MAIN CHANGED PARTS:

1. **Introduction**
2. **Conclusion**
3. **Section 1.6**

   ***old-version* p. 34**

   ***new-version* p. 30 a new paragraph has been added after the paragraph beginning with "The concept of modelling …."**
4. **Section 5.3.2**

   ***old-version* p. 186 para-4, p. 187 para-3 and p. 190 para-1**

   ***new-version* p. 156 para-1, p. 157 para-3 and p. 159 para-1**
5. **New introductions for each chapter to get rid of some repetitions**

Chapter 1

{ADDED IN}

In this chapter, the concept of open-development dependency that underlies MWDS is identified and compared and contrasted with closed-world dependency. An early example of the implicit use of dependency can be found in Ross's APT[1] language [ITT67], developed in 1967 for automatic programming of numerical machine tools. Several dependency-based applications, such as spreadsheets, Agentsheets, and several graphical modelling tools, are reviewed to illustrate the potential virtues of using dependency. The last section introduces a modelling framework, developed at Warwick over more than ten years, that adopts dependency similar to that reviewed in those applications as a fundamental concept.

{REMOVED}

1.0 Overview

This chapter reviews applications that feature dependency with particular emphasis on what we shall characterise as 'open dependency' (cf. Section 1-1). An early example of the implicit use of dependency can be found in Ross's APT[2] language [ITT67], developed in 1967 for automatic programming of numerical machine tools. Other examples of dependency-based software developed in diverse computer science disciplines to be reviewed in this chapter include the electronic spreadsheets of the 1970s, the interactive languages for graphics first introduced by Wyvill in 1975 [Wyv75] and the Information Systems Base Language (ISBL) query language designed by Todd in 1976 [Todd76]. The review also considers some current applications that make essential use of the dependency concept. The last section introduces a modelling framework, developed at Warwick over more than ten years, that adopts dependency similar to that reviewed in those applications as a fundamental concept.

[1] APT stands for "Automatically Programmed Tools", and the APT language.

[2] APT stands for "Automatically Programmed Tools", and the APT language.

1.7 Summary

The chapter has identified and discussed two distinct kinds of dependency: closed-world and open-development dependency. The latter is an essential concept in the modelling framework studied in this thesis. Dependency-based applications that put particular emphasis on open-development dependency have been reviewed to demonstrate the virtues of using dependency in diverse areas of computer science, for instance in spreadsheets, graphical modelling and database systems. Certain features of these applications, such as explicit editing of defining formulae and experimental-based (i.e. 'what-if') interaction, support open-development modelling activity.

The use of a definitive script to reflect an agent perspective, as introduced in this chapter, is one essential theme of MWDS. For instance, the spreadsheet – as a single-user application in Nardi's sense – will be interpreted as using open-development dependency to support modelling based on a single-agent perspective. In the next chapter, we will discuss in detail how MWDS can support single-agent modelling with reference to some simple case studies implemented using the tools developed at Warwick.

Chapter 2

{ADDED IN}

This chapter has two parts. The first describes the distinctive characteristics of a definitive script and reviews the computer tools that have been developed at Warwick to support MWDS. The second discusses the principles of MWDS in single-agent and multi-agent scenarios and the use of MWDS as a framework for the modeller's construal of an external situation. The chapter also introduces the concept of an agent-oriented LSD account and its supportive role in MWDS.

{REMOVED}
The concept of dependency has been implicitly used in a large number of software applications, as mentioned in Chapter 1. However, the essential principles and characteristics of dependency, particularly in the 'open-development' sense, have not yet been fully investigated.

2.0 Overview

The chapter will mainly discuss the essential features of dependency in open development, as represented (cf. Section 1.6) by modelling with definitive scripts (MWDS). Giving a more formal and detailed account of the use of definitive scripts in open-development modelling, as informally introduced in Section 1.6, the chapter can be divided into two parts. The first part (Section 2.1) will focus on the principles and characteristics of a definitive script and the significance of dependency, and give a brief explanation of computer support tools developed at Warwick together with some illustrative case studies. The second part (Section 2.2) will explore how MWDS can support single-agent modelling and then multi-agent modelling.

2.3 Summary

This chapter has discussed the principles and basic characteristics of a definitive script, and also given a brief review of tools and notations to support it. The virtues of definitive scripts in representing internal and external state in single-agent MWDS has been explored, along with a discussion of multi-agent MWDS. In the next chapter, we will exploit MWDS framework to support universal modelling together with many illustrative definitive scripts.

Chapter 3

{ADDED IN}
This chapter introduces the Abstract Definitive Modelling (ADM) framework for MWDS in all its aspects. The versatile use of definitive scripts in representing a variety of agent perspectives is illustrated by discussing how a wide range of models can be construed as ADM artefacts. These indicate the capacity for the ADM framework to support 'universal agent-oriented modelling'.

{REMOVED}

3.0 Overview

The last chapter has discussed various characteristics of MWDS in supporting single-agent and multi-agent scenarios in principle. This chapter, on the other hand, aims to exploit MWDS in practice through many definitive models. In order to achieve this, the Abstract Definitive Modelling framework is introduced and discussed in Section 3.1 to frame MWDS and hence to help in constructing the discussion of the variety of ways of using definitive scripts in various definitive models in Section 3.2.

3.2 Summary

In this chapter, the concept of an ADM artefact has been used to frame the wide range of potential applications for MWDS. Many definitive models with various characteristics have been discussed and explored to illustrate the versatility of MWDS. These indicate the capacity for the ADM framework to support 'universal agent-oriented modelling'.

Chapter 4

{ADDED IN}

This chapter discusses MWDS as a modelling activity with reference to a timetabling case study. The modelling activity is viewed from two complementary perspectives:

- as developing an instrument (the 'Temposcope') to give semi-automatic support to the timetabler [BWM+00];
- as constructing an interactive situation model (ISM) [Sun99] that embodies the modeller's growing understanding and experience of the timetabling scenario.

The qualities of MWDS in respect of interaction, comprehension, discovery and extension are also described.

{REMOVED}
Previous chapters have discussed the principles of MWDS and illustrated the characteristics of definitive models. This chapter focuses on the nature of the modelling activity itself. It also considers how we are able to apply MWDS in practice.

4.0 Overview

MWDS is based on the representation of state as observed and experienced within a given situation. Three basic concepts, observables, agency and dependency play a fundamental role. A definitive script is used to represent observables and the dependencies amongst observables in the referent. The key emphasis in MWDS is on interaction between the modeller and the situation and between the modeller and the computer-based model. The model is viewed as a physical artefact rather than as an abstract machine.

The activity associated with MWDS will be described with reference to a practical study in timetabling. It is typically difficult to find a systematic way of solving a timetabling problem, since the conditions or constraints on the timetable can change even as the timetable is being developed. This chapter discusses how MWDS can get around this problem by introducing a comprehensible representation of the current status of the modelling activity at every stage. This is illustrated in Section 4.1, which describes the use of MWDS first to construct the underlying data model for a timetabling problem (Section 4.1.1), then to add the visualisation and interactive features in the interface (Section 4.1.2), and finally to progressively build up solutions (Section 4.1.3). Throughout this model building, there is conceptually a single artefact that develops initially as the modeller's construal of the timetabling problem gets to be embodied in the artefact, and later as the modeller's construal of the problem is enriched through the interaction with the artefact. There are two ways to interpret the continuous evolution of the computer model:

- as developing the potential of the computer as a timetabling instrument (cf. [BWM+00]);
- as constructing an interactive situation model (ISM) [Sun99] that embodies the modeller's growing understanding and experience of the timetabling scenario.

MWDS has many characteristics that enable the modeller, and the user working in conjunction with the modeller, to adapt the emerging model to cope with new and unexplored situations. Relevant properties, such as interaction, comprehension, discovery, and extension are discussed in Section 4.2.

4.3 Summary

This chapter has illustrated how MWDS can be applied in constructing an application viz. the Temposcope. In this context, the developer initially focuses on capturing the data required for building the timetable. Through the refinement process, the developer, acting in the role of a timetabler, observes and learns what is required in the manual construction of the timetable. The constraints and relationships between observables introduced reflect the timetabler's concerns in building the timetable. The model can be used as a partially automated tool to help the timetabler to construct a timetable. The process of building the timetable involves many characteristic activities, such as interaction, comprehension, discovery and extension, that can be supported by MWDS.

Chapter 5

{ADDED IN}

As described in the introduction to the thesis, this chapter sets out to illustrate how MWDS can be viewed (controversially) as providing an alternative and unorthodox picture of the relationship between the world and our representation of it. In the experience of the author, such an alternative viewpoint is natural for the modeller engaged in MWDS. This leads the author to conceive the distinction between MWDS and traditional computer-based modelling (for example, in respect of openness, and experiential character) as fundamental rather than as a matter of degree. This is potentially a source of controversy for sceptical readers – one that becomes even more pronounced in Chapter 6, where such a fundamental distinction is presumed.

Representing state is a key concern in MWDS. The discussion in this chapter addresses the connection between state and behaviour in MWDS. By introducing the concept of an ADM device and a program device, it is possible to explore the link between MWDS and traditional modelling of state and behaviour. The ADM device and the program device are compared and contrasted with reference to four aspects of state – Situational, Explicit, Mental and Internal – that are of concern when we use and interact with a device. The two principal case studies in this chapter relate to:

- a simple program ('Jugs'), originally developed for the BBC micro-computer by R. Townsend [Town], that has been widely used in schools;

- a CSP specification for a chocolate vending machine introduced by C. A. R. Hoare [Hoare85],

which are respectively used to explore the explicit and internal, and the situational and mental aspects of state.

{REMOVED}
So far this thesis has addressed MWDS for representing state, especially from the perspective of exploratory modelling. The themes of this chapter are connecting state to behaviour in MWDS and exploring the link between MWDS and traditional modelling of state and behaviour.

5.0 Overview

Open-development and closed-world modelling offer different views of representing state and behaviour. MWDS offers a very open-ended view to construct the model while traditional modelling is closed, precise and circumscribed. Without placing some restrictions on the open-development modelling, we cannot compare these two approaches. We need to consider artefacts that are 'program-like' and have a specific use (e.g. a digital watch and the Jugs program in [Town]) in order to bring the two views of modelling state into line with each other. We adopt the term 'device' for such an artefact, to distinguish it from an ADM artefact. Four aspects of state, which are of concern when we interact with a device, are introduced and the explanation and elaboration of these four aspects of state: **S**ituational, **E**xplicit, **M**ental and **I**nternal, is included in Section 5.1. Section 5.2 discusses in detail the relation of these four aspects of state and MWDS using the Jugs model as a way of illustration.

Section 5.3 will explain how MWDS relates to traditional techniques (e.g. data structure, MVC, CSP, dataflow) for constructing programs. This section will compare and contrast how MWDS and traditional approaches handle these four aspects of state by dividing them into two groups: **I&E** and **S&M**. The summary of the chapter is written in Section 5.4.

5.4 Summary

This chapter has compared and contrasted MWDS and traditional programming with reference to the four aspects of state (SEMI). These four aspects are concerned with the use of a device in a situation. The concept of a program device and an ADM device are introduced to make the comparison between these two distinct approaches possible. A program device is designed and developed based on a preconceived and precise description. Its patterns of interaction to be used in possible situations are designed and prescribed in advance. Traditional approaches to developing requirements rely upon elicitation to create an idealised model of the interaction with a device that only partially reflects the implicit assumptions about how this interaction is mediated by actual observables (cf. the way that interaction in a game of Noughts-and-Crosses is explored in [BJ94]). In contrast, MWDS constructs an ADM device for cultivating requirements that puts the emphasis on the significance of experience and experimentation. It also tends to represent and handle the SEMI aspects of state in parallel, whilst a traditional approach tends to separate these aspects in developing the device. By using MWDS, we can connect software development with the analysis of perception and action.

Chapter 6

{ADDED IN}
As described in the introduction to the thesis, this chapter sets out to illustrate how MWDS can potentially be viewed as providing empirical roots for logical constructions. This agenda is only appropriate if the distinction between MWDS and classical programming is presumed to be fundamental in character. For convenience, this chapter is written from this perspective and is acknowledged to make controversial claims.

There are two quite different ways to interpret computer use. The classical theory of computation accounts for interaction with computers in terms of algorithms and formal models of state and automata [HU79]. The semantics of MWDS has an entirely different basis in which the key emphasis is upon the computer as a physical artefact and on experientially mediated interpretation of human-computer interaction (cf. Section 2.2). Beynon [Bey99] argues the case for a non-logicist foundation for AI that stems from such experientially mediated interpretations, and discusses the significance of languages and logics within this framework.

6.0 Overview

The key concept in our case study is the ‘data structure’. A data structure serves two purposes: it can be viewed as a physical artefact that metaphorically represents computer manipulation of data; it is also used to frame state transitions within a computer program (cf. Section 5.3.1). Our specific focus in this chapter is upon the *heap* data structure and its application in the heapsort algorithm. The two main sections of the chapter address:

- the use of MWDS to construct a model of the heap data structure as a physical artefact (such as a lecturer might draw on a blackboard when introducing heapsort);
- how a formal account of heapsort, based on Dijkstra’s Weakest Precondition (WP) formalism can be interpreted in terms of the definitive Heap model[3].

Section 6.1 introduces the heap structure and the conventional heapsort pseudo code. This is followed by a discussion of the development of the Heap model based on the observation-oriented and agent-oriented paradigm provided by MWDS. The model building is closely linked to the kind of experimental analysis that might have led to the discovery of heapsort, and the model is gradually developed in a manner similar to that discussed in [SRCB99]. The definitive Heap model supplies the basis for a non-standard implementation of heapsort, centred on the representation of state as observed and experienced by the modeller. This can be viewed as an ADM Heapsort device that has different characteristics from a conventional heapsort program.

Section 6.2 relates the experiential view of heapsort associated with the ADM Heapsort device to a formal specification of the heapsort algorithm using Dijkstra’s WP formalism [Dij76]. The integration of informal and formal views of heapsort within the Heap model is addressed in subsection 6.2.1, where we show how the states in the heapsort process that are characterised by pre- and post-conditions in WP can be interpreted and visualised in the Heapsort device. The characteristics of the Heapsort device as a non-standard model of heapsort are discussed in subsection 6.2.2. In the final subsection (6.2.3), we consider the relationship between MWDS and the classical theory of computation, using heapsort as a case study. This involves construing the different ways in which heapsort is performed by the ADM Heapsort device and by a conventional heapsort program.

Chapter 7

{ADDED IN}

In this chapter, some of the problematic issues and limitations of MWDS that have been encountered in the study and development of definitive models are reviewed, and some possible solutions are suggested.

{REMOVED}
The discussion so far has illustrated many characteristics of using MWDS to represent state as experienced and observed. Through the review and study of many definitive models developed

[3] Some material in this section is drawn from [BRS00]

with the EM tools, many problems and difficulties have emerged during different phases of modelling such as construction, debugging and maintenance. This chapter will discuss these problems, which may stem from the principles of MWDS itself or from the tools that are still under-developed, and then suggest some possible solutions. The discussion of the problems and possible solutions can lead to innovation and improve and clarify the principles and supporting tools.

7.0 Overview

In this chapter, the difficulties and problems that we have at the current stage of development of the tools and concepts are studied in section 7.1. It highlights the problems with model construction (Section 7.1.1) stemming from the fact that definitive principles are quite new and different from a traditional programming paradigm. This is followed (Section 7.1.2) by a discussion of the problems (in terms of comprehending, interacting, debugging and reusability) of maintaining large scripts and the issue of efficiency (Section 7.1.3).
Section 7.2 discusses possible solutions to cope with the problems of comprehending, maintaining and interacting with large definitive models. It can be divided into two main sections. One is concerned with tools to help in understanding and interacting with the models. The second is centred on automatically generating large scripts, which is a tedious process if carried out manually.

7.3 Summary

This chapter has discussed some of the limitations and difficulties that arise in MWDS, and suggested simple possible solutions to overcome some specific problems.