different in character from interacting with symbolic representations. Essentially this model has been used for two contrasting purposes: firstly as a learning aid, so that I personally could come to understand some of the basic concepts of lattice theory; and secondly as a research aid, so that exploration of connections between mathematical objects could be undertaken.

6.3 The Heapsort model

The heapsort model is intended to demonstrate the potential for the pedagogical use of EM to support understanding of formal algorithms. Heapsort is an advanced sorting method that relies on maintaining partial orders within a data structure known as a heap. For a fuller description of the heapsort algorithm, see [AHU82]. Meurig Beynon constructed the basic EM heapsort model discussed here, initially with the assistance of Amanda Wright [BRS+98]. Significant extensions to the model were made by Jaratsri Rungrattanaubol and the relationship between her model and a conventional program to teach heapsort is discussed in [Run02, Chapter 6].

The EM heapsort model is not intended to be a formal representation of the heapsort process, but rather an environment within which activities that are related to heapsort can be investigated in an experiential fashion. The emphasis is on exposing the empirical knowledge that contributes to the design of the heapsort algorithm. Conventional ways of providing computer assistance for teaching heapsort involve animating the algorithm and displaying the stages of the process (cf. the JELIOT system [BMS+02], Animated Algorithms [GDL]). However, this approach assumes that the learner has already understood the notion of a heap and the basic operations that can be performed upon it. The approach adopted in the EM heapsort model has more in common with that of the Brazilian educator Valdemar Setzer, who proposed that learners manually perform the sorting operations with physical objects [SH93].

In contrast to a conventional animation, the EM heapsort model can fulfil many different learning objectives [Bey98]. For instance, it can be used as:

1) an environment for testing a student's understanding of the concept of a heap and of the procedures used in heapsort.
2) a visualisation aid for the exposition of the heapsort algorithm.
3) a prototype for the implementation of heapsort in a conventional programming paradigm.
4) a platform for investigating variants of the heapsort algorithm.

This diverse range of learning objectives can be satisfied within a single EM learning environment by building up the partial heapsort model with combinations of agent actions that are stored in separate auxiliary files. In general these files are added to the model according to the current needs of the learner and the purpose for which they are using the heapsort model.

Beynon describes three stages in using the EM model to learn about heapsort [Bey98]:

1) experimental manipulation of a visual heap to understand the heap concept. This is possible because the heap is embedded in a definitive script that does not constrain the possible agent actions to be added to the model later.
2) the construction of state-based models to represent the stages in the heapsort process, allowing the user to trace the steps involved in heap-building and sort extraction through a sequence of manual operations. These stages are introduced as agents to perform parts of the sorting activity.
3) the introduction of automatic mechanisms to carry out the appropriate sequence of steps. This is achieved through the automation of sensible heapsort behaviours in particular patterns.

As mentioned above, the implementation of a conventional heapsort teaching program engages primarily with the activities at stage 3. Stages 1 and 2 are concerned with obtaining a solid construal of the concepts and basic operations involved in heapsort. Stages 1 and 2 admit exploratory learning and experiments to achieve the necessary background knowledge to fully appreciate the heapsort process. As Beynon remarks in [Bey98]:

> ‘the most effective way to present the model construction is to systematically introduce the underlying concepts as they might have been encountered in the discovery of the heapsort algorithm’.

Initial exploration of the heapsort model centres on understanding the heap structure. Figure 6.7 shows the heap data structure. A binary tree is a *heap* if each node satisfies the *heap condition*, which states that the value of a node is greater than that of both its children. Figure 6.7 shows how the visualisation of the heap structure provides cues to the learner about the state of the heap. Edges and nodes are coloured to reflect the current status of the heap condition at each node and the relationships between nodes and their children. For instance, if all the edges are blue then the tree is a heap.

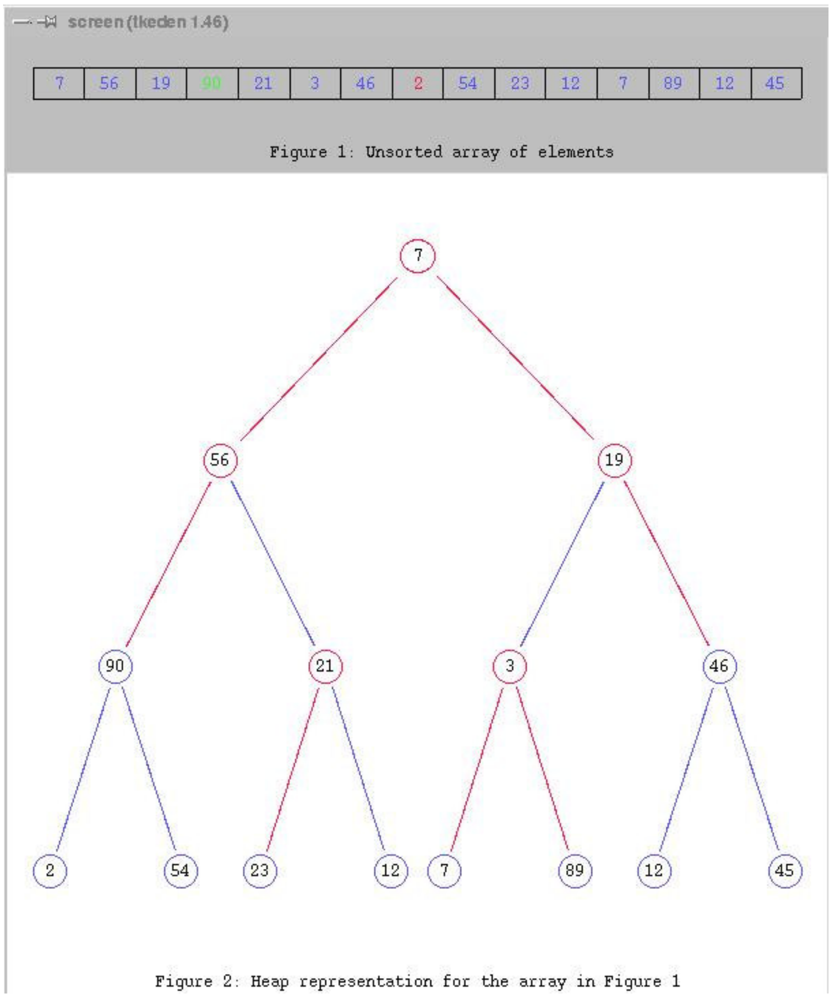


Figure 1: Unsorted array of elements

Figure 2: Heap representation for the array in Figure 1

Figure 6.7 – The heapsort model showing a representation of a heap

Embellishments to the basic model add agent actions that correspond to primitive operations on the heap and are a stepping-stone to understanding the heapsort algorithm. The simplest operation on the heap is exchanging a pair of elements. This can be performed manually, or by agents that can be attached to the nodes of the heap model to perform this exchange automatically, given a pair of indices. The learner can attempt to manipulate the tree into a heap by exchanging suitable pairs of elements. Through exploration, the learner will begin to comprehend sensible strategies for establishing a heap. For example, if a node does not satisfy the heap condition then the most effective strategy is to exchange values with the child node that has the larger value. The index of the child with the larger value is maintained by dependency and can be consulted as an observable by the agent at each node (cf. Figure 6.4 in [Run02]). Such agents can then automatically establish and maintain the heap condition at each node. The model can be set up so that these agents act autonomously to perform the complete process of heap construction. This is most appropriate when the learner has first understood how manual redefinitions can establish a heap.

All the basic operations required to understand heapsort are exercised in the heap construction phase (stage 2). From this point, it is a relatively easy to derive the complete heapsort algorithm, which repeatedly removes the root of the tree and then re-establishes the heap condition until the tree becomes empty (stage 3). As is typical of other EM models discussed in this thesis, the openness of the EM approach allows interaction and exploration outside the scope of normal heapsort operation. Rungrattanaubol outlines many possible scenarios for the use of the EM heapsort model that would be impossible with conventional educational software [Run02, pages 177–178].

The heapsort learning environment described above illustrates interaction with an EM model that conflates both model-use and model-building perspectives. This conflation of perspectives allows the integration of concrete and formal learning activities within a single learning environment. The definitions in the heapsort model are organised into files in a directory. Each file addresses a different feature of the model. Initially the learner is presented with a simple model of a binary tree within which they can explore the concept of a heap. This approach is beneficial in two respects:

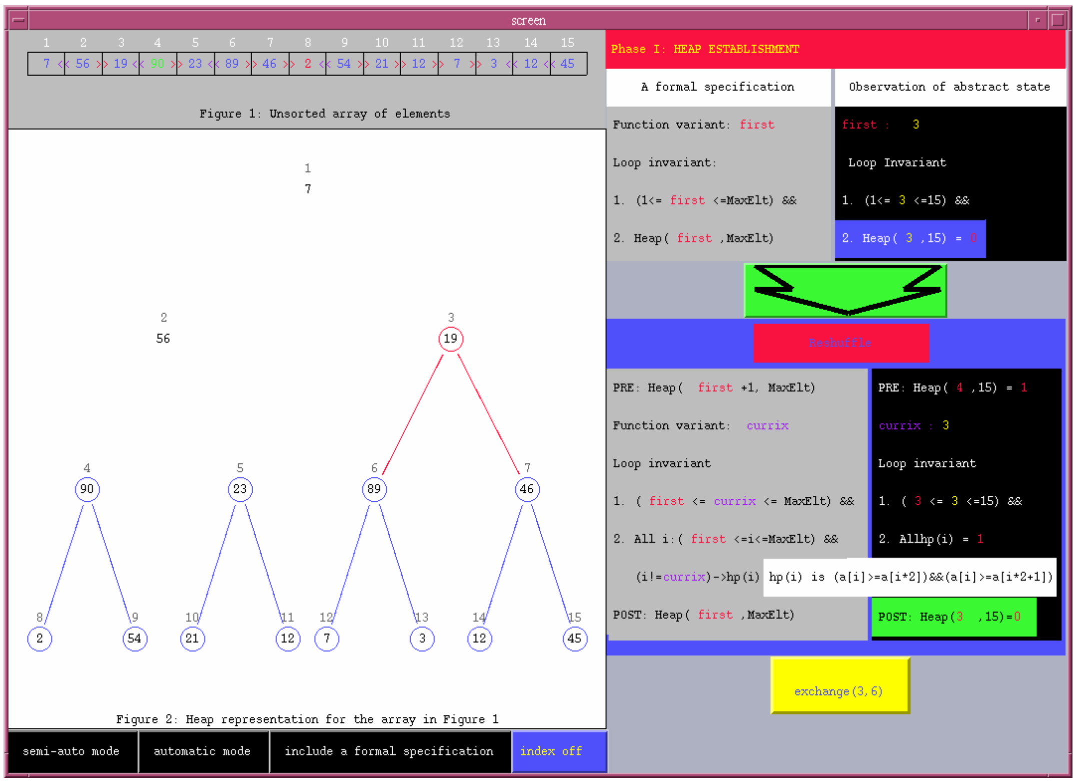
i) personally constructing the model gives the learner an appreciation of the key concepts and operations involved in heapsort.

ii) because of its interactive nature, the heapsort model allows the learner to experiment at each step in the construction to ensure that they understand the interaction between components.

Note that in the model building the learner has flexibility in how and when the supplementary files are added to the existing model, but in general they constrain the heapsort model towards the heapsort algorithm. The learner can follow a prescribed sequence of steps in the construction of the model (cf. the README file in [EMRep, heapsortBeynon1998]). The learner nonetheless has complete discretion over their interactions with the model, can interact outside the scope of conventional heapsort, or can manually perform heapsort operations. This flexible style of interaction is essential to experiential learning and provides the type of activities that can foster a full and deep understanding of the heapsort algorithm.

The heapsort model exemplifies the way in which EM can support the wide range of learning activities in the EFL. Stages 1 and 2 referred to above are concerned with understanding the observables and dependencies that are characteristic of the heap data structure and subsequently identifying the indivisible stimulus-response mechanisms that govern the behaviour of the heapsort algorithm. The discussion above has described how, in using the heapsort model, the learner initially focuses on the heap structure, then moves through manual operation of the sorting process to the automation of heapsort. The basic heap model is a definitive script that describes the state of the heap. It is gradually embellished with agent actions to represent necessary operations on the heap. This process reflects the emphasis on state-as-experienced as prior to behaviour-as-abstracted, fundamental to the EM approach, that was discussed in chapter 3. A further extension of the heapsort model that has been implemented by

Rungrattanaubol [Run02] complements the model with a formal specification of heapsort in the Weakest Precondition formalism [Dij76] (see Figure 6.8).



Phase I: HEAP ESTABLISHMENT

A formal specification	Observation of abstract state
Function variant: first	first : 3
Loop invariant:	Loop Invariant
1. (1<= first <=MaxElt) &&	1. (1<= 3 <=15) &&
2. Heap(first ,MaxElt)	2. Heap(3 ,15) = 0

PRE: Heap(first +1, MaxElt)	PRE: Heap(4 ,15) = 1
Function variant: currix	currix : 3
Loop invariant	Loop invariant
1. (first <= currix <= MaxElt) &&	1. (3 <= 3 <=15) &&
2. All i:(first <=i<=MaxElt) &&	2. Allhp(i) = 1
(i!=currix)->hp(i)	hp(i) is (a[i]>=a[i*2])&&(a[i]>=a[i*2+1])
POST: Heap(first ,MaxElt)	POST: Heap(3 ,15)=0

Figure 6.8 – Heapsort and its associated formal specification

In this extension, the logical relationships between key variables are themselves interpreted as observables at a high level of abstraction, as discussed in detail in [BRS00]. This extension of the heapsort model illustrates how EM can support learning activities across the whole of the EFL, from experimental interaction to formal reasoning.

6.4 The Robotic Simulation Environment

In this section, we describe a prototype EM learning environment developed to help learners understand the behaviour of robots in LEGO Mindstorms™, a system that allows computer programming and the construction of physical LEGO robots to be integrated. The learning environment to be described in this section is targeted at the particular scenario being used in Kids' Club, Joensuu, Finland. It shows how EM can be used to provide support for experiential learning activities.

The concept of out-of-school technology clubs for children has been successful in giving children a deeper understanding of, and confidence with, computer programming and technology in many different cultural contexts [RR96, RRC98, ESV^{+}02]. The primary aim of these clubs is to give children of various ages the chance to work on personally meaningful technology-based projects within a relaxed and informal setting. These clubs employ students and teachers to assist children in achieving their own personal goals without imposing rigid curricula or examinations on them. The first technology club for children was set up in 1993 and run by the Computer Museum in Boston in collaboration with the MIT Media Laboratory [Com03]. The aim was to create an atmosphere within which children and adults could collaboratively construct technological artefacts facilitated by computer technology. Many computer clubhouses now exist across the world under the auspices of the Intel Computer Clubhouse Network [Com03].

The Kids' Club, run by the Educational Technology Research Group based in the Computer Science Department at the University of Joensuu in Eastern Finland, is a recent initiative that shares the ideology of the Intel computer clubhouse. The motivations behind the Kids' Club are twofold: firstly to be an environment where children can undertake technological projects built on their own interests beyond the boundaries of the school curriculum; and secondly as a laboratory setting within which researchers can field test new educational technologies [ESV^{+}02]. Children are not merely participants in the technological phase of the project; they often contribute to the research that is being undertaken. The close involvement of the children in the research process is the main difference claimed by the Kids' Club organisers from the Intel Computer Clubhouses [ESV^{+}02].

There is a wide range of technology-based activities that children undertake in the Kids' Club. They have the opportunity to program computers, to use the Internet, to create interactive Java type animations and to use modern digital media such as video cameras. One task in which children have been heavily involved is in constructing LEGO™ Mindstorms robots (hereafter to be referred to as 'robots') and programming them to achieve tasks in a real-world environment. In the next section, we discuss the fundamentals of robot construction and programming and outline some of the conceptual problems facing learners. These problems motivate the EM learning environment to be discussed in section 6.4.2.

6.4.1 Building and programming robots

LEGO Mindstorms robot kits consist of conventional LEGO blocks and pieces, wheels, connectors, a programmable device, and sensors that can be attached together to build robots. Wheels are connected to motors that enable the robots to move. Robots have independently operating motors to drive the wheels on each side of the robot. This allows the robot to turn with only one motor running. Sensors can be attached to the robots that allow it to interact with its environment. For example, a touch sensor will return a positive value if it is touching an object in its environment. Each construction kit contains touch sensors, light sensors and colour sensors which are used in conjunction with the programming language so that the robot can interact with, and respond to, its environment. At the heart of a robot is the Robotic Command eXplorer (RCX) Programmable Brick. This is a large LEGO brick that provides the battery power for the motors and is also a computer that can store and run programs. The programs are written on a personal computer before being uploaded to the brick by an infrared communication device.

Robots can be constructed in many different ways. In our prototype environment, we have chosen one specific robot design. An example of such a robot can be seen in Figure 6.9: it has a motor attached to the pair of wheels on each side. In principle, our environment could be extended to encompass arbitrary robot design.

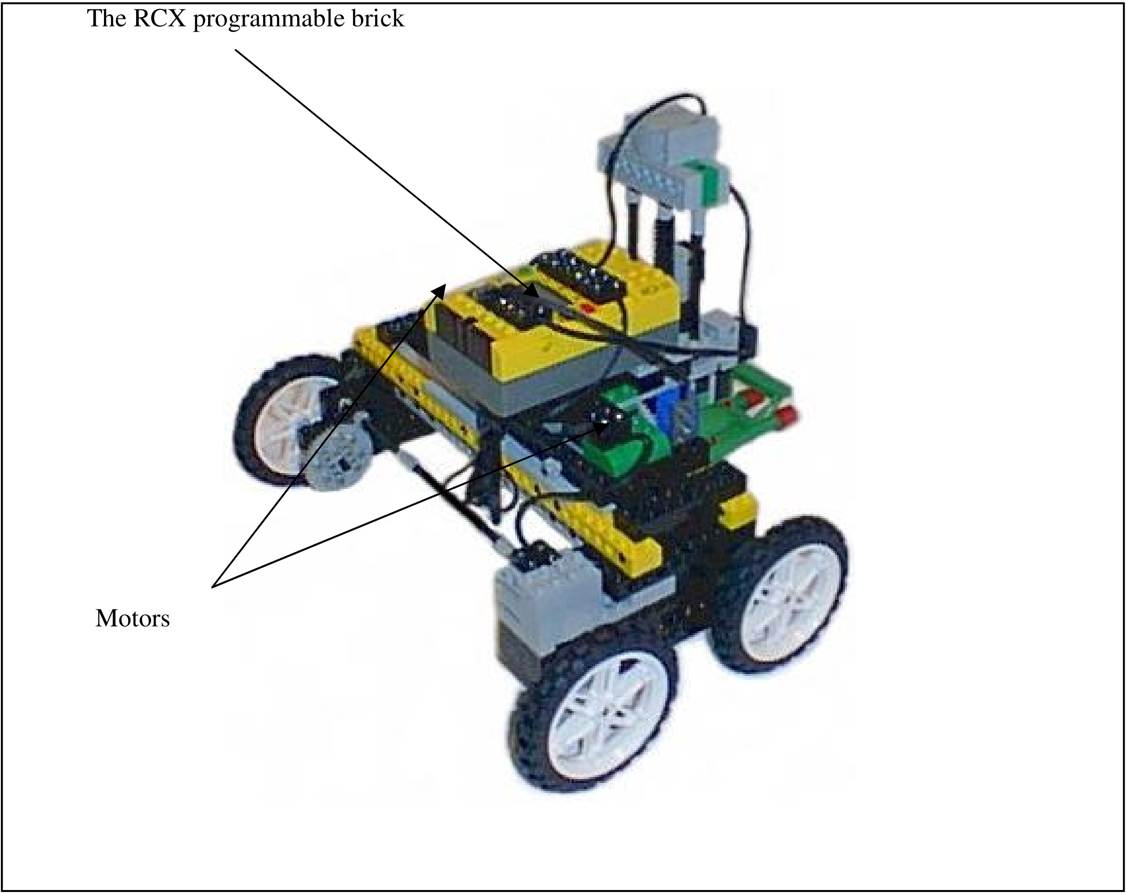


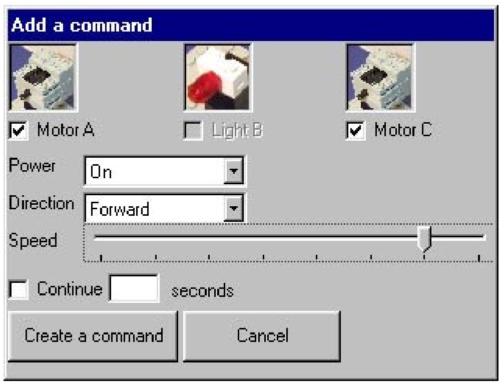
Figure 6.9 – An example robot and its main features

Robot programs are created on a computer and transferred to the programmable brick. The robot programming language has commands to control the motors that drive the wheels. The controls allow the robot to move forwards or backwards in a straight line, to rotate on the spot, to rotate in either direction centred on a wheel, or to move in a circular path forward or backward. Each motor can be turned on or off or have its speed or direction changed. The complex behaviour that can emerge from a set of simple commands requires a good understanding of how the commands map onto the real-world movement of the robot. There are commands that allow the robot to respond to input from the sensors. For instance, when a robot touches an obstacle, a touch sensor will return a true value that can trigger a change in the robot's movement. Programming language constructs such as 'if-then-else' or 'while-do' loops can be used to create more complex robot behaviours. Listing 6.1 illustrates how a robot can be programmed to move around a room. When it hits an obstacle, the robot reverses, changes direction and then moves forward again.

```
REPEAT FOREVER
     AB: On forward. speed 7.
     IF TS1 = 1
          AC: On reverse. speed 2. Continue 2 seconds.
          A: On forward. speed 2. Continue 1 second.
     END IF
END REPEAT
```

Listing 6.1 – An example robot program listing

Robot programs are developed on a computer using either a text editor or a specially designed programming environment such as the Instructive Portable Programming Environment (IPPE) [JKS02]. The IPPE allow commands to be built in a visual manner, thereby eliminating syntax errors. It uses a clipboard to store potentially useful commands before they are committed to a program. Commands can be added to a program from the clipboard in any order, and new ones can be placed on the clipboard at any time. In the IPPE, commands are created using dialog boxes such as that shown in Figure 6.10, where the command is 'Turn on motors A and C with speed 7' [JKS02].


Add a command

[✓] Motor A
[ ] Light B
[✓] Motor C

Power: On

Direction: Forward

Speed

[ ] Continue ___ seconds

Figure 6.10 – Using the IPPE to create a command

Complete programs can be transferred to the programmable brick using an infrared communication device. When the program has been uploaded to the robot it can be run and the behaviour of the robot can be observed. Following observation of the robot's behaviour, the program can be debugged if required by considering the problem, making changes to the program, transferring the program to the brick and running it. This leads to an iterative program development cycle comprising programming the robot, testing its behaviour and conceptual debugging (see Figure 6.11). Each of these phases occurs in a different context: programming on the computer, testing in the world and conceptual debugging in the mind.

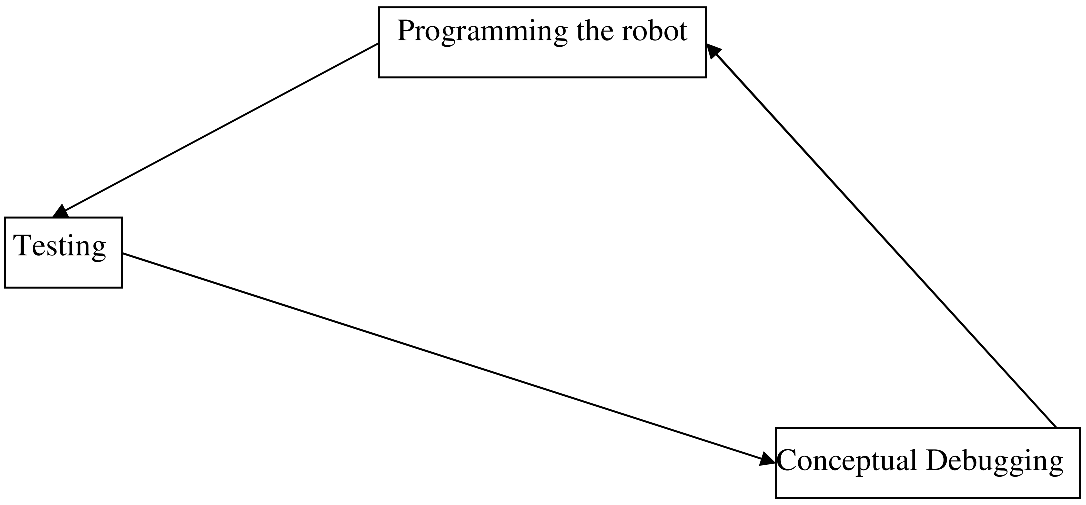


Figure 6.11 – The iterative robot programming cycle

The problem with the current programming environment, and the iterative developmental style outlined above, is that significant cognitive demands are placed on the learner. This becomes apparent when we consider the steps involved in a typical robot programming scenario.

In practice, each iteration of the development cycle outlined above can take several minutes. During this period, the learner has to remember the modifications they had made and what they were trying to correct, and at the same time check whether their revised program solves the problem through analysing the observed behaviour. If a shorter amount of time were required to complete a cycle, then – in addition to the intrinsic timesaving benefit – the cognitive demands on the learner would be drastically reduced.

A related problem in understanding a robot program is that the program code and the resulting behaviour are studied in separate environments. The learner's task – trying to grasp why the robot is not performing as it should, and working out how to debug the code – combines observation of the robot with analysis of the program code. This requires an understanding of how the available robot commands relate to primitive robot behaviour. If the learner has an inadequate construal of this relationship, the programming task is exceedingly difficult.

A further problem with the current robot programming approach – with its associated long feedback loop – is that it is difficult to perform experiments to determine the reasons for a robot's undesired behaviour. An environment in which commands and programs can be easily tested, that allows users to interact on a level appropriate to their understanding, promotes exploratory learning. This accords with diSessa's observation that a key factor in the design of learning environments that promote active learning is that it should be easy to explore personal hypotheses [diS01].

The demands discussed above can only be met if the learner has sufficient experience of the relationship between robot programs and robot behaviour. The construction of a robot simulation environment is motivated by the fact that novices do not possess the experience that is required to program successfully. A visualisation to make the relationship between the program code and the robot's behaviour more explicit would simplify the understanding of robot programming. This would reduce the cognitive load on the learner by bringing together the elements of programming, testing and conceptual debugging into a common environment. From an EFL perspective, robot programming in the existing environment assumes that the learner understands the relationship between robot program and robot behaviour and does not require support for the experiential learning activities that inform this understanding. This support can be established by providing features that enable exploratory learning to establish a solid construal.

In the next section we discuss a prototype EM robot simulation environment targeted at solving the problems associated with learning to program robots in the Kids' Club.

6.4.2 The Empirical Modelling Robotic Simulation Environment

The primary aim of the EM Robotic Simulation Environment (RSE) is to reduce the cognitive demands in learning to program the robots. The work that is reported in this section originated from group work with Pasi Eronen, Järi Järvela and Marjo Virnes at an Educational Technology summer school held in Finland in August 2002 [EJR$^+$02].

The RSE supports the development of a child's construal of robot behaviour by targeting a better understanding of how program commands are related to the behaviour of the robot. This is achieved through a layered environment that eliminates the long feedback loop associated with the current programming approach. With reference to the EFL, the RSE permits learners to undertake activities situated at the experiential end of the EFL, by providing different perspectives that can support the development of objective knowledge.

The RSE provides cognitive support for the key feedback loop that is driven by the problem that the learners are trying to solve (see Figure 6.12). Through observations about the robot behaviour in the environment they can test hypotheses and gain feedback on them. Solutions to their problems are the catalyst for successful concrete robot building. Learners' construals are used as the basis for solving new problems that are refined to suit the context of their new understanding.

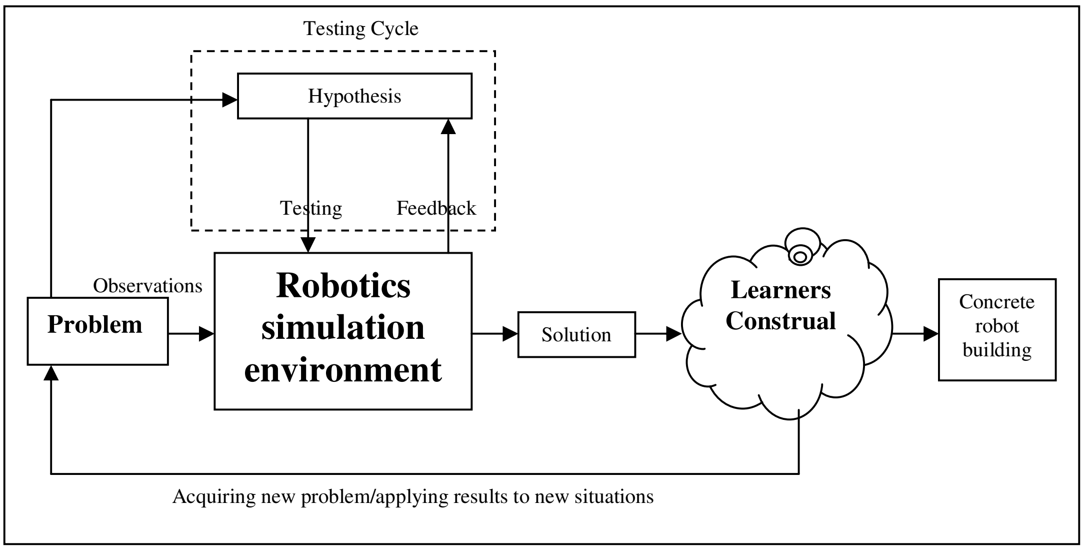


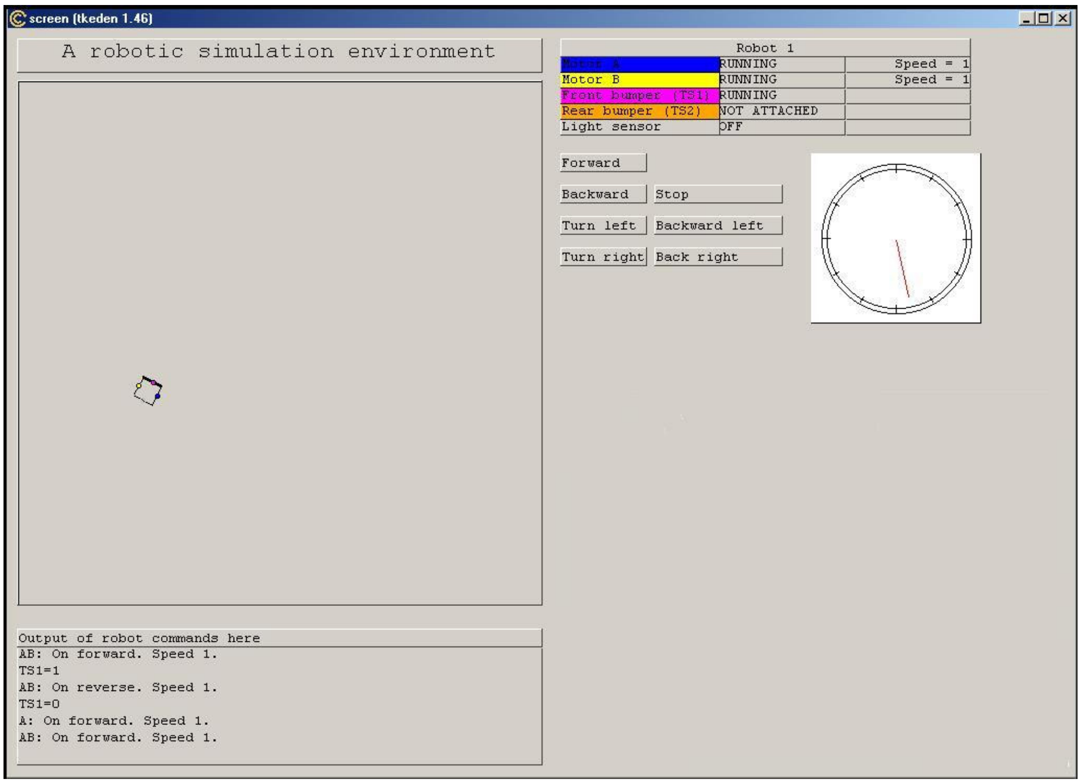
Figure 6.12 – Learning and the Robotic Simulation Environment [EJR+02]

The complexity and duration of the feedback loop is dramatically reduced in the RSE because of the computer-based environment and the quality and type of feedback provided. With the current approach, a large proportion of the time spent debugging is taken up with physical manipulation of the robot and observation of its behaviour in the real world. In a computer-based model, the resultant behaviour can be more easily correlated with the program that defines it through tracking the program commands as they are executed. For this purpose, the behaviour of the simulated robots must match that of their real-world counterparts, otherwise the environment would not allow for the transfer of knowledge from the RSE to robot programming. Ideally, the simulated robots should be programmed using the same programming language that is used to control the physical robots (cf. Listing 6.1). This feature has not yet been implemented in our prototype RSE but it would be possible to specify the programming language for the robot using the AOP (see section 5.4.1). In essence, the RSE allows the learner to concentrate on the essential features of programming the robots and supports experimental activities targeted at establishing or reinforcing their construal of robot behaviour [EJR+02].

6.4.3 Layering in the Robotic Simulation Environment

The RSE is designed to support learners at many different levels of competency [EJR+02]. Some learners will use the environment to learn about the basic concepts involved in robot programming. For instance, the learner may wish to explore the relationship between primitive robot movement and the programming commands needed to produce that movement. Other learners will already understand the basics of robot programming and will use the environment to test hypotheses. For instance, the learner may wish to confirm that a particular program successfully achieves its goal irrespective of the initial orientation of the robot. Both these agendas can be satisfied in a single layered environment. We now illustrate different types of use within the RSE by describing two different layers. For a fuller description, see [EJR+02].

The basic layer of the RSE allows learners to manipulate the robot using predefined controls to establish how robot movements are related to the underlying program code that produces them. This of course makes fundamental assumptions about the environment such as: the terrain is flat, the friction is uniform, and there is no wind. It also assumes that the robot has already been configured so that the touch and light sensors are attached. As depicted in Figure 6.13, the robot is controlled using a set of buttons labelled ‘Forward’, ‘Backward’ etc, to specify an intended movement of the robot. If the learner presses ‘Forward’ then the robot will move forward in a straight line until it receives a further command or until it hits an obstacle. By using these buttons and referring to the current status of the components, the learner can explore how the movement of the robot is related to the internal state of its motors. For instance, when the robot is moving forward, both motors are running in the same direction at the same speed. To provide scaffolding for writing robot programs, each time a button is pressed, the equivalent robot command for that movement is displayed using the syntax of the IPPE language. For instance, in Figure 6.13 pressing the forward button has set motors A and B to ‘`RUNNING`’ and the code output window shows ‘`A,B: on forward. speed 2`’. The important activity in this layer is concerned with understanding the basic repertoire of movements of the robot and the dependencies between commands and movements. The robots are assumed to possess some automatic stimulus-response mechanisms. For instance, if a robot touches a wall then the touch sensor will return a true value, which is registered in the code output window.



Robot 1		
Motor A	RUNNING	Speed = 1
Motor B	RUNNING	Speed = 1
Front bumper (TS1)	RUNNING	
Rear bumper (TS2)	NOT ATTACHED	
Light sensor	OFF	

```
Output of robot commands here
AB: On forward. Speed 1.
TS1=1
AB: On reverse. Speed 1.
TS1=0
A: On forward. Speed 1.
AB: On forward. Speed 1.
```

Figure 6.13 – The RSE being used to investigate the relationship between the motors and the robot’s movement

The exploratory nature of the RSE helps to develop the learner’s construal of the relationships between the movement of the robot, the setting of the motors attached to it and the equivalent program code needed to generate that behaviour. The learner needs to understand the fundamentals of robot behaviour in order to program the robots effectively. This support for establishing and reinforcing a learner’s construal is a major advantage that the RSE has over the IPPE.

A layer of the RSE to support more advanced learning gives learners the power to simulate the physical construction of a robot and write programs in the IPPE language. In programming the physical robot, sensors and motors must be attached to the robot, and wired to the programmable brick in order for program instructions to be passed to the motors and sensors. Forgetting to connect devices is a common mistake that children must learn to avoid. To support this in the RSE, attaching and connecting the motors are two separate actions, which reinforces the need for this to be carried out in the real world. To program the simulated robot the learner must enter code into the robot commands window (see Figure 6.15). In the current prototype, the robot is programmed using direct EDEN counterparts of the primitives and control structures available in the IPPE language. Because of this close syntactic similarity between EDEN and IPPE code, providing a translator from IPPE into EDEN using the AOP should not prove too technically demanding. By entering code fragments into the input window, it is envisaged that children will be able to write simple commands to replicate single operations, or construct programs using the full range of commands and high-level language constructs available in the IPPE language. When commands are being executed in the RSE, the status of the motors and sensors is kept up to date by dependency so that learners can use the environment to debug their real-world programs, or determine how their program influences the internal state of the robot.

The layers of the RSE are not intended to be completely separate and we envisage that learners writing their own programs could also use the direct manipulation controls to test a hypothesis about a situation they are trying to understand in their program. With reference to the EFL, this reflects the way in which learning activities migrate from the concrete to the abstract and vice versa as a learner consolidates his or her advanced understanding by exploring areas of uncertainty. The intention is that as learners become more competent they gain a better conceptual understanding of programming robots and more of their time is spent testing hypotheses and experimenting with real-world programming.

We now describe a practical example of how the RSE can be utilised by learners in trying to solve a programming task. The scenario described here is originally from [JKS02]. The objective is to write a program to navigate a robot around a rectangular obstacle, as shown in Figure 6.14.

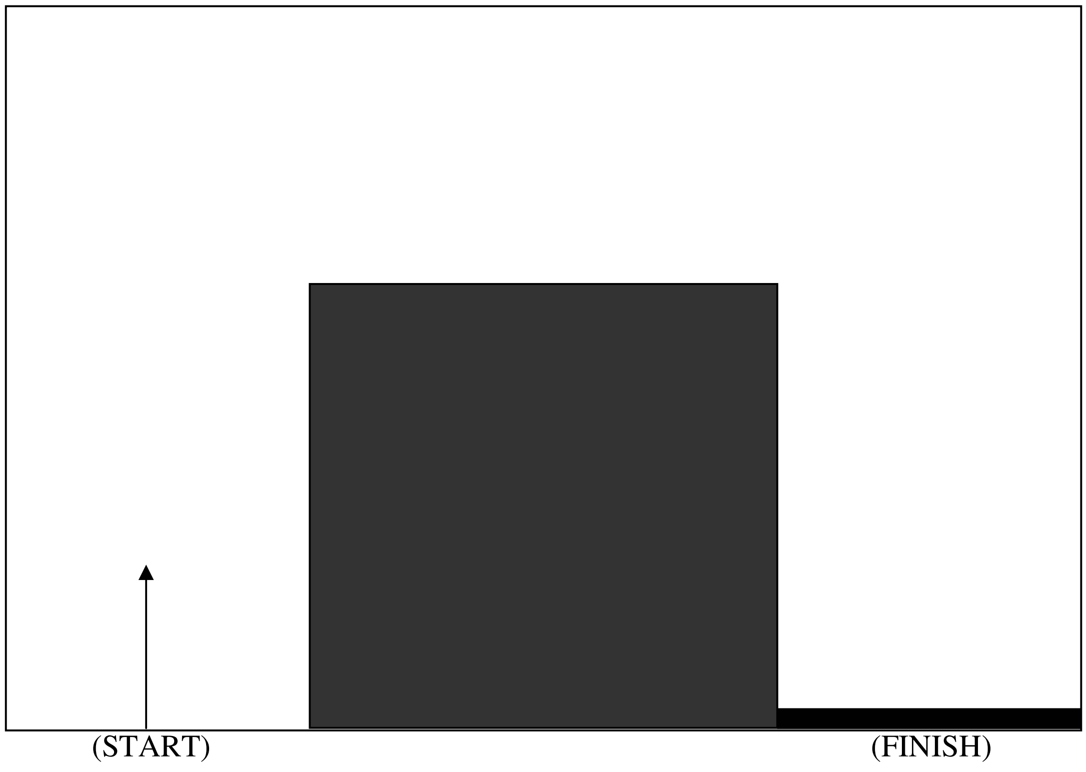
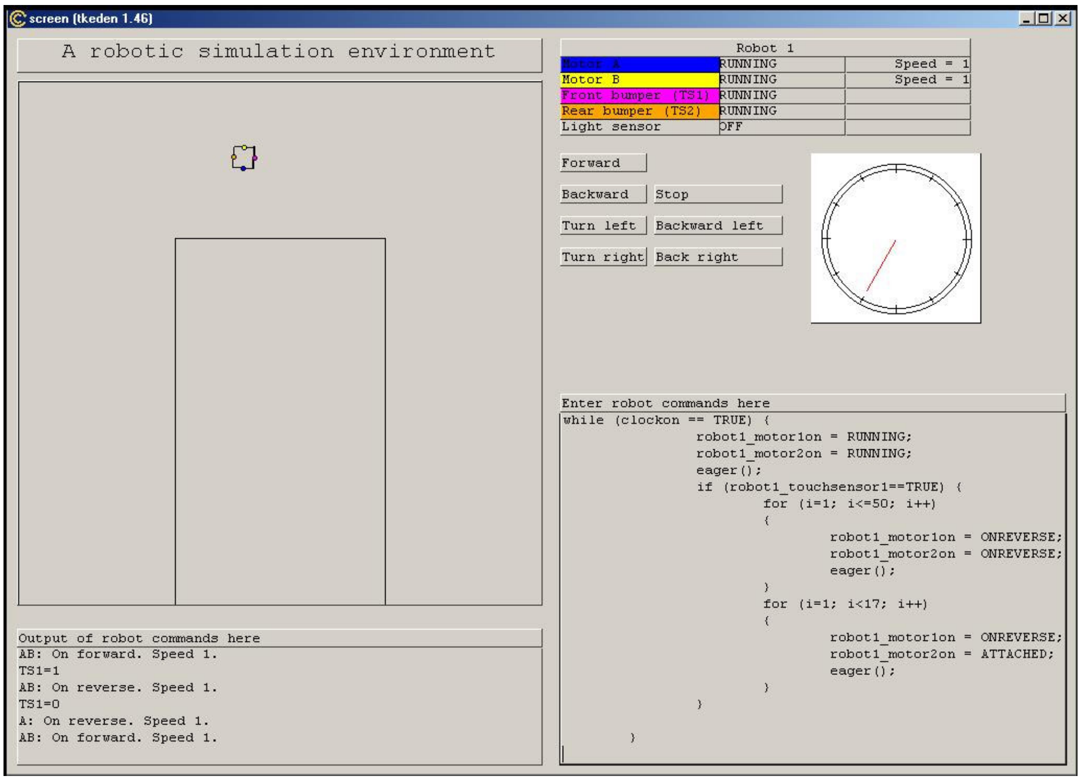


Figure 6.14 – An example task for a robot program to solve (from [JKS02])

Setting up the problem scenario in this example requires specifying the world in which the robot will interact. This simply contains four boundary walls to represent a square room and three interior walls to represent the obstacle. These walls are specified as definitions and can be changed whilst the robot is moving as when simulating a moving obstacle.

Learners can interact at either a direct manipulation level, or at a programming level in order to solve the problem. Direct manipulation may be used to understand the types of commands that are needed to solve the problem and to identify a starting point for writing a program to navigate around the obstacle. Alternatively learners can write a program for the robot. When the program is run, the behaviour of the robot can be observed and debugged as required. Figure 6.15 shows the RSE in use in solving the given problem.



Robot 1		
Motor A	RUNNING	Speed = 1
Motor B	RUNNING	Speed = 1
Front bumper (TS1)	RUNNING	
Rear bumper (TS2)	RUNNING	
Light sensor	OFF	

Enter robot commands here

```
while (clockon == TRUE) {
                robot1_motor1on = RUNNING;
                robot1_motor2on = RUNNING;
                eager();
                if (robot1_touchsensor1==TRUE) {
                        for (i=1; i<=50; i++)
                        {
                                robot1_motor1on = ONREVERSE;
                                robot1_motor2on = ONREVERSE;
                                eager();
                        }
                        for (i=1; i<17; i++)
                        {
                                robot1_motor1on = ONREVERSE;
                                robot1_motor2on = ATTACHED;
                                eager();
                        }
                }

        }
```

Output of robot commands here

```
AB: On forward. Speed 1.
TS1=1
AB: On reverse. Speed 1.
TS1=0
A: On reverse. Speed 1.
AB: On forward. Speed 1.
```

Figure 6.15 – The RSE in use in solving the task from Figure 6.14

This example shows that the RSE can be used to solve problems that are encountered in robot programming in the Kids Club. There are many ways robots can be constructed, and many tasks that the robots could be programmed to perform in the real-world. The RSE will not be able to support them all, but this example illustrates that it can play a role in establishing a conceptual understanding of robot programming.

In its present form, the RSE is not sufficiently developed to deal with robot programming in its full generality. However, the above example establishes proof-of-concept, and illustrates that the RSE can be applied to at least some of the problems that learners face when programming robots in the Kids' Club. There are many things that still need to be added to make it more suitable for its intended audience. Providing an interface for the IPPE language is one of the principal concerns. However, there are a number of other features that could be added to the RSE in order to enhance its usability. These include:

- a graphical interface to enable learners to create and manipulate the robots' environment interactively.
- a more realistic visual representation of the robot, potentially in 3D using the Sasami notation.
- support for developing and simulating different robot designs.

The RSE has further potential as a test bed for designing new features that enhance the robots. For example, it would be of interest to introduce communication sensors so that teams of robots could work collaboratively or competitively to achieve a task [EJR+02].

In this section, we have shown that the RSE can allow learners to gain an understanding of the concepts underlying robot programming without having to concentrate initially on specifying the behaviours of the robot in abstract program code. This approach allows the learner to intersperse concrete manipulation and abstract programming according to their current task and learning needs.

6.5 Chapter summary: Supporting learning across the EFL in Empirical Modelling

In this final section, we draw together the three characteristic case studies described in this chapter and relate them to the EFL. We argue that model building and model use – from the perspectives of many different learners – requires support for learning activities across the whole range of the EFL.

By way of illustration, in section 6.2, we discussed the MBF4 model from the perspective of a learner and a mathematical researcher. For a learner, the primary learning activity is in understanding the mathematical concepts embedded in the model. Learners require support for formal concepts through the visualisation of concrete examples, allowing them to move from the abstract end to the concrete end of the EFL. For the mathematical researcher, the primary learning activity is investigating potential connections between various mathematical objects. This task involves adding to the model in order to test out conjectures through experimentation with concrete examples.

Both model building and model-use can be associated with learning activities throughout the EFL. As discussed above, in respect of model building, the MBF4 model is primarily focused on the learning activities at the concrete end of the EFL. In respect of model use, the primary focus of the RSE in section 6.4 is to give learners access to an experimental environment in which they can seek to consolidate their understanding of programming LEGO Mindstorms robots. With reference to the EFL, this is associated with enabling learners to connect the formal symbols in robot programs with their concrete meanings in the world. The Heapsort model in section 6.3 – which integrates aspects of model building and model use – seeks to integrate the formal characteristics of the heapsort algorithm with the experiential elements that inform it. With reference to the EFL, this gives learners the scope to move either towards the abstract or towards the concrete in their exploration of heapsort.

The case studies described in this chapter – and more generally throughout the thesis – are practical evidence of the claim that EM can support learning across a wide range of learning activities and domain contexts.

Chapter 7 – Summary and Conclusions

7.0 Overview of the Chapter

In this chapter, we review the thesis, discuss possible future work, discuss limitations of the research and draw conclusions.

7.1 Review of the thesis

In this section, we review the contents of the thesis and evaluate them with reference to EM and learning.

In Chapter 2, we considered the qualities of software based on spreadsheet principles where learning is concerned. We identified a connection between the support spreadsheets offer to learning and their characteristics as tools for exploratory modelling. We examined how exploratory modelling is related to Cantwell Smith's semantic relation β and argued that it has two key aspects: negotiation and elaboration. Negotiation of the semantic relation β is essentially concerned with understanding the nature of a concept. Elaboration of the semantic relation β is essentially concerned with understanding how a concept can be applied in its wider domain context. We evaluated computer-based modelling tools in respect of supporting the two key aspects of the semantic relation β. We argued that spreadsheets are well suited to negotiation in certain domains but are limited in respect of elaboration. We also considered research products related to the spreadsheet and found that these also had limitations where supporting the semantic relation β is concerned. We introduced the practical EM tool, TkEden, and argued that it offers general support for both negotiation and elaboration of the semantic relation β, and hence is a good tool for exploratory modelling.

In Chapter 3, we outlined the challenges of exploiting ‘computers for learning’. These difficult challenges can be attributed in part to the differing concerns of the educationalist and the computer specialist. In learning, typical computer use can be separated into two types of activity: building models and using models. We argued that satisfying the requirements of the educationalist and the computer specialist requires that:

- in model building there is a close connection between the model construction approach and domain learning.
- in model use there is easy and flexible adaptability of software in response to different learning situations.

We introduced an experiential framework for learning that can be used to describe different domain learning activities on a spectrum that ranges from the concrete, empirical and private to the abstract, theoretical and public. We argued that this framework can be viewed as being generally applicable to any learning situation. We introduced the key features of EM: the development of construals; the emphasis on state-as-experienced being prior to behaviour-as-abstracted; and the concepts of observables, dependency and agency. We argued that model construction in EM can support learning activities, and migration between learning activities, across the whole range of the EFL. This close connection between EM and the EFL suggests that domain learning and model construction can be intimately linked.

In Chapter 4, we examined the relationship between model construction and domain learning from an educational perspective. We took a broad constructionist perspective on learning embracing bricolage and situated learning. We argued that all computer-based model construction approaches involve active knowledge construction through building a public entity, and hence satisfy Papert’s basic definition of constructionism. By relating instructionist and constructionist theories to the EFL, we established that instructionism is typically concerned with abstract learning activities and constructionism is typically concerned with concrete learning activities. We examined how three domain learning techniques: concept mapping, conventional programming and EM, are related to our broad constructionist perspective on learning and the EFL. We concluded that:

- concept mapping is primarily useful in brainstorming activities and that knowledge gained using it is typically set aside when constructing models.
- conventional programming is not well oriented to the broad perspective on constructionism adopted in this thesis because it emphasises planning, abstraction and circumscription. These emphases align programming with abstract learning in the EFL, and this detracts from its usefulness as an approach to model construction that promotes domain learning.
- an EM approach to model construction supports our broad perspective on constructionism and learning activities across the EFL, enabling effective domain learning to proceed in tandem with model construction.

In Chapter 5, we considered the advantages to learners, teachers and software developers in using EM to construct learning environments that support many different types of learning objective. We described three different types of learning that can be scaffolded: comprehending a fixed referent; exploring possibilities and invention; and learning domain-specific languages. In each of these types of learning environment, an initial seed model was embellished by extending or refining the existing model. The case studies used to illustrate scaffolding of EM learning environments exhibited advantages for:

- learners, since the learning environments lend themselves to exploration of the model domain and referent.
- teachers, since environments are – in principle – customisable resources that can be utilised to take advantage of their particular teaching requirements.
- software developers, since the approach to model development means that families of models can be easily created in response to teacher demands and flexibly adapted or extended to suit different contexts or learner competencies.

In Chapter 6, we described EM case studies illustrating model building and model use in relation to the EFL. We recalled the two categories in the use of computers for learning identified in chapter 3: the construction of computer-based models by learners; and the use of existing models where a user does not make any changes to the model. We argued that a third category was particularly appropriate in EM: where a partially complete model can be used as it stands, or extended to fulfil some learning criterion. The three case studies described in the chapter: the Free Distributive Lattice model; the Heapsort model; and the Robotic Simulation Environment give practical illustration to our claim that the use of EM in learning can support a wide range of learning activities within the EFL.

7.2 Future work

7.2.1 Empirical Testing

In this thesis, the focus has been on establishing a solid conceptual foundation for future EM-based educational research. It is the author's opinion that future work needs to be targeted at practical developments for use in educational situations.

The quality of computational resources in an educational environment is assessed with reference to the learning objectives to which they are being put. Many hundred models have been developed in practical EM case studies as a result of student project work and academic research. The majority of these models have not been developed with education in mind. There is a definite need to target schools more effectively with regard to the development of EM models as learning resources. In my view, this targeting should involve a close relationship with a particular school and model development based on their educational requirements. This arrangement would have a two-way benefit. The school – who are not so interested in the underlying computer implementation – will receive specifically relevant educational software. The research group would have the opportunity for valuable empirical testing of EM models in practical educational situations.

7.2.2 Comparative studies

To date, there has been no evaluation of EM in education in comparison with other programming languages or software environments. There is a need for comparative studies to ascertain whether there are quantitative and/or qualitative improvements in learning that occur in practice through the adoption of an EM approach. An exemplar for this type of comparative study can be seen in the Playground Project [Pla03], a 3-year European research project that compared the Toontalk programming environment [Too03] and a Logo-based language called Imagine [KB00]. Imagine was used to create a simple game-programming environment called Pathways [GKN+01]. Children built their own games on top of these environments, and these games were used as the vehicle for discussing how they coped with ideas such as building rules, cause and effects and issues of object-orientation (cf. section 5.3.3). The primary concern in the Playground project was to compare and contrast the quality of the children's evolving understanding of both their game domain and their programs in the two environments. Comparative studies of EM with other educational programming languages and software is essential to identify whether, in practice, the suitability of EM for learning argued in this thesis can be realised. Further, in this thesis we have argued that there are advantages in respect of domain learning when using EM models rather than conventional programming. We have explored this claim insofar as it relates to theoretical and conceptual issues. Practical evidence in support of such a claim can only arise from realistically scaled comparative testing, and not through the anecdotal accounts described in this thesis (and referenced from [Her02, Won03, Wan01]).

7.2.3 Developing an Empirical Modelling environment for children

The current EM tools are suitable for programmers and not for end-users, whether adults or children. Our own experiments in trying to introduce our tools to 17-18 year old students showed that non-computing students could not grasp the fundamental ideas underlying programming languages, such as functions or parameter passing. We had evidence that students grasped the conceptual ideas of analysing a domain such as the jugs game (discussed in section 2.4.4) in terms of observables, dependency and agency, but they could not translate these into the necessary computer definitions. The creation of a simpler development environment or visual language, developed in accordance with EM principles, could potentially enable students to construct their own models.

There have been many attempts at giving what Papert termed ‘the power of programming’ to children [Pap80]. Most share his vision for the Logo language: of providing the children with a tool with which they can construct their own models or programs according to their desires. Programming languages aimed at children typically take inspiration from a particular programming paradigm, and translate the important features of each paradigm into an appropriate and simple form. For example, Logo was a language aimed at children that was built on the procedural programming paradigm [Pap80]. Agentsheets (as discussed in section 2.3.3) uses a visual programming language to specify rules to give agents in a simulation [Rep93]. Toontalk associates computational notions such as procedures and message passing with actions such as training robots and sending birds back to their nests [Too03].
Without significant empirical testing, it is difficult to imagine how the principles of EM can be expressed using metaphors with which children will be comfortable. In order for an EM language aimed at children to be successful, it is essential to remove the syntactic barriers presented by our current notations. Previous research by Wong [Won98, Won03] has been targeted at developing end-user EM languages. However, the DMT tool (see chapter 4 and [Won03]) and the WING environment [Won98] – although they are both visual – are still too complicated for children to use in developing their own models. Both still require users to interact with complex syntax.

At present, the construction of models could proceed with children and expert modellers collaboratively building models. The child would provide the ideas and the expert would translate them into definitions and program code. It remains for future work to ascertain whether this is an appropriate mode of working from a pedagogical perspective. Careful planning and evaluation would be required to establish the educational benefits to the child in such an arrangement.

7.3 Conclusions

In this final section of the thesis, we identify limitations of the research described in this thesis and put forward the final conclusions that have been reached.

7.3.1 Reservations about the research

Although the EM research group has been in existence since the early 1980's, there has been relatively little empirical testing with software users, and very little use of the tools for model construction outside the University of Warwick. Although many models developed by the EM group have been deployed in 'one-off' trials with students, or demonstrated to teachers, only two have been more extensively tested in educational contexts:

- The current SQL-EDDI environment (see section 5.4.3) has been used over a three-year period by approximately eight hundred computer science students as part of an undergraduate database course. An earlier version of the EDDI interpreter was used in the previous two years by approximately five hundred computer science students. With this level of exposure, and the associated feedback from students, we have evidence for the educational merits of the environment (see [BBR+03] for more details).
- The Clayton Tunnel accident scenario (see section 2.4.5) has been used on several occasions as the basis for University of Warwick open days for 14-year-old children. Each session has lasted for an hour and involved children playing the roles of the participants in the situation and taking

part in discussions on railway safety issues. The educational benefit of such sessions is hard to evaluate, but the quality of the post-mortem discussions indicated that the close engagement with the model stimulated a positive learning experience.

The models described in chapters 5 and 6 that are targeted at educational settings (see e.g. racing cars (section 5.2.1), the OXO models (section 5.3.2), the clown-and-maze environment (section 5.4.2), the heapsort model (section 6.3) and the RSE (section 6.4)) have not been tested in educational contexts. To fully evaluate the claims made in this thesis, these models should be tested in the environment in which they are intended to be deployed. Only after such an evaluation can we be sure of the validity of our conclusions.

Writing by itself is not the ideal way to disseminate EM. In this thesis, we have argued that model construction in EM is different in character from developing a conventional program. It is difficult for the reader who is unfamiliar with EM to form a considered view on this claim without any practical exposure to developing models. In our opinion, such exposure is vital to fully appreciate the distinctive character of EM in relation to conventional programming. Whilst discussions in previous papers (see e.g. the discussion of the clock model in Appendix D and [Bey01] and the discussion of the lift model in [Bey03]) can give an impression of the open-ended and flexible nature of model construction, it is only through personal and practical engagement (as Papert himself would advocate!) that the reader can fully appreciate the nature of EM model development.

7.3.2 Conclusions of the thesis

In this thesis, we have explored the potential of EM in the application of computers for learning. This exploration has addressed its merits in respect of the construction of models, and the use of models.

In respect of model building, we have argued that in EM, the process of model construction is intimately linked with furthering domain understanding. This intimate link exists because of the support EM model construction offers for a wide range of learning activities, as reflected in the EFL. In particular, EM supports the concrete learning activities in the EFL that are more closely associated with constructing spreadsheets than writing programs. Furthermore, the model construction approach in EM allows the modeller to move at will between learning activities across the EFL in response to their learning needs at the time.

In respect of model use, we have argued that EM learning environments are flexibly adaptable in response to different learning contexts. This flexibility stems in part from the openness of the model development approach. The use of cognitive layering – the layered development of microworlds in which future layers are not preconceived, and can be flexibly adapted - has advantages for all the stakeholders in educational software development. Our case studies have illustrated that EM is able to support the wide range of different learning activities represented in the EFL, and confirm its potential as an approach for addressing the agenda of 'computers for learning'.

Bibliography

[4mat03] 4mation Educational Resources. Online at http://www.4mation.co.uk. [Accessed on November 1st, 2003].

[ABC+98] J.Allderidge, W.M.Beynon, R.I.Cartwright, Y.P.Yung. Enabling technologies for Empirical Modelling in Graphics. In *Proceedings of Eurographics UK*, 16th Annual Conference, pages 199-213, 1998.

[Acu03] ACUMEN spreadsheet system. Online at: http://www.threedgraphics.com/tdg/defaulta.asp. [Accessed on November 1st, 2003].

[Aep86] W.Aeppli. *Rudolf Steiner Education and the Developing Child*. Hudson, N.Y: Anthroposophic Press, 1986.

[AFC00] The Alliance For Childhood. *Fools Gold: A critical look at computers in childhood*. Report into the use of computers by children. Online at: http://www.allianceforchildhood.net/projects/computers/computers_reports.htm [Accessed on November 1st, 2003]

[Age03] Agentsheets Inc Website at http://www.agentsheets.com [Accessed on November 1st, 2003]

[AHU82] A.V.Aho, J.E.Hopcroft, J.D.Ullman. *Introduction to Algorithms and Data Structures.* Addison-Wesley, 1982.

[Akm00] V.Akram. Rethinking context as a social construct. In *Journal of Pragmatics* 32(6), pages 743-759, 2000.

[ARS96] J.R.Anderson, L.M.Reder, H.A.Simon. Situated learning and education, *Educational Researcher 25 No.4*, pages 5-11, May 1996.

[Ath02] J.S.Atherton. *Learning and Teaching*. Online at: http://www.dmu.ac.uk/~jamesa/learning/contents.htm, 2002. [Accessed on November 1st, 2003].

[Aus68] D.Ausubel. *Educational psychology: A cognitive view*. New York: Holt, Rinehart, and Winston, 1968.

[AZ93] L.Anderson-Inman, L.Zeitz. Computer-based concept-mapping: Active studying for active learners. *The Computing Teacher*, 21(1). 6-8, 10-11. (EJ 469 254), August/September 1993.

[BAB+96] W.M.Beynon, D.Angier, T.Bissell, S.Hunt. DoNaLD: A line-drawing system based on definitive principles. *Research Report RR-86*, Department of Computer Science, University of Warwick, 1983.

[BAD+01] M.Burnett, J.Atwood, R.Djang, H.Gottfried, J.Reichwein, S.Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. In *Journal of Functional Programming* 11(2), pages 155-206, March 2001.

[BB95] T.Buzan, B.Buzan. *The Mind Map Book*. BBC Consumer Publishing, 1995.

[BB96] G.Brassard, P.Bratley. *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[BBC+01] A.F.Blackwell, C.Britton, A.Cox, T.R.G.Green, C.Gurr, G.Kadoda, M.S.Kutar, M.Loomes, C.L.Nehaniv, M.Petre, C.Roast, C.Roe, A.Wong, R.M.Young. Cognitive Dimensions of Notations: Design tools for Cognitive Technology. In *Proceedings 4th International Conference on Cognitive Technology: Instruments of Mind*, University of Warwick, August 2001, LNAI 2117, Springer-Verlag, pages 325-341, 2001.

[BBF84] R.Burton, J.S.Brown, G.Fischer. Skiing as a model of instruction. In B.Rogoff, J.Lave (eds), *Everyday cognition: Its development in social context*, Cambridge MA: Harvard University Press, pages 139-150, 1984.

[BBR+03] W.M.Beynon, A.Bhalerao, C.Roe, A.Ward. A computer-based environment for the study of relational query languages. In *Proceedings of the Teaching, Learning and Assessment in Databases Workshop*, Coventry, United Kingdom, 14th July, pages 104-108, 2003.

[BBY92] W.M.Beynon, I.Bridge, Y.P.Yung. Agent-oriented modelling for a vehicle cruise controller. In *Proceedings ESDA Conference*, ASME PD-Vol 47-4, pages 159-165, 1992.

[BC95] W.M.Beynon, R.I.Cartwright. Empirical Modelling principles for cognitive artifacts. In *Proceedings IEE Colloquium: Design systems with users in mind: The role of cognitive artifacts*, Digest No.95/231, 8/1-8/8, December 1995.

[BCD89] J.S.Brown, A.Collins, S.Duguid. Situated cognition and the culture of learning. *Educational Researcher, 18*(1), pages 32-42, 1989.

[BCH+01] W.M.Beynon, Y-C Chen, H-W Hseu, S.Maad, S.Rasmequan, C.Roe, J.Rungrattanaubol, S.Russ, A.Ward, A.Wong. The computer as instrument. In *Proceedings 4th International Conference on Cognitive Technology: Instruments of Mind*, University of Warwick, August 2001, LNAI 2117, Springer-Verlag, pages 476-489, 2001.

[Bec00] K.Beck. *eXtreme Programming explained*, Addison Wesley, 2000.

[Ben99] F.Bennett. *Computers as tutors: Solving the crisis in education*. Faben Inc., 1999.

[Ben01] M.Ben-Ari. Constructivism in Computer Science Education. In *Journal of Computers in Mathematics and Science Teaching*, 20 (1), pages 45-73, 2001.

[Bey74] W.M.Beynon. Combinatorial aspects of piecewise-linear maps. In *Journal London Mathematical Society* (2) 7, pages 719-727, 1974.

[Bey83] W.M.Beynon. A definition of the ARCA notation. *Research Report RR-54*, Department of Computer Science, University of Warwick, 1983.

[Bey86] W.M.Beynon. The LSD notation for communicating systems. *Research report RR-87*, Department of Computer Science, University of Warwick, 1986.

[Bey87a] W.M.Beynon. Definitive Principles for Interactive Graphics. *Research report RR-93*, Department of Computer Science, University of Warwick, 1987.

[Bey87b] W.M.Beynon. Monotone Boolean Functions as Combinatorially Piecewise Linear Maps. *Research report RR-109*, Department of Computer Science, University of Warwick, 1987.

[Bey97] W.M.Beynon. Empirical modelling for educational technology. In *Proceedings of Cognitive Technology 1997*, pages 54-68. University of Aizu, Japan, IEEE, 1997.

[Bey98] W.M.Beynon. Modelling state in mind and machine. *Research Report RR-337*, Department of Computer Science, University of Warwick, 1998.

[Bey99] W.M.Beynon. Empirical modelling and the foundations of artificial intelligence. In *Proceedings Computation for Metaphors, Analogy and Agents,* Lecture Notes in Artificial Intelligence 1562, Springer, pages 322-364, 1999.

[Bey01] W.M.Beynon. Liberating the computer arts. In *Proceedings of the 1st International Conference on Digital and Academic Liberty of Information (DALI'2001)*, University of Aizu, Japan, 2001.

[Bey03] W.M.Beynon. Radical Empiricism, Empirical Modelling and the nature of knowing. In K.Freyburg, H-J.Petsche, B.Klein (Eds). *Proceedings of the WM 2003 Workshop on Knowledge Management and Philosophy.* Luzern, April 2003. Online at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-85/ [Accessed on November 1st, 2003].

[BFP+03] K.Brady, M.B.Forton, D.Porter, C.Wood. *Rules in school*. Northeast foundation for children, 2003.

[BGA56] J.Bruner, J.Goodnow, A.Austin. *A study of thinking*. New York: Wiley, 1956.

[Bir95] G.Birkhoff. *Lattice theory (3rd edition)*. American Mathematical Society Colloquium Publications, Vol. 25, 1995.

[BJ94] W.M.Beynon, M.S.Joy. Computer Programming for Noughts-and-Crosses: New Frontiers. In *Proceedings PPIG'94*, Open University, pages 27-37, January 1994.

[BMS+02] M.Ben-Ari, N.Myller, E.Sutinen, J.Tarhio. Perspectives on program animation with Jeliot. *Software Visualization: International Seminar*. Dagstuhl Castle, Germany, Lecture Notes in Computer Science 2269, pages 31-45, 2002.

[BNO+90] W.M.Beynon, M.T.Norris, R.A.Orr, M.D.Slade. Definitive specification of concurrent systems. In *Proceedingss UKIT'90*, IEE Conference Publications 316, pages 52-57, 1990.

[Boe76] B.W.Boehm. Software Engineering. In *IEEE Transactions on Computers*, Vol. C-25, No.12, pages 1226-1241, 1976.

[BR99] H.J.Becker, J.Ravitz. The influence of computer and internet use on teachers' pedagogical practices and perceptions. Journal of Research on Computing in Education, 31(4), pages 356–384, 1999.

[Bra78] C.Brainerd. *Piaget's theory of intelligence*, Englewood Cliffs, NJ: Prentice-Hall, 1978.

[Bro95] F.P.Brooks. Jr. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, 1995.

[Brö95] P.Brödner. The two cultures in engineering. In *Skill, Technology and Enlightment*, Springer-Verlag, pages 249-260, 1995.

[Bro01] C.Brown. An agent-based parsing system in Eden. Final year project, Department of Computer Science, University of Warwick, 2001.

[BRR00] W.M.Beynon, S.Rasmequan, S.B.Russ. The use of Interactive Situation Models for the development of business solutions. In *Proceedings workshop on Perspective in Business Informatics Research (BIR-2000)*, Rostock, Germany, March 31-April 1, 2000.

[BRS$^+$98] W.M.Beynon, J.Rungrattanaubol, P.H.Sun, A.E.M.Wright. Explanatory models for open-ended human computer interaction. *Research report RR-346*, Department of Computer Science, University of Warwick, 1998.

[BRS00] W.M.Beynon, J.Rungrattanaubol, J.Sinclair. Formal specification from an observation-oriented perspective. *Journal of Universal Computer Science,* Vol. 6 (4), pages 407-421, 2000.

[BRW$^+$01] W.M.Beynon, C.Roe, A.Ward, A.Wong. Interactive situation models for cognitive aspects of user-artefact interaction. In *Proceedings 4th International Conference on Cognitive Technology: Instruments of mind*, University of Warwick, LNAI 2117 Springer-Verlag, pages 356-372, 2001.

[BS98] W.M.Beynon, P.H.Sun. Empirical Modelling: A new approach to understanding requirements. In *Proceedings 11th International Conference on Software Engineering and its Applications*, Vol 3, Paris, December 1998.

[BS99] W.M.Beynon, P.H.Sun. Computer-mediated communication: A distributed Empirical Modelling perspective. In *Proceedings of the 3rd International Conference on Cognitive Technology (CT'99)*, pages 115-132, August 1999.

[BSR99] M.Burnett, A.Sheretov, G.Rothermel. Scaling up a 'What you see is what you test' methodology to testing spreadsheet grids. In *1999 IEEE Symposium on Visual Languages*, Tokyo, Japan, pages 30-37, September 13-16, 1999.

[Buc90] J.Buckle. *Computational aspects of lattice theory*. PhD Thesis, Department of Computer Science, University of Warwick, March, 1990.

[BWM+00] W.M.Beynon, A.Ward, S.Maad, A.Wong, S.Rasmequan, S.B.Russ. The Temposcope: A computer instrument for the idealist timetabler. In *Proceedings of the third international conference on the practice and theory of automated timetabling*, Constance, Germany, pages 153-175, 16-18th August 2000.

[Car90] J.M.Carroll. *The Nurnberg Funnel: Designing minimalist instruction for practical computer skill*. Cambridge, MA: MIT Press, 1990.

[Car94] A.Cartwright. *Application of definitive scripts to CACD*. PhD Thesis, Department of Computer Science, University of Warwick, July 1994.

[Car99] R.I.Cartwright. *Geometric Aspects of Empirical Modelling: Issues in Design and Implementation*, PhD Thesis, Department of Computer Science, University of Warwick, May 1999.

[Car00] B.Carter. Solids Animation Simulator And Modelling Interface. Final year project, Department of Computer Science, University of Warwick, 2000.

[CCS96] D.Canfield-Smith, A.Cypher, K.Schmucker. Making programming easier for children. *Interactions*. Volume 3, Issue 5, Sept/Oct, 1996.

[Che01] Y-C.Chen. *Empirical Modelling for participative business process reengineering*. PhD Thesis, Department of Computer Science, University of Warwick, December 2001.

[CG95] A.Cockburn, S.Greenberg. TurboTurtle: A collaborative microworld for exploring Newtonian physics. In *ACM Conference on Computer Supported Cooperative Learning (CSCL 95)*, Bloomington, Indiana, pages 62-66, Lawrence Erlbaum Associates Inc, October 1995

[CK95] M.Campbell-Kelly. Development and structure of the international software industry, 1950-1990. In *Business and Economic History*, Volume 24, no. 2, pages 94-107, Winter 1995.

[Cod70] E.F.Codd. A Relational Model of Data for Large Shared Data Banks. *CACM 13*, No.6, 1970.

[Com03] The Computer Clubhouse (Boston). Website - http://www.computerclubhouse.org (Accessed on November 1st, 2003).

[Coo93] P.A.Cooper. Paradigm shifts in Designed Instruction: From behaviourism to cognitivism to constructivism. In *Educational Technology*, Vol 33 (5), pages 12-19, 1993.

[Cov98] M.Covington. *The will to learn: A guide for motivating young people*. New York: Cambridge University Press, 1998.

[CRB$^+$98] E.Chi, J.Riedl, P.Barry, J.Konstan. Principles for information visualization spreadsheets. *IEEE Computer graphics and applications*, Volume 18, No.4, pages 30-38, July/August 1998.

[CRB00] Y-C.Chen, S.B.Russ, W.M.Beynon. Empirical Modelling for Business Process Reengineering: An experience-based approach. In *Proceedings workshop on Perspective in Business Informatics Research (BIR-2000*), Rostock, Germany, March 31-April 1, 2000.

[CW95] I.B.Cohen, R.S.Westfall (Selected and Edited), *Newton*, Norton, 1995.

[Dat87] C.J.Date. *A guide to INGRES*. Addison-Wesley, 1987.

[Dat00] C.J.Date. *An introduction to database systems* (7th edition). Reading, Mass.:Addison-Wesley, 2000.

[DD93] K.Dunn, R.Dunn. *Teaching secondary students through their individual learning styles: Practical approaches for grades 7-12*. Allyn & Bacon: Boston, 1993.

[DD00] C.J.Date, H.Darwen. *Foundations for future database systems: The Third Manifesto* (2nd edition). Reading, Mass.:Addison-Wesley, 2000.

[Deu88] M.S.Deutsch. Focusing real-time systems analysis on user operations. *IEEE Software*, September 1988.

[Deu89] M.S.Deutsch. Enhancing testability with scenario oriented engineering. In *Proceedings of the 6th International Conference on Testing Computer Software*, 1989.

[Dew16] J.Dewey. *Democracy and Education*. New York: MacMillan, 1916.

[Dew38] J.Dewey. *Experience and Education*. New York: MacMillan, 1938.

[DFE03] DfES (Department for Education and Skills). Fulfilling the potential: Transforming teaching and learning through ICT in schools, DfES publications, 2003.

[Dij76] E.W.Dijkstra. *A discipline of programming*. Prentice Hall, 1976.

[diSA86] A. diSessa, H. Abelson. Boxer: A Reconstructible Computational Medium.*Communications of the ACM*, 29 (9), pages 859 - 868, 1986.

[diS97a] A.diSessa. Twenty reasons why you should use Boxer (instead of Logo). In M.Turcsanyi-Szabo (Ed.), *Learning and exploring with Logo: Proceedings of the sixth European Logo conference*, pages 7-27, 1997.

[diS97b] A.diSessa. Open toolsets: New ends and new means in learning mathematics and science with computers. In E.Pehkonen (Ed.) *Proceedings of the 21st Conference of the International Group for the Psychology of Mathematics Education.* Vol. 1. Lahti, Finland, pages 47-62, 1997.

[diS01] A.diSessa. *Changing minds: Computers, learning and literacy*. MIT Press, 2001.

[Dri00] M.P.Driscoll. *Psychology of Learning for Instruction* (2nd Ed). Boston: Allyn and Bacon, 2000.

[Dug99] V.Duggal. CADD primer: A general guide to Computer Aided Design and Drafting. Mailmax Publishing, 1999.

[EMRep, <project_name>] The Empirical Modelling Repository. Online at http://empublic.dcs.warwick.ac.uk. The second argument is the name of the project in the repository.

[EJR$^+$02] P.J.Eronen, J.Jarvela, C.Roe, M.Virnes. Using Empirical Modelling to simulate robotics in Kids' Club. *2nd Annual Finnish/Baltic Sea Conference on Computer Science Education (Koli Calling*), University of Joensuu, Finland, 18-20 October, pages 63-68, 2002.

[Ern95] P.Ernest. The one and the many. In L.P.Steffe, J.Gale (eds), *Constructivism in education*, pages 459-486, Hillsdale, NJ: Lawrence Erlbaum Associates, 1995.

[ESV$^+$02] P.J.Eronen, E.Sutinen, M.Vesisenaho, M.Virnes. Kids' Club as an ICT-Based learning laboratory. *Informatica Journal* 13 (4), pages 1-12, 2002.

[FB00] C.N.Fischer, W.M.Beynon. Empirical Modelling of Products. In *Proceedings International Conference on Simulation and multimedia in engineering education*, Phoenix, Arizona, pages 20-26, The Society for Modelling and Simulation International, 2001.

[Fri70] P.Freire. *Pedagogy of the oppressed*. Penguin Books. 1970.

[FRK+01] K.Fernandes, V.Raja, J.Keast, W.M.Beynon, P.S.Chan, M..Joy. Business and IT perspectives on AMORE: a methodology for Object-Orientation in re-engineering enterprises. In P.Henderson (Ed.), *Systems Engineering for Business Process Change: New Directions*, Springer-Verlag, December 2001.

[FWS92] R.A.Finke, T.B.Ward, S.M.Smith. *Creative Cognition: Theory, Research and Applications.* The MIT Press, 1992.

[Gar93] H.Gardner. *Frames of mind: The theory of multiple intelligences*. (2nd Edition), New York: Basic Books. 1983.

[GBJ+00] D.Grune, H.Bal, C.Jacobs, K.Langendoen. *Modern Compiler Design*, Wiley, 2000.

[GDL] P.Gloor, S.Dynes, I.Lef. Animated Algorithms (A Hypermedia Learning Environment for Introduction to Algorithms). CD-ROM.

[Geh96] D.Gehring. Spreadsheets and programming. In *Proceedings of the PPIG '96 Student Workshop*, pages 53-58, 1996.

[GG00] J.Glynn, T.W.Gray. *The beginner's guide to Mathematica Version 4*, Cambridge University Press, 2000.

[GKL+01] P.Gerdt, P.A.M.Kommers, C-K.Looi, E.Sutinen. Woven stories as a cognitive tool. In *Proceedings 4th International Conference on Cognitive Technology: Instruments of mind*, University of Warwick, LNAI 2117 Springer-Verlag, pages 233-247, 2001.

[GKN+01] R.Goldstein, I.Kalas, R.Noss, D.Pratt. Building Rules. In *Proceedings 4th International Conference on Cognitive Technology: Instruments of mind*, University of Warwick, LNAI 2117 Springer-Verlag, pages 267-281, 2001.

[Gog96] J.A.Goguen. Formality and informality in requirements engineering. In *Proceedings 2nd International Conference on Requirements Engineering*, pp 102-108, 1996.

[Goo90] D.Gooding. *Experiment and the making of meaning*. Kluwer Academic Publishers, 1990.

[GON92] C.Graci, R.Odendahl, J.Narayan. Children, chunking and computing. *Journal of Computing in Childhood Education* 3, 3/4, pages 247-258, 1992.

[Gre89] T.R.G.Green. Cognitive dimensions of notations. In *People and Computers V*, A.Sutcliffe, L.Macaulay (Eds), Cambridge University Press: Cambridge, pages 443-460, 1989.

[Gro02] T.A.Grossman. Spreadsheet engineering: A research framework. In *Proceedings EUSPRIG 2002,* pages 23-34, 18th-19th July, 2002.

[Har87a] D.Harel. *Algorithmics*. Addison-Wesley, 1987.

[Har87b] D.Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, Vol. 8, pages 231-274, July 1987.

[Har88] D.Harel. On visual formalisms. *Communications of the ACM*, Vol. 31, No. 5, May, pages 514-530, 1988.

[Har03] A.Harfield. Agent-oriented parsing with Empirical Modelling. Final year project, Department of Computer Science, University of Warwick, 2003.

[HCC01] Conference Paper Call for the *IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01) (Symposium on End-User Programming)*, Stresa, Italy, September, 2001. Available Online at http://iis.cse.eng.auburn.edu/~SEUP/ [Accessed on November 1st, 2003].

[Hea99] J.Healy. *Failure to connect: How computers affect our children's minds - for better and worse*. Simon and Schuster, 1999.

[Her02] T.Heron. *Programming with dependency*. MSc dissertation, Department of Computer Science, University of Warwick, September 2002.

[HK92] E.Heeren, P.A.M.Kommers. Flexibility of expressiveness: A critical factor in the design of concept mapping tools for learning. In P.A.M.Kommers, D.Jonassen, T.Mayes (eds.) *Cognitive tools for learning*. Berlin: Springer-Verlag, pages 85 - 101, 1992.

[HO96] B.R.Hergenhahn, M.H.Olsen. *An introduction to theories of learning*. Prentice Hall, 1996.

[How82] W.S.Howell. *The empathic communicator*. Belmont, CA: Wadsworth, 1982.

[HS73] J.R.Hartley, D.H.Sleeman. Towards more intelligent teaching systems. In *International Journal of Man-Machine Studies, 5(2)*: pages 215-236, 1973.

[Hud94] S.Hudson. User interface specification using an enhanced spreadsheet model. *ACM transactions on graphics* 13(4), pages 209-239, 1994.

[Ill71] I.Illich. *Deschooling society.* Marion Boyars, London, 1971.

[IMC+98] T.Igarashi, J.D.Mackinlay, B-W.Chang, P.T.Zellweger. Fluid visualization of spreadsheet structures. In *Proceedings of IEEE Symposium on Visual Languages*, pages 118-125,1998.

[Ins93] D.Helfgott, M.Helfgott, B.Hoof. Inspiration, The visual way to quickly develop and communicate ideas, Inspiration Software Inc. http://www.inspiration.com, 1993. [Accessed on November 1st, 2003]

[Jam90] W.James. *Principles of psychology*. (2 volumes), Holt, New York, 1890.

[Jam96] W.James. *Essays in Radical Empiricism*. Bison Books. 1996.

[JCJ+92] I.Jacobson, M.Christeron, P.Jonson, G.Overgaard. *Object-oriented Software Engineering: A use-case driven approach.* Addison-Wesley, 1992.

[JKS02] I.Jormanainen, O.Kannusmaki, E.Sutinen. IPPE - How to visualize programming with robots. In *Second Program Visualisation Workshop, The 7th Annual Conference on Innovation and Technology in Computer Science Education*, Aarhus, Denmark, June 24-26, 2002.

[Joh83] P.N.Johnson-Laird. Mental models: towards a cognitive science of language, inference and consciousness, Cambridge: Mass.: Harvard University Press, 1983.

[Joh03] J.M.Johnson. Then, Now and Beyond: From lofty beginnings to the age of accountability. In *Learning and Leading with Technology*, Volume 30, Number 7, International Society for Technology in Education, April 2003.

[Jon91] D.H.Jonassen. Objectivism versus Constructivism: Do we need a new philosophical paradigm? In *Journal of Educational Research*, 39 (3), pages 5-14, 1991.

[Jon92] D.H.Jonassen. Evaluating constructivistic learning. In T.M.Duffy, D.H.Jonassen (eds), *Constructivism and the technology of instruction: A conversation*, Hillsdale, N.J: Lawrence Erlbaum Associates, pages 137-148, 1992.

[Kah01] K.Kahn. Is Toontalk a colleague, competitor, successor, sibling, or child of Logo? In *Proceedings of EuroLogo*, 2001.

[Kay84] A.Kay. Computer Software. *Scientific American* 251(3), pages 52-59, September 1984.

[Kay96] A.Kay. Revealing the elephant: The use and misuse of computers in education. EduCom 31 (4), 1996.

[KB00] I.Kalas, A.Blaho. Imagine… New Generation of Logo: Programmable pictures. In *Proceedings of WCC2000*, Beijing, pages 427-430, 2000.

[KC98] H.Kirkpatrick, L.Cuban. Computers make kids smarter - right? TECHNOS Quarterly, Vol.7, No.2, Summer 1998.

[Ker92] J.Kernane. Address to the Maryland Instructional Computer Coordinators Association conference in Baltimore, 1992. (Appeared in Driver/Education Newsletter, 1992).

[KJM92] P.A.M.Kommers, D.H.Jonassen, T.J.Mayes. *Cognitive tools for learning*. Berlin: Springer-Verlag, 1992.

[Kno70] M.Knowles. *The modern practice of adult education: Andragogy versus Pedagogy*. New York: Association Press, 1970.

[Kno90] M.Knowles. *The Adult Learner: A neglected species* (4th Edition). Houston: Gulf Publishing, 1990.

[Kol84] D.Kolb. *Experiential Learning: Experience as the source of learning and development*. Prentice Hall, 1984.

[KR96] Y.B.Kafai, M.Resnick. Introduction. In Y.B.Kafai, M.Resnick (eds). *Constructionism in practice: Designing, thinking, and learning in a digital world*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1996.

[Lan03] M.Langer. *Microsoft Office Excel 2003 for Windows: Visual QuickStart guide*. Peachpit Press, 2003.

[Lav88] J.Lave. *Cognition in practice: Mind, mathematics and culture in everyday life.* New York: Cambridge University Press, 1988.

[Law85] R.Lawler. *Computer experience and cognitive development: A child's learning in a computer culture*. Ellis Horwood, 1985.

[LCS03] Logo Computer Systems Inc. http://www.microworlds.com. [Accessed on November 1st, 2003].

[Lev68] C.Levi-Strauss. *The savage mind*. University of Chicago Press. 1968.

[Lev94] M.Levoy. Spreadsheets for images. In *Computer Graphics (Proceedings SIGGRAPH '94)*, Volume 28, No.4, pages 139-146, ACM Press, New York, 1994.

[Lew51] K.Lewin. *Field theory in social science*. New York: Harper Collins, 1951.

[Lew90] C.Lewis. NoPumpG: Creating interactive graphics with spreadsheet machinery. In E.Glinert (ed.) *Visual Programming Environments: Paradigms and systems*, IEEE CS Press, Los Alamitos, California, pages 526-546, 1990.

[LJ98] M.Loomes, S.Jones. Requirements engineering: A perspective through theory-building. In *Proceedings 3rd International IEEE Conference on Requirements Engineering*. IEEE Computer Society Press, pages 100-107, 1998.

[LW91] J.Lave, E.Wenger. *Situated Learning: Legitimate peripheral participation*. New York: Cambridge University Press, 1991.

[Maa02] S.Maad. *An Empirical Modelling approach to software system development in finance: Applications and prospects*. PhD Thesis, Department of Computer Science, University of Warwick, March 2002.

[Mar92] J.Martin. *Rapid Application Development*, Prentice Hall, 1992.

[Mer91] M.D.Merrill. Constructivism and Instructional Design. In *Educational Technology*, 31 (5), pages 45-53, 1991.

[Min86] M.Minsky. *The society of mind*. New York, NY: Simon & Schuster, 1986.

[MKT93] E.McLean, L.A.Kappelman, J.P.Thompson. Converging end-user and corporate computing. In *Communications of the ACM* (36:12), pages 79-92, December 1993.

[MO94] B.Means, K.Olsen. The link between technology and authentic learning. In *Educational Leadership*, vol.5, pages 15-18, February 1994.

[MPG+02] N.A.M.Maiden, P.Pavan, A.Gizikis, O.Clause, H.Kim, X.Zhu. Making Decisions with Requirements: Integrating i* Goal Modelling and the AHP. In *Proceedings REFSQ'2002 Workshop*, 9-10 September 2002, Essen, Germany, 2002.

[Nar93] B.Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, 1993.

[Nau95] P.Naur. *Knowing and the mystique of logic and rules*. Kluwer Academic Publishers, 1995.

[Nau01] P.Naur. Anti-philosophical Dictionary. naur.com publishing, 2001.

[Nes97] P.E.Ness. *Creative software development -- An Empirical Modelling framework*. PhD Thesis, Department of Computer Science, University of Warwick, 1997.

[New01] L.Newton, L.Rogers. *Teaching science with ICT*. Continuum International Publishing Group, 2001.

[NG84] J.D.Novak, D.B.Gowin, *Learning how to learn.* Cambridge University Press, Cambridge, 1984.

[NH92] R.Noss, C.Hoyles. Looking back and looking forward. In R.Noss, C.Hoyles (Eds.), *Learning mathematics and Logo*, pages 431-468, MIT Press, 1992.

[NH96] R.Noss, C.Hoyles. Windows on mathematical meanings: learning cultures and computers. Dordrecht: Kluwer, 1996.

[NM91] B.Nardi, J.Miller. Twinkling lights and nested loops: Distributed problem solving and spreadsheet development. International Journal of Man-Machine studies 34, pages 161-194, 1991.

[Nor83] D.A.Norman. Some observations on mental models. In D.Gentner, A.L.Stevens (Eds.), *Mental models*, Lawrence Erlbaum Associates, pages 7-14, 1983.

[Nor98] D.A.Norman. *The Invisible Computer: Why good products can fail, the personal computer is so complex, and information appliances are the solution*. Cambridge, Mass., MIT Press, 1998.

[NS72] A.Newell, H.A.Simon. *Human problem solving*. Prentice Hall, 1972.

[OED97] The Oxford English Dictionary, Oxford University Press, 1997.

[OGL03] The OpenGL graphics library website. Online at http://www.opengl.org. [Accessed on November 1st, 2003].

[Opp97] T.Oppenheimer. The computer delusion. *Atlantic Monthly*, July 1997. Also Online at: http://www.theatlantic.com/issues/97jul/computer.htm. [Accessed on November 1st, 2003].

[Opp03] T.Oppenheimer. *Flickering minds: The false promise of technology in the classroom and how learning can be saved*. Random House, 2003.

[Ost96] J.Ostwald. *Knowledge construction in software development: The evolving artifact approach*. PhD dissertation, University of Colorado at Boulder, 1996.

[Pai01] J.Paine. Safer spreadsheets with model master. In *Proceedings EuSprig2001*, Amsterdan, Netherlands, pages 17-38, 2001.

[Pan00] R.R.Panko. Two corpuses of spreadsheet errors. *Proceedings of the 33rd International Conference on System Sciences*. Maui, Hawaii, January 4-7, 2000.

[Pap80] S.Papert. Mindstorms: *Children, Computers and powerful ideas*. New York: Basic Books, 1980.

[Pap93] S.Papert. *The Children's Machine*. New York: Basic Books, 1993.

[PH91] S.Papert, I.Harel. Situating Constructionism. In S.Papert, I.Harel (eds). *Constructionism: Research reports and essays*, Ablex Publishing, pages 1-11, 1991.

[PH96] R.R.Panko, R.P.Halverson Jr. Spreadsheets on trial: A survey of research on spreadsheet risks. *Hawaii International Conference System Sciences*, Maui, Hawaii, Jan 2-5, 1996.

[PH97] R.R.Panko, R.P.Halverson Jr. Are two heads better than one? (At reducing errors in spreadsheet modelling?), In *Office Systems Research Journal* (15:1), pages 21-32, Spring 1997.

[Pik89] B.Pike. Creative training techniques handbook. Lakewood Publications, 1989.

[Pla03] The Playground Project - Online at http://www.ioe.ac.uk/playground/ [Accessed on November 1st, 2003].

[Pol57] G.Polya. *How to solve it: A new aspect of mathematical method.* Princeton, NJ, 1957.

[Pos92] N.Postman. *Technopoly: The surrender of culture to technology*. Vintage Books, 1992.

[Pra98] D.Pratt. *The construction of meaning in and for a stochastic domain of abstraction*. PhD dissertation, Institute of Education, University of London, 1998.

[PRS+94] J.Preece, Y.Rogers, H.Sharp, D.Benyon, S.Holland, T.Carey. *Human Computer Interaction*. Addison-Wesley, 1994.

[RA97] A.Repenning, J.Ambach. The Agentsheets Behaviour Exchange: Supporting social behaviour processing. *CHI 97, Conference on Human Factors in Computing Systems, Extended Abstracts*, (Atlanta, Georgia), pp26-27, ACM Press.

[Ras01] S.Rasmequan. An approach to computer-based knowledge representation for the business environment using Empirical Modelling. PhD Thesis, Department of Computer Science, University of Warwick, November 2001.

[RB02] C.Roe, W.M.Beynon. Empirical Modelling principles for learning in a cultural context. In *Proceedings 1st International Conference on Educational Technology in Cultural Context*, University of Joensuu, Finland, pages 151-172, September 2002.

[RBF00] C.Roe, W.M.Beynon, C.N.Fischer. Empirical Modelling for the conceptual design and use of products. In *Proceedings International Conference on Simulation and multimedia in engineering education*, Phoenix, Arizona, pages 27-32, The Society for Modelling and Simulation International, 2001.

[Rep93] A.Repenning. *Agentsheets: A tool for building domain-oriented dynamic, visual environments*. PhD Dissertation, University of Colorado at Boulder, 1993.

[Rep00] A.Repenning. Agentsheets: An interactive simulation environment with end-user programmable agents. In *Proceedings of the IFIP Conference on Human Computer Interaction* (INTERACT '2000), Tokyo, Japan, 2000.

[Res87] L.B.Resnick. Learning in school and out. *Educational Researcher*, 16(9), pages 13-20, 1987.

[Res94] M. Resnick. *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds.* MIT Press, 1994.

[RI01] A.Repenning. A.Ionnidou. Engaging learners through simulation-based design. In C. Daetwyler (Eds.), *The use of Computers in (medical) Education* (German title: Zeitschrift für Hochschuldidaktik) Austrian Association for Didactics in Higher Education, 2001.

[RIA98] A.Repenning, A.Ioannidou, J.Ambach. Learn to communicate and communicate to learn. *Journal of Interactive Media in Education*, 98 (7), www-jime.open.ac.uk/98/7, 1998.

[RIZ00] A.Repenning, A.Ioannidou, J.Zola. Agentsheets: End-user programmable simulations. In *Journal of Artificial Societies and Social Simulation*, Vol.3, No.3, June 2000. Online at http://www.soc.surrey.ac.uk/JASSS/3/3/forum/1.html [Accessed on 1st November, 2003].

[RJM$^+$98] M.Roussos, A.Johnson, T.Moher, J.Leigh, C.Vasilakis, C.Barnes. Learning and Building Together in an Immersive Virtual World. In *Presence*, vol 8, no 3, June, 1999, special issue on *Virtual Environments and Learning*; W.Winn, M.J.Moshell (Eds.) MIT Press, pages 247-263, 1998.

[Rob02] Robocup 2002 - http://www.robocup2002.org/ [Accessed on November 1st, 2003]

[Roe99] C.Roe. *The EMPTY project*, MSc thesis, Department of Computer Science, University of Warwick, 1999.

[Rog90] B.Rogoff. *Apprenticeship in thinking: Cognitive development in social context*. New York: University Press, 1990.

[Rol82] L.T.C.Rolt. *Red for danger*. Pan Books 4th Edition, 1982.

[ROP88] M.Resnick, S.Ocko, S.Papert. Lego, Logo and Design. *Children's Environmental Quarterly*, 5(4), pp. 14-18, 1988.

[RR96] M.Resnick, N.Rusk. Access is not enough: Computer clubhouses in the Inner City. *American Prospect*, no. 27, pages 60-68, July-August 1996.

[RRB00] S.Russ, S.Rasmequan, W.M.Beynon. An experience-based approach to decision support systems. *Working conference on Decision Support through Knowledge Management*, Stockholm, Sweden, 2000.

[RRC98] M.Resnick, N.Rusk, S.Cooke. The Computer Clubhouse: Technological fluency in the Inner city. In D.Schon, B.Sanyal, W.Mitchell (Eds). *High Technology and Low-income communities*. Cambridge: MIT Press, pages 266-286, 1998.

[RRP+98] A.Repenning, M. Rausch, J. Phillips, A. Ioannidou. Using Agents as a Currency of Exchange between End-Users. In *Proceedings of the WebNET 98 World Conference of the WW, Internet, and Intranet*, Orlando, FL, Association for the Advancement of Computing in Education, pp. 762-767, 1998.

[RRR00] S.Rasmequan, C.Roe, S.Russ. Strategic decision support systems: An experience-based approach. In *Proceedings of the 18th International Association of Science and Technology for Development (IASTED) Conference on Applied Informatics*, Innsbruck, Austria, 14-17th February 2000.

[RS94] A.Repenning, T.Sumner. Programming as problem solving: A participatory theater approach. *Workshop on Advanced Visual Interfaces (AVI'94)*, pages 182-191, Bari, Italy, 1994.

[Run02] J.Rungrattanaubol. *A treatise on modelling with definitive scripts*, PhD Thesis, Department of Computer Science, University of Warwick, 2002.

[Rus97] S.B.Russ. Empirical Modelling: the computer as a modelling medium. *BCS Computer Bulletin*, pages 296-301, April 1997.

[Sae67] P.Saettler. *A history of instructional technology*. New York, N.Y: Mc Graw Hill, 1967.

[SGH94] E.Soloway, M.Guzdial, K.E.Hay. Learner-centered design the challenge for HCI in the 21st century. *Interactions*, pages 36-48, April 1994.

[SH93] V.W.Setzer, R.Hirata Jr. Algoritmos e sua análise: uma introdução didática. São Paulo: *Caderno da Revista do Professor de Matemática*, Brazilian Mathematical Society, Vol. 4, No. 1, pp. 1-38, 1993. English translation: Algorithms and their analysis: A pedagogical introduction. Online at http://www.ime.usp.br/~vwsetzer/alg/alg-eng.html. [Accessed on November 1st, 2003].

[She03] Sherston Educational Software. Online at http://www2.sherston.com. [Accessed on November 1st, 2003].

[Ski74] B.F.Skinner. *About behaviourism*. Random House Inc, 1974.

[Smi97] B.Cantwell-Smith. One hundred billion lines of C^{++}. *CogSci News*, Volume 10, Number 1, Spring 1997.

[Sol93] E.Soloway. Should We Teach Students to Program? *CACM,* 36(1), October 1993.

[Ste94] M.Steinberger. Where does Programming fit in? *Logo Update.* Vol. 2 (3), 1994.

[Suc87] L.A.Suchman, *Plans and situated actions: the problem of human-machine communication*, Cambridge University Press, 1987.

[Sun99] P.H.Sun, *Distributed Empirical Modelling and its application to software system development,* PhD Thesis, Department of Computer Science, University of Warwick, July 1999.

[Tal95] S.Talbott. *The future does not compute: Transcending the machines in our midst.* O'Reilly and Associates Inc, 1995.

[Tap98] D.Tapscott. *Growing up digital: The rise of the Net generation.* McGraw Hill, 1998.

[Tcl03] Tcl/Tk Developer Xchange. Online at http://www.tcl.tk/ [Accessed on November 1st, 2003]

[Tod76] S.J.P.Todd. The Peterlee Relational Test Vehicle-- A system overview, *IBM Systems Journal* 15(4), pages 285-308, 1976.

[Too03] Toontalk website - http://www.toontalk.com. [Accessed on November 1st, 2003].

[TP91] S.Turkle, S.Papert. Epistemological Pluralism and the revaluation of the concrete. In S.Papert, I.Harel (eds), *Constructionism: Research reports and essays 1985-90*, MIT media laboratory, 1991.

[Tur95] S.Turkle. *Life on the screen: Identity in the age of the Internet*. New York: Simon and Schuster, 1995.

[Tur96] M.Turner. *The literary mind*. Oxford University Press. 1996.

[UNC03] The United Kingdom National Curriculum. Online at http://www.nc.uk.net/index.html. [Accessed on November 1st, 2003].

[VK96] A.Varshney, A.Kaufman. FINESSE: A financial information spreadsheet. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 70-71, 1996.

[Vyg62] L.Vygotsky, *Thought and language*, Cambridge, MA: MIT Press, 1962. Original work published in 1934.

[Wan03] R.Wang. Modelling for software system development: Object-Oriented and Empirical Modelling perspectives. MSc dissertation, University of Warwick, January 2003.

[Wil93] U.Wilensky Connected Mathematics: Building Concrete Relationships with Mathematical Knowledge. PhD Thesis. Cambridge, MA: MIT Media Laboratory, 1993.

[Wil99] U.Wilensky. NetLogo. http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL, 1999.

[WL90] N.Wilde, C.Lewis. Spreadsheet-Based interactive graphics: From prototype to tool. *ACM Conference on Human Factors in Computing Systems*, Seattle, Washington, pages 153-159, 1990.

[Won98] A.Wong. *Implementing a definitive notation for windowing and graphics using Modd*, Final year project, Department of Computer Science, University of Warwick, 1998.

[Won03] A.Wong. *Before and Beyond Systems: An Empirical Modelling approach.* PhD Thesis, Department of Computer Science, University of Warwick, January 2003.

[YB01] T.Yeshno, M.Ben-Ari. Salvation for bricoleurs. In *13th workshop of the Psychology of Programming Interest Group*, Bournemouth, UK, pages 225-235, 2001.

[Yun90] Y.W.Yung. *EDEN: An engine for definitive notations*. MSc dissertation, Department of Computer Science, University of Warwick, September 1990.

[Yun93] Y.P.Yung. *Definitive Programming: A paradigm for exploratory programming*, PhD Thesis, Department of Computer Science, University of Warwick, January 1993.

Appendix A – An example of constructing a model for the simple game of Jugs.

The game of jugs is formulated as follows: There are two jugs of known quantities that have no intermediate markings on them. The aim of the game is to achieve a specific quantity of water in one of the jugs. The only permissible operations are to empty or fill a jug or pour water from one jug to the other such that the destination is full or the source is empty.

To aid in the elaboration of the model construction each of the lines of the model have been numbered, although these numbers are not part of the script. To begin we can define the capacities of the two jugs, which we will denote A and B respectively.

```
1. %eden
/* set some capacities of the jugs */

2. capA = 5;
3. capB = 7;
```

We use the line drawing notation to specify the display of the jugs. The size of the jugs has a fixed width in this model and a height that is dependent on the capacities of the jug.

```
/* develop an interface for the jugs */
4. %donald

5. int scalefactor
6. scalefactor = 100

7. point Abotleft, Abotright
8. line Abase, Aleft, Aright
```

```
Abotleft = {100,100}
Abotright = Abotleft + {300,0}
Abase = [Abotleft,Abotright]
Aleft = [Abotleft, Abotleft+{0,capA!*scalefactor}]
Aright = [Abotright, Abotright+{0,capA!*scalefactor}]

point Bbotleft, Bbotright
line Bbase, Bleft, Bright

Bbotleft = Abotleft + {400,0}
Bbotright = Bbotleft + {300,0}
Bbase = [Bbotleft,Bbotright]
Bleft = [Bbotleft, Bbotleft+{0,capB!*scalefactor}]
Bright = [Bbotright, Bbotright+{0,capB!*scalefactor}]
```

In the DoNaLD notation, variables have to be declared and are strongly typed unlike in EDEN. The ‘=’ sign in DoNALD has the same functional equivalence as the `is` operator in EDEN in maintaining dependencies. The reference points of the jug display are dependent on the base point of the jug (`Abotleft`). The position of jug B is dependent on jug A. The variable `scalefactor` is used to scale the display of the jug heights to be visible. The ‘!’ operator is used to reference EDEN variables in DoNaLD. Statements are terminated by line breaks unlike EDEN which uses the ‘;’ symbol

```
%eden

/* initial contents of the jugs */

contentA = 0;
contentB = 0;
```

We can now define the amount of water in the jugs. Note that comments in the EDEN notation are between the /* */ symbols.

```
/* some information about the jugs */

24. Afull is (capA==contentA);
25. Bfull is (capB==contentB);
26. Aempty is (contentA==0);
27. Bempty is (contentB==0);
```

These definitions are commonsense observations of the jug situation. A jug is only full if its content is equal to its capacity and is empty if it has no water. These conceptual observables define information we can ascertain by looking at a jug.

```
/* now we can define the surface of the liquid */

28. %donald

29. line Asurface
30. Asurface = [Abotleft+{0,contentA!*scalefactor},
Abotright+{0,contentA!*scalefactor}]
31. line Bsurface
32. Bsurface = [Bbotleft+{0,contentB!*scalefactor},
Bbotright+{0,contentB!*scalefactor}]
```

We have now added a water surface to the jugs model in the line drawing. The surface is represented as a line from the left side of the jug to the right side of the jug. Its height is dependent on the content of the jug.

```
%eden

/* now we define some jug operations */

proc fillingA {
  if (!Afull) {
      contentA++;
      eager();
      fillingA();
  }
}

proc fillingB {
  if (!Bfull) {
      contentB++;
      eager();
      fillingB();
  }
}

proc emptyingA {
  if (!Aempty) {
      contentA--;
      eager();
      emptyingA();
  }
}

proc emptyingB {
  if (!Bempty) {
      contentB--;
      eager();
      emptyingB();
  }
}

proc pouringAtoB {
  if ((!Bfull)&&(!Aempty)) {
      contentA--;
      contentB++;
      eager();
      pouringAtoB();
  }
}

proc pouringBtoA {
  if ((!Afull)&&(!Bempty)) {
      contentB--;
      contentA++;
      eager();
      pouringBtoA();
  }
}
```

These six actions are the important state transitions. They conform to the basic operations in the jugs game, namely emptying, filling and pouring. They can be tested by running them in the input window with commands of the form ‘`emptyA();`’. Each procedure runs as a stepwise operation since we cannot (in practice) determine the current content of a jug by inspection if it is partially full. Each step consists of checking whether an enabling condition is satisfied, making a stepwise change to the contents, updating the display and then continuing by recursively calling the same action.

```
/*now develop a scout interface for the main operations*/

%scout

window fillA = {
  type: TEXT
  string: "Fill A"
  frame: ([{10, 10}, {80, 30}])
  border : 1
  sensitive: ON
};

window fillB = {
  type: TEXT
  string: "Fill B"
  frame: ([{90, 10},{160, 30}])
  border : 1
  sensitive: ON
};

window emptyA = {
  type: TEXT
  string: "Empty A"
  frame: ([{170, 10}, {240, 30}])
 border : 1
 sensitive: ON
};

window emptyB = {
  type: TEXT
  string: "Empty B"
  frame: ([{250, 10}, {320, 30}])
104.   border : 1
105.   sensitive: ON
106. };

107. window pourAB = {
108.   type: TEXT
109.   string: "A -> B"
110.   frame: ([{330, 10}, {400, 30}])
111.   border : 1
112.   sensitive: ON
113. };

114. window pourBA = {
115.   type: TEXT
116.   string: "B -> A"
117.   frame: ([{410, 10}, {480, 30}])
118.   border : 1
119.   sensitive: ON
120. };
```

The lines 78-120 define interface windows to activate the emptying, filling and pouring operations. The user does not then need to know the procedure names but can perform operations using a graphical user interface. The Scout notation is used to define the windows. Each window is sensitive to mouse button events (sensitive: ON). When the mouse is clicked in a window a definition for that mouse event is created. This can be used to trigger an action based on that variable's new value.

```
121. display operations = <fillA/ fillB/ emptyA/ emptyB/
pourAB/ pourBA>;

122. screen = operations;
```

Lines 121 and 122 group the set of windows into a display and then set the screen to display those windows on the interface, as shown in Figure A.1.

```
123. %eden

/* now we can define some routines to be triggered on the
mouse click on a window */

124. proc filljugA : fillA_mouse_1 {
125.      if (fillA_mouse_1[2]==5) { /*on button
release*/
126.           fillingA();
127.      }
128. }

129. proc filljugB : fillB_mouse_1 {
130.      if (fillB_mouse_1[2]==5) { /*on button
release*/
131.           fillingB();
132.      }
133. }

134. proc emptyjugA : emptyA_mouse_1 {
135.      if (emptyA_mouse_1[2]==5) { /*on button
release*/
136.           emptyingA();
137.      }
138. }

139. proc emptyingjugB : emptyB_mouse_1 {
140.      if (emptyB_mouse_1[2]==5) { /*on button
release*/
141.            emptyingB();
142.        }
143. }

144. proc pourAB : pourAB_mouse_1 {
145.        if (pourAB_mouse_1[2]==5) { /*on button
release*/
146.            pouringAtoB();
147.        }
148. }

149. proc pourBA : pourBA_mouse_1 {
150.        if (pourBA_mouse_1[2]==5) { /*on button
release*/
151.            pouringBtoA();
152.        }
153. }
```

The lines 124-153 define actions to be taken on mouse button presses in the windows. Definitions of the form ‘window’_mouse_1 are created each time a mouse event is received in a sensitive window. It is a list specifying the button that has been pressed or released and the window coordinates at which it has been pressed. The second element of the list has a value of 5 when the button is released and this condition causes the action to be executed. We have now constructed a working model that allows us to play the game of jugs.

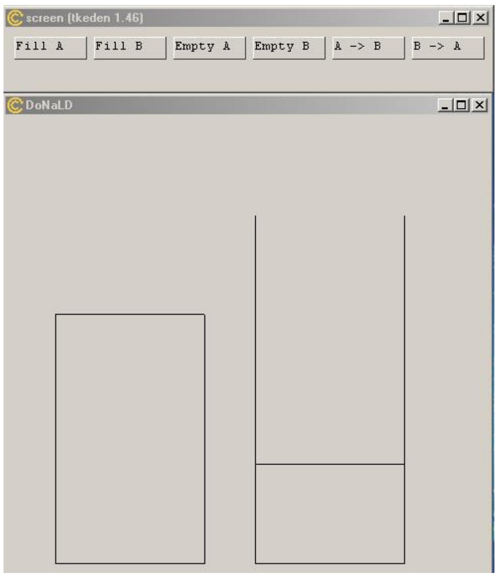


Figure A.1 – The simple jugs model

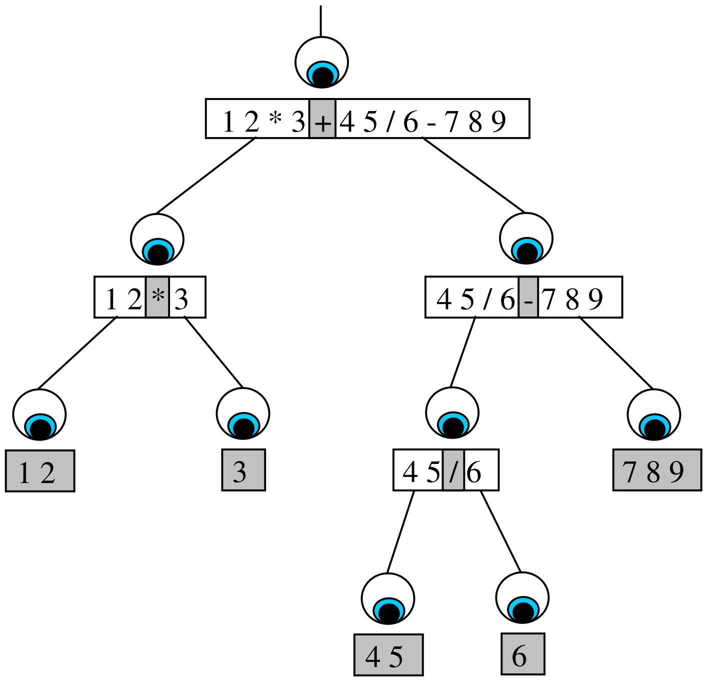
Appendix B – An example of building a parser

This appendix is taken from Antony Harfield's final year project on the agent-oriented parser [Har03]. It includes information about the agent-oriented parser that is essential for understanding the discussion in section 5.4.1.

Getting Started

Parser concepts

In order to learn to use the Agent-oriented Parser in Eden we must put aside our prior knowledge of parsing. Conventional parsers read each input character, one at a time, and not until the entire string is read can any meaning be derived. Instead, it is useful to think about how we, as humans, read languages (natural or otherwise). When we read sentences, we do not remember each character, or even each word, but our brains register the important words. From this we are able to derive meaning.
An agent is an independent entity capable of acting on and interacting with an environment. In the Agent-oriented Parser, the agents have a set of rules (a grammar) with which to parse any input. Each agent will take an input string and a rule, with which it will determine whether the rule can be applied. If the rule is applied, then more agents could be generated to work on substrings. The agent will fail if it cannot apply the rule.
A rule specifies a string that must be observed in the input string. For example, an agent might have the input '1+2' and the most important string it must find could be '+'. When the agent observes a match, it will pass the remaining substrings (if any) to new agents. These are called child agents.



The diagram shows how agents could parse a string. In this example the input is an arithmetic expression and the rules are such that the expression is parsed using standard operator precedence.

Writing notations

A notation is defined as a set of rules, which specify the behaviour of our agents. A rule is simply an Eden list. The basic template for a rule:

```
myrule = [ operation, pattern, [ rule, ... ], [ "fail", rule ]
      ];
```

An `operation` is a string containing the name of the operation an agent should perform. These will be explained in detail later. A `pattern` defines what string the agent is trying to find (or observe) in the input. A `rule` is a string containing the name of a rule (a variable name of an Eden list).

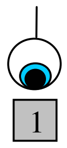
The first item in the list is always an operation. The second item is always the string to be matched. The third item is optional, it is a list of rules to apply to the resulting substrings – the number of substrings depends on the operation to be performed. This is the minimum that a rule can contain. It is likely that most rule definitions will have a fail clause, such that if the agent fails then it will try another rule. The fail clause is a two item list, the first item is always the string "fail" and the second item is a string containing the name of a rule.

Developing a calculator notation

A simple calculator is able to accept digits and arithmetic operators. It will calculate the result of the input expression. Calculators also allow nested expressions using brackets.

Start with a simple model

Recall that our Agent-oriented Parser uses an agent to observe a string in the input. The simplest agents therefore match the input entirely. On a calculator, the user can input just a digit, and the result will be that same digit.



The first operation we will be introduced to is "literal". This operation attempts to match the entire input string (see diagram right). It does not create any child agents. Lets define a rule that matches the digit ‘1’:

```
number1 = [ "literal", "1" ];
```

To invoke the parser, you must supply an input string and a starting rule. The only input our rule should match is ‘1’, so to test this run the parser:

```
dfparse("1", "number1", []);
```

It should accept. The first parameter is the input and the second is the starting rule. Try other inputs to get the parser to reject.

Our language is not very powerful, being only able to accept the digit ‘1’. Now we will introduce the fail clause. We can get our parser to try another rule should the operation fail:

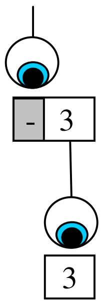
```
number1 = [ "literal", "1", [ "fail", "number2" ] ];

number2 = [ "literal", "2" ];
```

The language will now accept either of the digits ‘1’ or ‘2’. We could extend this method to accept all the digits.

Iteratively developing the model

Now that our calculator can parse digits, we will introduce operators to the input. This is an important concept in Empirical Modelling, the ability to build up a model in an iterative fashion.

Our current model allows us to parse positive numbers, so an obvious extension is to allow negative numbers too. We can define an integer as a number with an optional minus symbol (-) in front of it.



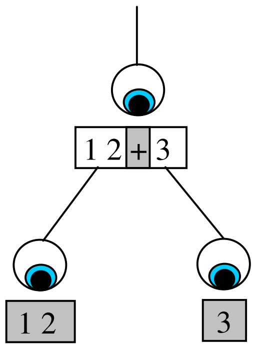
The next agent operation we shall introduce is "prefix". This operation matches a string at the beginning of the input. If a match is made then a child agent is created with its input as the unmatched part of the agents input (see diagram right). This operation can be used to detect a minus symbol at the beginning of our number:

```
    term = [ "prefix", "-", "number1", [ "fail", "number1" ] ];
```

Notice that we use the same rule for the child (third item) and the fail clause. The parser will try to match a minus sign at the start of the input, if it matches then it will match the remainder of the input as a number, else it will match the entire input as a number.

A similar operation is "suffix" which does exactly the same as "prefix", but at the end of a string. An example of using suffix is to remove the semi-colon from the end of a string (i.e. to parse a statement in C/Java/Eden)

The parser should now accept any positive and negative numbers. We shall now look at how we can parse arithmetic operations (e.g. +, -, *, /). As always we will begin with something simple and build the model up. We will first try to parse expressions containing only additions of terms, where our terms are the integers we learnt to parse above.



This is where the most powerful agent operation comes in. The "pivot" operation searches the input (left to right) for a specified string. If it finds a match, then it creates two child agents, one for the left substring and one for the right substring (as shown in the diagram on the right). Our parser would pivot on the addition sign:

```
expr = [ "pivot", "+", "expr", "expr", [ "fail", "term" ] ];
```

An "expr" agent is looking for an addition sign, and if it finds one it creates two children that also search for expressions. If the agent finds no addition sign on the input, then the fail clause specifies that the input must be a term.

In order for our parser to recognise expression containing other operators it is necessary for us to have a rule for each string to be matched:

```
expr = [ "pivot", "+", "expr", "expr", [ "fail", "expr2" ] ];

expr2 = [ "pivot", "-", "expr", "expr", [ "fail", "expr3" ] ];

expr3 = [ "pivot", "*", "expr", "expr", [ "fail", "expr4" ] ];

expr4 = [ "pivot", "/", "expr", "expr", [ "fail", "term" ] ];
```

Notice that we search for operators in their reverse precedence order. This can be explained by taking an example, say the input is '1+2*3'. First we would pivot on the addition sign giving us two substrings '1' and '2*3'. We have broken the calculation down into two sub-calculations which we will add together later. The deepest level will get calculated first, which in this example is '2*3'. Therefore we are observing the rules of precedence correctly.

Regular expressions

Using just the pivot and literal operations gives you all the power you need to develop languages. However, the rules would be quite cumbersome without regular expressions. The Agent-oriented Parser has 3 other operations that deal with basic regular expressions.

The first is "read_all", which is a regular expression version of the "literal" operation. This will attempt to match every character in the input string with a set of characters specified in the rule. For example, the following will match any number using a single rule (compare with our inefficient earlier method):

```
number = [ "read_all", [["0","9"]] ];
```

The second item in the list is a list of tuples. The tuples define the range of characters included in the set to be matched (inclusive). Note that the "read_all" operation accepts the empty string.

The other two regular expression operations are "read_prefix" and "read_suffix", which operate in much the same way but you also specify the number of characters to attempt to match. For example, the following will match one letter of the alphabet at the beginning of the input string:

```
    letter = [ "read_prefix", [[["a","z"],["A","Z"]],1], "nextrule"
          ];
```

Perl-style regular expressions

The above regular expressions lack power, for example, you cannot specify rules for real numbers or identifiers. Our definition of a "number" will accept the empty string – not desirable in most cases!

Three more regular expression operations exist in the latest version of the Agent-oriented Parser. These are "literal_re", "prefix_re" and "suffix_re". The first matches the whole input, where as the other 2 match the beginning and the end of the input respectively. The pattern to be matched is a perl-style regular expression. For example, the following correctly parses a number:

```
    number = [ "literal_re", "[0-9]+" ];
```

More information on perl regular expressions can be found in any good perl book or on the web.

Blocks

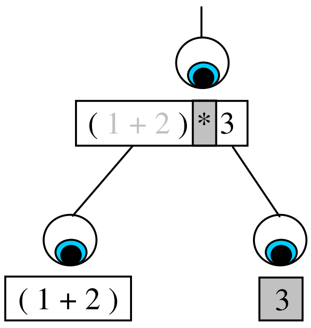
Now we shall look at adding brackets to our notation. This will give our calculator the ability to work out expressions like ‘(1+2)*3’. We have seen how to use the "prefix" and "suffix" operations, so we can use these to parse brackets:

```
    expr5 = [ "prefix", "(", "expr6", [ "fail", "term" ] ];

    expr6 = [ "suffix", ")", "expr" ];
```

This gives us a bit of a problem. Think about the input '(1+2)*3'. The first agent will use the pivot operation on the '+', leaving two substrings '(1' and '2)*3'. Instead, we really want the parser to pivot on the '*' and break the input into the two smaller expressions '(1+2)' and '3'. The parser needs to be sensitive with our brackets.
This is where blocks are a very useful feature of the Agent-oriented Parser. We can define a block and instruct agents to ignore that block. To define a block:

```
bras = [ ["(", ")"], ["bras"] ];

addblocks("bras");
```

The first statement is the block definition. The first item of the list is a pair of strings, the first being the starting string of the block and the second the end string. The second item is a list containing names of blocks that may be contained within the block. The second statement adds the block definition to the environment.



Now for a particular rule we can specify it to ignore blocks. For our calculator notation, we want to ignore any strings between brackets when the agent is looking for an arithmetic operator (+,-,*,/). We add an ignore clause to our rule:

```
expr = [ "pivot", "+", "expr", "expr" ,

         [ "ignore", ["bras"] ],

         [ "fail", "expr2" ] ];
```

This behaviour is demonstrated in the diagram (right). The string within the brackets is 'greyed-out' because it is ignored. Hence the most important string to be observed is the multiplication sign (*) and a pivot is made.
It is important to remember that when an agent 'ignores' a block it is not removing that block from the input. It is perhaps better described as preserving the block. The agent simply preserves the contents of the block and leaves it for another agent to parse.

Scripting

A parser that either accepts or rejects an input is of little use unless it produces some output. Our simple calculator needs some way of outputting the result of an expression. This is achieved with agent actions. If an agent does not fail to match the input then it can optionally perform some actions. Each action can be performed before or after the actions of the agent's children.

The format for including agent actions in a rule definition is similar to the other optional components of a rule. Here we modify our "term" rule by adding an action that prints out some random comment:

```
term =
  [ "literal_re", "[0-9]+",
    [ "action",
      [ "now", "writeln(\"somerandomcomment\");" ] ] ];
```

The third item in the list above is the action declaration. This sublist begins with the "action" tag to recognise it from the other optional tags. The items following the head tag are the commands to execute. Each command is a list containing only 2 items, the first being either "now" or "later" depending on whether the command will be executed before or after the child agents. The second part of the command is the command string which is typically some eden code to be executed.

An agent has some data associated with it that can used in its actions. Each agent also has a unique variable associated with it. This agent data can be substituted into the command string using the following:

```
$i = the input string to the agent
$j = the name of a variable containing the input string
$t = the token/string that was matched by the agent
$v = the variable name that belongs to the agent
$s1 = the first substring of the input
$s2 = the second substring of the input (and so on for 3rd,
      4th, ..)
```

```
$p1 = the variable name of the first child agent (parameter 1)

$p2 = the variable name of the second child agent
```

Now we can make our "term" rule more useful by adding an action that stores the term value in the agent variable:

```
term =

  [ "literal_re", "[0-9]+",

    [ "action",

      [ "now", "$v = $t;" ] ] ];
```

We would then add actions to our other rules. The "expr" rule can do the addition of the two sub-expressions:

```
expr =

  [ "pivot", "+", [ "expr", " expr" ],

    [ "action",

      [ "later", "$v = $p1 + $p2;" ] ],

    [ "fail", "term" ] ];
```

Take a look at the final calculator notation at the end of the document for more examples of agent actions.

Note: The dollar sign is a special character in the command string. If you want to print a single dollar sign ($) in your command string then you must follow it by another dollar sign ("$$" will produce a single dollar in the command string).

The original version of the Agent-oriented Parser had a different method for writing scripts, using the "script" tag. Although the parser will still accept these scripts, it is recommended you use the "action" notation. For more information on the "script" tag, refer to Chris Brown's third year project [Bro01].

Installing notations

Now that we are happy with our calculator notation, it is probably a good idea to make it more accessible. We can install new notations into the Eden environment, which can then be used in scripts as you would existing notations (e.g. %eden, %donald, %scout). In tkeden this will add a radio button for our new notation to the environment.

First we must create an initialisation rule:

```
calc_init = [ "\n", "calc", [] ];
```

The first item in the list is the string to split the input on. For our calculator notation we would like to separate each command by the end-of-line character (\n). For other notations you may wish to split the input on other characters (e.g. semi-colon for C/Java/Eden). The second item is the starting rule. The third item is a list of blocks to ignore in the splitting procedure.

To install the notation in the environment:

```
installAOP("%mynotation", "calc_init");
```

Notations must begin with a percent (%) character.

You can now switch to the new notation by typing `%mynotation`. Do not forget to switch back to `%eden` for Eden code!

The final calculator notation

```
calc_init =

  [ "\n", "calc", [] ];


calc =

  [ "prefix", "", "calc_expr",

    [ "action",

      [ "later", "writeln('=',$p1);" ] ],

    [ "fail", "calc_err" ] ];


calc_expr =

  [ "pivot", "+", [ "calc_expr", "calc_expr" ],
    [ "ignore", ["bras"] ],

    [ "action",

      [ "now", "$v is $p1 + $p2;" ] ],

    [ "fail", "calc_expr2" ] ];


calc_expr2 =

  [ "pivot", "-", [ "calc_expr", "calc_expr" ],

    [ "ignore", ["bras"] ],

    [ "action",

      [ "now", "$v is $p1 - $p2;" ] ],

    [ "fail", "calc_expr3" ] ];


calc_expr3 =

  [ "pivot", "*", [ "calc_expr", "calc_expr" ],

    [ "ignore", ["bras"] ],

    [ "action",

      [ "now", "$v is $p1 * $p2;" ] ],

    [ "fail", "calc_expr4" ] ];


calc_expr4 =

  [ "pivot", "/", [ "calc_expr", "calc_expr" ],

    [ "ignore", ["bras"] ],

    [ "action",

      [ "now", "$v is $p1 / $p2;" ] ],

    [ "fail", "calc_expr5" ] ];
```

```
calc_expr5 =
  [ "prefix", "(", "calc_expr6",
    [ "action",
      [ "now", "$v is $p1;" ] ],
    [ "fail", "calc_term" ] ];


calc_expr6 =
  [ "suffix", ")", "calc_expr",
    [ "action",
      [ "now", "$v is $p1;" ] ],
    [ "fail", "calc_err" ] ];


calc_term =
  [ "literal_re", "[0-9]+",
    [ "action",
      [ "now", "$v = $t;" ] ],
    [ "fail", "calc_err" ] ];


calc_err =
  [ "read_all", [],
    [ "action",
      [ "now", "writeln(\"calc: syntax error\");" ] ] ];


installAOP("%calc", "calc_init");
```

Extensions

If you want to experiment with this notation, then here are a few ideas of how to extend it:

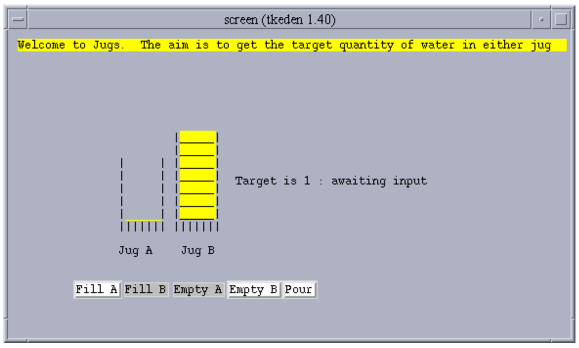
- Add some common constants like ‘pi’ and ‘e’.
- Introduce power and square root functions.
- Add memory capabilities like you would normally find on a calculator (e.g. M+, MR, etc).

Appendix C – Glossary of models used in the thesis

This appendix contains brief descriptions of the EM models used as case studies in this thesis. For each model, we give a reference to its location in the EM model repository [EMRep], a short description of the model and a screenshot of it in use. For further details on individual models, consult the documentation files provided with each model in the repository.

The jugs model – [EMRep, jugsBeynon1988] and [EMRep, jugsPavelin2002]

This is a model of a simple educational game first developed for the BBC computer in the 1980's by the Chiltern Advisory Unit. The game revolves around trying to measure a specified amount of water where there are two jugs of known quantities that have no markings on them. A set of basic operations is available on the interface to empty a jug, fill a jug, or pour water between the jugs.

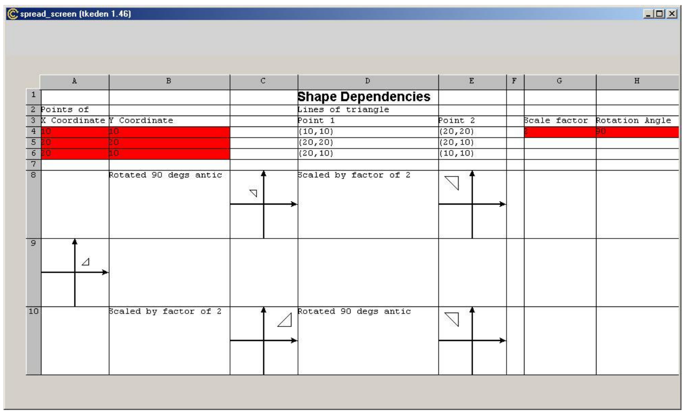


Welcome to Jugs. The aim is to get the target quantity of water in either jug

Target is 1 : awaiting input

The spreadsheet model – [EMRep, spreadsheetRoe2002]

This spreadsheet created using TkEden illustrates connections between spreadsheets and modelling. The model can replicate the essential functionality of conventional spreadsheets and can show how the

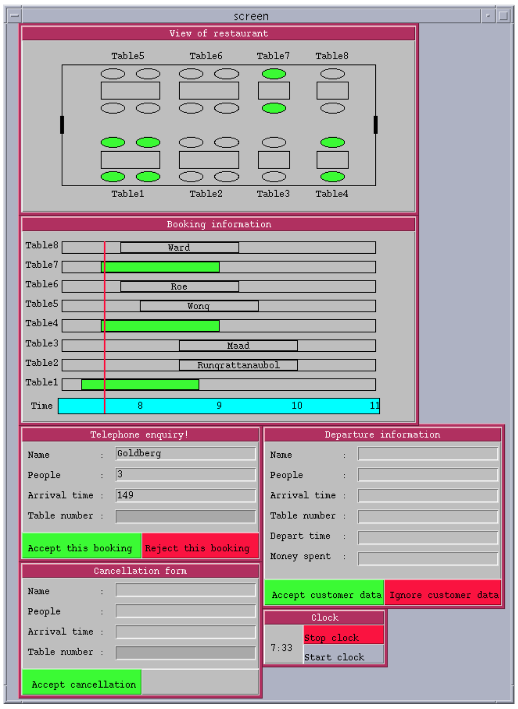


	A	B	C	D	E	F	G	H
1				Shape Dependencies				
2	Points of			Lines of triangle				
3	X Coordinate	Y Coordinate		Point 1	Point 2		Scale factor	Rotation Angle
4	10	10		(10,10)	(20,20)		2	90
5	20	20		(20,20)	(20,10)			
6	20	10		(20,10)	(10,10)			
7								
8		Rotated 90 degs antic		Scaled by factor of 2				
9								
10		Scaled by factor of 2		Rotated 90 degs antic				

generalised notion of dependency in EM allows the spreadsheet model to support a wider variety of types. It also illustrates agent actions in a spreadsheet.

The restaurant model – [EMRep, restaurantRoe2000]

This case study illustrates how a model can be constructed to investigate restaurant management. The user of the model can play through fictional scenarios of bookings in the restaurant to gain experience of how to best allocate tables in order to maximise profit. The model contains a seating plan for a fictional restaurant, a timetable booking sheet and a set of forms to enter data in. Scenarios can either be created manually or a random sequence of events can be generated to simulate the activity on an evening.



Telephone enquiry!

Name :	Goldberg
People :	3
Arrival time :	149
Table number :	

Departure information

Name :	
People :	
Arrival time :	
Table number :	
Depart time :	
Money spent :	

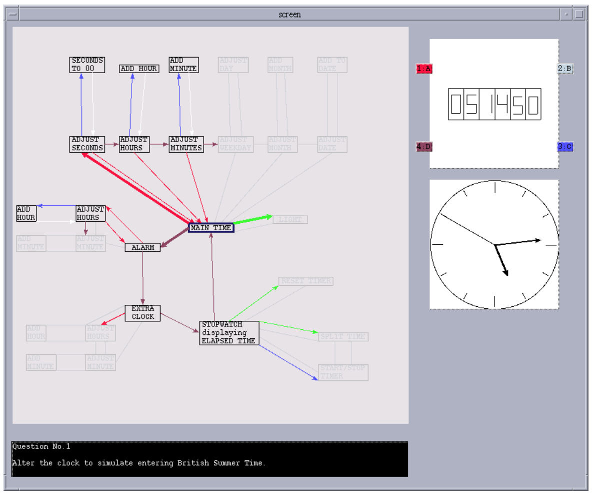
Cancellation form

Name :	
People :	
Arrival time :	
Table number :	

Clock: 7:33

The digital watch – [EMRep, digitalwatchRoe2001]

The digital watch model has been developed by a number of different people over a period of eight years. It consists of a digital watch display with a number of buttons to activate its functionality, an analogue clock display and a graphical depiction of all the states that the watch can be in. States emerge when they have been visited for the first time.

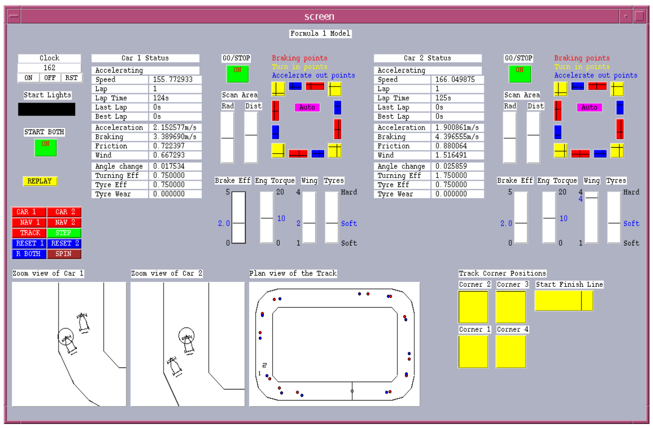


Question No.1

Alter the clock to simulate entering British Summer Time.

The racing cars model – [EMRep, racingGardner1999]

This model can be used to explore the setup of racing cars in order to minimise lap times around a track. It is layered so that learners are exposed to more functionality at each successive layer. The final layer of the model is shown in the screenshot, and contains two fully customisable cars that race against each other on a partially customisable track.



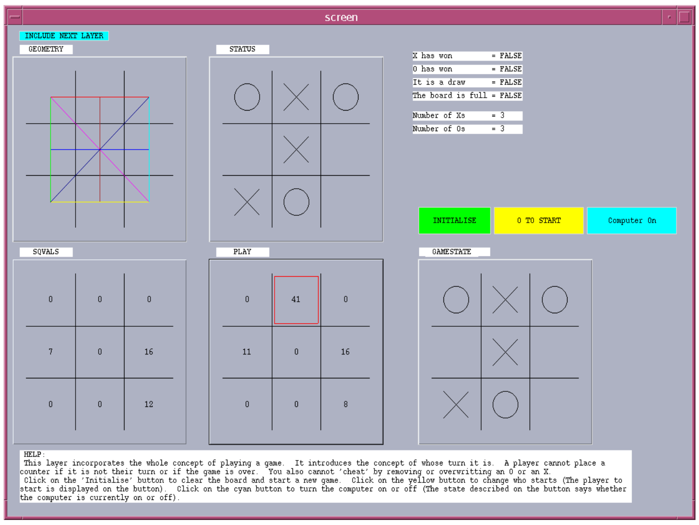
Clock: 162

Car 1 Status	
Accelerating	
Speed	155.772933
Lap	1
Lap Time	124s
Last Lap	0s
Best Lap	0s
Acceleration	2.152577m/s
Braking	3.389690m/s
Friction	0.722397
Wind	0.667293
Angle change	0.017534
Turning Eff	0.750000
Tyre Eff	0.750000
Tyre Wear	0.000000

Car 2 Status	
Accelerating	
Speed	166.049875
Lap	1
Lap Time	125s
Last Lap	0s
Best Lap	0s
Acceleration	1.900861m/s
Braking	4.396555m/s
Friction	0.880064
Wind	1.516491
Angle change	0.025859
Turning Eff	1.750000
Tyre Eff	0.750000
Tyre Wear	0.000000

The OXO model – [EMRep, oxoJoy1994], [EMRep, oxoGardner1999] and [EMRep, 3doxoRoe2001]

This is a layered model that introduces different concepts of noughts-and-crosses at each layer. The layers introduce the board and its geometry, the pieces to be placed on the board, the rules by which the pieces can be placed on the board, and strategy considerations for a computer player. At each layer there is no prescription about future layers that are to be added.



X has won = FALSE
0 has won = FALSE
It is a draw = FALSE
The board is full = FALSE

Number of Xs = 3
Number of 0s = 3

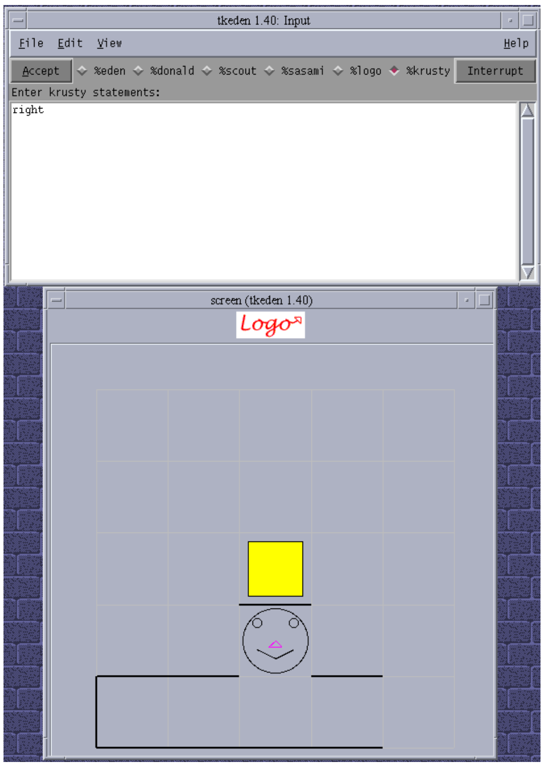
0	0	0
7	0	16
0	0	12

0	41	0
11	0	16
0	0	8

HELP:
This layer incorporates the whole concept of playing a game. It introduces the concept of whose turn it is. A player cannot place a counter if it is not their turn or if the game is over. You also cannot 'cheat' by removing or overwritting an 0 or an X.
Click on the 'Initialise' button to clear the board and start a new game. Click on the yellow button to change who starts (The player to start is displayed on the button). Click on the cyan button to turn the computer on or off (The state described on the button says whether the computer is currently on or off).

The clown-and-maze model – [EMRep, krustyRoe2002]

The clown-and-maze model illustrates how languages for interaction can be interactively extended from very simple languages. In this model, the initial language contains just four simple directional commands and gradually the functionality of a Logo style language is added. The language for interaction can be interactively altered whilst the model is running. The aim of the underlying model is to direct a clown around a maze to find treasure.

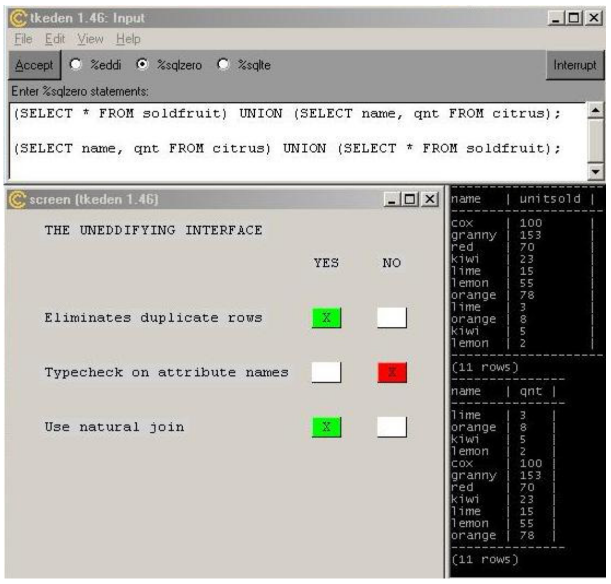


Enter krusty statements:

```
right
```

The SQL-EDDI model – [EMRep, sqleddiWard2003]

The SQL-EDDI environment has been used on the 2nd year Introduction to Database Systems module at the University of Warwick to explore the relationship between relational algebra and relational query languages. It comprises an interpreter for a subset of SQL and a relational algebra language ("EDDI"). 'The Uneddifying Interface' is used to control the way in which queries are evaluated and illustrate flaws in the design of SQL.



Enter %sqlzero statements:

```
(SELECT * FROM soldfruit) UNION (SELECT name, qnt FROM citrus);

(SELECT name, qnt FROM citrus) UNION (SELECT * FROM soldfruit);
```

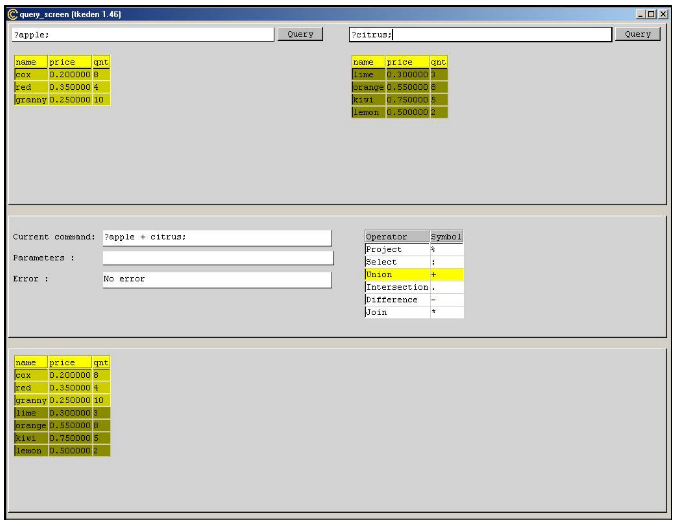
THE UNEDDIFYING INTERFACE

	YES	NO
Eliminates duplicate rows	🟩 X	
Typecheck on attribute names		🟥 X
Use natural join	🟩 X	

```
------------------------
name    | unitsold  |
------------------------
cox     | 100       |
granny  | 153       |
red     | 70        |
kiwi    | 23        |
lime    | 15        |
lemon   | 55        |
orange  | 78        |
lime    | 3         |
orange  | 8         |
kiwi    | 5         |
lemon   | 2         |
------------------------
(11 rows)
---------------
name    | qnt |
---------------
lime    | 3   |
orange  | 8   |
kiwi    | 5   |
lemon   | 2   |
cox     | 100 |
granny  | 153 |
red     | 70  |
kiwi    | 23  |
lime    | 15  |
lemon   | 55  |
orange  | 78  |
---------------
(11 rows)
```

The RAT – [EMRep, ratRoe2003]

The Relational Algebra Tutor was developed to show how the basic operations of relational algebra produce output tables from one or two input tables. The operations are colour coded so that the learner can see directly how an output table is created. The syntax of the resultant relational algebra query in the EDDI language is displayed.



```
?apple;
```

name	price	qnt
cox	0.200000	8
red	0.350000	4
granny	0.250000	10

```
?citrus;
```

name	price	qnt
lime	0.300000	3
orange	0.550000	8
kiwi	0.750000	5
lemon	0.500000	2

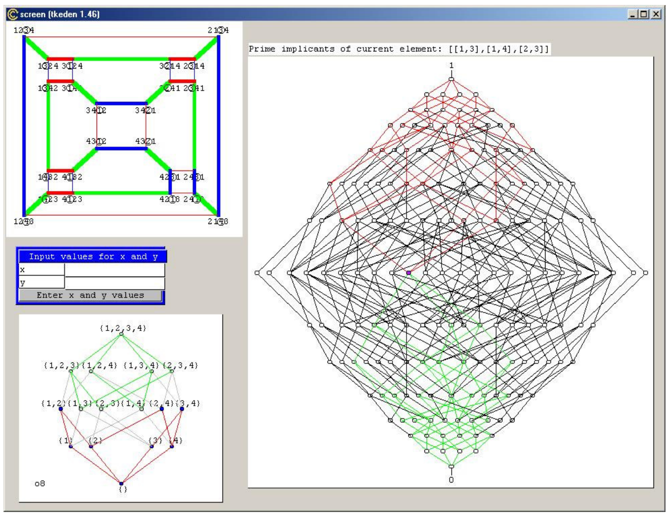
```
Current command:  ?apple + citrus;
Parameters :
Error :           No error
```

Operator	Symbol
Project	%
Select	:
Union	+
Intersection	.
Difference	-
Join	*

name	price	qnt
cox	0.200000	8
red	0.350000	4
granny	0.250000	10
lime	0.300000	3
orange	0.550000	8
kiwi	0.750000	5
lemon	0.500000	2

The MBF4 model – [EMRep, mbf4Beynon2003]

The MBF4 model illustrates how several independently developed models can be integrated into a single model using dependency. The model was developed to explore connections between different realisations of an abstract area of mathematics surrounding lattice theory.



```
Prime implicants of current element: [[1,3],[1,4],[2,3]]
```

The heapsort model – [EMRep, heapsortBeynon1998]

The heapsort model is an example of a partially built model to which a learner can add small fragments of script to embellish the basic model. The initial model consists of a heap structure that a learner can manipulate to gain understanding of the essential nature of a heap. Agent actions can be added to the model to introduce the basic transitions of heapsort that are eventually directed at automation of the heapsort algorithm. By building up the model gradually, a learner can introduce new definitions as their competency increases.

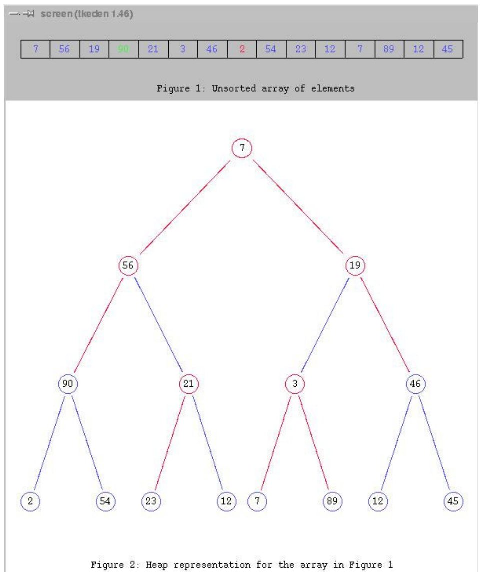
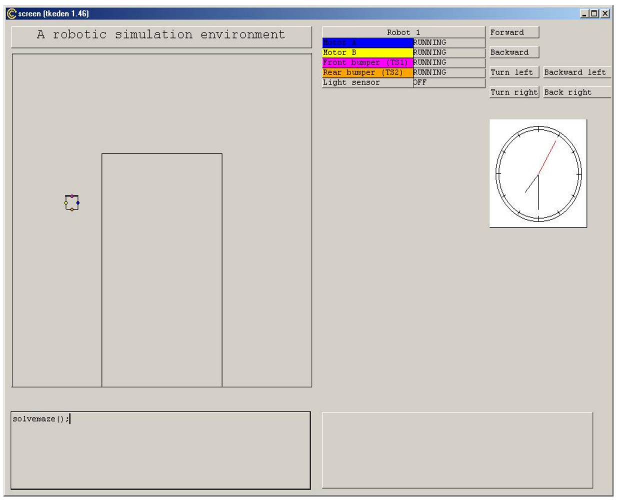


Figure 1: Unsorted array of elements

Figure 2: Heap representation for the array in Figure 1

The RSE – [EMRep, rseRoe2003]

The robot simulation environment is a prototype for understanding programming of LEGO Mindstorms robots. The environment is an exploratory laboratory where learners can directly manipulate robots in order to understand how the movement of the motors and the state of the sensors is related to the real-world behaviour of the robot.



Robot 1	
Motor A	RUNNING
Motor B	RUNNING
Front bumper (TS1)	RUNNING
Rear bumper (TS2)	RUNNING
Light sensor	OFF

```
solvemaze();
```

Appendix D – Example model building interactions in EM

The following extract is taken from the paper 'Liberating the Computer Arts' by Meurig Beynon [Bey01]. This extract describes model building in EM with reference to the construction of a simple analogue clock.

4.2. Principles and Tools of EM

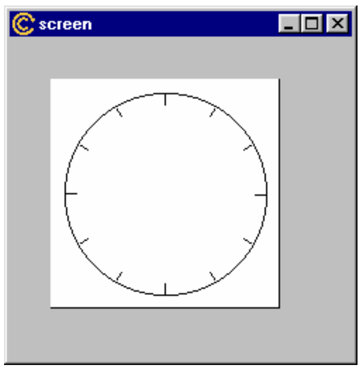
The principles and tools of EM will be briefly described and illustrated with reference to a simple but extended example. This focuses on model-building activity surrounding an imaginary analogue clock.

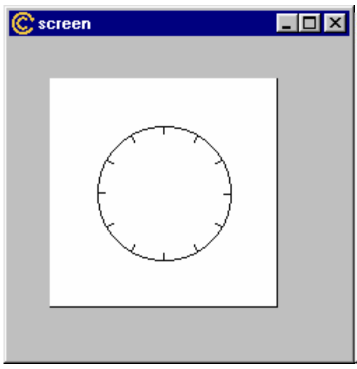
The primary focus for representation in EM, as in art – and in contrast to computer programs that are targeted at behaviour, is on situation. In the initial situation in which we observe the modelling activity, the model consists of the outline of the clock. This is defined by a family of definitions of variables (a definitive script) in a special-purpose notation (a definitive notation) for line-drawing – DoNaLD.

```
%donald

viewport CLOCK


openshape clock


within clock {

    real sixthpi

    line eleven, ten, nine, eight, seven, six, five, …, one

    line noon
    point centre

    real radius

    circle edge


    sixthpi = 0.523599

    radius = 150.0

    eleven = rot(noon, centre, -11 * sixthpi)

    ...

}
```

The variables in this script represent observables in the clock: the rim of the face, represented by the circle `clock/edge`, its centre `clock/centre` and the divisions `eleven, ten, nine,` ... etc that indicate the hours.

The artefact that is defined in this fashion is depicted on a screen, as in Figure 1.1:





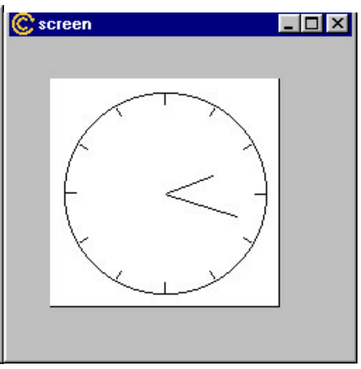


Figure 1.1: The clock face   Figure 1.2: The scaled face   Figure 1.3: With time set

The definitions in the script establish dependencies between the variables similar to those in a spreadsheet. Interaction with the script takes place in an environment in which the values of variables are always open to redefinition. For instance, the redefinition

```
clock/radius = 100.0
```

has the effect of making the clock smaller, simultaneously changing all the positions of the divisions and the rim of the clock. The display also depends directly upon the values of variables in the script, and is simultaneously updated (see Figure 1.2).

The hands of the clock are not yet displayed, but there are already variables and definitions in the script that refer to them:

```
within clock {
  line minHand, hourHand
  real minAngle, hourAngle
  real size_minHand, size_hourHand
  int t
  size_minHand, size_hourHand = 0.75, 0.5
  minAngle = (pi div 2.0) - float (t mod 60) * (pi div 30.0)
  hourAngle = (pi div 2.0) - float (t mod 720) * (pi div 360.0)
  minHand = [centre + {size_minHand*radius @ minAngle}, centre]
  hourHand = [centre + {size_hourHand*radius @ hourAngle}, centre]
  centre = {200, 200}
  ....
}
```

They are not currently displayed because the value of the integer `t`, which represents the current time, has yet to be determined. Assigning different values to `t` readily establishes that `t` can be interpreted as the time elapsed in minutes from midnight, so that (for instance) assigning `t` to 138 sets the clock to time 2.18 (see Figure 1.3.).

The modeller's interaction with the clock script is not directed towards any particular goal or constrained by a preconceived interpretation. The modeller views interaction with the artefact from the perspectives of many different human agents, shifting perspective arbitrarily, much in the way that an artist saturates their imagination through attention to the emerging work of art. For instance, in developing a play a playwright might reflect upon a particular situation from the viewpoint of the fictitious characters in the play, the actors, the audience or the producer. The simultaneous consideration of all these viewpoints is not associated with a separation of concerns, but with a dwelling in the situation so as to draw out all its possibilities and enrich the experience of the author in the writing and the audience in the appreciation of the work. This holistic approach is vital to the activity, and is complemented by an openness and responsiveness to what is encountered that characterises creative thought.

The act of setting the time on the clock supplies a modest illustration. It might be that the clock is to be sold, and the hands placed in the most aesthetically pleasing configuration. It might be that the user is setting the clock to the current time. The time on the clock potentially represents an aspect of the clock that is beyond user control. The script is intended to reflect all these possible interpretations, and support them in so far as they can co-exist in the modeller's imagination. The novelty in this approach lies in the ontology of the model: there is no prescribed interpretation, only certain interpretations that may acquire particular significance and permanence as the modeller's imagination shapes the model and the referent.

The description of EM activity is framed in terms of 'how the modeller construes the situation'. Observables, agency and dependency are the key concepts used to express construals. Where there is a fixed external physical referent, construal is concerned with how the observations of the referent can be explained with reference to agency, dependency and observables. It is also appropriate to regard the relationship in which the modeller chooses to stand to the referent as a form of construal, where the emphasis is placed upon the agency of the observer rather than agency that operates within the referent. Interaction with the script can be associated with elaborating a construal of either kind. The parametrisation of the clock face in terms of `noon` and `radius` is concerned with construal from the modeller's perspective. As a simple illustration of how construal applies to the clock mechanism itself, the following revised definition for `hourAngle` establishes the dependency between minute and hour hand that is typically present in a mechanical clock:

```
within clock {

      minAngle = (pi div 2.0) - float (t mod 720) * (pi div 30.0)

      hourAngle = (pi div 2.0)  - ((pi div 2.0) - minAngle) div 12.0

      ...

}
```

Through this redefinition, `hourAngle` depends upon `minAngle`, and is no longer directly defined in terms of the time t. Note also that the definition of `minAngle` has also been modified (so that it records the reflects the number of minutes elapsed over a 12 hour period rather than a single hour) so that it delivers the correct result for `hourAngle`. This can be interpreted as reflecting whether or not we are taking the state of the internal mechanism of the clock into account in observing the position of the minute hand. The relationship between these two scripts illustrates how the removal of dependency can be associated with optimisation and information loss.

The practical techniques that the modeller can use to support construal include the use of definitions to express dependency and the introduction of triggered procedures to represent agent actions. The underlying framework is supplied by the EDEN interpreter, which serves both as an "evaluator of definitive notations" (in particular for DONALD) and as a hybrid definitive/procedural environment that interfaces with the user and the computer. Through EDEN it is possible to increment the variable `clock/t` repeatedly for instance, so as to simulate the clock in operation. The speed at which it is appropriate to carry out this update is at the discretion of the modeller: it may be useful to observe the hands in slow motion, as in the construction of the rotating clock described below, to run it as fast as possible to obtain an overall impression of the clock behaviour, or to synchronise the update of variable `clock/t` with the system clock.

The above discussion illustrates many of the distinctive features of EM in a simple setting. The most significant issue is that the context is such that the term modelling is not entirely apposite (so obscure is the referent and the goal of the modeller): there are many roles for agency, the conventions for observation and interpretation are fluid, and modelling activity is as much concerned with exploring what the referent is as with representing it.

4.3. The SIN principle in EM

The messiness of our real engagement with the world is at odds with the systematic models of behaviour to which a science aspires. In building artefacts that can support this engagement, the idea of making things easy for the user is suspect. It is appropriate to eliminate unnecessary frustration, but not to suppose that all frustration can be eliminated (cf. Donofrio's [IBM Vice-President] "what we want, when we want it, where we want it"), or even that that would be a desirable goal. Art works often both with and against the tools and the medium, and this is not something that can be designed away. One aspect of the artist's skill is to overcome the limitations of the instrument: "a bad workman blames his tools".

The evidence is that in certain respects the tools of EM are not easy to use. Even after making allowance for some obvious flaws in the tools – and acknowledging that a bad designer blames the workman – there is an essential reason for difficulty in use. It is quite usual in conventional programming to achieve an end without explicitly considering what assumptions have been made in order to achieve it. In EM, there is no choice but to engage with the experience that should inform our constructions. This activity is expensive in terms of human time and effort – but there is no substitute for it. The potential advantage of the EM approach is that, when we subsequently identify problems, the model itself can help us to access the knowledge that informed our original solution, and that is required to improve this solution.

An actual illustration drawn from developing a rotating clock model is useful at this point. I wish to highlight the noon position on the clock by marking it with a longer line segment. I choose to do this because of the limitations of the tools: the attributes of a line drawing are not preserved if I create a new image of it by rotation. I first think of refining the line `noon`, but realise that this will affect all the other divisions by dependency. This means that I shall introduce a new line `noon2` to mark the noon division. My idea is to derive `noon2` by elongating the line `noon`. This I can do by addressing its endpoints `noon.1` and `noon.2`. My first attempt is:

```
within clock {
        line noon2
        noon2 = [noon.1, noon2*2]
        ...
}
```

This is identified as a cyclic definition, since `noon2` appears on the right hand side of the definition of `noon2` – a mistake precipitated by my own choice of notation. I correct the redefinition of `noon2` thus:

```
noon2 = [noon.1, noon.2*2]
```

This is a conceptual mistake – it results in a spike at `noon`, not a longer line – I am forgetting that the elongation has to be along the direction of the line `noon`. I next try:

```
noon2 = [noon.2 + (noon.1 - noon.2)*2];
```

This a type checking mistake – I am getting confused about what sub-expressions are lines and what are points. To correct this, I enter:

```
noon2 = [noon.2 + (noon.1 - noon.2)*2, noon.2]
```

This is a mistake because the endpoints `noon.1` and `noon.1` are not the way round that I thought they were. Only then do I get what I wanted:

```
noon2 = [noon.1, noon.1+ (noon.2 - noon.1)*2]
```

The interpreter itself poses its own syntactic challenges to correct input, and the steps detailed above were further complicated by such vagaries. Pencil and paper also played a role in supporting his interaction. On several counts, EM offers poor quality end-user interaction, but it is unusual in that it supports a degree of engagement even in error and misconception.

The experimental form of my interaction points to a significant danger: that EM encourages sloppiness in thought and practice. That said, I know that the kind of activity exposed above is quite characteristic of my inner thought processes, and (modulo the vagaries of the tools) that it is harder to trace this on paper or in my head. The end result is also much more satisfactory – not only do I derive the correct answer, but I construct an environment in which my mistakes and misconceptions have been captured and recorded to an extent that is otherwise problematic.

As highlighted in this example, 'making mistakes' is an essential part of employing EM. For the traditional programmer, this is a difficult concept: the most significant mistakes that the skilled programmer makes are in the early stages of design, and hopefully never reach the implementation. Though it can be frustrating and embarrassing to follow through the experimental phases of design in EM interactively – and whilst it is tempting to focus on flaws in the notations, the interface or the interpreter – there is some virtue in exposing our imperfect thought processes.

4.4. Illustrating the SIN principles of representation in EM

This section develops the theme of the simple clock model to illustrate how representation operates in EM. The significance of using EM in representation is best understood by considering the continuity in the cognitive activity that accompanies the modelling. When a conventional program is executing, there are at least two aspects of its state that are relevant. There is the computational state, with which the user may or may not be interacting, which – in so far as it is intended to be interpreted by the user – is meaningful in terms of the application of the program. There is also –

at a meta-level – the state of the program code itself, which is not typically known to the user.

When making a redefinition in a script, the modeller can be changing the associated state in what conventionally would be regarded as affecting both of these aspects. On the one hand, the state that is visible to the user may be changed. On the other, the underlying pattern of dependency maintenance ("part of the program code") may also be changed. The aspiration for EM, to some considerable degree supported even by our current tools, is for it to be possible to change the program code without disrupting that part of the state with which the user is engaging. An analogy can be made between a conventional programming paradigm and the way in which a traveller might use a vehicle built by a mechanic at a workshop, travel about in it to discover its limitations, then return to the workshop so that it can be modified to overcome these. The aspiration for EM is that the mechanic and his workshop can journey with the traveller, to effect modifications in the context where they are needed, potentially with more first-hand appreciation of the requirement.

A few examples will serve to illustrate how EM helps to address issues concerned with situation, ignorance and nonsense.

- Representing situation

The representation of situation in EM can be illustrated in many ways. For instance, to set the clock to Japanese time:

```
%eden

uk_time is tnsecs / 60;

/* tnsecs = the number of seconds that have elapsed since a fix date
*/

japan_time is uk_time + 480;

_clock_t is japan_time;
```

To represent a broken clock, in which the minute hand hangs loose:

```
%donald

clock/minAngle = - pi div 2
```

It is also possible to take account of observables present in the situation, but previously unrecorded. For example, to add a secondhand that is coloured red (see Figure 3):

```
%donald

within clock {

      line secHand

      real secAngle, size_secHand

      secHand = [centre + {size_secHand*radius @ secAngle}, centre]

      size_secHand = 0.8

}

%eden

sec_mod_60 is tnsecs % 60;

A_clock_secHand = "color=red";
```

- Representing ignorance

The primary respect in which EM deals with missing knowledge is through supporting variables whose value is as yet undefined. The graceful handling of the unspecified time on the clock discussed above is a simple illustration.

Another kind of ignorance is that associated with exploratory design, where something is known only after it is constructed and recognised in interaction. As a simple illustration, the modeller can act in the role of a clock designer via the redefinition:

```
within clock {

      circle inner_edge

      real width_edge

      inner_edge = circle(centre, radius - width_edge)

      width_edge = 20.0

      ...

}
```

Such a redefinition is here to be construed as changing the clock itself.

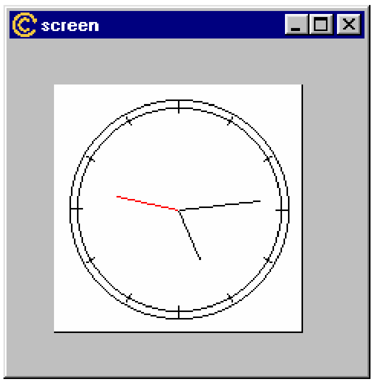


Figure 2: Hand and Rim added

Further experimentation with values of `width_edge` elicits tacit knowledge about what is an acceptable design through interaction.

Conventional programming is knowledge-driven, in that a program is designed with a specific functionality in mind. Being able to make use of a model even though we are ignorant about how we might wish to use it is possible to the extent that we have an effective construal. The rotating clock, in which the clock pivots freely about its centre according to the moments associated with its hands, was conceived as an opportunistic extension of the clock model. This construction illustrates the way in which automatic agency, such as here represents the clock rotating into equilibrium as time passes, is developed from – or as if from – a pattern of interaction by the modeller. For instance, the modeller might compute the moments of the clock with minute hand and hour hand pointing vertically downwards, then when the rotation of the clock lies halfway between these positions, and proceed by binary search to locate the rotation that makes the moment of the clock zero. This can be represented by using the pattern of observation:

```
%eden

hA = _hrAngle;

mA = _mnAngle;

momentH is moment(_hrAngle, _mnAngle, hA);

momentHM is moment(_hrAngle, _mnAngle, (hA+mA)/2.0);

momentM is moment(_hrAngle, _mnAngle, mA);
```

– where `_hrAngle` and `_mnAngle` refer to the current positions of the hour and minute hands in the normal sense, and the function `moment()` returns the moment of the clock about the angle specified by its third argument – and assigning `hA` or `mA` to `(hA+mA)/2` according to the sign of `momentHM`. This binary search can then be performed automatically by an agent that responds to the changing time. This illustrates how EM can be used as a way of specifying the functions that maintain dependencies themselves.

The use of this model to explore the dynamics of the rotating clock is another more familiar sense in which EM is concerned with the discovery of knowledge rather than its exploitation.

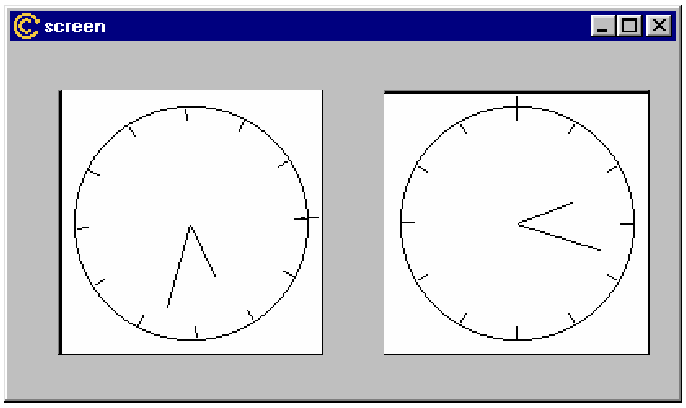


Figure 3: The rotating clock

- Representing nonsense

A much-neglected concern in our representation of the world is the way in which our minds impose relationships upon possible situations and events. We are accustomed all the while to organise our experience according to the degree to which it is familiar and 'makes sense'. This leads us to say "I don't think that's possible", or "if that can happen, then that is also plausible". Though nonsense suggests the antithesis of sense, it in fact refers to what is – in this mind-space – near enough to sense to blend with it in some respects. Perhaps the most important feature of the EM representation of a situation by a script is that it establishes such relationships: namely, nearness as assessed by the kind of redefinition that is required to transform one script to another, and by the kind of agency that would be involved. The extent to which these relationships match those that we encounter in the world reflects the quality of our construal, as determined by the observables, dependency and agency we identify. The clock study is a useful source of examples.

It is easy to modify the clock so that the length of the second hand depends on the time:

```
_clock_size_secHand is (float(sec_mod_60)/60.0) *
_clock_size_minHand;
```

In the days of the mechanical clock, this would have been implausible if not nonsensical, but it would seem unremarkable on a computer desktop. There are simple redefinitions to create a mirror image clock:

```
%scout

window clockwin = {

    type: DONALD

    box: [clockwinNW, clockwinSE]

    pict: "CLOCK"

    xmin: 30

    ymin: 370

    xmax: 370

    ymax: 30

    bgcolor: "white"

    border: 1

};
```

or an the upside-down clock:

```
%donald

clock/radius = -100.0
```

These examples highlight the role that agency of the observer plays in discriminating sense from nonsense: the conventions by which we read the time are in principle so arbitrary. The rotating clock with nothing to distinguish noon from other divisions would be more absurd as a timepiece.

Another extension of the basic clock model serves to illustrate how visual art exploits both the sense of space and 'the space of sense'.

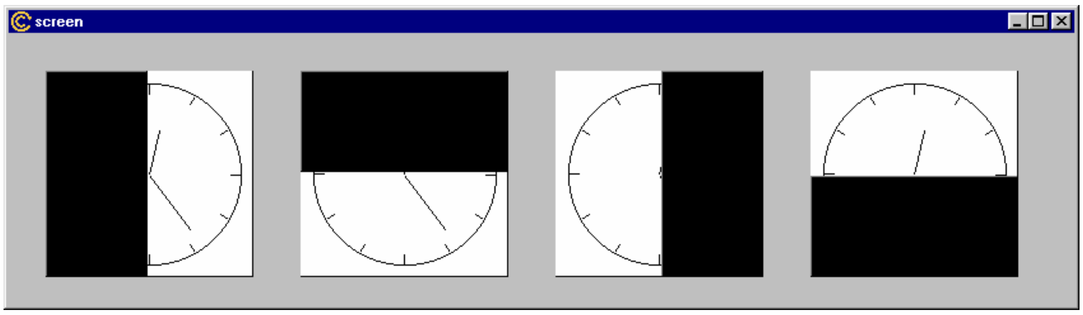


Figure 4: A model loosely based upon Richard Wentworth's The Warwick Dials (1999)

This model is loosely based on a construction on display in the Warwick Arts Centre. It is made by a simple extension of the original model in which the line drawing of the clock is displayed in several windows, each partially occluded by a blank window whose size can be altered by redefining the variable `blankedge`.

```
%scout

integer blankedge = 83;


window blank1 = {

    type: DONALD

    box: [clockwinNW, clockwinSE - {blankedge*clockwinScale, 0}]

    bgcolor: "black"

    border: 1

};

...


display     screen     =     <blank1/clockwin/blank2/clockwin2/blank3/
clockwin3/blank4/clockwin4>;
```

In the space of sense, there is a distinction between nonsense and meaninglessness. In EM, this can be explored by transforming artefacts through random redefinition of variables. The arbiter in matters of sense is the observer, who may or may not be able to connect in any way with the experience offered by the transformation of an artefact. The redefinition of variables that correspond to observables beyond the control of any recognised agent, and redefinitions that subvert our physical intuitions about the permanence and reliability of objects are some of the most effective in destroying the semantic relation, as in the corrupted clock in Figure 5:

```
within clock {

      sixthpi = 1.0

      minHand = [centre + {0.75*radius @ minAngle}, hourHand.1]

      ...

}
```

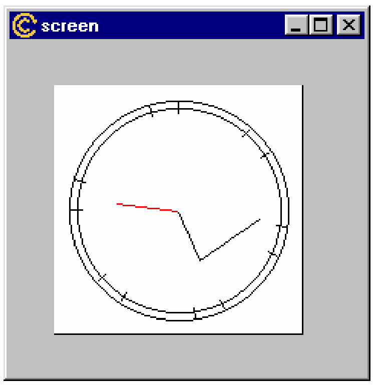


Figure 5: The corrupted clock