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.

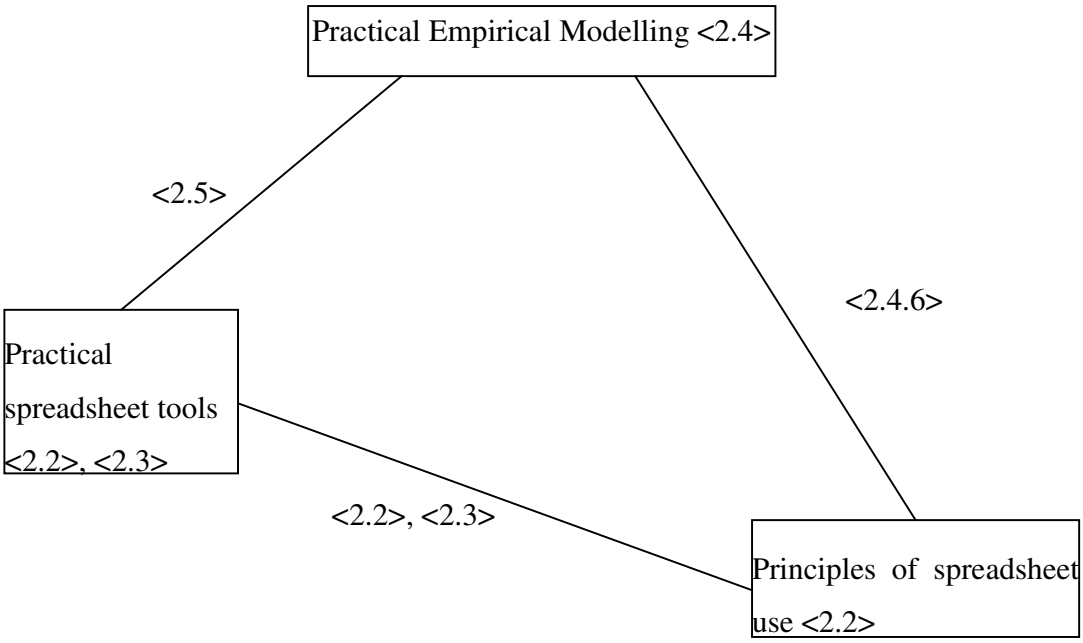


Figure 2.1 – Connecting Empirical Modelling, practical spreadsheet tools and principles of spreadsheet use

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
```

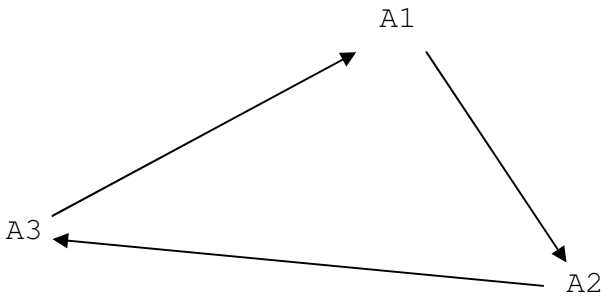


Figure 2.2 – Cyclic dependency.

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.

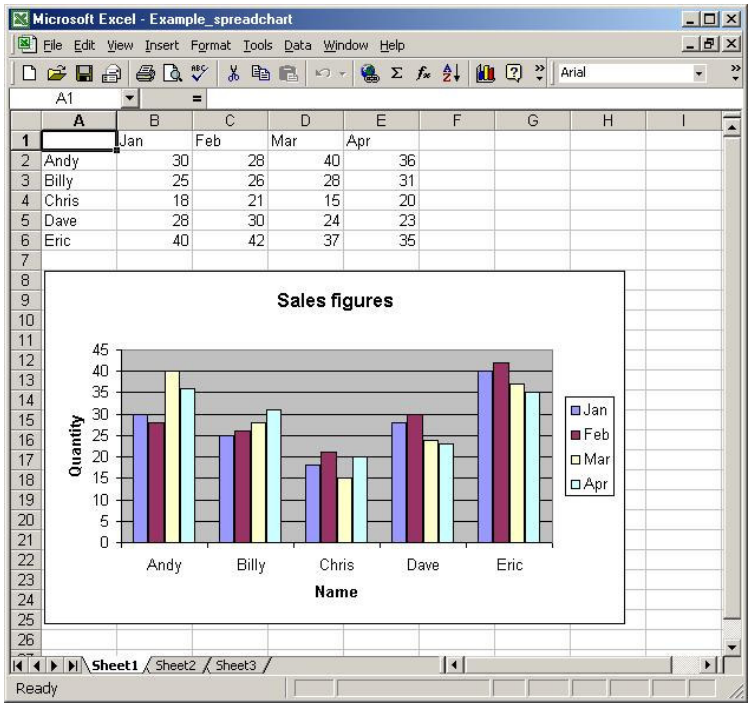


Figure 2.3 – An example spreadsheet and chart in Excel.

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).

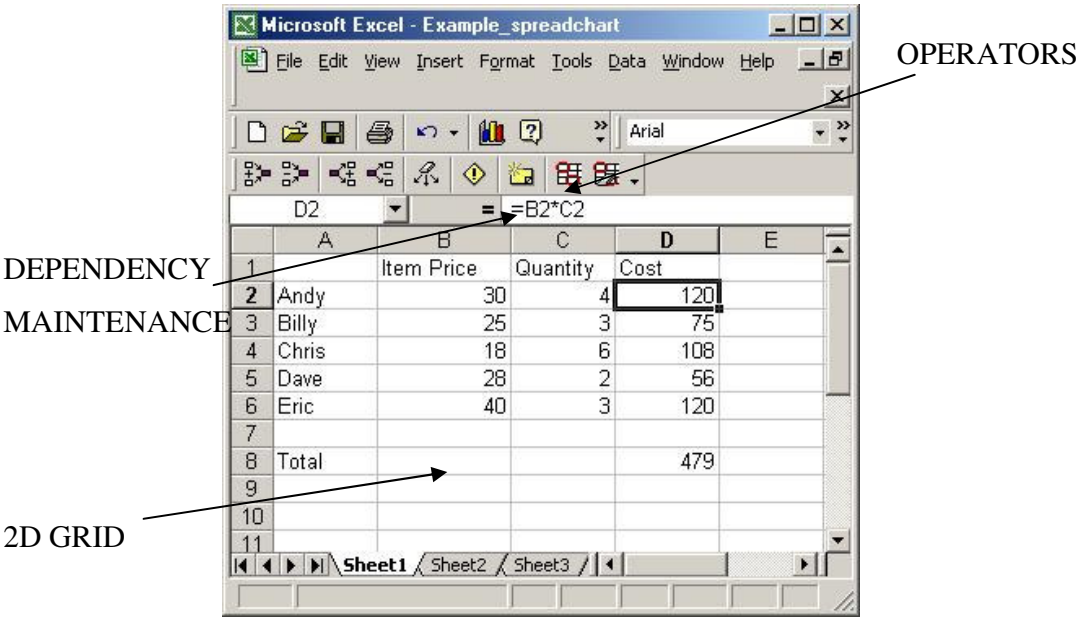


Figure 2.4 – A diagram of an Excel spreadsheet with the key characteristics of the paradigm being highlighted

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.

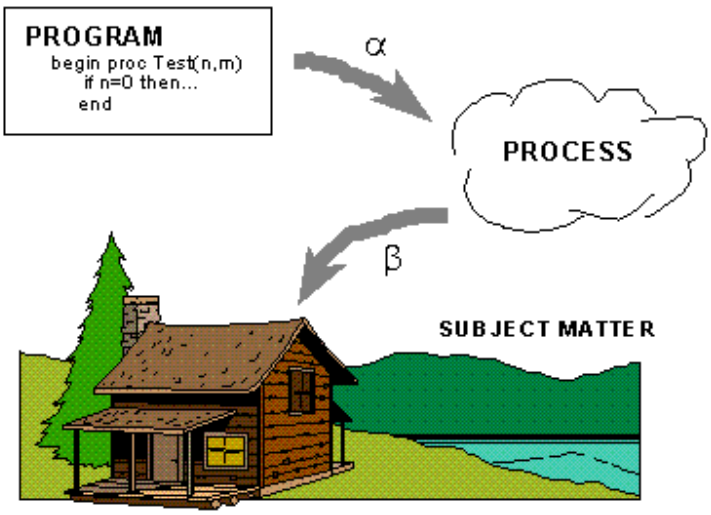


Figure 2.5 - Cantwell-Smith's program, process and subject matter [Smi97]

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)
```

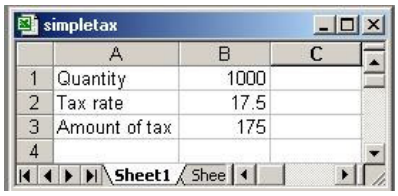
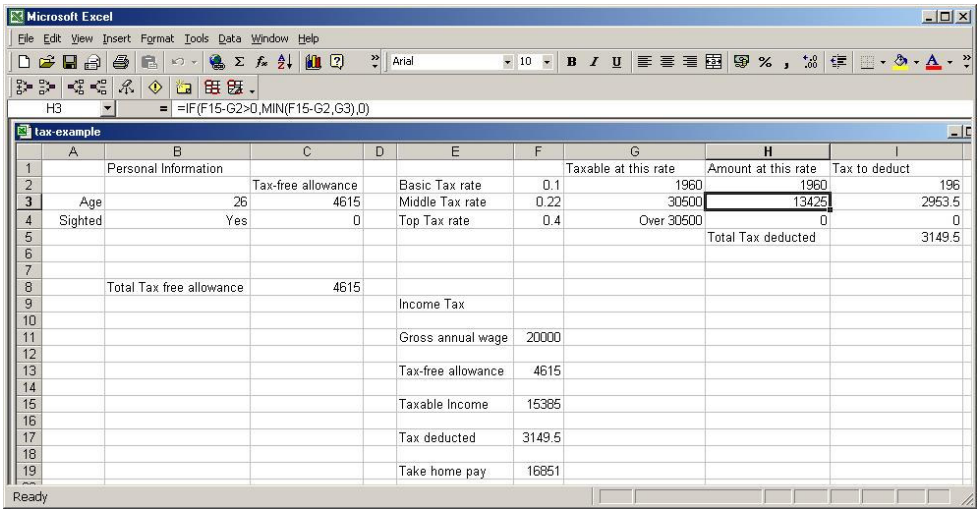


Figure 2.6 - A simple tax spreadsheet

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.



	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.

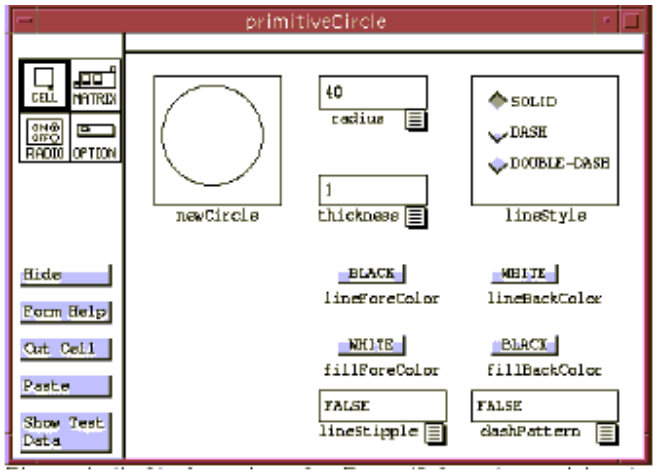


Figure 2.8 – Example of a Forms/3 form with a set of cells to define a circle and its attributes.

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]
```

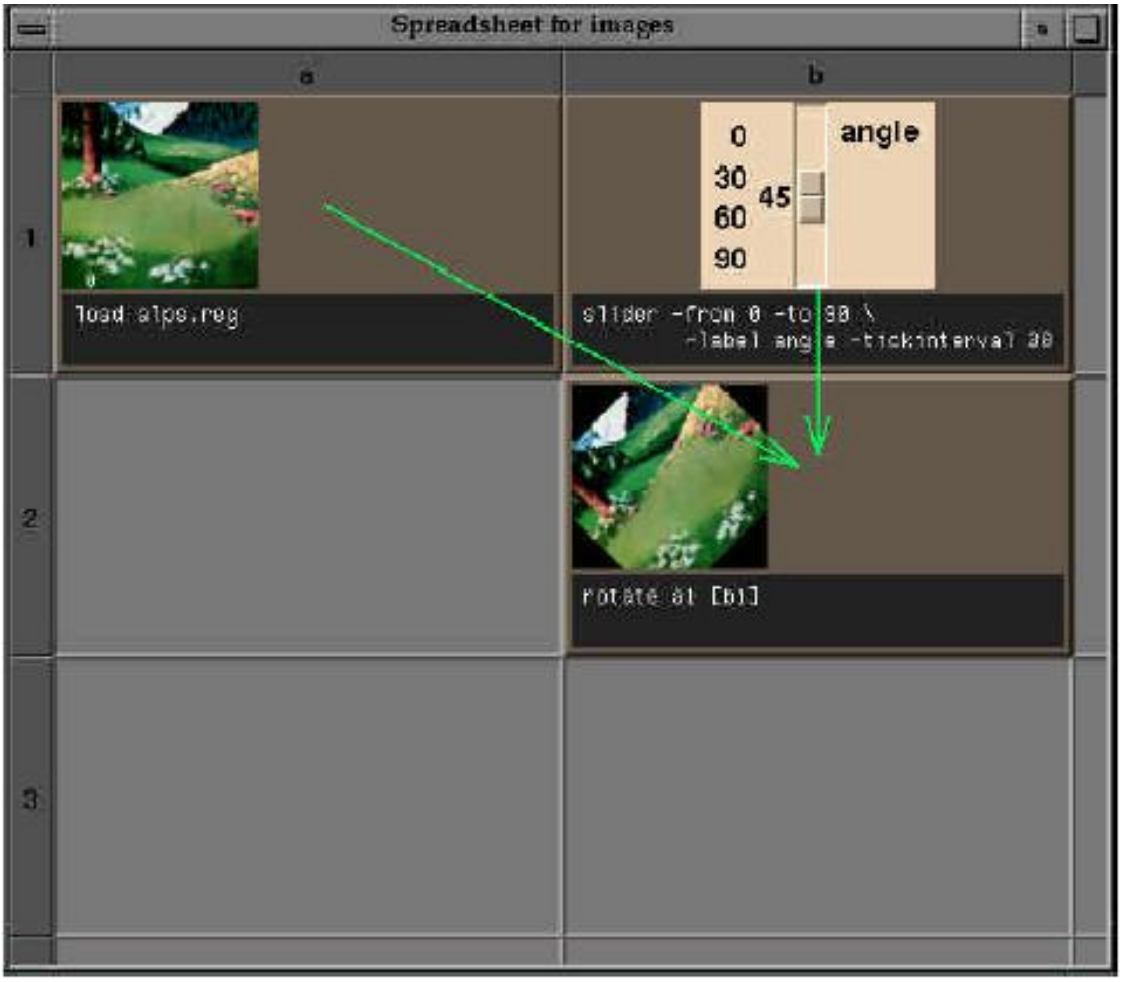


Figure 2.9 – An example of a spreadsheet for images, taken from [Lev94]

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.

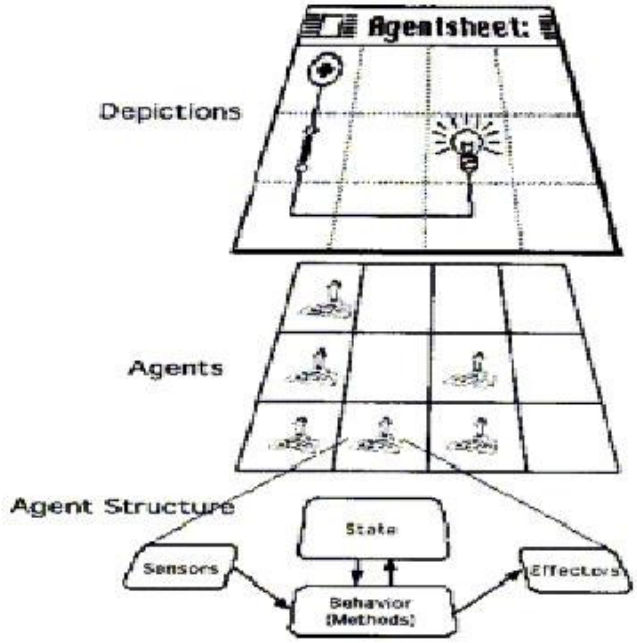


Figure 2.10 – Structure of an Agentsheet (taken from [Rep93])

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.

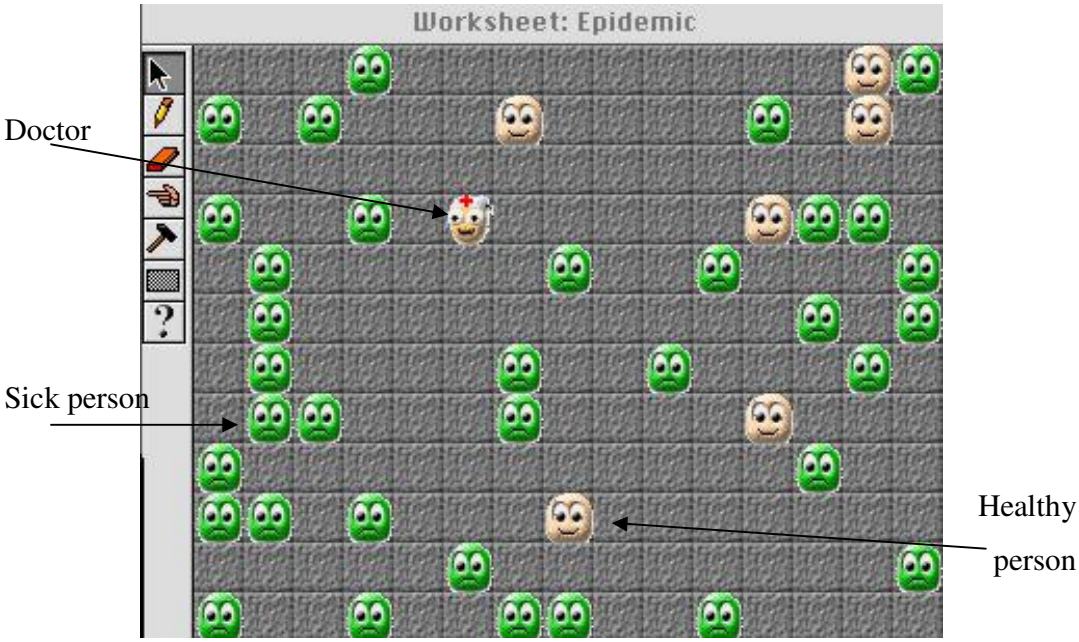


Figure 2.11 – A screenshot of the Agentsheets epidemic model

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.

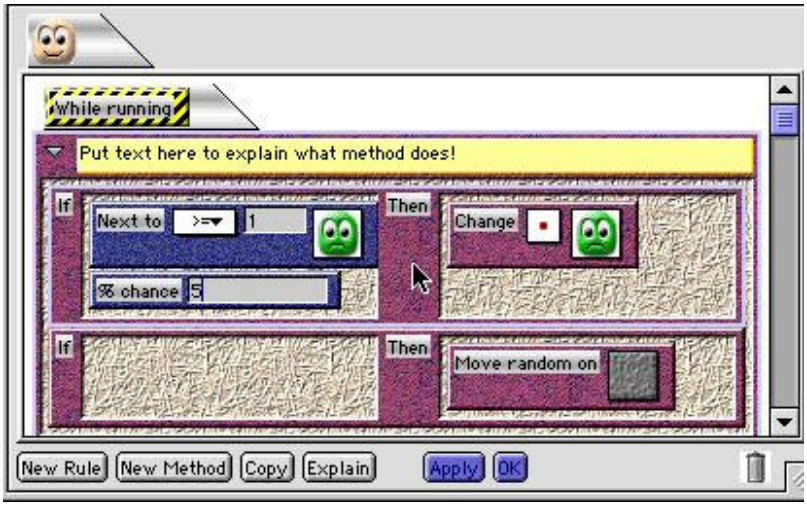


Figure 2.12 – A VisualAgenTalk rule for a person in the epidemic model.

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.

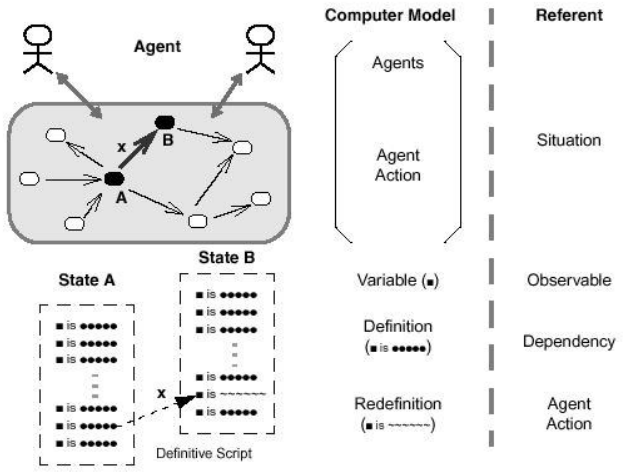


Figure 2.13 - The significant concepts associated with Empirical Modelling

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.

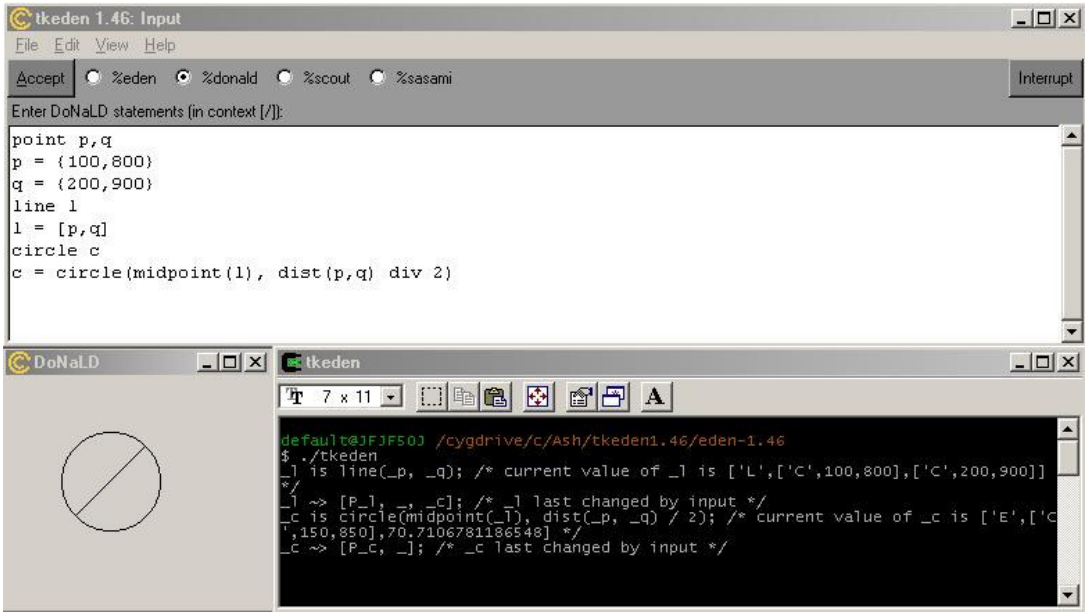


Figure 2.14 – The three windows in the TkEden modelling environment.

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) )
```

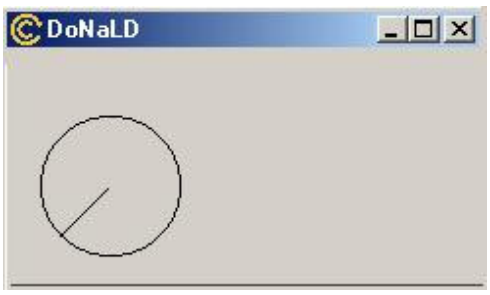


Figure 2.15 – An example DoNaLD fragment to define a circle

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>;
```

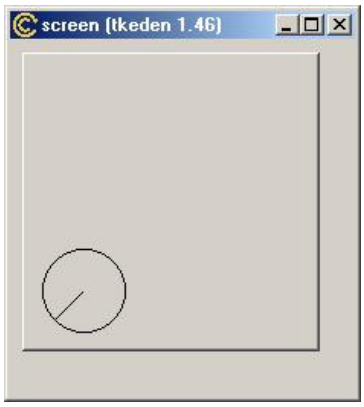


Figure 2.16 – An example of a SCOUT fragment to display the drawing in Figure 2.15 in a window

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
```

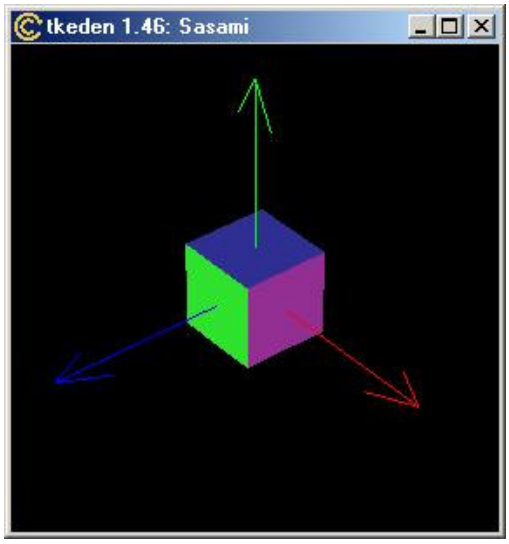


Figure 2.17 – An example of a Sasami fragment to display a coloured cube

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}]
```

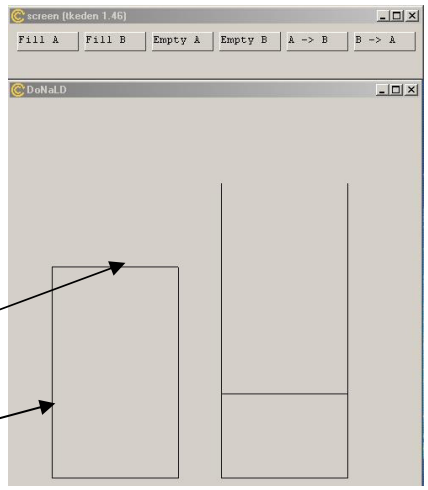


Figure 2.18 – The Jugs model from Appendix A

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.

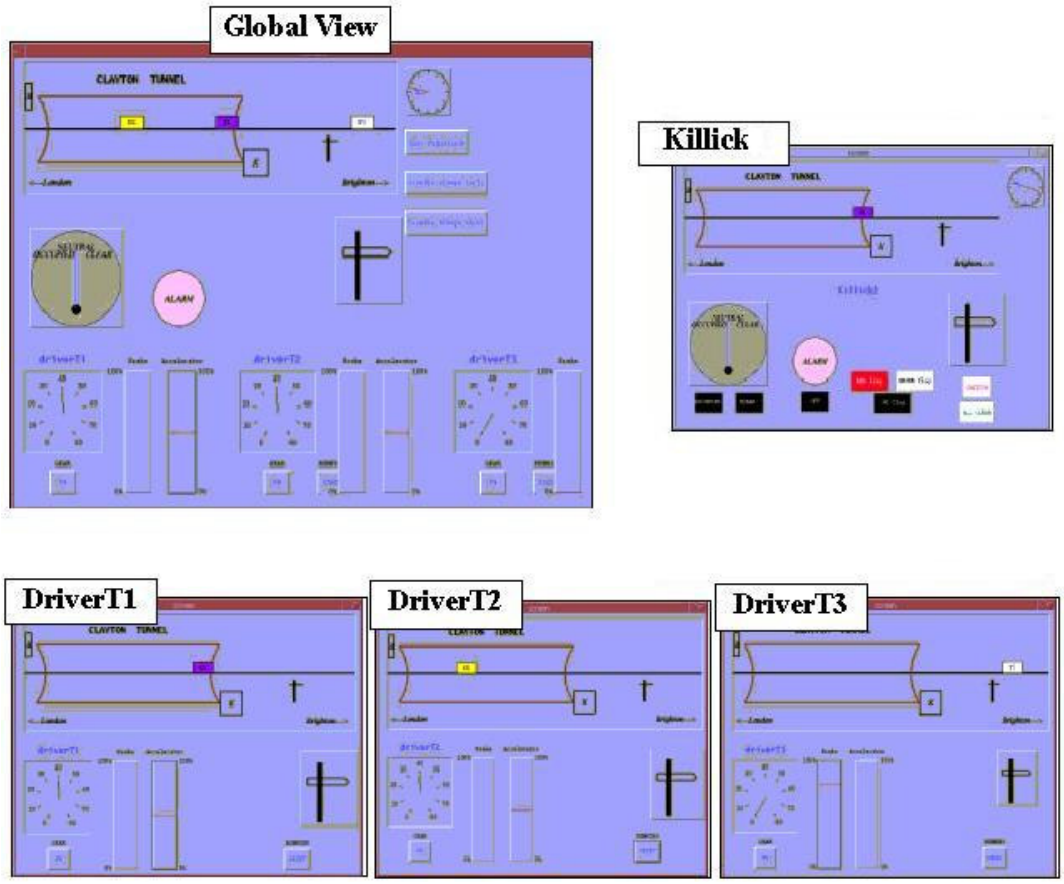


Figure 2.19 – Screenshots of the Clayton Tunnel simulation from the perspectives of each of the participants.

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);
```

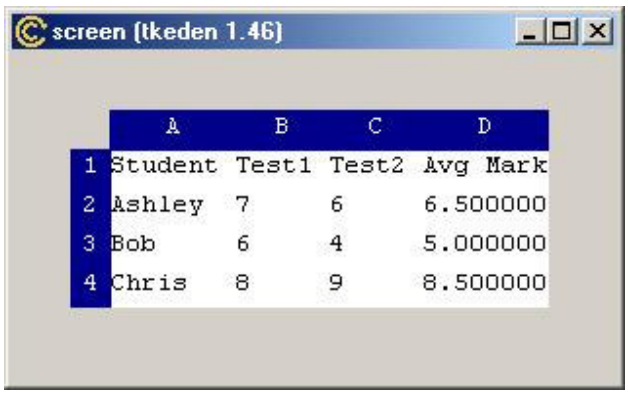


Figure 2.20 – A small example TkEden spreadsheet

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.

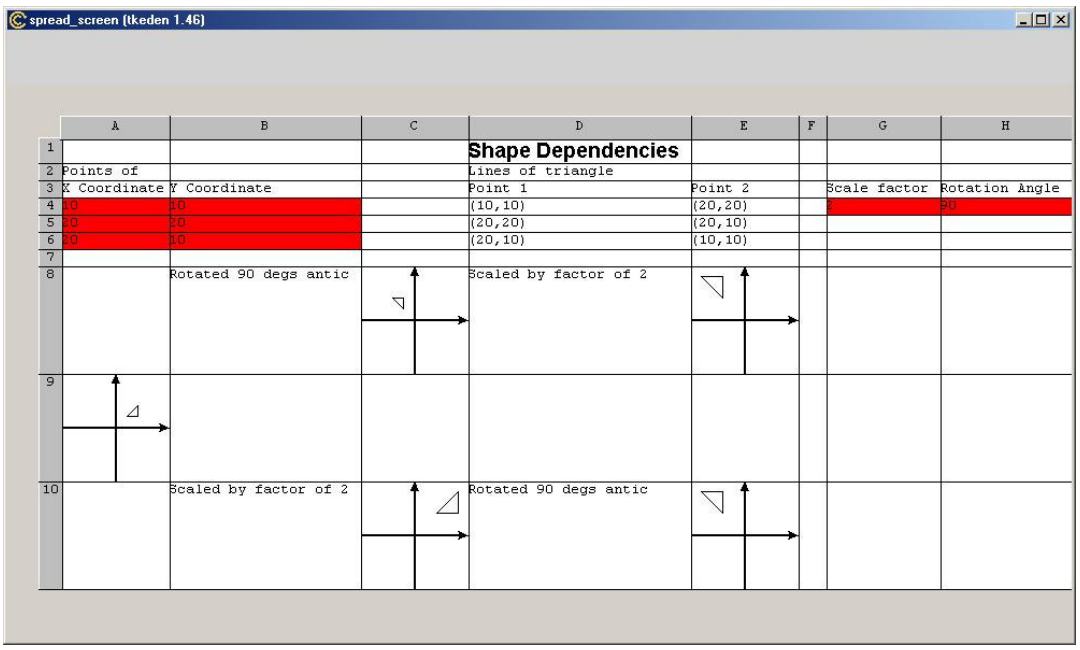


Figure 2.21 – The TkEden spreadsheet illustrating geometrical shapes in a spreadsheet

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.

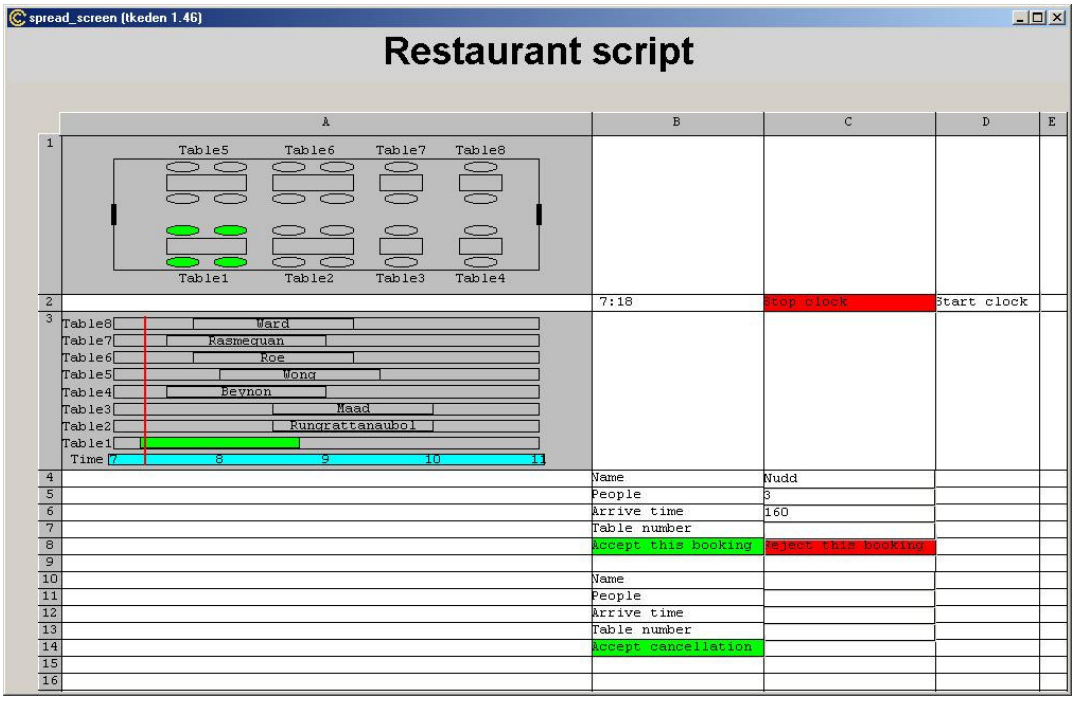


Figure 2.22 – The restaurant model in a spreadsheet

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.