THE UNIVERSITY OF

WARWICK

Empirical Modelling
for Participative Business Process Reengineering

by

Yih-Chang Chen

Thesis

submitted to the University of Warwick
in partial fulfillment of the requirements for admission to the degree of
Doctor of Philosophy

Department of Computer Science
The University of Warwick

Coventry, United Kingdom
December 2001

dedicated to Jung-He and

Ming-Mei, my dad and mum

Contents

List of Tables

List of Figures

Acknowledgments

I am deeply indebted to many persons who have provided help, support and encouragement. First of all, I would like to thank my supervisor Dr Steve Russ for his unvaluable help, advice, patience, and unselfish support throughout the preparation of this thesis. I would surely not have been able to finish this thesis without his help. Warm thanks also to Dr Meurig Beynon, my adviser, for his fruitful discussions and comments during this research. I would also like to thank my colleagues and other friends both in Empirical Modelling Group and in this department.

I also owe thanks to all my friends here during the period of my MSc study in LSE and doctoral programme in Warwick, who made my life of study rich and joyful: Dr James Backhouse (my supervisor in LSE), Dr Sedat Agan, Harun Esen, Russell Lewis, James Nyman, all members of WTC, and to all other friends in UK, Taiwan and Turkey for their help and friendships.

Finally, I would like to express my heartfelt gratitude to my dear parents Jung-He and Ming-Mei for sharing their life-wisdom and giving crucial advise in difficult situations. Also warm thanks to my elder brother Yih-Lang. Without their support this thesis would never have been completed.

- · — · — · — · — · — · — · -

Further thanks goes to Dr Chrystopher Nehaniv for his constructive comments during the viva and to Steve Russ and Meurig Beynon for their useful help on this final version.

Declarations

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

The perspective of Empirical Modelling for Business Process Reengineering has been published in (Chen et al., 2000a). The various aspects concerning the application of EM to Participative Process Modelling have been presented in (Chen et al., 2000b). The view of Interactive Situation Models relating to software system development and the framework of SPORE have been proposed in (Sun et al., 1999). Some of the technical work for the example of the warehouse management system has been described in (Chen et al., 2000a), (Chen et al., 2000b) and (Sun et al., 1999).

Abstract

The purpose of this thesis is to introduce a new broad approach to computing – *Empirical Modelling* (EM) – and to propose a way of applying this approach for system development so as to avoid the limitations of conventional approaches and integrate system development with business process reengineering (BPR). Based on the concepts of agency, observable and dependency, EM is an experience-based approach to modelling with computers in which the modeller interacts with an artefact through continuous observations and experiments. It is a natural way of working for business process modelling because the modeller is involved in, and takes account of, the real world context. It is also adaptable to a rapidly changing environment as the computer-based models serve as creative artefacts with which the modeller can interact in a situated and open-ended manner.

This thesis motivates and illustrates the EM approach to new concepts of *participative BPR* and *participative process modelling*. That is, different groups of people, with different perceptions, competencies and requirements, can be involved during the process of system development and BPR, rather than just being involved at an early stage. This concept aims to address the well-known high failure rate of BPR. A framework SPORE (situated process of requirements engineering), which has been proposed to guide the process of cultivating requirements in a situated manner, is extended to participative BPR (i.e. to support many users in a distributed environment). Two levels of modelling are proposed for the integration of contextual understanding and system development. A comparison between EM and object-orientation is also provided to give insight into how EM differs from current methodologies and to point out the potential of EM in system development and BPR. The ISMs (interactive situation models), built using the principles and tools of EM, are used to form artefacts during the modelling process. A warehouse and logistics management system is taken as an illustrative case study for applying this framework.

Abbreviations

ADM ..... Abstract Definitive Machine

AI ..... Artificial Intelligence

AMORE ..... A Methodology based on Object-Orientation for Reengineering Enterprises

BPR ..... Business Process Reengineering

BPRC ..... Business Processes Resource Centre

CORBA ..... Common Object Request Broker Architecture

CREWS ..... Cooperative Requirements Engineering With Scenarios

DEM ..... Distributed Empirical Modelling

DoNaLD ..... Definitive Notation for Line Drawing

DSS ..... Decision Support System

EDDI ..... Eden Definition Database Interpreter

EDEN ..... Evaluator for Definitive Notations

EM ..... Empirical Modelling

GST ..... General System[s] Theory

HCI ..... Human-Computer Interaction

IS ..... Information System[s]

ISM ..... Interactive Situation Model

IT ..... Information Technology

LSD ..... Language for Specification and Description

LSE ..... The London School of Economics and Political Science

MIS ..... Management Information Systems

MIT ..... Massachusetts Institute of Technology

OMG ..... Object Management Group

OMT ..... Object Modelling Technique

OO ..... Object-Orientation; Object-Oriented

OOBE ..... Object Oriented Business Engineering

OOSE ..... Object Oriented Software Engineering

RE ..... Requirements Engineering

SCCS ..... Source Code Control System

SCOUT ..... Notation for Screen Layout

SPORE ..... Situated Process of Requirements Engineering

SSM ..... Soft System Methodology

TQM ..... Total Quality Management

UML ..... Unified Modelling Language

CHAPTER ONE

Introduction

The core of this research is aimed at introducing an innovative approach (Empirical Modelling) to business process reengineering and the development of associated information systems. This first chapter starts with the aims and motivation for the research and this thesis. It includes the main fields and focus of this research, and gives the preliminary overview of the challenges and potential problems faced in these subjects and sometimes met by the author and other members in this group. At the end of this chapter, the outline of the thesis is presented.

1.1 Research Motivation and Aims

The motivation for this thesis stems from the author's master study in the London School of Economics and Political Science (LSE) during the year of 1995/1996 focusing on the security and management issues of information systems (Chen, 1996). At that time (five years after the first appearance of BPR in 1990), the concept of BPR was still quite new and attracted considerable attention amongst academics, business and industries. The keen interest in BPR can be attributed to the fact that lots of leading companies were anxious to stay close to their customers. BPR was originally developed by former MIT professor Michael Hammer both in his article (Hammer, 1990) and book (Hammer and Champy, 1993). To sum up, BPR is about completely rethinking processes without too much analysis of the existing practices. In BPR, the power of information technology (IT) is a central theme. It encourages the use of IT to radically redefine the business. Although BPR is regarded today as one of the prime managerial

approaches for business survival and increasing business competitiveness, still about 50 to 70 percent of the BPR projects have failed to achieve the results at which they aimed. These failures can be partly explained by the poor attempts to gain a complete understanding of the business processes and the subsequent deployment of inappropriate IT systems to support them. One main issue of BPR is to investigate whether or how IT can generate new process designs or how it can contribute to the performance of business processes (that is, to explain the role of IT as an enabler or a supporter). This thesis focuses on the topics of BPR and system development together partly because BPR is tightly connected with IT. And the hardware and software of IT have the ability to support the operational and informational, and even managerial, activities of business processes.

Today, it is widely admitted that the successful deployment of information systems within an organisation depends heavily on a good understanding of the customers' needs and on the quality of the requirements analysis for the supporting systems. That is to say, if the system developed does not meet the actual needs of users, the users may fail to use or even discard that system because for them that system is regarded as an ineffective means of assistance or even an obstacle to the achievement of their goals. Thus during the projects of BPR and the process of developing supporting information systems for the newly designed business processes, the task of requirements analysis is one of the most challenging and the most error-prone activities. One reason for this is that it involves a communication-intensive activity, and usually the business management and personnel (in the rest of this chapter we will call all of them 'stakeholders') lack the clear understanding of what the desired system should do for them and tend to change their minds about the functionalities of that proposed system. Traditionally the analyst elicits requirements from stakeholders in informal ways, synthesises the information and then develops a representation of the system requirements (such as requirements specifications or models). These requirements models developed are typically based on the analyst's personal understanding of the requirements after he has synthesised the information. And the stakeholders are only involved during the requirements determination process[1]. This leads to one of the motivations of this research: to identify an appropriate channel between the stakeholders and analysts for their communication and negotiation, or the process of jointly developing models, during the requirements elicitation as well as

the other stages of system development and BPR, which enables the participation, good understanding and speedy acceptance by both parties.

Now let us turn to the issues of software system development. As software systems become more and more complicated, it becomes more and more difficult for designers and programmers to cope with such complexity. Further, some large programs may need several years to develop, and the programmers often meet situations in which some significant parts of the system are still under development with none of their original developers. Normally the principal efforts of system development are to deal with the partitioning of the program and technical knowledge among individual designers. But with each partition, the members of the designing team may change several times during the development, and thus further partition the program.

One way to deal with the complexities of large system development is to abstract and circumscribe the details of programming. This characteristic of abstraction and circumscription is popular in object-oriented development methodologies. This raises one of the motivating issues of this research: the expressive power provided by system development notations, i.e. the richness of the concepts they can offer and in terms of which the customer's requirements can be expressed. Normally these notations reflect the domain of application for which the models has been designed. But what is essential is that the modelling concepts should not only support the modelling of requirements which are inherent to the software system, but also the modelling of the *environment* of that system (people, other devices, etc). To this end, suitable modelling concepts should include facilities for modelling the real-world situations. In the context of the development of information systems to support the business processes, the results of requirements should include the specifications not only of the software to be implemented, but also of the environment around this software, that is the business, which includes the interactions between human and automated agents. This is what many conventional methods fail to achieve, especially those

1. Galliers (1995) further describes this situation as: "the [business managers] are often happy in the mistaken belief that information technology can be left to the technologists, and many [system developers] are happier to have an information systems strategy and an information systems development that are more concerned with technological issues than with business imperatives -- with as little as possible involvement from business executives". (p. 52)

focusing more on software development (design and implementation) rather than on requirements. Because they are aimed at, and limited to, the modelling of software artefacts, such as data-flows, data-base or control and communication processes, they do not offer enough expressive power to capture the real-world happenings.

This raises another issue: that of 'preconception' in system development. The task of defining data-flows or communication processes involves preconceiving the rules of event-condition-action through which the action triggering is specified. That is to say, at any moment, when an event occurs and if a condition on the current state is met, then the action should occur. For this, the system designers need to evaluate the set of events which occurred as well as the set of conditions currently prevailing in order to determine the set of actions. But actually no one can preconceive or predict all the causalities in the real world, especially when developing information systems for the use of business or modelling the business situations which consist of human beings. When dealing with the modelling of the real-world things such as human behaviour, we may need to consider the uncertainty which is associated with the occurrence of actions.

The approach taken in this thesis involves a more holistic view of the system development environment, which should also include the business environment in which the system is or will be running. We assume that no methodology provides a complete solution. Instead the focus of the research is mainly on 'programming through modelling' rather than on a system architecture which aims to develop new systems.

Exploring the existing environment in a new way will enable us to address the unforeseen complexity in the business domain. This will be critical to the success of BPR. As we can see, business management or BPR analysts do not have the adequate knowledge and understanding of the technology. Software developers may not understand the principles of management enough to know what they should communicate to the managers in order for them to use and manage the IT systems. This situation forms a conceptual gap which divides the world of system design from the world of BPR techniques, i.e. the gap between what the users want and what the software developers think the users

want. The existence of this gap shows itself not only in the poor understanding of business processes by the IT experts, but also in the difficulties that BPR experts meet when they try systematically to uncover the business process from the existing software system. This gap leads to difficulty in transferring knowledge between the people who are designing the information systems required for a BPR implementation and their customers.

Mingers (1995) suggests that this situation may be made even more difficult when requirements and design are focusing on technical characteristics of the system. This makes the developers see the situation differently from how it is seen by the users. As he describes:

> Users are concerned with business tasks and objectives and formulate requirements in these terms. They are concerned not with the system itself, but with what it can achieve. This is a problem that the [system] analyst must overcome by trying to see the world as the user does, but conventional methodologies have little to offer in this respect. (p. 20)

From the perspective of participative BPR we are aiming to bridge this gap. During the modelling process (broadly including BPR modelling and system development), all the participants playing various roles (stakeholders, analysts, designers, etc) are expected to apply their appropriate knowledge to the modelling process by constructing artefacts to serve as inputs and providing feedback as well as to generate outputs. The relationships between different roles can be understood through the modelling process under the distributed environment, in order to identify the interactions and in turn enable the appropriate tasks for system development and BPR to be formulated. For example the first level of interactions are between system developers and business stakeholders (cf. Figure 1.1). The responsibility of system developers is to develop the supporting information systems for the business. The developers include software developers and hardware developers, and in the software development organisation the software developers may comprise project managers and software engineers. Then we can find the second level of interactions between the project manager and software engineers. For example, it is the responsibility of the software engineers to communicate the appropriate information about the state of application development to the project managers. This enables the manager to fulfil his responsibility to the clients, which may be business management or personnel. Similarly the business stakeholders can

be divided into customers, management and personnel, each of them have a specific set of experiences and beliefs about how the business process is organised and running. For example in BPR, if their values and attitudes are not deeply considered in the redesign of new processes, then their old attitudes or behaviours may conflict with the new process. The distributed tools of EM provide an environment for participative process modelling in which stakeholders, analysts and modellers are represented and included in the process. We can conclude that the increased user involvement in the modelling process will raise the level of participants' contributions in the development of systems. User participation enables the creation of better models than that solely created by the analyst. To this end, the purpose of this research is to investigate how the continuous interaction with and the involvement of users throughout the modelling process of BPR and system development that will increase the probability of implementation success.

The primary purpose of this thesis is to investigate the suitability of Empirical Modelling as the framework for a new approach to software development and business process reengineering. Relevant

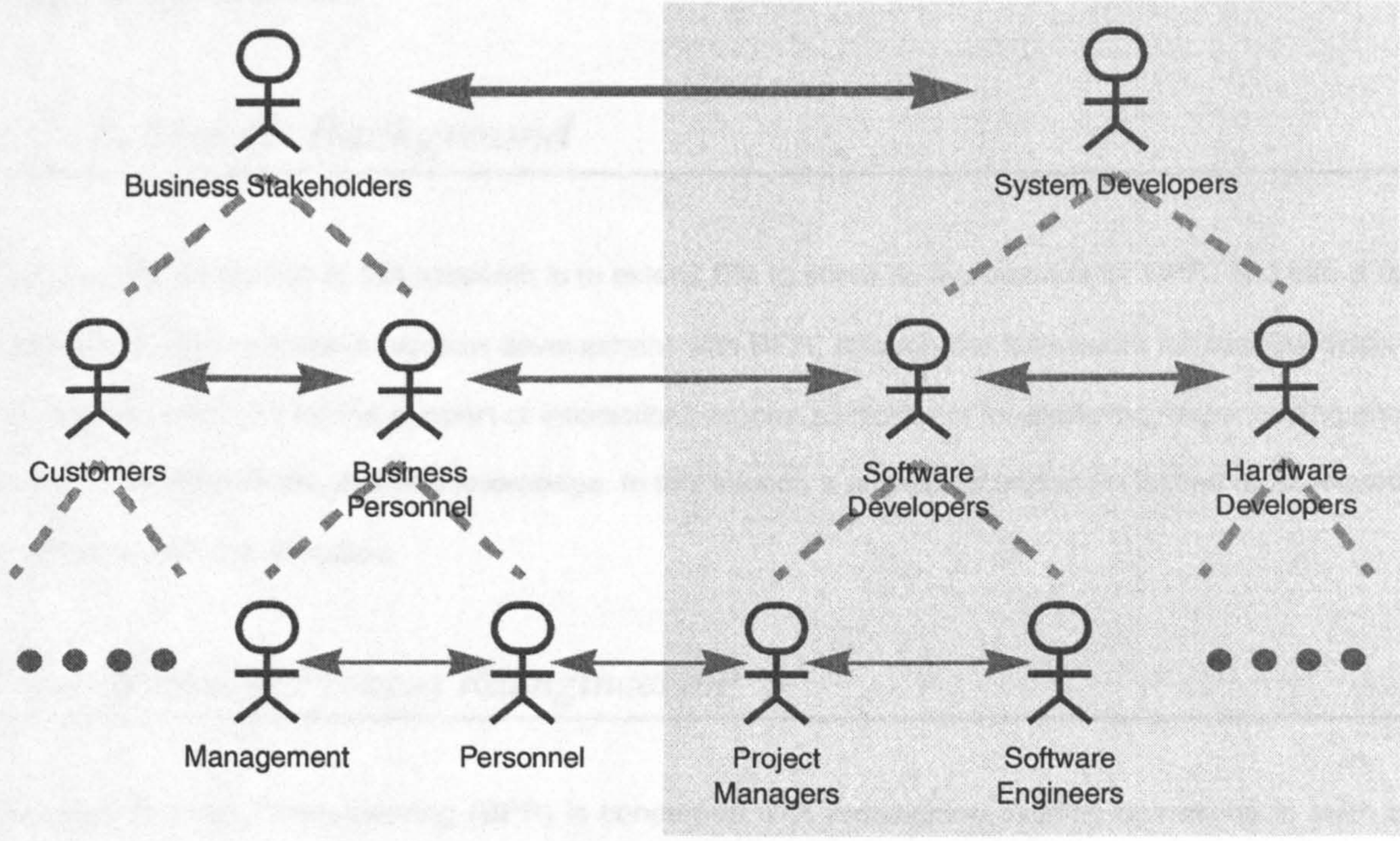


Figure 1.1 Various Roles of Participants in the Process of BPR

researches into EM in system development have been done by two previous PhD students in this group (e.g. Ness, 1997 and Sun, 1999). However we should note that, as has been discussed in these previous research works, EM does not involve the definition of well-structured steps or procedures that follow formalised rules and algorithms, and should not be viewed as a technical process like conventional system development. The emphasis of the research is on the need to improve the modeller's understanding and shape his experience through experiments and observations, i.e. regarding the software system to be developed as a computer-based model, and furthering the development of this software system through modelling. Through the interaction with the computer-based model, the modeller can conduct the modelling activities in a situated manner just as human agents solve problems in the real world. This situated characteristic of the modelling process enables the modeller to design and learn (through the feedback from his interaction with the model), and enrich his experience. In this way the modeller can also adapt the system (the model) to its evolving environment. When adopting EM as an approach to BPR, we need also to consider the interactions between participants. Thus this thesis tries to clarify the characteristics of EM from the perspective of BPR, in particular in relation to user participation in the modelling process of BPR.

1.2 Research Background

The aim and motivation of this research is to extend EM to serve as the process for BPR. The aim is to integrate the EM practice of system development with BPR, through the framework for user participation and the EM tools for the support of interactions among participants for exploring, experiencing and communicating their insights and knowledge. In this section a number of important issues most related to this research are identified.

1.2.1 Business Process Reengineering

Business Process Reengineering (BPR) is concerned with redesigning existing operations in such a way as to exploit new technologies and to serve customers better. It involves radical changes to busi-

ness processes to achieve dramatic improvements in quality, cost, speed, etc. Today this concept has become popular both in industries and in business management schools. But the term BPR is sometimes used in different senses: it may represent either the cases of minor process improvements or the radical changes in both business processes and organisational structure. Conventionally the process of BPR includes a description of the business activities and the deliverables involved. These deliverables are usually represented in the form of business models which focus on the organisation's structure and dynamics, together with the well-defined and preconceived process for the development of the new business models.

The following summarises some characteristics of BPR that give a preliminary view of how EM is appropriate as an approach to BPR:

- **Process Orientation** (from structure to process): The process orientation of BPR changes the perspective from *structural* relationships between two hierarchical levels of an organisation to the *interaction* processes between individuals and between departments. That is, the principal concern of BPR is to facilitate radical change at process level rather than through organisational functions[2]. One motivation for this is that people in business have started to think "how to develop *new* products or services *effectively* rather than to produce *old* products or services *efficiently*". This explains why the framework presented in this thesis is aimed at the creative development of innovative systems rather than at providing the structured steps and methodical transformation of models between different phases that are characteristic of a conventional development lifecycle. As we shall discuss in chapter 4, EM modelling aims to achieve *effectiveness* – measured by the overall outcome of such processes rather than the speed or immediate results during the process – for the whole development.

- **Customer Orientation**: BPR is radically customer-oriented because the original purpose of BPR is to get closer to customers. That is, the newly redesigned processes should not only sup-

2. The dramatic development of IT has enabled many completely new modes of operation in both business and industry. As a result, many companies are moving toward combination not division of human labour. Thus the previously divided tasks are today being re-unified into coherent business processes.

port the organisation's objectives, but must also satisfy the customers' requirements. But the human factors inside the business should also be taken into account. This broadly includes human, organisational, cultural and political issues. It has been determined that one of the main reasons for BPR failure is the neglect of the human element – most BPR approaches take much account of the scale of change but fail to consider such change through people. As Peltu et al. (1996) comment, "failure to give priority to human factors at a time of radical change can break an existing social contract within an organisation". Thus it is essential that the relevant human agents should be integrated into and involved in the process of redesign.

- **Holistic View of Process** instead of Piecemeal Engineering: Because of its process-oriented characteristics, BPR takes a holistic view of the network of cross-functional processes within an organisation. Such a holistic view can overcome the problem of piecemeal engineering of isolated parts (such as one department) of a business which often results in solutions which are sub-optimal and only appropriate for that part[3]. Such problems closely relate to the issues of systems thinking and reductionism which will be further discussed in chapter 2. As BPR, whether interpreted as continuous improvement or radical innovation, aims at redesigning processes to reduce cost and time and to increase customers' satisfaction and organisational flexibility, a holistic and deep understanding of the business is needed to identify the significant issues. In the thesis, we propose an approach – similar in spirit to the systems approach to be described in chapter 2 – that takes a holistic view of the many different influences on BPR. This is crucial in enabling us to define the relationships amongst IT, human agents and the organisation, and to analyse the nature of these entities and their impact upon an organisation, because these can only be assessed in terms of the total impact.

3. Crowe et al. (1996) point out that most existing computer-based support for reengineering was developed from the bottom upwards, as determined by the traditional product life cycle, and applied piecemeal to closed problem areas as they arise.

1.2.2 System Development

Two activities are sometimes distinguished within system development (or software development): requirements engineering (RE) and software engineering. Normally software engineering refers to the whole lifecycle of software development, and requirements engineering is one central activity in that life-cycle (Figure 1.2). Some researchers separate requirements engineering from software engineering as they argue some problems and questions addressed in requirements engineering cannot be tackled as software engineering[4]. Requirements engineering starts from 'goals' expressed by customers and then elaborates a requirements specification (either in the form of documents or models) in which the system to be developed is defined precisely. This requirements specification[5] is then the starting point of the activity of software engineering, which will develop a software system to meet the requirements. Normally there are the following tasks involved in requirements engineering:

- **Elicitation** The analyst collects the information about the customers goals and problem. This can be done through informal (non-technical) ways such as interviews, discussions, observations or study of relevant documents about the organisation.

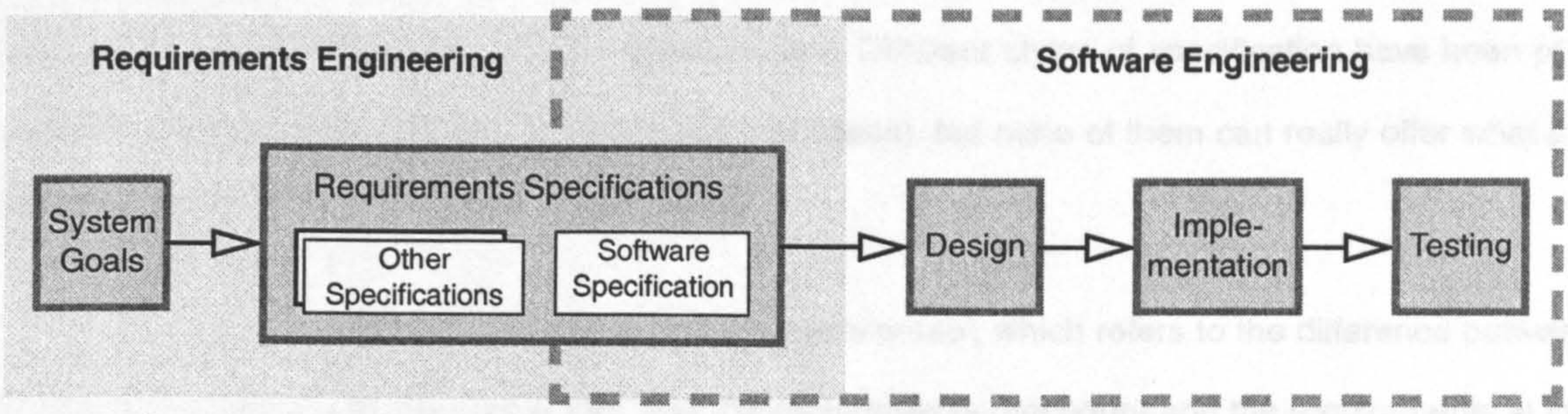


Figure 1.2 Requirements Engineering and Software Engineering

4. This is not the main issue discussed in this thesis. Relevant information can be found in many RE books, *Requirement Engineering* journal, IEEE conferences of *International Symposia on RE* and *International Conferences on RE*, IFIP Working Group (IFIP 2.9), etc.
5. Or precisely, software specification. Because the requirements specification also contains other non-software specifications, for example hardware specifications. Thus in Figure 1.2 the boundary of software engineering only includes the component of software specification in the requirements specifications.

- **Modelling** After collecting the relevant information about the customers' requirements, the analyst will construct models as the representation of the problem domain. Then the analyst studies the models he has built with a view to detecting problems such as inconsistency or the problem of integrating new models with the rest of requirements.
- **Validation** If the analyst detects any problems, he needs to communicate these problems to customers and try to solve them. If no problems are detected, the analyst will formalise the descriptions of requirements and show them to customers in an appropriate way.

Thus the tasks in requirements engineering comprise the interaction between the customers and the analyst in order to progressively clarify the requirements (i.e. in elicitation and validation). But it is obvious that the customers are not explicitly involved in the modelling task. This may lead to discrepancies between the formally described specifications of requirements, and the imprecise, incomplete and even inconsistent goals and wishes of customers in their minds. Further the transition from requirements to design may be more difficult for the end users to follow due to the changes in representation. Also through Figure 1.2, we can find that the customers' goals cannot be directly mapped into the tasks in software engineering (i.e. design, implementation and testing) because customers are not involved in these processes and thus their requirements were abstracted during the modelling in requirements engineering[6]. Various forms of formal modelling methodologies have been developed to support the analysis and verification of the software specifications. Different styles of specification have been proposed (functional, object-oriented or textual like use cases), but none of them can really offer what the customers can use to express their requirements.

Another potential problem is so-called 'requirements creep', which refers to the difference between the requirements specification developed after the requirements procedure and the requirements at the time when the actual product is built (cf. requirement engineering through 'theory-building' in (Loomes and Jones, 1998)). There are many reasons why the problem of requirements creep occurs: the stake-

6. Another aspect of this problem is that we can view the traditional approaches for requirements as relying on a 'snapshot' of a specific state (or at a specific time). The state of the requirements specification only mirrors the real-world state at that time as well as containing other information related to the past.

holders change their minds about the functionality of the proposed system, or (as frequently happen) new or modifying requirements emerge during the processes of design, implementation and testing[7]. It is commonly agreed that this problem is a significant source of excessive cost and time in system development. In today's system development, there is much iteration (e.g. in the Spiral Model) in the refinement process due to the misunderstanding or misinterpretation of the requirements.

The third problem is that the requirements engineering focuses on the system functionality which measures what should be provided by the proposed system in terms of functionalities and services. But the analyst cannot broadly check the adequacy of that system against the constraints within an organisation which are related to business rules or to the way the allocation of the work is organised. That is, the requirements specifications only specifies 'what' the system will do at a high level of abstraction and does not explain the reasons behind that choice of system.

To sum up, better tools and open-ended modelling artefacts are needed to support the communication and negotiation, and the requirements refinement process. The development of an appropriate modelling processes to support the whole lifecycle of software development and maintenance is one of the key issues in this research. It is the aim of this thesis to bridge the gaps mentioned through the concept of 'participative process modelling' which will be detailed in chapter 6.

1.3 Thesis Outline

This thesis draws extensively on the concepts and principles of Empirical Modelling in its approach to system development and BPR. The thesis is divided into eight chapters, of which this introductory chapter gives a background to the thesis. The remainder of this thesis is organised as follows.

7. Davis and Hsia (1994) also give the similar reasons to this problem: (1) Needs are often impossible to realise until after a system is built. (2) Even when requirements are stated up front, it is likely that they will change immediately after deployment. (3) The time between the requirements phase and product delivery is usually too long to pinpoint how specific requirements engineering techniques contributed to a product's success or failure.

Chapter 2 discusses the general concepts of system development. It starts from the concept of 'system' and the systems approach which have affected the thinking in various subjects including the development of computer systems. Then how a computer system is developed and what problems the designers always face when developing the system are discussed. The origins and key concepts of object-oriented (OO) methods are also described in this chapter. This includes the difference in analysis and design between OO and conventionally structured methods, the claims and problems of OO methods, and the influence of OO on system development. At the end of the chapter, the focus is on the concept of circumscription and the details of the link between circumscribing and programming. This includes the discussion of two paradigms: closed-world and open development, and an account of evolutionary design. The discussion of the problems associated with circumscription foreshadows the advantages of developing systems using the EM paradigm, i.e. through the construction of computer-based artefacts to support the open development. This will be further discussed in chapters 5 and 6.

Chapter 3 presents a background of BPR. It provides a survey of the related work in the field of BPR and discusses the problems met during these BPR projects. It is commonly agreed that BPR is important but also problematic for business. Thus it is necessary to identify the factors affecting the successes and failures of BPR, before commencing the reengineering project or proposing an approach to reengineering the processes. The problems facing BPR discussed here are divided into two main categories: the role of IT in BPR and those associated with human factors affecting the success of BPR. The discussion of the role of IT concludes that the development of appropriate IT systems is crucial for the success of BPR work; whereas the discussion of human factors will give strong support to our proposed approach to BPR, which emphasises the significance of user participation in the process of BPR. Furthermore, as the failures of BPR projects are partly due to the use of inappropriate applications or the poor design of supporting computer systems, this chapter investigates the relationship between system development and BPR. This includes the relationship between IT[8] and the organisation, and the deficiencies of conventional system development for BPR.

8. The systems described in the chapter include the existing systems, the future (proposed) systems and the legacy systems which increasingly raise problems in today's business.

Chapter 4 introduces Empirical Modelling as a human-centred and situated computer-based approach to modelling which is being developed at the University of Warwick. It reviews and illustrates the basic concepts and principles of Empirical Modelling which form the basis of this thesis. In addition to introducing the essential concepts, this chapter also describe the construction of EM models -- the interactive situation models (ISMs), and the notations and software tools developed in this group to support EM for open-ended development. Finally the modelling process of EM, and its key features, are discussed. The fundamental comparison between the EM process and the phase-based processes of other modelling methods is also given in the last section. On the basis of the discussion in this chapter, the following chapters will consider the use of EM as an approach to BPR. By further comparing EM with object-oriented methodologies, the advantages and limitations of using EM can be made clear. This comparison will be discussed in chapter 5.

Chapter 5 compares the EM approach with object-orientation and with Jacobson's use case approach. The aim of these comparisons is to give insight into how EM differs from other development methodologies in character. However, it is still necessary and important to find an appropriate framework for comparison because these activities are quite different in nature, e.g. modelling vs programming or system development. Firstly in this chapter a summary of the use-case related works, such as Jacobson's OOSE and Object Advantage (Jacobson et al., 1992 and 1995), is presented. This also includes the discussion of the potential difficulties when using use cases in the development of software systems. Then a comparison is made based on two concerns: the first, the different natures of their development processes, involving the construction of the models; and the second, the different natures of the artefacts they employ, viz. the LSD specification, visualisation and animation in EM, and the structure of the artefacts in OO methods (e.g. models built using Unified Modelling Language (UML)). Even though these two concerns are compared separately, they are in fact closely inter-related: the processes and activities of EM and OO are essentially sequences of modellers' situated actions performed on artefacts. That is to say, the character of the processes and other aspects of activities are determined by the nature of artefacts (Ness, 1997).

Chapter 6 brings together the previous two chapters and is the core of the thesis. It considers how Empirical Modelling can be used as an approach to system development and BPR. For developing useful systems, the EM approach is based on the alternative views of the notions of computer, program and programming that are outlined in (Ness, 1997): the *computer* is viewed as *artefact* rather than just an electronic device; *programming* is viewed as *configuring systems* rather than just as software development; and a *program* is viewed as a *system configuration* rather than just the stored program. These concepts and those in chapter 4 together highlight the importance of user participation in the modelling process. The concepts of *participative process modelling* and *participative BPR* introduced in this chapter focus on the activities with which relevant users as well as modellers (both are participants) are involved when employing the approach. This chapter mainly presents the heuristics which guide the modellers through the process of EM. Examples from the case studies in the next chapter are provided to illustrate such heuristics.

Chapter 7 presents examples for the case studies. Two examples are used to illustrate the features of the EM approach as it applies in two different fields: the digital watch example for system development; and the warehouse management system for BPR. Though the examples described here are quite simple, they are significant to illustrate the principles of EM application suggested in chapter 6, and their investigation raises a wide range of challenging issues.

This thesis ends in chapter 8 by drawing conclusions from the discussions and results in previous chapters. It brings together the major findings and summarises the contributions of the thesis. Then it continues with the discussion of the limitations of the present research and proposes suggestions for future research work in the areas of system development and business process reengineering.

CHAPTER TWO

System Development Re-Considered

This chapter starts from the concept of 'system' and the systems approach developed in 1960s which affected the thinking about computer system development. Then it turns to how a system is to be developed and the problems the developers, including software developers, always face when designing systems. The origins and key concepts of object-oriented methods will come next and how such ideas affect the process of software system development will be discussed. Finally the focus will be on the concept of circumscription, and in detailing the linkage between circumscribing and programming. Two paradigms, closed world and open development, which are related to this topic especially to the circumscription of knowledge, will also be discussed. This final section will also include the idea of 'evolutionary design' which is closely related to the EM approach. As this chapter covers work throughout the 1980s to date (the first three sections), and re-examines the issues in relation to circumscription (the last section), we call this 'system development re-considered'.

2.1 System Ideas and System Development

The idea of 'system' is commonly used in an informal sense as an abstraction from collections of real-world objects or human activities in the world[1], for example, the 'software system' or 'education system'. In this section we focus on the subject of 'systems': systems thinking, systems theory and system devel-

1. According to Harrington (1991), we talk of systems when referring to the completeness of a particular concept or to a process (e.g. "you cannot fight the system").

opment, then how systems theory and the use of systems ideas have been applied in organisations. Checkland's SSM is one example of an off-shoot of such work where systems thinking is a way of thinking about complex situations. Also we will try to find out how such systems theory and systems thinking have been applied significantly to computers and to the design process of software.

2.1.1 The Concept of 'System'

Today in technology we have been led to think not in terms of single machines but in terms of 'systems', especially as the more and more widespread use of computer systems has led to the increasing use of the term 'system'. Jordan (1968) lists fifteen different definitions of the term 'system' which cover a wide range of subjects, such as methodology, biology, physical and social sciences. But he concludes that

> a thing is called a system to identify the unique mode by means of which it is seen. We call a thing a system when we wish to express the fact that the thing is perceived/conceived as consisting of a set of elements, of parts, that are connected to each other by at least one discriminable, distinguishing principle. (p. 52)

This is the so-called 'core meaning' of 'system' and he comments that all these fifteen definitions are concrete exemplifications of this core meaning[2]. Jordan suggests that the term 'system' is used as an interaction between what is 'out there' and how we organise it 'in here', i.e. the interaction between the real world and how people observe or think about it[3].

Checkland (1993) says that a system has these properties: boundaries, inputs and outputs, components, structures, the ways of retaining its integrity, and "the coherency principle which makes it defensible to describe the system as a system". His discussion of systems leads to a new way of thinking about the world (i.e. systems thinking) based on which Soft Systems Methodology (SSM) was developed. The works of Boulding (1956) and Jordan (1968) were also the attempts at observing the world in systems

2. Checkland (1993) points out that the overall argument of Jordan's paper is that there is a core meaning of 'system' which makes it proper to attach it to many different things perceived in the real world outside ourselves.
3. To put it more simply, we use the term 'system' to describe to another person a set of elements and the nature of their relationships, so that the person can conceive of the set as a single, compound entity.

terms. The thinking of Checkland's SSM tries to find out how such systems ideas can help to tackle the unstructured problems which the conventional reductionist methods failed to address. Before further discussing the details of systems thinking in SSM, we should have a broad insight into systems theory and the approach of system development in general.

Systems Thinking

The concepts of 'system' have brought a big change in our views of the world, from the *mechanistic* view of the world by Descartes and Newton to a *holistic* view (Capra, 1996), that is, seeing the world as an integrated [functional] whole and not simply a collection of parts[4]. This holistic view in the twentieth century has led to a new way of thinking: *systems thinking*.

During the 1950s and 1960s, systems thinking had a strong influence on both engineering and management sciences, where systems concepts were applied to cope with *complexity* and solve practical problems[5]. Because the situations of business and science became more and more complex, scientists and managers had to concern themselves with not only the large numbers of individual components but also with the effects of the interactions of those components. Thus many scientists and managers began to analyse their strategies and methodologies using systems concepts. As Checkland (1993) describes:

> The systems engineer must also be capable of predicting the emergent properties of the system, those properties, that is, which are possessed by the system but not its parts. (p. 129)

Conventional Cartesian mechanistic thinking regards the world as a *machine* which can be understood completely by analysing its parts, i.e. the behaviour of the whole can be understood entirely from the

4. The emphasis on the parts is called mechanistic, reductionist or atomistic; the emphasis on the whole is called holistic, organismic or ecological (Capra, 1996).
5. In relation to complexity, Simon (1996) infers three 'eruptions' each refer to a different aspect of complexity: "The post-WWI interest in complexity, focusing on the claims that the whole transcends the sum of the parts, was strongly anti-reductionist in flavour. The post-WWII outburst was rather neutral on the issue of reductionism, focusing on the roles of feedback and homeostasis (self-stabilisation) in maintaining complex systems. The current interest in complexity focuses mainly on mechanisms that create and sustain complexity and on analytic tools for describing and analysing it". (pp. 169-170)

properties of its parts. Thus the method of *analytic thinking* (promoted by René Descartes) for coping with complexity is to break complex phenomena into different parts in order to understand the behaviour of the whole from the properties of its parts. In contrast, the main difference in a holistic perspective is the shift of emphasis from the parts to the whole. That is, the nature of the whole is different from, and even more than, the sum of its parts. Systems thinking embraces a holistic perspective but also emphasises *interaction* rather than just wholeness[6]. According to the systems view, the essential properties of a system are the properties of the whole and none of its parts have such properties, because such properties arise from the interactions and relationships among the parts. The properties of the parts are not *intrinsic* properties because the latter can only be understood within the context (or organisation) of the whole. In differentiating systems thinking from reductionist analysis, Capra (1996) characterises systems thinking as *contextual thinking* or *environmental thinking*:

> Analysis means taking something apart in order to understand it; systems thinking means putting it into the context of a larger whole. (p. 30)

The characteristic of systems comes from the properties of the whole (for example, the functionality or the purpose), which in turn arise from the configuration of relationships among the parts. The configuration of relationships is called a *pattern* which is also the focus of systems thinking in *cybernetics*[7]. Capra emphasises the importance of the idea of pattern as "the understanding of life begins with the understanding of pattern". Systemic properties are properties of patterns, "what is destroyed when a living organism is dissected is its pattern". The reductionist theories fail to cope with the issues of pattern because pattern is non-material and irreducible.

Checkland suggests that systems thinking is based on two pairs of core ideas: *emergence and hierarchy*, and *communication and control*. This shows another criterion of systems thinking: the ability to

6. Harrington (1991) characterises the holistic view as a perspective; whereas the systems view is a methodology.
7. The attention of pattern of organisation is the explicit focus of cybernetics (Simon, 1996; Capra, 1996), even though the development of cybernetics was independent of the one of general systems theory (both after World War II). Norbert Wiener, who inspired and invented the name of cybernetics, expanded the concept of pattern from the patterns of communication and control which are common to animals and machines to the general idea of pattern as a key characteristic of life (Capra, 1996).

shift our attention back and forth between systems levels (Capra, 1996). That is, in the systems view, different systems levels represent different levels of complexity and at each level the properties of the observed phenomena will not exist at lower levels. Such systemic properties of a particular level are called 'emergent' properties because they emerge at that particular level (Checkland, 1993; Capra, 1996). Further in the systems view, every 'object' is regarded as a network of relationships. Thus when we perceive objects as a network of relationships, what we observe are also an interconnected network of concepts in which *there are no foundations*. That is, the material world is a dynamic web of interrelated events and no properties of any part is fundamental. Such a multi-levelled structure is called a 'hierarchy'[8].

The concept of *scientific objectivity* is also influenced by systems thinking, as Capra (1996) describes:

> In the Cartesian paradigm, scientific descriptions are believed to be objective, I.e. independent of the human observer and the process of knowing. The new paradigm [of systems thinking] implies that epistemology – understanding of the process of knowing – has to be included explicitly in the description of natural phenomena. (p. 39)

Because in a systems view the world is regarded as an interconnected web of relationships, the configurations of relationships (the patterns) are identified depending on different observers, methods and the processes of observation. Thus systems thinking means a shift from objective to 'epistemic' science (Capra, 1996); and the method of questioning is the main focus of systems theory. As Capra says: "what we observe is not nature itself, but nature exposed to our method of questioning". In addition, that there are no foundations in the web of nature leads to the insight of approximate knowledge, i.e. this recognises that all the scientific concepts and theories are limited and approximate, and can never provide a complete understanding. This is in contrast to the Cartesian paradigm which asserts the certainty of scientific knowledge. Thus, according to Capra's view, in science, we can only make limited and approximate descriptions of the described phenomenon.

---

8. Capra calls it 'the web of life' instead of a hierarchy to avoid misleading since the latter is derived from human hierarchies.

Systems Theories

Capra (1996) points out that systems thinking emerged simultaneously in several disciplines during the first half of the twentieth century, especially during the 1920s. The emergence of system theory, as Boulding (1956) describes it, came because of the increasing need for systematic theoretical constructs which can discuss the general relationships of the world, especially an increasing number of hybrid disciplines which connect with many different subjects[9]. Boulding presented the idea of *General Systems Theory*[10] (GST) which was used to "describe a level of theoretical model-building which lies between the highly generalised constructions of pure mathematics and the specific theories of the specialised disciplines". Conventional theories such as mathematics attempted to organise general relationships into a system, but a system may not have such connections to the real world. GST considers all 'thinkable relationships' abstracted from the concrete situation or knowledge of experience, and tries to make a connection between "the specific that has no meaning and the general that has no content". Based on the systems concepts that the world is a web of interrelated events and has no foundations for such a network of concepts, Boulding presents a preliminary hierarchy of real-world complexity. He suggests that the use of such a hierarchy can reveal the gaps in knowledge. He also argues we should never accept as final "a level of theoretical analysis which is below the level of the empirical world which we are investigating". This is why he puts in the title that GST is the skeleton of science because

> it aims to provide a framework or structure of systems on which to hang the flesh and blood of particular disciplines and particular subject matters in an orderly and coherent corpus of knowledge. (p. 208)

Earlier, von Bertalanffy (1968) introduced *General System Theory*[11]. By regarding the living organism as a whole, rather than simply a set of components with relationships between components, von Berta-

---

9. Boulding gives some examples such as cybernetics arising from electrical engineering, neurophysiology, physics, biology and economics; organisation theory arising from economics, sociology, engineering and physiology.
10. The name is slightly different to von Bertalanffy's (1968) *General System Theory* but Boulding attributes many of his ideas of GST to von Bertalanffy.
11. An earlier discussion of von Bertalanffy's ideas can be traced back to his paper: "General System Theory: A New Approach to Unity of Science", in *Human Biology*, Vol. 23, December 1951, pp. 303-361 (referred from Boulding, 1956).

lanffy shows an important distinction between systems which are *open*[12] to their environment and those which are *closed*. He emphasises that systems thinking is *process thinking* and based on this he formulated a theory of 'open systems' which combined the various concepts of systems thinking and organismic biology into the theory of living systems. As living systems have a wide range of shapes in phenomena, such as organisms, social systems or ecosystems, von Bertalanffy believes that a GST can offer the conceptual framework for unifying different disciplines[13]:

> General system theory should be ... an important means of controlling and instigating the transfer of principles from one field to another, and it will no longer be necessary to duplicate or triplicate the discovery of the same principle in different fields isolated from each other. At the same time, by formulating exact criteria, general system theory will guard against superficial analogies which are useless in science. (pp. 80-81)

Thus GST is regarded as a meta-disciplinary theory which abstracts from the special properties of specific disciplines and aims to provide a common language and theory for people in different disciplines to express and solve the problems of many different disciplines, especially with regard to the 'interdisciplinary movement'.

Systems Approach

There are two kinds of approach: the 'scientific approach' comes from science, just as 'systems approach' comes from the systems concepts. Checkland (1993) classifies both the scientific approach and the systems approach as meta-disciplines and both have a particular way to regard the world:

> The scientific outlook assumes that the world is characterised by natural phenomena which are ordered and regular ... and this has led to an effective way of finding out about the regularities – the so-called 'laws of Nature'. The systems outlook, accepting the

12. He calls such systems 'open' because they need to feed on a continual flux of matter and energy from their environment to stay alive.
13. According to Capra, before the 1940s the terms 'system' and 'systems thinking' had been used by several scientists, but it was von Bertalanffy's concepts of an open system and a general systems theory which established systems thinking as a major scientific movement. Checkland (1993) calls him the founder of the 'systems movement'.

> basic propositions of science, for it is a part of the scientific tradition, assumes that the world contains structured wholes ... which can maintain their identity under a certain range of conditions and which exhibit certain general principles of 'wholeness'. (p. 6)

The scientific approach involved reductionism, which tried to emphasise reducing the observed situation in order to increase the chance of experiment obtaining some observations which are reproducible. This approach assumes that the components of the whole are the same when examined singly, and the principles of assembling these components into the whole are also straightforward (von Bertalanffy, 1968; Checkland, 1993; Capra, 1996). These assumptions may be reasonable in physical science. But when moving to more complex phenomena such as social phenomena, it will be difficult to answer how to make the separation of the whole into 'components' or whether it is right to make such a separation. These can be also illustrated by the concepts of *restricted science* and *unrestricted science*. Physical science or chemistry is restricted science where a limited range of phenomena are needed to study, [reductionist] experiments in the laboratory are possible, and hypotheses which are expressed mathematically can be tested by quantitative measurements[14] (Checkland, 1993). By way of contrast, in an unrestricted science such as biology or social systems, the effects are so complex that experiments cannot be controlled easily or be precise. Quantitative models of physical science cannot be appropriately applied here and the effects of unknown factors (mainly from interaction and dependency of elements in systems) on our observations are also high.

The systems approach[15] aims to cope with the problems in unrestricted science by observing the world in a 'system way', i.e. as a set of systems. The following is a comment on the systems approach by one researcher in 1950s:

> Systems, of course, have been studied for centuries, but something new has been added. ... The tendency to study systems as an entity rather than as a conglomeration of parts is consistent with the tendency in contemporary science no longer to isolate phe-

14.Thus the greater the precision of the quantitative predictions and the more successful the experiments, the greater the confidence in the hypothesis.

15.Another reaction against the scientific approach is 'ethnomethodology' which reconciles a radical empiricism with the situatedness of social data, by looking at how members of a group actually organise their behaviour (Goguen, 1994 and 1996).

> nomena in narrowly confined contexts, but rather to open interactions for examination and to examine larger and larger slices of nature. (Ackoff, 1959)

Checkland (1993) argues that the process of the systems approach is similar to the one which occurs in human thinking, and the way we observe in systems approach is a world-view or *Weltanschauung*. "We attribute *meaning* to the observed activity by relating it to a larger image we supply from our minds". The phenomena we observe (i.e. the essential properties) are only meaningful in the context of the whole: the *Weltanschauung*. As systems thinking has brought a new way of thinking in many scientific disciplines, many new concepts have been developed from the systems approach, even in restricted sciences in which conventionally reductionist experiment and quantitative measurements were thought to be enough to find out the principles. Examples include the quantum theory in physics. Conventional physicists believed that all physical phenomena can be reduced to some properties of elements (the material particles). However in the 1920s the quantum theory showed the fact that such particles can further dissolve at the subatomic level into wave-like patterns of probabilities (Capra, 1996). These patterns represent the probabilities of interconnections but not of the things. In such a view, our world is not regarded as a set of building-blocks, but as a complex web of relationships between the parts of a whole. Capra sums up the influence of systems concepts on quantum physics:

> Whereas in classical mechanics the properties and behaviour of the parts determine those of the whole, the situation is reversed in quantum mechanics: it is the whole that determines the behaviour of the parts. (p. 31)

Systems Classes

There are many similarities between the two systems ontologies[16] of Checkland (1993) and of von Bertalanffy (1968). The following summarises Checkland's systems classes, which are mainly influenced by Boulding's hierarchy and Jordan's taxonomy[17] in their attempts to survey the whole of the real world in systems terms.

16.According to von Bertalanffy, 'systems ontology' refers to the kinds of system that there can be.

17.By analysing these works, Checkland justifies that the four is the absolute minimum number of systems classes need to describe the whole of reality.

Natural Systems Checkland defines natural systems as the systems "whose origin is in the origin of the universe and which are as they are as a result of the forces and processes which characterise this universe". He further says that natural systems are evolution-made and irreducible wholes[18].

Designed Physical Systems These are, the systems which are man-made and are the result of human conscious design[19]. Both natural systems and designed physical systems can be categorised as *real systems* in von Bertalanffy's term which means "entities perceived in or inferred from observation, and existing independently of an observer". The main difference between natural and designed physical systems is their origins. Designed physical systems exist because they are needed for the human activity system (which will be described later).

Designed Abstract Systems This is similar to *conceptual systems* in von Bertalanffy, which means the ordered conscious products (symbolic constructs) of the human mind, such as logic, mathematics, poems or philosophies. Checkland argues that "they will exist as a result of a positive act related to some objective – elucidation, the enlargement of knowledge, or an inner urge to express the inexpressible".

Human Activity Systems The concept of human activity systems is that of a number of human activities related to each other which can be viewed as a whole. Checkland argues that the fact these activities form a human activity system is emphasised by the existence of other systems (often designed systems) which are associated with them[20]. The main difference between natural systems and human activity systems is that in the latter we cannot establish *full* public knowledge because the entities perceived are not independent of an observer, as in natural systems[21]. Thus reductionist methods which are suitable in restricted science may not be appropriate for unrestricted science.

18.For example, according to Checkland (1993), carbon dioxide is not reducible to carbon and oxygen, we can even find its interatomic distance and bond angles.

19.Sometimes we call them *artificial systems* because they are designed to achieve some human purposes.

20.These "other systems" take on a quite different character when they are viewed as part of a human activity system, because of the impact of the human intervention.

21.That is, according to Checkland, that human free will (i.e human freedom of choice in selecting actions) means that the principles and logic of natural systems cannot be totally applied to human activity systems.

'Social Systems' People in a group (i.e. a social system) have their responsibilities as well as having some expectations of other members. Checkland suggests social systems can be placed between natural systems and human activity systems. Because social systems observed in the real world should be a mixture of a rational assembly of activities (the human activity system) and a set of relationships (the natural system). The important aspect of social systems is that human beings are components which are active participants in social phenomena. As with human activity systems, we cannot observe and explain human activities in the same way as natural systems. Also we cannot predict what will happen as such systems consist of both intended and unintended effects or they may even be influenced by the growth of human knowledge that is unpredictable as well[22]. By contrast with finding 'laws of Nature' in natural phenomena by scientific approaches, in social systems we can only reveal the *trend* rather than *laws*, and reduce the observed subject only to what Checkland refers to as the *logic of situations* rather than social reality[23].

Checkland's Soft Systems Methodology

Today systems thinking has been the paradigm of thinking holistically and has much influenced the later development of systems ideas as well as the development of the use of systems ideas in particular disciplines. SSM, as Checkland describes, is the combination of these two developments and aims to see if systems ideas can help to tackle the problems of 'management'[24] (the broader term for social phenomena). He summarises four reasons for using the systems concept in management problems:

> Firstly, many problems are seen to recur when they are looked at as problems of systems; hence solutions may be transferred. ... Secondly, the systems view enables problem-solving to concentrate on the *processes* by which things are done, rather than on (*ad hoc*) 'final outcomes'. Thirdly, systems may provide the objective standard by which problems can be organised for solution. ... From objective standards we may be able to

22.Checkland also points out that not only complexity in social systems but also the non-availability of experimental objects will be problems.

23.Over a long time this logic of situations will increasingly change because of the growth of human knowledge and humans being involved.

24.That is, the real-world problems or the problems of decision in social systems.

> gain greater insight to generalise on business phenomena. Fourthly, ... many business problems are 'mixed', that is they have both qualitative and quantitative attributes. Systems analysis is the newest technique to be brought to these mixed problems. (Checkland, 1993, p. 147)

The early experiences of SSM in trying to apply the systems approach outside technical areas can be found in his book *Systems Thinking, Systems Practice*[25] (*STSP*, Checkland, 1993). In connecting with the four systems classes mentioned above, Checkland (1993) summarises the following would-be problem solving approach:

> Firstly, there will be much to *learn* from natural systems. ... Secondly, the problem solver is free to *use* designed systems, whether physical or abstract, to achieve his ends. And thirdly, ... since human beings are purposeful, that we may seek to 'engineer' human activity systems, using the word 'engineer' in its broadest sense. (p. 125)

The starting point of SSM is to consider an organisation as a system with functional sub-systems (such as production, finance, human resources, etc). The real-world experiences can enable us to build the knowledge of these [sub-]systems and support a better design and operation of systems in real situations. Similar to systems thinking, the process of SSM is a *learning cycle* by which the tentative ideas can be applied to inform the practice which then becomes the source of richer ideas. Then these ideas and the ways of using such ideas can be extended as a result of practical experiences. The development of SSM as a learning process is based on the following key thoughts (Checkland, 1999):

- Every situation in the research is a human situation in which people are attempting to take *purposeful* action which is meaningful for them. This thought leads to the idea of modelling purposeful human activity systems: the sets of human activity "which together exhibit the emergent property of purposefulness".
- Because of the complexity of human activity systems, there may exist many models for systems interpreted for different purposes[26]. Checkland suggests that the first choice of models should be

25.'Systems practice' here implies a way to find out how to use the systems concept to solve problems.

the one that is the most relevant in exploring the situation. Then we decide the perspective or view-point for each activity, i.e. the *Weltanschauung*, based on which model will be built.

- Similar to the various interpretations of purpose, we may find many models to represent the situation. This makes SSM different to conventional systems engineering. SSM thus is regarded as an inquiring process as it is not working with the 'obvious' problem to find out solutions. In contrast, it works with the idea of a *situation* which people may regard as problematical for various reasons. SSM in this case is regarded as an organised *learning system*.

Checkland emphasises that the term 'system' in SSM is not applied to the world but instead applied to the *process* of dealing with the world[27]. This is different to a conventional approach of systems engineering which regards the world as a set of interacting systems, some of which can be engineered to make them work better. This also frames the distinction between two kinds of systems thinking: the latter is 'hard systems thinking' and SSM is 'soft systems thinking'. Their difference is illustrated in Figure 2.1. Harrington (1991) says that the hard approach views a system as a machine with definite and identifiable input/output processes, and the system here is defined as a way of getting things done. Therefore this approach emphasises the 'doing' aspect of systems methodology and thus on the physical domain. In contrast, the soft approach is organically-oriented because many aspects or elements, which make up the system, are not easily controlled and defined, and the interactions among them depend on circumstances. It is not only linked with the physical domain but also the perceptual domain. Wholeness and relationships are important because the soft system only has meaning as a whole, i.e. the parts have meaning only when relating to the whole. As the interaction of parts is essential for the systems as a whole, the emphasis of the soft approach is on the maintenance of stability. To summarise, the hard approach is prescriptive analysis whereas the soft approach is descriptive analysis (Harrington, 1991).

---

26. When modelling purposeful activity, many interpretations of 'purpose' can be possible. Checkland (1999) takes the Concorde project as the example. It is commonly regarded as an *engineering* project to produce supersonic aircraft (technical world-view). But it is also treated as a *political* project because at its origins President de Gaulle of France was vetoing the British entry into the European Common Market and this project is the result of negotiation (political world-view).
27. This idea is similar to von Bertalanffy's as he emphasised systems thinking as process thinking.

In the learning process associated with SSM, the models used are *devices* for helping to structure an exploration of the problem situation. This is different from the models used in hard systems thinking as they represent the parts of the world outside ourselves. Checkland (1999) makes these comments on the models in SSM:

> They are not models *of* ... anything; they are models *relevant to debate* about the situation perceived as problematical. They are simply devices to stimulate, feed and structure that debate. (p. A21)

To conclude, SSM presents a more complex interpretation which does not readily give us solutions. It attempts to describe, so as it enables us better to understand the situation rather than prescribe.

Computer 'Systems' and Software 'Systems'

Today computers are argued to be more complex than most things people build because of their large numbers of *states* (Brooks, 1987). This makes the conceiving and developing of computers become dif-

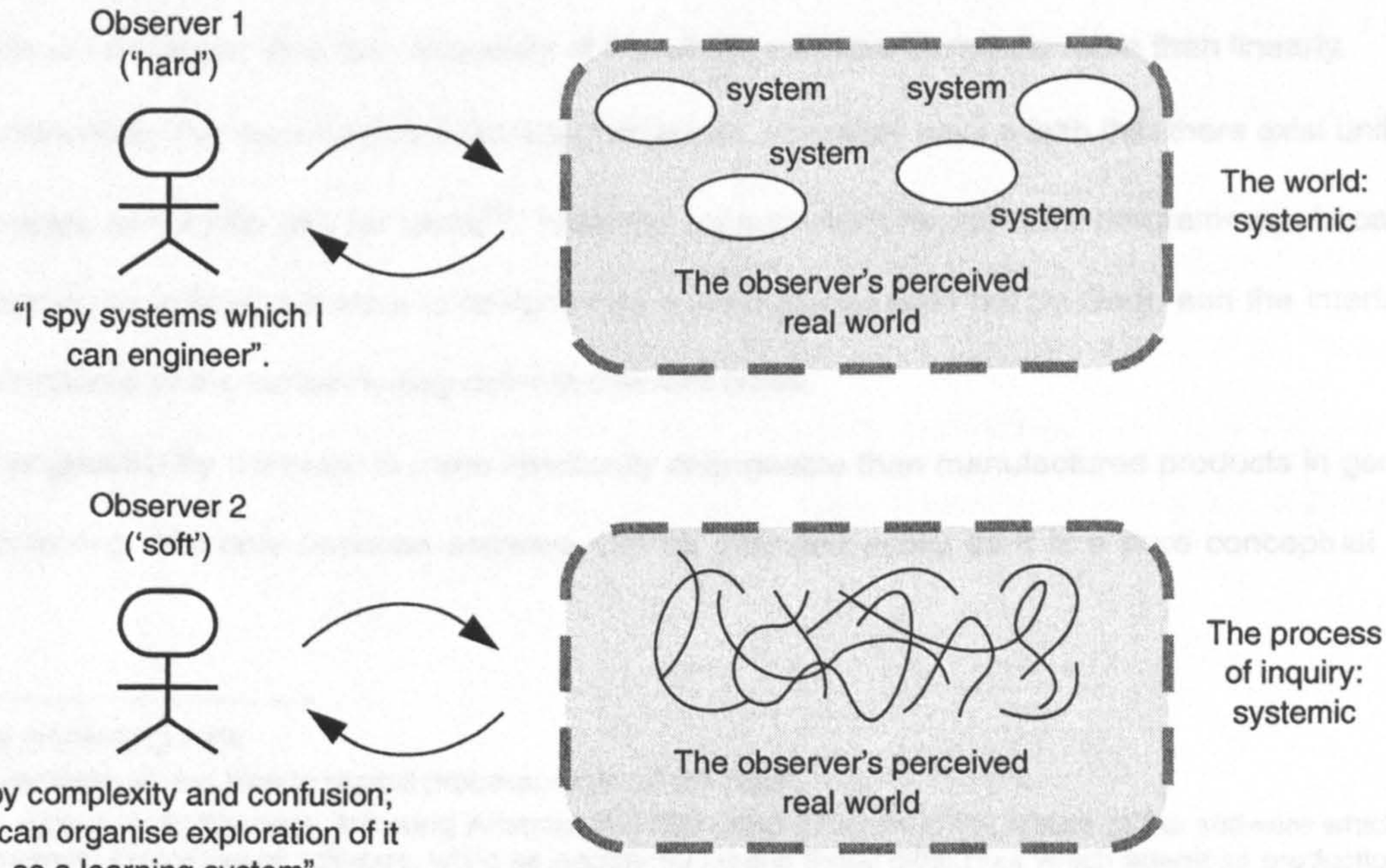


Figure 2.1 The Hard and Soft Systems Stances (adapted from Checkland, 1999)

ficult. However, the model 1401, announced by IBM in October 1959, was not so much a computer as a computer *system*. At that time many computer companies were obsessed with the designing of CPUs[28] and thus did not consider the system to be designed and its external context as a whole[29] (Campbell-Kelly and Aspray, 1996). However IBM recognised that customers were more interested in the solution to business problems rather than a solution for computer design. Thus they started on the total-system view of computers, i.e. the systems thinking, which considered programming, customer transition, service, logistics, etc. This change of thinking, as Campbell-Kelly and Aspray comment, brought the success of IBM 650 and was the factor that most differentiated IBM from its competitors.

It is commonly recognised by most software programmers that the greatest difficulty of writing software is that the programmers have to anticipate all possible outcomes of computers. Brooks (1987) considers the difficult part of writing software is the development of the conceptual construct. He points out four essential[30] difficulties in software technology:

- **Complexity** The complexity of software comes from the increasing numbers of different elements, not only merely repeating the same elements to large size. As the interaction of such elements is non-linear, thus the complexity of the whole software increases more than linearly.
- **Conformity** For conventional [restricted] sciences, scientists have a faith that there exist unifying principles and which can be found[31]. However no such faith for software programmers because, according to Brooks, software is designed by human beings (and not by God), and the interfaces or structures of the software may differ at different times.
- **Changeability** Software is more constantly changeable than manufactured products in general engineering. Not only because software can be changed easily as it is a pure conceptual con-

28.Central processing units.

29.It was common at that time to regard processors as 'all the rage'.

30.Brooks names *essential* here, following Aristotle, the difficulties inherent in the nature of the software which are also essential properties of software, while as *accidental* means those difficulties which attend its production but are not inherent.

31.Brooks reports that "Einstein repeatedly argued that there must be simplified explanations of nature, because God is not capricious or arbitrary".

struct, but also the function or behaviour of a system is decided by software, and generally system function is the most changeable part because of some pressures[32].

- **Invisibility** Software is invisible. Thus it is hard to find a proper representation for it. Diagrams are commonly used to represent software structure, but sometimes we need several diagrams to represent different aspects of software such as dependency or time sequence. Brooks makes the criticism that structures of software still remain inherently unvisualisable which "not only impedes the process of design within one mind, it severely hinders communication among minds".

The use of the term 'software systems' may be traced back to the 1960s, when the software programmers found the difficulty of exploiting the potential of the rapidly developing hardware and the difficulty of managing big projects which were going disastrously wrong. The conference entitled 'Software Engineering' held in Garmisch, Germany in October 1968[33] marked the first time software development was regarded as a engineering discipline[34]. From this point on, systems thinking had a prominent influence on software development. Due to the difficulties of software construction mentioned above, people found that systems concepts could be applied with conceptual integrity to cope with the complexity in problems, especially those whose solution involves the combination of software and hardware. As the requirements of software escalated rapidly, programs became 'software systems' (Beynon and Russ, 1994). Today, software plays such a critical role in organisations that the term 'software systems' is commonly used in both engineering and management, and has even become a discipline. The introduction of systems concepts in software due to the development of the software and the design of organisational structures are mutually dependent. They cannot be developed in isolation and also they affect each other. Warboys et al. (1999) conclude that "a software system in its real world domain of application can be thought of as a system within a system".

32.Such as users wanting to extend its functions or due to new devices of computers.

33.Campbell-Kelly and Aspray (1996) say that "the real importance of the meeting was to act as a kind of encounter group for senior figures in the world of software to trade war stories".

34.Brooks (1995) comments that the goal of applying engineering principles to software production in the 1970s was to increase the quality, testability, stability and predictability of software products, not necessarily the efficiency of software production.

2.1.2 Software System Development

Software system development is a process in which system developers design, implement and maintain the software and models of computer systems[35]. Traditionally computers were designed to replace some other devices or skilled operators for the sake of efficiency and effectiveness. In business, computers were developed to analyse and automate the existing data-flows and databases which means running the business processes that were paper-based before. Harel (1992) calls this 'one-person programming' because such requirements are known in advance and are straightforward relative to the construction of software. Structured methods for system development were first developed in the 1970s which gave systems developers useful ways to build effective requirements during the late 1970s and 1980s. However as the requirements escalated, the problems of the *software crisis* emerged because systems took too long and cost too much to develop, and even after that they were not working as we expected. Furthermore, there exists a problem between the rapidly growing software code and the shortfall of personnel to maintain the code. According to O'Callaghan (1994), maintenance costs for legacy systems are reaching 90% for the total cost of software. Thus other methodologies, such as object-orientation, to software development were advocated to deal with the software crisis.

The Systems Boundary

It is important to define the system boundary while developing a system. The boundary is defined by Checkland (1993, p. 312) as "the area within which the *decision-making process* of the system has power to makes things happen, or prevent them from happening", or more generally, "a distinction made by an observer which marks the difference between an entity he takes to be a *system* and its *environment*". Generally inside the boundary are the system itself and other elements the designers intend to control; outside the boundary is the environment with which the system will interact. Regnell (1999) designates the former the *target systems* and the latter the *host system*. However it is not easy to define

35.We use the term "system development" here to mean "software system development" which focuses on the functionality or behaviour of computer systems, i.e. on *what* the system does rather than *how efficiently* it performs.

the system boundary as we need to decide which elements will be either within the system or a part of the environment. This is a issue of *circumscription* which will be focused on later in this chapter.

Cook and Daniels (1994) identify three categories of software system boundary: (1) *hard* system: the system boundary is explicit and the functionality or behaviour of the system is totally dependent on its software; (2) *semi-hard* system: the introduction of software will not change the behaviour of the whole system, but the role of software depends on non-technical factors such as cost or political issues; (3) *soft* system: the introduction of software will consequently impact the system and the environment in unpredictable ways, thus later we need to change the software in order to reflect and incorporate such change. We may regard the systems developed in a conventional way as 'hard' systems because their boundary is defined in advance and the system behaviour is perceived as totally dependent on software. Systems developed in the EM way may be regarded as 'soft' systems because their boundary is not explicitly fixed and is changing all the time. Thus the behaviour of systems and their interaction cannot be predicted and this is why we are focusing on observation and experiments in the EM models. 'Semi-hard' systems are more related to information systems as we would like to automate some operations and processes in an organisation but hope this will not impact the overall organisational operation. Thus traditionally the introduction of information systems into organisations is on a piecemeal basis and we assume that the software does not affect the basic structure of the organisations. However in many cases the introduction of information systems are changing the overall organisational processes, and this is the main issue BPR is focusing on. This issue will be further discussed in section 3.3.

System Development Process

Morris et al. (1996) suggest that system development during the 1970s was moving towards *engineering* techniques rather than *programming* or *management* techniques. That is, it was based on graphical models which represented various views of the proposed system ('engineering techniques'), rather than carried out by the programmer without any external reference ('programming techniques') or supported

only with textual specifications and documentation ('management techniques'). Warboys et al. (1999) consider the term 'engineering' from a more general viewpoint not confined to technical sciences:

> *Engineering* is the application of scientific principles to the solution of real-world problems. It is not about finding the truth, it is about intervening in the real world with artificial constructs intended to ameliorate some condition otherwise affecting people. It is about making best of the knowledge that we have, albeit imperfect, incomplete and uncertain, and finding safe, pragmatic and economic solutions within known constraints. ... The engineering of processes is a way of solving problems arising in the context of an organisation, in particular those organisational problems whose solution lies in exploiting the use of coordinative and integrative technologies. (p. 55)

In relating to systems thinking, the way we observe the world is a process of questioning and through which the limited and approximate knowledge is obtained[36]. In summary, engineering as systems approach relates to problem-solving via the construction of models (i.e. the artificial artefacts). The process of engineering is mainly based on personal experiential knowledge acquired from much trial and error, learning from observing which solutions work or fail. Each solution will be the model for the next stage. The process of engineering may also change people's behaviour. Sometimes we say *system engineering* or *software engineering* which relates to software development mainly focusing on solving problems in software design. In general the process of software development consists of three essential stages:

- **Requirements:** requirements elicitation, capture and approval (*what* functionality/behaviour the system should have);
- **Design:** study the requirements and design a solution (*how* the specific functionality, defined in requirements, to be built);
- **Implementation:** implementation, coding, testing and maintenance.

Figure 2.2 shows an integrated view of software development process which starts from requirements elicitation and analysis, through design, implementation and testing phases, finally to validation and cer-

36.Because there are no foundations in the web of our life.

tification before releasing the system. Regnell (1999) says that the development process starts with the external view of the system, where the system functionalities are defined in requirements specification. It then continues to the design and implementation stages which focus more on internal system issues, and finally it returns to the external view for validation. The external view (or *black-box* view) focuses on external functionality as observed by the users of the system. The internal view (*white-box* view) focuses on system architecture as well as the structural (and behavioural) properties of elements in the system[37]. There are also two views for software testing: black-box testing means testing system functionalities whereas white-box testing refers to structural testing. When the system has been implemented, its function is determined by its program.

With reference to Checkland's characterisation of scientific and systems approaches, there are a number of reasons why the conventional system development process can be categorised as a scientific approach. It is based on the subdivision of the development process (i.e. requirements, design and

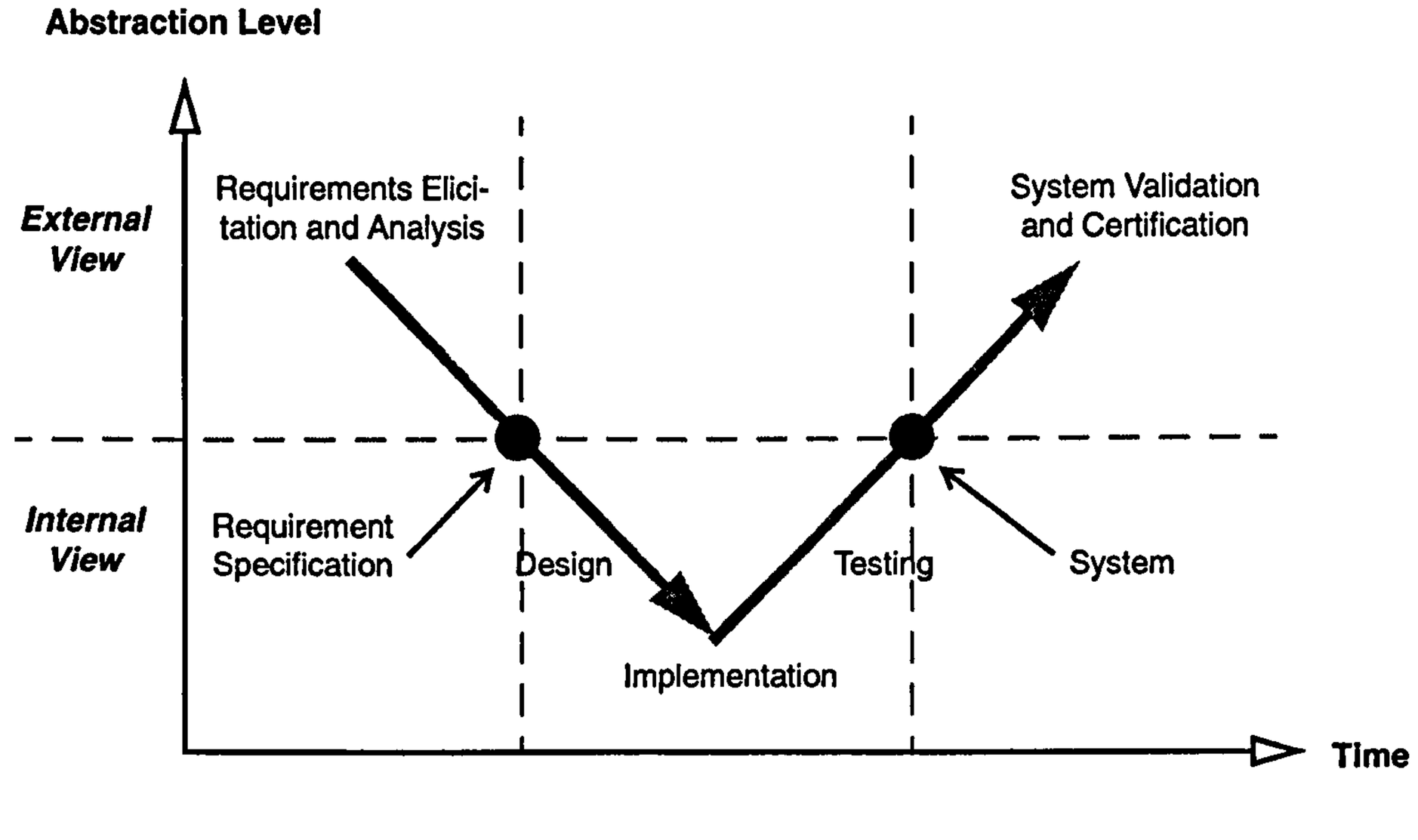


Figure 2.2 Different Abstraction Levels over Time (adapted from Regnell, 1999)

37.Checkland (1993) has a similar viewpoint in SSM – he classifies the perspectives of the observer on the behaviour of systems in two ways: either he can concentrate exclusively on the inputs and outputs and in this case treat the systems as 'black-box'; or he can describe the internal state of the systems in terms of variables, or the path of state changing influenced by external conditions.

implementation). It can also be viewed as a linear-phased process, even if sometimes there is an iteration across these phases for enhancing specifications until systems are released. On this basis, both the sequential 'waterfall' approach and the iterative 'spiral' approach will be categorised as scientific approaches. The development process is fragmented into different procedures and the completion of each phase is prerequisite to the beginning of the next one. The system to be developed is viewed as a machine with definite and identifiable input and output processes. The emphasis of such approaches is on the 'doing' aspect (i.e. getting things done) and the domain is mainly the physical domain[38]. One of the reasons, as described by Loomes and Nehaniv (2001), that software engineering has been dominated by such a lifecycle model of design is that "the design problems are still formulated with respect to a single, reified, pre-supposed system, referred to, already before the fact, as 'the system'". In their discussion of the reification of 'the system'[39], they point out that:

> Within this [life-cycle] model, there is the assumption that 'the system' exists in some embryonic from the very start of the process, and that the task of the designer is to nurture development and allow maturity to be reached in good health. (p. 27)

The common feature of these approaches is assuming the development is a one-time effort (Lehman, 1991) and the modification and enhancement to the systems are treated as a new or different development process. This may result in high maintenance costs and inflexible systems which are difficult to adapt to changes in the dynamic business environment. And the sequential subdividing process in the development may restrict the designer's response to dynamic situations. Further the systems development, as well as modification and enhancement, may take a long time and be difficult to carry out because of the 'frozen' functionality of the systems. The same task may be classifiable in two different phases, e.g. the software changes in testing phase may not be different from the changes in the maintenance phase. Such problems have led many companies to adopt a non-structured approach, such as

38. This is in contrast to the systems approach in which the aspects (or elements) or a system are not easily controlled and defined (because the interactions among them depend on circumstances). The emphasis is on the maintenance of stability (cf. SSM) and the domain includes both the physical and perceptional domain.
39. They say that concepts of 'the system' are treated as an entity with a single identity whose existence is tacitly accepted and validated in unspoken implicit assumptions. For example, 'the system' as discussed by designers, the representation or prototypes of 'the system' in its development, 'the system' as a physical, deployed entity, or the maintained or modified versions of 'the system'.

object-orientation, in their system development. But even in object-orientation, the development and maintenance are still treated as two different activities.

Structured Analysis and Design

Structured methods for system development were widely used during the 1970s and 1980s. They took the reductionist view that the best way to manage complexity was to limit the designer's view of the domain. That is, a program should be modular (i.e. divided into many small parts) so that designers could write a module with little knowledge of the code of other modules. The designers begin with an overview of software artefact and, when this has been satisfactorily developed, their attentions shift to a lower level of detail. The behaviour model (or functional model) for the proposed systems is built via a decomposition of functionalities which leads to a network of concurrent processes. The claim is that modularisation is a mechanism for improving the flexibility and comprehensibility of systems and shortening the time of development[40]. Structured methods focus on the processing aspect because traditionally people believe that the systems are designed to perform work. Structured methods, reflecting a reductionist view, make a distinction between data and processing, which enables the designers to observe the problem in terms of three dimensions: the processing, the information structure and the time sequence[41], and then integrate these views into whole systems. Some programming languages, such as COBOL, C or Pascal, promote/encourage such distinction.

There are two main paradigms for system development: *waterfall model* (Figure 2.3) and *spiral model* (Figure 2.4). The waterfall model decomposes the developing process into sequential phases, and the sub-process of each phase is totally dependent on the results of the previous phase. O'Callaghan (1994) says that because the structured analysis decomposes the functionality of proposed systems into small ones, waterfall model is thus the main paradigm used in these conventional methods.

40.Parnas (1996) asserts that the effectiveness of such modularisation depends on the criteria of dividing the system into modules.

41.Sully (1993) gives some examples of tools focused on each dimension: the data flow diagram (DFD) addressed the processing; the entity relationship diagram (ERD) addressed the information structure; and the state transition diagram (STD) addressed the sequence behaviour.

He further argues that the main problem in the waterfall model is its application-centred character. That is, the overall functionality of the application has to be defined firstly in order to define other small modules. Also it may become difficult to define and analyse the functionality for complex systems, and if errors made in the early stage, it will be very expensive if such errors are not found until all the software has been developed. Spiral model may sort out such problems but still have some disadvantages when applied to structured methods.

System development using structured methodologies raises some problems. Structured analysis defines the requirements from three dimensions (processing, information and time) from which it may be easy for designers to construct specifications. However systems developed in this way may be resistant to change and extension. The reason is such methods focus on the *processing* rather than the *concepts* (or objects) of the system (Sully, 1993). Sully criticises that processing of a system is the most volatile aspect of a system which may change rapidly over time. In contrast object-orientation focuses on 'concept' and aims to build systems more resilient under change[42]. Other potential problems may occur from transformation between requirements specification to implementation. As Morris et al.

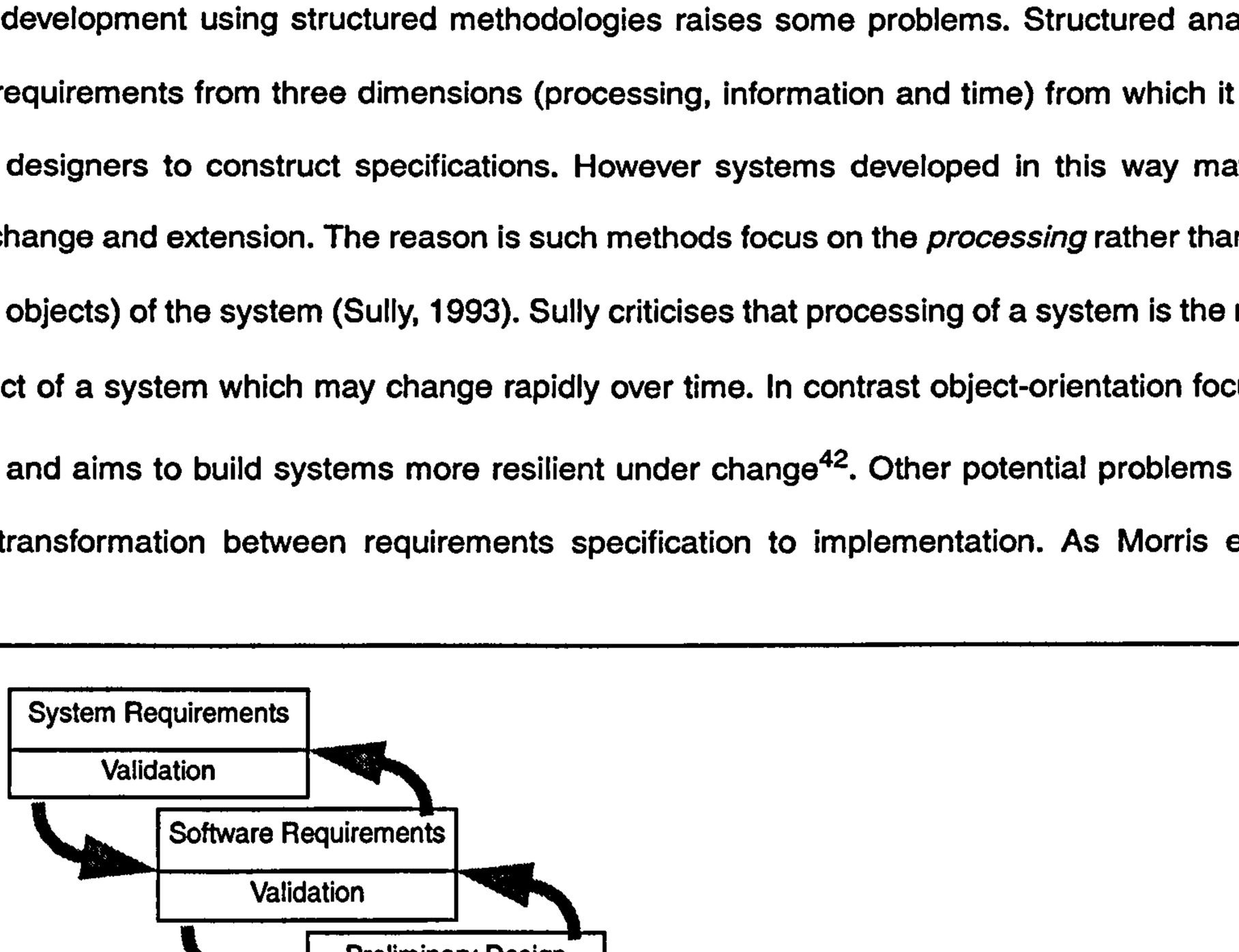


Figure 2.3 The Waterfall Model Lifecycle
(adapted from Boehm, 1976)

(1996) describe, the weakness of all structured methods is that the effort of analysis may be left behind, just when it has reached the peak of its development, and the development is going into the implementation stage. That is, a new structure will be defined in implementation which tries to map the products of earlier analysis to some executable functions. Error is likely to occur during this transformation and it will become worse if such error occurs at later stages[43].

2.2 Object-Orientation

The object-oriented languages have been around since mid-1960s and became popular in academic research in the 1980s. But the realisation of object-orientation in both industry and business has started from the 1990s due to the reduction of memory cost and increasing processing power. Another reason is that systems developed by OO methods have shown themselves to be more resilient under change. Today object-orientation applies not only to tools and a way of thinking in computer programming, but its

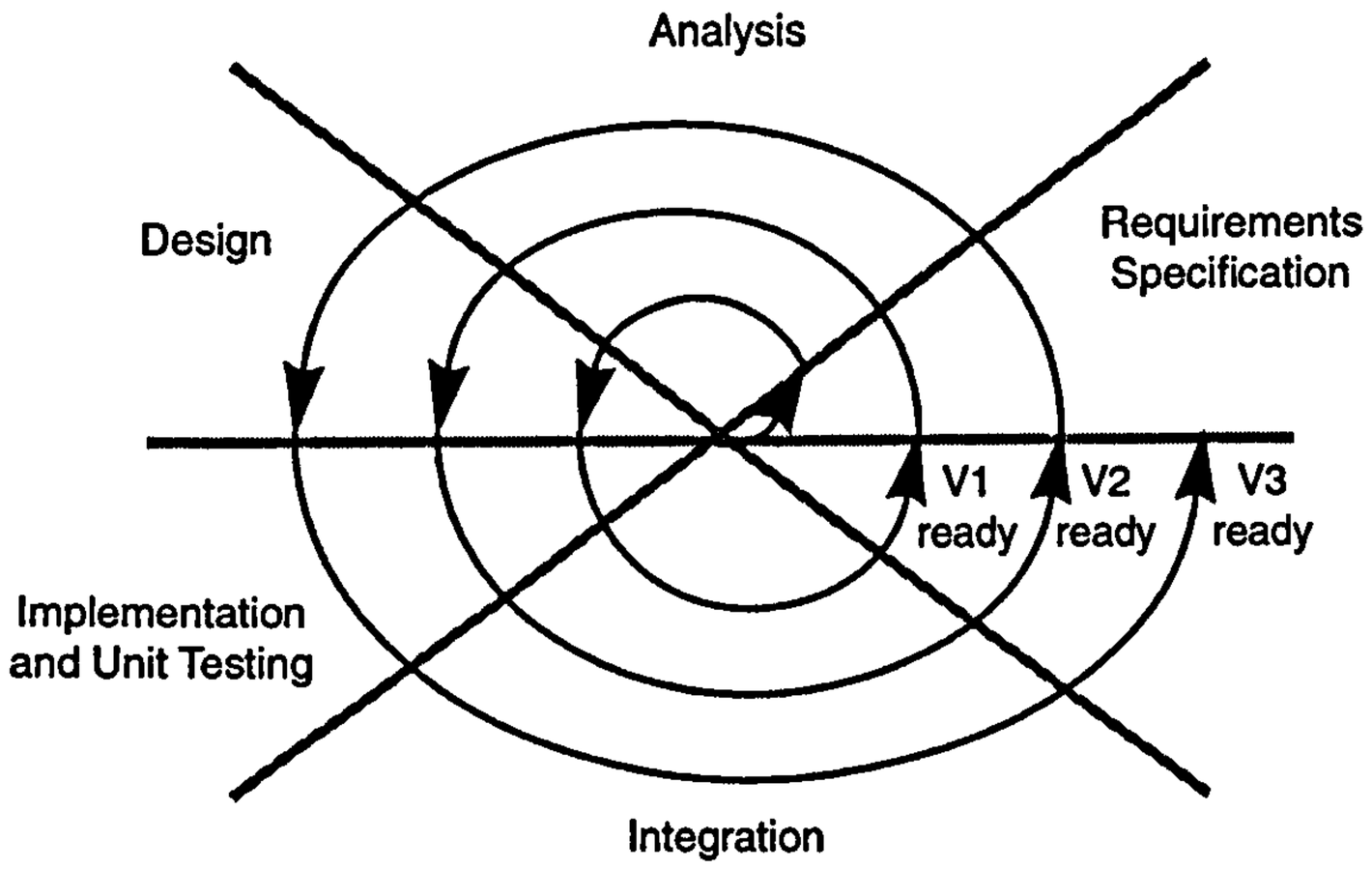


Figure 2.4 The Spiral Model Lifecycle (adapted from Boehm, 1981)

42.In OO, all the processing units are organised around the object concept, and the detailed design of processes can be done by implementing objects.

43.OO methods try to avoid such problems by ensuring that the objects identified in analysis phase will evolve to the objects of implementation.

concepts are also commonly used in the information processing industry (i.e. information technology) and BPR. This section summarises the general issues of object-orientation and describes the influence of object-orientation in system development.

2.2.1 The Origins and Key Concepts of Object-Orientation

It is widely accepted that the original ideas of *objects* came from Simula which is regarded as the first object-oriented language (Sully, 1993; Graham, 1994; Cook and Daniels, 1994). Simula first introduced the concept of *classes* and *inheritance* to programming languages and the main purpose for which Simula was designed was expressing discrete-event simulations. The influence of Simula in object-oriented languages can be seen today in the way that OO models aspire to mimic the real world and to map the behaviour of parts of the world. As Sully (1993) remarks, OO solutions are viewed as simulations of the real world, and this principle was 'discovered' when using Simula. These core motivations for object-oriented modelling are also represented in the EM approach to be introduced in chapter 4. When we perceive solutions via computer models which mimic the real world, the language we use can reflect our mental model of proposed systems. When we simulate real world subjects, we can *observe* the effects of different parameters rather than *calculate* them[44].

Object-orientation became popular both in industry and business and widely discussed in academia in the 1990s. The reasons can be summarised as following:

- **Software Crisis** The software crisis is the problem discussed in subsection 2.1.2 which emerged because of the rapidly growing volume of software and the shortfall of development personnel to maintain the code. Further it becomes more and more difficult to meet the user's requirements in software quality as conventionally we only highlight the expectations of system functionality. OO is claimed to be a good solution for the crisis because it provides an environment

44.For example in simulation we can observe the car behaviour when passing an intersection via queue lengths and time delay.

for higher productivity and makes software more reusable, which requires less effort for development and maintenance.

- **'Natural' Abstraction** The term *natural abstraction* (O'Callaghan, 1994) means a higher level of abstraction which can be regarded as a common 'language' between users and designers. Thus the natural abstraction can map easily from the user's domain to the designer's domain. The concept of objects can be the high level of abstraction through which designers can structure their concepts easily and use such abstraction to model the problems. At the same time, designers and users can 'talk' to each other in this common 'language'. O'Callaghan concludes that "as the structuring concepts are chosen from the *problem domain* rather than imposed on the problem from the domain of computer science", this makes OO more natural than structured methods.
- **Reuse of System Components** It is desirable that systems should be developed as economically as possible with the minimum duplication of effort due to the software crisis. For this reason a library of reusable components is required. The objects in OO which bind the processing and data in a unit can be the reusable components of system development. Also the *design patterns* for software architecture are another example of reuse in system development. They constitute a 'grass roots' effort to build on the collective experience of skilled people (Buschmann et al., 1996). As such experts have already developed solutions to recurring design problems, using these design patterns can help to base the development on proven solutions of standard subproblems.

Sully (1993) identifies three main characteristics of object-oriented language: *object-based* means the OO language has the facility for encapsulation and to represent object identity[45]; *class-based* means the facility for offering an object-based unit and abstraction; *object-oriented* means the facility to offer class-based units inheritance, self-recursion and a higher degree of object character. The term *encapsulation* – that data and the operations performed on it are combined together (encapsulated) into objects – means that OO can provide a boundary around the object which can hide the complexity (i.e data and methods) by limited visibility and information hiding[46]. This can help designers to concentrate

45.Gammack (1995) comments that two concepts, encapsulated objects and messaging, suffice for a system to be called object-based.

on the conceptual view of objects without considering how to implement the code or other physical realisation. The *inheritance* mechanism in OO languages allows the specification of subclasses to inherit data from the higher-level class. In this way the designers have more flexibility where they need only to recognise the general and the specific parts then add the extra behaviour.

The Unified Modelling Language (UML) is the synthesis of object design languages. UML merges Booch notation, Object Modelling Technique (OMT) and Object-Oriented Software Engineering (OOSE) and was released in 1997 by Object Management Group (OMG) for a standard object-oriented methodology[47]. UML includes a set of consistent diagrams which can be used to describe the systems requirements, design and implementation. It seems to be more complete than other OO methods in support for the modelling of complex systems.

The Difference between Structured and OO Methods

Object-orientation can be viewed as initiating a totally new paradigm for programming compared with structured methods. In general, OO methods are *structural* and *problem-oriented* approaches whereas structured methods are *functional* and *solution-oriented* approaches (Graham, 1994; Morris et al., 1996). The following summarises some issues raised while discussing the difference between OO and structured methods.

- **The Paradigm Shift** The first difference is in the perception of designers when they start to develop the software. Through OO methods their thinking is on how to design the objects which can be reused many times, in comparison to the structured methods where their thinking is on how to design the specifics for each function of the proposed system. Thus OO methods are problem-oriented because designers need to understand the problems well in order to design suitable objects. It is also reuse-oriented because at first the designer thinks of the generalised units and

46.Or we can say each object in the system is acting as a 'black box' which only showing limited visibility (but not internal details) to external observers. The object defines a visible interface which declares to other objects (i.e. its client objects that use it) what operations on the hidden data are recognised.

47.It was submitted by Rational's Grady Booch, Jim Rumbaugh and Ivar Jacobson, as well as other major companies which include i-Logix, Digital, HP, ICON Computing, Microsoft, MCI Systemhouse, Oracle, Texas Instruments and Unysis.

based on these many subsequent applications can be developed. Table 2.1 shows three differences between these two paradigms.

Old Paradigm	Object-Oriented Paradigm
Separate data from processing.	Integrate data and processing.
Build distinct applications from specifications.	Place each part at its correct place within the whole.
Design processing by top-down functional decomposition.	Design processing within the context of an object model.

Table 2.1 A Comparison of Paradigms (from Cook, 1994)

- **Modules versus Objects** The concepts of modules and objects may be similar but their characters are quite different. In structured methods, the main function of proposed systems is decomposed into many sub-functions which are allocated into modules. Thus modules are interdependent on each other and most of them are not usable outside the system. The main function of the system is built up by these modules in a hierarchical structure. It may take a short time to develop such modules initially but maintenance may become difficult to control and too expensive in large systems. In contrast, the objects in OO methods can localise change within one unit by information hiding[48] and so promote reuse. The design of suitable objects may take more time initially because it involves thinking about the use of objects in many contexts, but later it will become easier and more effective to use objects. In this way the object-orientation can encourage us to take a wider view of system development. Figure 2.5 shows the different efficiencies among unstructured, structured and OO developments along with the lifecycle of software development and maintenance.
- **Scientific Approach versus Systems Approach** We have already discussed the scientific approach and systems approach in subsection 2.1.1 (under 'Systems Approach"), and focused on their different ways of problem solving. OO is problem-oriented because designers have to

48.O'Callaghan (1994) views the OO systems as a network of co-operating 'virtual machines' because the behaviour of each machine is defined by its protocol of operations, rather than just a single, subdivided machine with the main function at the top.

understand the problems (i.e. the whole context) in order to design reusable objects. Thus the OO development is facilitated by a whole approach and Sully (1993) also has a similar viewpoint. He points out that the key difference between OO and structured approaches is that the designer should consider all the enterprise concepts which are likely to be involved in order to define objects for the application. In conclusion we may regard the OO approach as a kind of systems approach because we take an enterprise-wide view of system development. This is in contrast to structured methods as scientific approach because its emphasis is on the decomposition of system functionality and the construction of modules – a set of independent system components, and its development is generally a linear-phased process. As each module is to perform a single function, any change to that function requires a change in only one, identifiable, part of the system (Crowe et al., 1996).

2.2.2 The Claims and Problems of Object-Orientation

Object-orientation is claimed to be the most natural way to understand and model the problem domain in which a system will operate. The whole idea of OO is that the object is an abstraction which more closely mimics the real world than conventional structures. Further, the encapsulation of process and

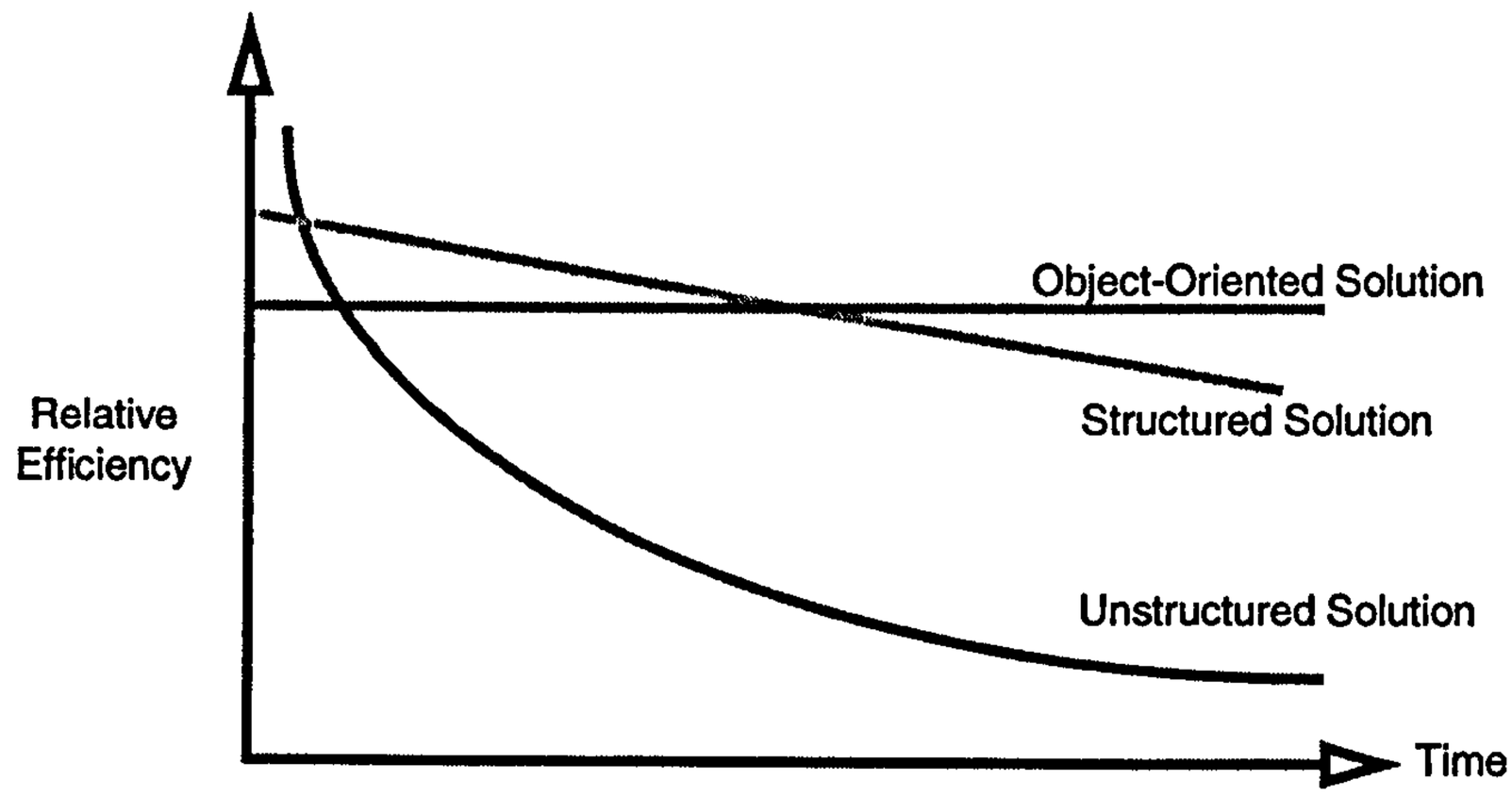


Figure 2.5 The Relative Efficiencies of Unstructured, Structured and Object-Oriented Development (adapted from Sully, 1993)

data into objects take place around the domain rather than processing. This makes OO systems being more resilient to change as the concept (or data) is less volatile with time. Other claims of OO include increasing productivity, reducing cycle time, reducing the maintenance task and leading to more reliable and efficient systems. All of these are due to the reuse of proven components (i.e. the objects).

While discussing the difference between OO and structured methods, we can find some potential advantages with object-orientation[49]. For example, the paradigm shift leads to economy in applying OO in system building where the designers are 'assembling' a system by prefabricated parts which are stored in libraries. The comparison between modules and objects can also lead to economy as the average size of objects is typically smaller than that of modules. Further we also regard OO as a 'natural' abstraction and as a common 'language' between users and developers. This will enable users to be more active participants in system development. Finally Cook and Daniels (1994) suggest that one important strength of OO is its ability to integrate heterogeneous systems. It can be predicted that in the future the focus on a particular programming language will become less and less when systems are constructed from parts written in different languages, which will communicate via language-independent structures such as CORBA[50].

Object-orientation has attracted more and more attention in the field of computer science and information technology. One claim is that the designer can use objects as a uniform approach throughout the developing process, this makes the transition from OOA to OOD easy and smooth (Morris et al., 1996). That is, it is assumed the objects identified in the problem domain (analysis) have the same meaning in the solution domain (design). However some researchers (e.g. Kaindl, 1999; O'Callaghan, 1994; Morris et al., 1996) argue this it is difficult for OO to move directly from OOA to OOD because their objects represent different things. To summarise these researchers' points, OOA objects are objects in a *problem* domain which are commonly-held abstractions for real-world things; whereas OOD objects are objects in *solution* space which are specialist abstractions concerning a particular domain of

49.Because of the advantage in object-orientation, we can see many conventional languages have started to grow extensions, such as C and C++, Pascal and Object Pascal, etc.

50.OMG's Common Object Request Broker Architecture.

interest. People can also find these two objects have different characters as they are used in totally different models. OOA model is a model for part of problem domain which is an abstraction of real-world things (both existing or non-existing) and developed before the proposed system is built. OOD model is a model of the internal structure of the proposed system which is an abstraction of a solution in terms of objects for building the system. Thus these models have different purposes: OOA model is to help the analyser to understand the domain and capture requirements; whereas OOD is to help designers to understand the nature of proposed systems. Thus the objects used in OOA and OOD are representing different things and this makes the transition from OOA to OOD difficult and problematic.

There is another potential problem with the claim that OO enables the designer to model the real world closely. There are several reasons (some of which are also applied to structured methods) why it is difficult to accommodate real-world features within a model of software:

- The world is a social system, and we cannot simply apply the principles and methods for investigating natural systems to the social phenomena as we are a part of the social system. Because software is a part of the world, the introduction of software will alter the nature of the social system (i.e. will change the 'soft' system boundary).
- The world is dynamic and unpredictable but a model of software is typically static and predictable because its behaviour is preconceived. This static nature of a model of software means that software system "may actually thwart the dynamism of [the world]" (Warboys et al., 1999).
- OO models use message-passing between objects. As Cook (1994) and Cook and Daniels (1994) observe, this is appropriate for describing software execution, but cannot describe what happens in the world.

Some researchers of SSM (e.g. Stowell, 1995b; Checkland, 1993 and 1999; Checkland and Scholes, 1990; Lewis, 1993 and 1995) conclude that this problem stemmed from the ill-defined philosophy of system development – *ontology* vs *epistemology*. That is, the confusion of treating the status of 'entities' or 'objects' as ontological labels for parts of the real world or epistemological devices through which the

nature of the real world may be explored (Lewis, 1995). For epistemological philosophy they recognise the models might be an epistemological device, as Odell (1992) emphasises:

> Object-oriented (OO) analysis should *not* model reality – rather it should model the way reality is understood by people. The understanding and knowledge of people is the essential component in developing systems. (p. 45)

Thus Lewis concludes that OO analysis is still displaying the same conceptual confusions of structured methods and the same lack of concern for the essential philosophy.

2.2.3 The Influence of Object-Orientation on System Development

The introduction of object-orientation does not only change the programming process for system development, but also it influences people's thinking on software modelling and the software process. Fichman and Kemerer (1997) summarise that the influences of OO include new approaches to analysis and design, greater use of iterative and incremental development, and increased emphasis on component reuse. The influence of OO on designed systems (system development) and on human activity systems (i.e. human factors and organisations) is the subject of the next two subsections.

New Culture of System Development

Meyer (1990) argues that object-orientation is not just a programming style but also an implementation of a new view of what the software is, and this view implies a rethinking of the software process. He summarises the comparison between the two cultures of structured and OO methods in Table 2.2. He names conventional culture as *project culture* because its subject is an individual project; whereas the culture of OO as *component culture* as the subject is reusable components.

The reusability of objects can make the development of systems economic, and localise change and minimise effort during the development and maintenance[51]. One important feature of this reusabil-

51.Further using components from the OO library can also guarantee more performance as their behaviour is prescribed and proven once.

ity is the change in the nature of systems. Traditional system development regards the system to be built as an 'end-product' which is based on a preconceived purpose. Developing systems in this way may ignore some potential contribution the systems may be able to make to the whole development infrastructure. In applying OO in reuse, the systems to be developed are required to be adaptive to further requirements, as is appropriate for the evolutionary ('E-type') systems which will be described later in this chapter. The EM approach to development leads to systems with similar properties that have a flexible behaviour to meet the needs of different people and different environments.

	Project Culture	Component Culture
Outcome	Results	Tools, libraries
Economics	Profit	Investment
Unit	Department	Industry
Time	Short-term	Long-term
Goal	Program	System
Bricks	Program elements	Software components
Strategy	Top-down	Bottom-up
Method	Functional, structured, analysis/ design, entity-relation, dataflow, etc.	Object-oriented
Languages	PDL, C, Pascal, etc.	OOPLs such as Java, C++, Smalltalk, etc.

Table 2.2 A Comparison between Project Culture and Component Culture (from Meyer, 1990)

Human Factors and Organisations Affected by Object-Orientation

One of the claims of OO is that it provides a 'common language' for both users and designers. This enables users to have a more active and participatory role in the development, and designers can have a more active attitude to users and response to new requests efficiently due to the use of pre-existing components. The symbols or diagrams used in OO approach (rather than textual specification in conventional ways) are claimed by Sully (1993) to be better graphical metaphors for effective use, since "there will be the fewest possible concepts to juggle with" and this offers 'cognitive economy' for users. Another potential advantage of this is that people may find it is easier to remember class structures (i.e. by the concepts of abstraction and inheritance in OO).

Even though OO has the potential for increasing reuse, many organisations are still failing to take advantage of this. The following are some comments from managers (during interviews by Fichman and Kemerer, 1997):

> "Reuse is good but comes at a cost. People who have sold OO talk about benefits but not the cost".
>
> "The objects that get reused tend to be generic, not domain specific".
>
> "Not many objects are common across systems. When they are, they have different behaviours".
>
> "By the time you define all the behaviours, the business may have changed".
>
> "It sounds like reuse is the right way to go but it does not happen in practice".

Through the interview, Fichman and Kemerer find three reuse barriers: the lack of motive to build for reuse; no processes or tools to manage the reuse environment; and disagreement about the kind of reuse (large or fine grained). In summary, investing in OO for reuse involves overcoming barriers which "potentially put at risk that weight of investment, and only those who are utterly convinced that the weighing of costs and benefits will be in their favour are prepared to take the risk" (O'Callaghan, 1994).

In addition to applying OO in developing software systems, the concept of objects has also been applied to modelling business processes in an organisation. That is, the configurability of objects is used to enable some changes in business processes (i.e. BPR) in order to achieve more productivity for the organisation and produce more value for its customers. For example, the 'business objects' are designed to simulate corporate procedures as well as translate into software objects while implementing the software system to support the business processes. Using OO methodologies can allow for rapid evolution of business objects in response to business conditions. Thus far there have been many researchers focusing on applying OO methods to BPR. For example the work by Jacobson et al. (1995) on BPR (the Object Advantage) is based on their previous software development methods (i.e. OOSE). The use case driven method is firstly applied in requirements gathering and later on applied to model

business processes. Section 3.3 will give details of some researches focusing on linking system development and BPR.

2.3 Circumscription

Circumscription is commonly used in the development of software systems. For example the circumscription of state by the values of a type and of behaviour by the operations of a type or by rules for state change. There is a close link between circumscribing and programming and system boundary. Today it is commonly agreed that the major benefit can be obtained from the application from development activities before coding, such as problems definition, requirements analysis and systems specification (Lehman, 1989). This leads to the development of modelling methodologies which define systematic processes and techniques for formulation of concept and realisation. They are circumscribed because they provide mechanised support for individual activities but also restrict these activities. This section will summarise the relationship between circumscription and programming. Further the role of circumscription and how knowledge is represented and constructed in different paradigms of modelling will also be described.

2.3.1 The Concept of Circumscription

Most of the traditional approaches of software development are trying to formalise the development process abstractly without reference to the particular characteristics of the product to be developed (Sun, 1999). This is mainly because the subjectivity of human beings can influence the process. Also conventionally it was believed that the correctness of a program can be guaranteed as a consequence of a rigorous (mathematical) development process. Thus most methodologies of software development try to minimise the subjective influence of human being in order to ensure the quality of the proposed systems. These prescriptive methodologies tend to specify particular modelling tools and techniques for representing the systems requirements and architectures, as well as set the fixed patterns of activities with rigid algorithms for guiding and controlling the process and thinking of developers. Such circum-

scription is organised according to a particular background and pre-understanding[52]. For example, the structured methods for system development divide the process into several phases in a linear order (both in waterfall model and in spiral model). In each phase the developers have some proven methods, techniques and tools to follow. Pressman (2000) describes this phase-based development process as 'linear thinking'. The aim of such a circumscribed approach is to design high-quality systems with finite resources in a predictable way. We suggest that these prescriptive methodologies focus on only the micro-level aspects, i.e. the constructional aspects of systems such as algorithms or data structures. This differs from the open development and evolutionary paradigm due to Lehman which focuses on the macro-level (or systems level) aspects relating to system-wide attributes and which will be detailed later in this section. Although a prescribed methodology may not provide the deep understanding of the solution domain, it still leads the designers to examine the solution domain more comprehensively and systematically, and to deal in a more uniform way with potential solutions (Beynon et al., 2000c).

Mathematical Models for Circumscribed Behaviour

There are two kinds of systems behaviour: one is defined by the changes of state that are *immediately experienced* and the other is associated with *preconceived* reliable patterns of state change (Beynon, 1994). Traditional mathematical models are concerned with the latter one, where the context for observation is circumscribed and what is to be observed can be anticipated. Since circumscribed behaviour involves some preconception about what is to be observed, it implicitly involves an expression of faith. Beynon infers that we can often use a mathematical model as a basis for description and prediction of a circumscribed behaviour.

As the world is changing and essentially dynamic, its control and exploitation using computing techniques requires the development of restricted representations, made finite, discrete and static through the development of models (Lehman, 1989). Von Bertalanffy (1968) mentions the advantages of math-

52.Winograd and Flores (1987) explain that in most cases this pre-understanding reflects the rationalistic tradition, which includes biases about objectivity, the nature of 'facts' and their origin, and the role of the individual interacting with the computer.

ematical models: unambiguity, possibility of strict deduction, verifiability by observed data. To build a mathematical model for systems behaviour, we create an abstract representation for the observed phenomenon where the effects of state change can be predicted from theory, and for capturing the attributes which are considered relevant and essential to the problems and proposed solutions. The quality of the model depends entirely on how precisely the actual system's behaviour is reflected in the mathematical model (Beynon, 1994). For example we can ask "Does the mathematical model relate the behaviour of a system to its structure?". The way in which a system is developed will affect its response to some exceptional circumstances and failure conditions. But typically an abstract mathematical model does not consider this. Thus the quality of the mathematical model depends on how successfully the system behaviour has been circumscribed. And the process of circumscribing behaviour represents the abstract patterns of human activity which provide the basis for prediction.

Humans versus Computers

The aim of software development is to help people in the manual process of everyday tasks by automatic or semi-automatic systems. But it is commonly the case that computer systems do not provide the same flexibility as manual processes. For example in the late 1950s and early 1960s, many US banks and insurance companies were seeking to use computers to attempt automation, but soon recognised that it was inappropriate to install computers as electronic clerk-replacements (Lehman, 1997). It is important to compare human manual processing of tasks prior to automation with computer processing of the same tasks after the automation, in order to determine the effect of circumscription on system development.

In the manual process (say business process for example), the human judgements are usually influenced by some factors through their exploratory activity. Thus human beings explore some possibilities of different organisations for the resources and make qualitative judgements. The way in which human beings interact with the physical artefacts in the manual process is far less circumscribed than interaction with computers which are programmed in conventional ways. This feature of manual activity is similar to the scientific experiment prior to the identification of the theory because it enables an open-ended

exploration. But this kind of 'opportunistic extension' of functionality may be hindered by the conventionally constructed software systems, "especially where optimised algorithmic processes have been implemented" (Beynon et al., 2000c). Also as such development of computer systems is based on solutions, Lehman (1989) observes that it becomes increasingly difficult or even impossible to return to the previous way of doing business (i.e. the manual process). That is, people are forced by the computer systems to work according to the archetypal plan, and the short-cuts and flexibility in the manual process are no longer possible. This is because the judgements made by humans concern the relationship between the proposed solution (or abstract data) and its referent in the real world, which involve observations and cannot be preconceived and preprogrammed. A rigid algorithm cannot capture all the human awareness of the situation. In this case the developers of software systems need to preconceive any possibilities of integration of the manual and automated activities. As Lehman describes, the knowledge on which their inference of what is required is based represents an historical view.

It is essential to integrate both the manual and the automatic processes in order to exploit the advantages of automation but also retain the features of human manual interaction with physical artefacts. But thus far current systems designed in conventional ways cannot support such integration. Beynon et al. (2000c) argue that the difficulties of integration at the cognitive level stems from the traditional computational paradigms. For example, the difficulty of transition from OOA to OOD stems mainly from the fact that the programming methods rely on circumscription of the application prior to automation. There also exists a distinction between the treatment of observables by humans and the treatment of data by computers, which makes a difference between the conception/construction of software and its execution (Beynon et al., 2002). For the former (software conception/construction), human imagination and knowledge of the real world is essential, whereas for the latter (software execution), human interaction and interpretation is invoked in preconceived ways. Thus it is concluded by Beynon et al. (2002):

> There is a fundamental mismatch between abstract data that is interpreted by the human in direct association with its counterpart in the real-world referent and situation, and abstract data that is manipulated according to computational rules that can only take account of prespecified and preprogrammed features of this association.

We show in this thesis that the situated character of EM approach can allow the modeller to develop artefacts in an open-ended way and without circumscribing the domain.

2.3.2 Open Development versus Closed World

Brödner (1995) reports that in the history of engineering, there are two assumptions on the nature of humans and the function of technology, on the way of seeing the world, and on the human's being in the world:

> One position, ... the 'closed world' paradigm, suggests that all real-world phenomena, the properties and relations of its objects, can ultimately, and at least in principle, be transformed by human cognition into objectified, explicitly stated, propositional knowledge. ... The counterposition, ... the 'open development' paradigm, does not deny the fundamental human ability to form explicit, conceptual and propositional knowledge, but it contests the completeness of this knowledge. In contrast, it assumes the primary existence of practical experience, a body of tacit knowledge grown with a person's acting in the world. This can be transformed into explicit theoretical knowledge under specific circumstances and to a principally limited extent only. ... Human interaction with the environment, thus, unfolds a dialectic of form and process through which practical experience is partly formalised and objectified as language, tools or machines (that is, form) the use of which, in turn, produces new experience (that is, process) as basis for further objectification. (p. 249)

The *closed world* paradigm is characterised by the tradition of rationalism and logical empiricism, which was the predominant way of seeing the world in the enlightenment age and has been the mainstream of Western science and technology (Brödner, 1995). The assumption of this paradigm is that the human world and human work can be entirely modelled as logical propositions and mathematical relations. In particular with the development of computer systems and cognitive science, the closed world paradigm views the world as a 'system' which is controllable, analysable and formally describable. Thus in software engineering, the results of the closed world paradigm are the systems which offer a preconceived framework for interaction between designers, the models and the external world. Traditional [mathemat-

ical] models suit the close-world culture because once they are created, they are independent of its external referents. This means that their relationship to the referents is defined only at the time when the modelling is undertaken, i.e. they represent the modeller's knowledge and understanding at a particular point of time. The knowledge represented in such models is static and circumscribed. Mathematical models are commonly used in the analysis of problems, because computer-based simulation and requirements analysis are based on developing mathematical models and then implementing these as computer programs. The abstract function of the models is based on patterns of interaction which stem mainly from the modeller's understanding of historical results. Whenever the external referents change, the models have to be revised because there is no automatic link between them. To do this may not be easy as a large number of scenarios need to be evaluated.

However we should notice that the continuously changing environment and uncertainty of human behaviour make the predictability and repeatability of the closed world paradigm unrealistic (Winograd and Flores, 1987; Lehman, 1989; Brödner, 1995; Warboys et al., 1999; Sun, 1999; Beynon et al., 2000c). The problem of software development methodologies within this paradigm is that even though they allow for the incremental evolution of requirements, they assume that there is a point in time at which all developers' views will 'converge' and produce a final view of requirements. Paul (1993) terms this the *fixed point theorem*. Such a view neglects the fact that the real-world domain in which the computer systems exist is dynamic and changing. Also in software development, the constructed computer program is related to its external referent only through the relationships which are intended and preconceived by programmers. The computer may terminate operation within some situations which are unintended by the programmers. It is also not clear to what extent we can preconceive which information about the objects is significant in the process of design, or what exceptional scenarios of use need to be considered. Warboys et al. say that it is often the case that the engineer cannot address all aspects of a problem, but will address that part which is capable of being addressed. But because solutions cannot be found for some aspects of the problem does not mean they are unimportant. As they say, "the aspect which is pre-eminent is the fact that owners of systems have in the past been obliged to set down a set of requirements for their systems without much regard for potential problems". Winograd and Flores

comment that this embodies *blindness,* as the essence of intelligence cannot act appropriately when there are some pre-definitions of the problem or the space of states in which to search for solutions. That is, when we are in the domain we have defined, "we are blind to the context from which it was carved and open to the new possibilities it generates". These new possibilities usually create a new openness for design.

With regard to business processes, Warboys et al. point out that "the human issues centre upon the still-developing debate about whether it is useful to prescribe and enforce a detailed prescription of process through the support system". There are many influences on business processes and in many cases it is not possible to define the process which can cope with all possible influences. For instance, the software system is designed to automate manual processes and achieve the goals of the company, but in the closed-world culture the system may become unfit as it cannot be changed as the organisational goals change. The result may be that the users have no flexibility in their operation but are required to work in a way which is defined externally by programmers. "The battery-user has been invented!" (Warboys et al., 1999).

Knowledge Construction versus Knowledge Representation

In general the knowledge manipulation for system development involves two processes (Sun, 1999): *knowledge construction* in which the developer captures the knowledge about the proposed systems, and *knowledge representation* which records the developer's knowledge by means of media such as documents, models or computer artefacts. Conventionally software methodologies focus on knowledge representation and provide the systematic and algorithm-based ways to represent the knowledge in a circumscribed fashion. For example the usual form of knowledge representation in AI is through facts, rules and goals (expressed in propositions). This reflects the closed world paradigm in which formalised mathematical descriptions are used to specify the system behaviour. The knowledge represented in such models or computers is restricted to propositional knowledge in the form of symbolic structures, predicates and rules.

As the models in closed-world culture only represent the developer's understanding at a particular time when the models are created, they are independent of their external referents. Thus if the models fail to reflect the real-world situation, something in the real world domain may be misunderstood. Some researchers argue that the conventionally abstract descriptions which are context-free are inadequate to represent knowledge of a situation fully. Winograd and Flores (1987) say that "knowledge is *always* the result of interpretation, which depends on the entire previous experience of the interpreter and on situatedness in the tradition". A similar emphasis is reflected in EM in the shift of focus from knowledge representation to knowledge construction by which the developer's knowledge is constructed by the context of the real-world domain and in a situated manner (Sun, 1999). This perspective will suit the open development paradigm as the knowledge evolves and is open to change. The focus on knowledge construction does not deny the essential need for knowledge representation: knowledge representation helps the developer to organise the information needed in order to realise a system, whereas knowledge construction enriches the knowledge in a situated manner. For knowledge construction, the knowledge of the developer associated with the system development will change continually due to his growing understanding (or *construal*: the personal understanding of a situation) throughout the process and the new information emerging from the real world domain. It is difficult for conventional models under the closed-world culture to track such change. We shall illustrate in the next chapter how the EM approach takes both knowledge representation and construction into account with the generation and comparison of experience through interaction with computer-based artefacts, but firstly we define the open development paradigm for system development.

Open-Ended Modelling for System Development

The need for interaction between humans and computers makes it impossible to construct a computer model which faithfully represents the real-world situation in the closed-world perspective. As Beynon et al. (1996) argue, whatever formal propositional account is given for the behaviour of the system, *experience* is necessarily involved in the interaction between humans and the system. They point out that the open aspect of computer systems can be reflected in two ways: (1) the user can attribute interpretations

to the system responses which are beyond the scope of what has been formally specified (or circumscribed); (2) the user is enabled to develop skills in interaction with the system which are outside the scope of the abstract system specification. For software development, the software program can be understood as a statement of beliefs about the behaviour which will take place under certain circumstances. However the validity of the beliefs on which the behaviour is founded is always elusive (Warboys et al., 1999). Especially when the software system is employed in an organisation, it requires a precise understanding of the wider implications of the behaviour of the software program with its environment. This is difficult because our environment is complex as well as unpredictably dynamic. When considering the influence of social factors on the system development, Goguen (1994; 1996) concludes the following points:

- Requirements are situated, which result from negotiations whose outcome depends on the participants (their interests, position, background, etc). Furthermore, the construction, interpretation and updating of requirements are also situated as they can only be understood in relation to the concrete situation (i.e. requirements are *local*).
- Requirements are an inextricable part of the development process. That is, requirements are *emergent* – they do not already exist in the minds of users and designers but gradually emerge from the interactions between them. Important requirements issues may arise throughout the life-cycle of system development.
- Requirements are *contingent* – they evolve throughout the development process. Because as development proceeds, new ideas, specification, code, etc are produced which in turn generate new objects and relations and inevitably modify requirements. This is the issue of the 'evolutionary paradigm' for system development which will be detailed in the next subsection.

The *open development* paradigm suggests that the focus should be on interaction that is adaptable in order to cope with the wide variety of situations. That is, systems should not be built by following fixed patterns but be developed by the interaction between actors and the environment. Winograd and Flores (1987) say that the description of programs is based on an *idealisation* in which we preconceive the functionality of computers, but in the actual use of computers there is a critical larger domain in which

new issues arise from the breakdowns of hardware and software. Further, besides these technical aspects, there are the concerns of the people who design, build and use the systems. As Winograd and Flores assert: "The world determines what we can do and what we do determines our world". The creation of new systems will create new ways which did not exist before and a framework for actions and interactions which may not make sense to people before (cf. section 3.3). Loomes and Nehaniv (2001) observe this situation and describe that the requirements, the system, and even the users may drift, due to the result of situation/interaction change and the pressures within the network of users and systems. They use the word-processing systems as the example to illustrate the drift in requirements and context of use which create new needs. Originally the uses of the first word-processing package was typists and the package was developed as an electronic typewriter. But as new functions and technology are invented and added, such as linking within a network, the requirements of such a system are not merely spelling checks or fonts of variable sizes, but will also include portability, compatability and security. Also, the users of the system will 'drift' from typists to the authors or the readers of the document. Through this example, Loomes and Nehaniv conclude that the direction of technological change creates, subverts and co-opts the needs of its users:

> *What the user wants* is now determined by many pressures on the network of organisations that rely on word-processing software, and what the user now *needs* to survive in the organisation. This is how '*wants*' has become '*needs*'. (p. 35)

Open development has several advantages. It addresses the important concern that, when interacting with the systems, humans perceive the connection between their actions (or interactions) and the effects of the interaction with the systems. As Brödner (1995) points out, open development means that, rather than restricting the user to specific procedures, the use of a system incorporates some conventions for interaction that enable the user to explore and experience its usability. Open development is also associated with the essential involvement of the designer in interpretative activity associated with interaction with the emerging system during the development process (Beynon et al., 1996). This involvement can be reflected in the development of models and in interaction with the external world.

2.3.3 The Evolutionary Paradigm for System Development

Organisations are human activity systems which are complex as well as dynamic. As mentioned in the last section, it is important for organisations as a whole to be adaptable and flexible, thus the software as a serving system must also be adaptable and flexible. However a software designer cannot predict the role of software systems and how they will contribute to an organisation, especially when such systems will also change the environment. Warboys et al. (1999) infer that software systems should evolve in the way which is consistent with the evolution of the organisation or should even provide a means for the evolution of the organisation. Lehman (1992) also has a similar viewpoint:

> A computer program or software system is a model of a domain and of an application or problem in that domain. For *real* applications in *real* domains, that is for *real world* applications, the domain is essentially continuous, unbounded and dynamic whereas the software model is essentially discrete, bounded and, unless changed by human decision and action, static. Moreover, there will always be a time delay between a decision to change the software and satisfactory completion of the change. The software model is, therefore, an approximation, essentially incomplete and with embedded assumptions. ... With the real world forever changing, an increasing number of assumptions embedded in the system will, unless corrected, become invalid. Even if by some miracle there are none, this can never be known. The system becomes progressively less satisfactory and unpredictable. Uncertainty of behaviour follows.

This means the software development is not a one-off and linear process. It is characterised by uncertainty because we cannot predict the consequences of the new software design. In conventional system development, the software is built and, once installed, it will require only maintenance[53]. Generally the concerns of the maintenance phase are product updates, minor enhancements, error correction and substantial requirement changes (hence rewrites). Warboys et al. (1999) take the view that, while they are in the maintenance phase, software systems should be treated as legacy systems. But Goguen

53.The maintenance phase here is broadly including intensive requirements, design and implementation. But Lehman (1989) suggests the more appropriate term for this phase is *system evolution* as the conventional meaning of maintenance (i.e. preventing change from its current state) is inappropriate when referring to software.

(1994) believes that most of the effort for large systems goes into the maintenance phase. Because not enough effort can be put into being precise in earlier phases and much more is going on during maintenance, i.e. reassessment and re-doing of requirements, design, documentation, testing, etc.

Thus we should treat software as an 'ever to be adapted *organism*' but not as a 'to be produced once *artefact*' (Lehman, 1989), and new design paradigms should provide useful solutions for the uncertainty situation, for example to design reliable systems which can retain usefulness through the process of adaptation and be able to accommodate the features of human behaviour. Warboys et al. (1999) characterise 'adaptation' as the process of implementation of changes to itself which will diminish the difference between a perceived current state and a perceived preferred state. From the systems viewpoint, we need methods for evaluating the systems models by analysing their functionality *during* but not after the development. For the whole system, this goes beyond merely validating and verifying the micro-level aspects such as the data structures or algorithms. We need to develop models in which the artefact can fit the context and satisfy the purpose. This should be done with the wider consideration of the dynamic nature of the contexts in which the systems evolve. Beynon et al. (2000c) add that *compromise* is an important element in human adaptation of solutions: the relaxation of what was considered to be an absolute constraint at the beginning. The software systems and the process of their development must both be adaptable, that is, the design and the system itself should be evolutionary in order to adapt to the environmental changes. The evolutionary paradigm for system development has been considered and discussed by Lehman (1989; 1991; 1997) and Warboys et al. (1999).

The Theory of Evolutionary Design

In (Warboys et al., 1999), after reviewing some software paradigms, such as the Analysis-Synthesis-Evaluation (ASE) paradigm, Artificial Intelligence paradigm, Algorithmic paradigm and Formal Design paradigm, the authors conclude that although each paradigm can address some significant parts of problems, each paradigm is still limited by the 'bounded rationality' (the term from Simon, 1996) of the observer (e.g. the designer). That is, the designer cannot be sure of the completeness of the design. This is because the problems in the real world are not usually well-defined, and what we see is condi-

tioned by the events which define the context. Thus sometimes we may ask if the problem in the domain is in fact what it seems to be. According to Warboys et al., problem situations may continue to be poorly defined and therefore might have to be revisited during development until the problem statement and solution are compatible. This is why they characterise the systems as hybrid and what we want is "to discover the information which has lead to the current situation in which we find ourselves and further what information we might need to make the right decision in this situation". They propose the *Evolutionary Design* paradigm which views the software development process as evolutionary:

> A view of the design process where each elaboration of the design is tested against the requirements, and the result of this test determines the form of the next cycle. It recognises that the requirements for the design may need to be changed, so, over the period of design activity, there is continuous testing and mutual adaptation of both design and requirements and they converge to ultimately be consistent. It takes an empirical scientific view of the design activity. (p. 60)

This means the software designers in this paradigm are forced to resort to *satisfactory* rather than *optimal* designs. It is because the solutions, or the final systems, can be influenced by the viewpoint from which the problem is approached, for example the way the requirements are elicited and analysed. There is no right or best solution as the goals of the organisation may change, so the activities would change and new functionalities would be needed for the systems. As the process of software development is regarded as an evolutionary process, a design at any stage is regarded as a tentative solution to the proposed systems[54]. Warboys et al. suggest that this paradigm encourages us to recognise the link between natural and artificial sciences, the use of testable hypotheses as a method, the nature of evolutionary systems and hence the need for an incremental development approach.

54.That is, the *implementation* in any step of the development process becomes the *specification* for the next step. From this perspective, Lehman (1989) determines that the terms 'specification' and 'implementation' reflect points of view but not absolute properties. The only absolute specification is the application itself which undergoes continuing evolution.

The E-Type Systems

Lehman (1989; 1991) identifies three types of software system: S-type, P-type and E-type. An *S-type* program has a well-defined domain which can be represented by a fixed and pre-stated specification. For an S-type program the only criterion of acceptability is its *correctness*, in a strict mathematical sense, for the specification and implementation. Thus program correctness is always regarded as a specific relationship between specification and implementation. A *P-type* program is the one which is required to produce an acceptable solution to solve some stated problems. It is an intermediate classification between S-type and E-type. The criterion of success is that the solution obtained in execution is correct in the sense stated in the problem statement, and the solution may need to be modified if side effects are observed during execution.

An *E-type* program is required to solve a problem in a real-world domain (the 'E' stemming from 'evolution'). The operational domain of the E-type program is unbounded and keeps changing, and the acceptability of the program is determined by its *consequences*. As the E-type program cannot be developed completely and precisely, the criterion of its assessment is the user *satisfaction* with each execution of the system rather than the correctness relative to a fixed specification. Its relationship with the real-world domain is critical and, according to Lehman, the acceptability of the system is dependent on subjective human judgement. It cannot be judged against some separate specification or problem statement but is understood in relation to needs and expectations which arise in the domain of which it is a part (Warboys et al., 1999). Further, as an E-type system is not entirely formal, Lehman describes how validation techniques such as testing or simulation are used to address the *uncertainty* which stems from the non-formal parts of the system. However as the validation techniques are themselves incomplete and imprecise, "the outcome, in the real world, of software system operation is inherently uncertain with the precise area of uncertainty also not knowable" (Lehman, 1991).

Conventional mathematical models which emphasise correctness are suitable for the development of S-type programs. Based on the prescribed and fixed specifications, these models aim to develop the right software (validation) and the software right (verification) (Sun, 1999). Such models provide some

proven techniques and tools for software developing in order to ensure the completeness and accuracy of the software. Lehman does not deny this, but he discusses that all systems relating to the real world are E-type and cannot be entirely formal, even though the end result of the process, the executable code, is totally formal in that it serves as a computer program. This is because the model (or representation) developed in the first step involves a non-formal representation mainly from mental abstraction of referents in the real world, whereas other further steps can produce formal (the S-type) elements. Thus any systems will have to display such characteristics of non-formal parts and omissions of this will be the major source of misuse and unsatisfactory results. The process of E-type system development is a transformation from a non-formal representation of a real-world application to a formal representation of a part of the program. It is possible to develop an S-type program in an E-type context. But the S-type models can only be of limited help for the E-type system. One reason is that their specification can only be a provisional description and is liable to change. Another reason is that the E-type systems have other properties (non-formal parts) which may affect their characteristic emergent properties. Thus Lehman suggests regarding an E-type software as a model of the application, its participants (human and mechanical), the operational domain and activities in that domain. This is similar to Warboys' Theory of Evolutionary Design Paradigm which is claimed to act as a framework able to use other paradigms.

As the E-type software must continually evolve to satisfy the conditions and the requirements of the changing environment, Lehman assesses *feedback* as the most important resource for revising the model because the feedback mechanism plays a significant role in determining the performance and dynamics of the software process (Lehman, 1998; Kahen and Lehman, 2000). Also the feedback nature of software processes has direct implications for the developers of E-type software. In relating to EM, the feedback may take the form of learning from experience by the developer, or take the form of observation or interaction with artefacts. Such feedback can further enable the developer to modify his future behaviour. Lehman says that the software evolution process is a closed loop feedback system. That is, the installation of the software will change the operational domain as well as change the application. Additional activities will also arise from the operation of the systems and there will be changes in the activities associated with the application. Thus the application and the operational domain will be

changed by the output of the process. Another important aspect of feedback in software processes is that it explains why process innovation has only a limited impact at the *global level* (the 'systems' level in Boulding's sense), i.e. the process improvement has to be assessed at the global level. This is because, as Lehman says, the characteristics of systems evolution are largely determined by human and organisational factors, rather than the process and technology used to achieve that evolution. Traditional programming methodologies, which mainly focus on improving the individual development activities and technical processes by proven languages and formalised methods or tools, will not be sufficient. Table 2.3 summarises the behaviour statements (the laws[55]) from Lehman's observation of the growth of IBM OS/360-370 and other systems, which, as Lehman claims, can provide useful inputs to understanding of the software process.

No.	Brief Name	Law
I (1974)	Continuing Change	E-type systems must be continually adapted else they become progressively less satisfactory in use.
II (1974)	Increasing Complexity	As an E-type system is evolved its complexity increases unless work is done to maintain or reduce it.
III (1974)	Self Regulation	Global E-type system evolution processes are self-regulating.
IV (1978)	Conservation of Organisational Stability	Unless feedback mechanisms are appropriately adjusted, average effective global activity rate in an evolving E-type system tends to remain constant over product lifetime.
V (1978)	Conservation of Familiarity	In general, the incremental growth and long term growth rate of E-type systems tend to decline.
VI (1991)	Continuing Growth	The functional capability of E-type systems must be continually increased to maintain user satisfaction over the system lifetime.
VII (1996)	Declining Quality	Unless rigorously adapted to take into account changes in the operational environment, the quality of E-type systems will appear to be declining.
VIII (1996)	Feedback System	E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.

Table 2.3 Current Statement of the Laws of Software Evolution (from Lehman, 1997; 2000)

55.They use the term *laws* because the observed pattern encapsulate common behaviour that is external to the technical development process (from the developer's perspective).

2.4 Concluding Remarks

The concept of 'systems' has brought a big change in our view of the world. That is, seeing the world as an integrated whole rather than simply a collection of parts. Systems thinking contains more than holistic thinking as it also emphasises interaction and not just wholeness. Thus systems thinking is also contextual thinking and process thinking. Many concepts in scientific disciplines have been developed from the systems approach. Checkland's SSM based on systems thinking aims to cope with complexity in everyday life. The focus of SSM is on the 'system' which is applied to the process of dealing with the world rather than acting on the world. This makes SSM differ from hard systems thinking which focuses on the 'doing' aspect and on the physical domain. The development of SSM is a learning process and links with both physical and conceptual domains.

The development of computer systems can be of benefit when applying systems concepts to cope with complexity in problems. Conventional systems methodologies in the closed-world paradigm try to formalise the development process based on rigorous mathematical algorithms for minimising the human subjective influence as well as ensuring the quality of the results. The knowledge represented in a circumscribed fashion in such models is restricted to propositional knowledge which is static and circumscribed. What we need is knowledge construction in which the developer's knowledge is constructed in the real-world domain through experience and in a situated manner, i.e. to represent the knowledge in an open-ended way. This meets the culture of open development paradigm and evolutionary design in which the focus is on the interaction between actors and the environment in order to cope with a wider variety of situations rather than following fixed patterns of activities. This is especially important for BPR whose aim is seeking overall business process improvement. We have to consider both the computer supported business processes as well as the development process of the software on which the computer systems are built. The computer system in the domain of BPR is best regarded as an E-type system which needs to be continually adapted to support the organisations with its changing goals, and the changing application in the dynamic environment. For this we need models which can reflect the dynamic properties of the systems, the domains and the interaction between all the agents.

In chapter 4 we will discuss the principles and essential concepts of our experience-based approach – Empirical Modelling. The aim of our EM approach is to allow the results of evolution to feed directly into the decisions about system development. We will show that by EM the evolution and feedback through the interaction with computer-based artefacts can also be useful for human beings to take a global view in the dynamic context.

CHAPTER THREE

Business Process Reengineering

Business process reengineering (BPR) has been receiving attention from industries as well as the academic community, because it is likely to change management practice and working processes in organisations in the future. However it is commonly agreed that BPR is important but also problematic. In this chapter we explore the principles and assumptions of BPR and identify the factors affecting its successes and failures. Especially we highlight some major debates currently found in the literature of BPR. These debates include the definitions used to describe business processes and BPR, the scale of the changes involved in BPR, and the significance and role of information technology (IT) in BPR, especially IT systems. As the main theme of this thesis is applying EM to BPR, it is essential to understand some factors which cause BPR projects failure due to the poor design of the supporting systems under the conventional paradigm.

3.1 Business Process Reengineering: Introduction

BPR is known by many names, such as 'core process redesign', 'new industrial engineering' or 'working smarter'. All of them imply the same concept which focuses on integrating both business process redesign and deploying IT to support the reengineering work. In this section we attempt to explore two questions: where does BPR come from and what is involved in BPR (i.e. its principles and assumptions).

3.1.1 What is BPR?

Generally the topic of BPR involves discovering how business processes currently operate, how to redesign these processes to eliminate the wasted or redundant effort and improve efficiency, and how to implement the process changes in order to gain competitiveness. The aim of BPR, according to Sherwood-Smith (1994), is "seeking to devise new ways of organising tasks, organising people and redesigning IT systems so that the processes support the organisation to realise its goals".

The Definition of BPR

It is argued by some researchers (for example, van Meel et al., 1994; MacIntosh and Francis, 1997; Peltu et al., 1996) that there is no commonly agreed definition of BPR. Peltu et al. consider that this lack of an accepted definition of BPR makes it difficult to assess the overall success or failure of its concept. Thus it is essential to make clear what the definition of BPR is before we propose any framework and techniques for BPR. The book *Reengineering the Corporation: A Manifesto for Business Revolution* by Hammer and Champy (1993) is widely referenced by most BPR researchers and is regarded as one of the starting points of BPR. The following is their definition of BPR:

> [Reengineering is] the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service and speed. (p. 32)

Another BPR father, Davenport (1993), describes 'business process redesign' as:

> ... the analysis and design of workflows and processes within and between organisations. Business activities should be viewed as more than a collection of individual or even functional tasks; they should be broken down into processes that can be designed for maximum effectiveness, in both manufacturing and service environment.

These definitions suggest that we should concentrate on *processes* rather than functions (or structures) as the focus of the (re-)design and management of business activity. The definitions of the term 'proc-

ess' by different researchers are also slightly different. For example, Hammer and Champy (1993) define a process as:

> a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer. (p. 35)

For Davenport (1993) it is:

> A process is a specific ordering of work activities across time and space, with a beginning, an end, and clearly identified inputs and outputs: a structure for action. (p. 5)

And Warboys et al. (1999) define a process as:

> A process is structured change, i.e. there is a pattern of events which an observer may recognise across different actual examples (or occurrences) of the process, or which may be made manifest, or implemented, in many different occurrences. (p. 32)

In BPR, the process to be reengineered is the so-called *business process*. Davenport describes a business process as "simply a structured, measured set of activities designed to produce a specified output for a particular customer or market". Riemer (1998) describes business processes in an object-oriented style: "business processes are series of steps that change states of business objects (that is, customers, orders and inventory), thereby causing business events". However we should note that BPR is concerned with customer-orientation. Thus the outputs of business processes should not only achieve the company's objectives, but also need to satisfy customers' requirements. From these definitions we can conclude that business processes start and end with customers, and the value of business processes is dependent upon customers.

The Origins of BPR

Some researchers argue that the original concept of *reengineering* can be traced back to the management theories of the nineteenth century. As one report in *The Financial Times* (1994):

> The purpose of reengineering is to make all your processes the best-in-class. ... Frederick Taylor suggested in the 1880's that managers use process reengineering methods to

> discover the best processes for performing work, and that these processes be reengineered to optimise productivity. ... In the early 1900's, Henri Fayol originated the concept of reengineering: To conduct the undertaking toward its objectives by seeking to derive optimum advantage from all available resources. (p. 8)

Similarly, Galliers (1998) observes that "BPR ... far from being a new departure, is in fact a reversion to the *classical* school[1] of strategic thinking popularised in the 1960s". That is, organisations make such radical changes when they meet competitive pressures which challenge their current processes. BPR can be viewed as a response to such change and therefore fits in the classical school of strategy where organisations adjust themselves to new forms in order to maximise their profits. However it is commonly agreed that BPR first came and attracted academic and industrial attention in 1990 as a result of two papers by Michael Hammer (on reengineering, see Hammer, 1990) and Thomas Davenport (on business process redesign, see Davenport and Short, 1990). In 1993 they further published two key books (Hammer and Champy, 1993 and Davenport, 1993) which brought widespread attention to the emerging field of BPR.

The concept of BPR is widely regarded as having been introduced as a perceived solution to the economic crisis and the recession of the late 1980's and early 1990's (Butler, 1994; Arnott and O'Donnell, 1994). As Butler describes it: "the '80s were a time for financial reengineering ... the '90s are for technological reengineering". Hammer and Champy (1993) propose that "BPR can help organisations out of crisis situations[2] by becoming leaner, better able to adapt to market conditions, innovative, efficient, customer focused and profitable in a crisis situation".

Before BPR emerged (and even today), it was widely accepted by industries and business enterprises that a work should be broken down into its simplest (and most basic) tasks. This leads to the structure of enterprises becoming hierarchical – or functional – in order to manage such divided tasks.

---

1. Galliers points out that the four schools (approaches) to business strategy since the 1960s are classical, processual, evolutionary and systemic.
2. Hammer and Champy gives a brief history of management problems in a chapter ("The Crisis that will not Go Away"), in which they point out three forces (the three Cs) are driving today's business environment to become more and more complex and thus 'the crisis will not go away": (1) *customers* take charge; (2) *competition* intensifies; (3) *change* becomes constant.

These hierarchical or functional structures were commonly used for a period. However enterprises of these structures later encountered some problems, especially when the competitive environment has changed beyond what we can recognise[3]. Today, many enterprises face competition from the global business environment as well as the fact that the taste of customers is becoming complex. As Hammer (1990) argues, "in order to achieve significant benefits, it is not sufficient to computerise the old ways, but a fundamental redesign of the core business processes is necessary". New organisational structures, which are more suitable to today's environment in which enterprises can understand their current activities and find potential problems, are needed. MacIntosh and Francis suggest that it is becoming more important "to develop new products effectively than to produce old products efficiently". By introducing fast developing information technology, enterprises try to redesign their structures and seek new ways of operation, which results in many enterprises moving toward *combination* but not *division* of labour. Hammer and Champy conclude that previously divided tasks are now being re-unified into coherent business processes. Thus one reason why BPR becomes popular is that it provides a mechanism to make the changes better to fit the competitive environment to which the enterprises must adapt themselves in this new and post-industrial age.

3.1.2 The Key Concepts

BPR seeks to break from current processes and to devise new ways of organising tasks, organising people and making use of IT systems so that the resulting processes will better support the goals of the organisation. This activity is done by identifying the critical business processes, analysing these processes and redesigning them for efficient improvement and benefit. Vidgen et al. (1994) define the central tenets of BPR as:

- radical change and assumption challenge;
- process and goal orientation;

3. For example, MacIntosh and Francis (1997) point out some problems: information could not easily be transferred without repeated, manual reprocessing and the layers of management served to relay and communicate information across and through the enterprise.

- organisational re-structuring;
- the exploitation of enabling technologies, particularly information technology.

That is, by focusing on business objectives, we analyse the processes of the organisation, eliminate non-essential or redundant procedures, and then use IT to redesign (and 'streamline') organisational operations.

BPR as Radical Change

BPR is a radical change, rather than incremental change. Hammer and Champy (1993) highlight this tenet as:

> Re-engineering is ... about rejecting the conventional wisdom and received assumptions of the past. ... Reengineering is the search for new models of organising work. Tradition counts for nothing. Re-engineering is a new beginning. ... To succeed at reengineering, you have to be a visionary, a motivator, and a leg breaker. (p. 49)

Similarly Davenport (1993) advocates radical change:

> Objectives of 5% or 10% improvement in all business processes each year must give way to efforts to achieve 50%, 100%, or even higher improvement levels in a few key processes. ... [Radical change is] the only means of obtaining the order-of-magnitude improvements necessary in today's global marketplace. ... Existing approaches to meeting customer needs are so functionally based that incremental change will never yield the requisite interdependence. (p. 1)

One reason the change in BPR is radical rather than incremental is "to avoid being trapped by the way things are currently done" (Vidgen et al., 1994). Dr Robinson of IBM UK highlights rapid IT innovation and increasingly intensive global competition as two main reasons why organisations have had to consider the introduction of radical change[4] (cf. Peltu et al., 1996). Robinson (1994) concludes that radi-

4. He further explains the progress of microelectronics in making IT cheaper and more powerful had been combined with advances in telecommunications to transform the opportunities available to re-invent business processes, management methods and organisational cultures and structures.

cally re-visioned processes drive the *shape* of the organisation, rather than current *structures*. Even such radical changes are not limited to inside one organisation but forge with other organisations, which generate new views of an organisation (Vidgen et al., 1994):

> Possible [radical] changes to the organisation are not limited to internal re-orderings, ... Links can be forged with other organisations even though they are competitors. This leads to a view of the organisation as *a fluid mix of interests* rather than a fixed entity with an objective existence. It is recognised in the BPR literature that advances in technology bring opportunities that were difficult to imagine before the technology had been created. There is a sense of innovatory solutions looking for problems and the exploitation of unexpected consequences that cannot be predicted by a purely conceptual approach. At its best, BPR can be seen as a mix of *conceptual thinking* and *practical experience* gained through creative experimentation and faith.

BPR, DSS and TQM

When discussing radical change in BPR, we find that BPR, DSS (decision support systems) and TQM (total quality management) have much common with each other. Firstly they are all focusing on business processes. Arnott and O'Donnell (1994) characterise DSS as relevant to BPR as it was the first information system (IS) movement to explicitly focus on the fundamental redesign of business processes rather than on the efficient application of a new computer technology. Also BPR and DSS have a common aim which is to improve business processes via radical change. The most significant difference between BPR and DSS is the *scope* of analysis: BPR focuses on the whole organisation whereas DSS focuses on one individual decision.

BPR is also different from TQM in that BPR concentrates on major discrete changes to business processes, whereas TQM concentrates on minor continuous improvement to business processes. That is, the improvements in TQM are smaller than the ones in BPR. Butler (1994) elucidates the difference between BPR and TQM as:

> [TQM approach] which favours steady incremental gain, may often take a number of years to complete. For firms in highly competitive industries, this lagtime can allow com-

petitors to forge ahead. In contrast, results from BPR can be realised within 12-18 months, but it is a far riskier undertaking, and should not be regarded as a 'quick fix' solution.

Furthermore, whereas BPR is commonly viewed as a top-down solution from management[5], TQM involves staff from all levels for problem solving and suggests bottom-up improvement. Employees' resistance to change has been identified as one major barrier to the success of BPR (cf. subsection 3.2.2). It was reported by MacIntosh and Francis (1997) that those companies that had introduced TQM prior to taking on board BPR, faced less resistance to change. As we believe that these two approaches are compatible, we propose in this thesis a concept of 'participative BPR' which combines both of them. Section 3.2 will describe the problems of BPR due to human factors, and motivate the concept of participative BPR. The details of participative BPR will be further described in chapter 6.

Process-Orientation: From Structure to Process

Many current business processes – with their functional structures – were designed to enable efficient management by separating processes into small tasks that could be performed by less skilled workers with little responsibility. Under this structure, the important decisions were made by the higher skilled and more trusted managers. Traditional (structural) approaches to a business engineering generally follow this sequential order: firstly business strategy is proposed, then the business structures and processes are planned, and finally they are implemented with IT.

In comparison, BPR is regarded as process-oriented which is trying to overcome some problems raised by hierarchical structures. That is, BPR as a process-orientation changes the structural relationships between management and employers into the interactive processes between them. BPR aims to break radically the existing process structures and replace them by fundamental and innovative solutions. The functional structure is a vertical structure in which there may exist barriers to separate the

5. It is argued by some researchers that reengineering cannot be led from the bottom of organisations, because it may be blocked by organisational boundaries, "like a wave dashing against a sea wall" as Stewart (1993) describes.

functions in organisations. BPR emphasises business processes which are regarded as horizontal flows and cut across organisational functions. MacIntosh and Francis (1997) justify the claim that BPR highlights the delays, errors and inefficiencies which are introduced when passing information and work from one function to another.

3.2 Problems Facing BPR

In 1996 Davenport published an article entitled *Why Re-engineering Failed: The Fad that Forgot People* in which he reports:

> To most business people in the United States, re-engineering has become a word that stands for restructuring, lay-offs, and too often, failed change programmes ... companies that embraced [re-engineering] as the silver bullet are now looking for ways to re-build the organisation's torn fabric. (Davenport, 1996)

Also in 1998 it was reported that only around 30% of BPR projects were regarded as a success (Galliers, 1998). The earlier promise of BPR had not been fulfilled. There are many reasons for the limited success of BPR. Some explanation of such high rates of failure for BPR projects have been discussed in BPR literature. For example, employees' resistance of change as they consider BPR as threats to their jobs (i.e. the increase in short-term contracts and lack of promotion); Galliers (1998) and Gerrits (1994) point out that currently BPR approaches lack detailed guidance and support for the actual implementation of reengineering: many publications describe the situation before and after BPR but do not discuss the path to reach the final situation; Chen et al. (2000a) explain that one reaction to this failure was to retain faith in IT as a dominant support and just admit that since it could not adapt – or at least not at acceptable levels of cost – then business activities must adapt to IT. For example:

> The pendulum has swung from 'continuous reengineering and re-inventing' to 'pick an application package and force our business processes to comply with the package'. (Riemer, 1998)

In this section we focus mainly on the role of IT in BPR and on human factors which affect the success of BPR. The reason is that there is still a gap between the need to model business process innovations and the capabilities of IT to support the task. In the last section, we propose participative BPR and modelling business processes, as the main idea of our EM approach to BPR is to consider the wider context of a desired 'system' in terms of the purposes, people and other resources which form the environment of the system.

3.2.1 Picking an Application and Changing Business Processes to Fit

One central tenet of BPR is to exploit IT to support 'radical change'. Some authors view IT as the central enabler of BPR. However BPR has not really worked as its proponents expected. Davenport and Short (1990) attribute this problem a lack of appreciation of the deeper issues of IT, and stress that an awareness of the capabilities of IT can influence the business redesign process. They claim that IT has traditionally been used to hasten work but not to transform it and BPR is about using IT to do things differently. Thus IT plays an important role in BPR. Properly adopting IT can improve the competitive position of organisations. But inappropriately adopting IT may create barriers to responding to the rapidly changing business environment[6]. Further, simply picking IT packages cannot achieve successful BPR if it is simply used to speed up the process rather than reengineer it. As Davenport (1993) comments:

> ... information and IT are rarely sufficient to bring about the process change; most process innovations are enabled by a combination of IT, information and organisational/ human resource changes. (p. 95)

Hammer (1990) also has a similar arguments about IT in organisations:

> IT could either 'pave the cowpaths' of bureaucracy – unless the organisation changed drastically, its IT would continuously reflect and reinforce bureaucratic and functional

6. For example, these IT tools may be designed for functional hierarchies, or they are TQM type tools – which are only in effect designed to support incremental improvements – cannot achieve the radical change in BPR projects.

> structures – or IT could help to create a leaner, flatter and more responsive organisation, a suggestion which is thus distinctly divergent from neo-classical economics, but only implicitly.

That is to say, while the information systems, or in a narrow sense the 'data-processing systems' (see the quotation from Galliers (1995) below), provide fast processing and response, they often fail to provide the flexibility for human communication – sometimes with serious consequences. This means IT may sometimes have a negative impact merely automating the existing processes[7], but it could also have a positive impact if we deploy it correctly in appropriate organisational arrangements. For BPR we need to change the structure and nature of operations, i.e. IT is the enabler to reengineer their processes and is an important driving force for business transformation. Galliers (1995) emphasises that the desire in BPR is to

> develop a flexible *information* systems environment (i.e. one that *informs*) rather than simply developing *data-processing* systems (i.e. systems that automate an operational task). The latter may be accomplished by replicating observable actions, while the former requires considerable awareness of the context in which information may be required and the manner in which it is likely to be interpreted to enable a required activity or decision to be made.[8] (p. 60)

As we will further detail in section 3.3, the introduction of information systems will not merely automate existing business practices, but will also shape the business. The purpose of reengineering the business is focusing on the latter in order to secure competitive advantage for the business.

3.2.2 Human Factors in BPR

We have found that the focus of the BPR literature is on IT and process redesign techniques. However there are other complex issues such as human, organisational, cultural and political issues. Some

---

7. Or we can say that mere automation of existing processes can at best lead to more *efficient*, but no more *effective*, processes.
8. He further differentiates both 'information' and 'data': information is the collection of data. "Information may be understood as being both *enabling* and *contextual*, while data is context-free and simply the raw material from which information (meaning) may be *attributed*".

researchers determine that one of the main reasons for BPR failure is the neglect of the human element – the approach takes too much account of the scale of changes and fails to consider such change through people. Corrigan (1996) describes this situation thus:

> Given [BPR's] focus on business processes, many researchers have highlighted the lack of attention given by [BPR] to the human dimensions of organising, emphasising "how employees, not just processes must be re-engineered or debugged if they are to run effectively in systems".

Davenport (1993) emphasises that the success of reengineering programmes is dependent on and concurrent with effective organisational and human resource change. Thus it is essential to take a wider (contextual) view of these influences on the success of BPR (cf. subsection 3.3.1).

Resistance to Change

BPR aims at the change in the organisation that is for the best. However as BPR is a radical rather than incremental change, it is not surprising that 'resistance to change' has been identified as a major barrier to the success of BPR (Corrigan, 1996; Quaddus, 1994; Peltu et al., 1996). Corrigan identifies one common situation in most organisations adopting BPR (through his interviews):

> Some of the interviewees pointed to the fact that employees were expected to agree and go along with goals and changes in working life that have been determined by senior management, and that they were the last to know about how change would affect them.

For these people, BPR is perceived as a threat to their jobs, either a threat directly to their existence or a threat to the quality and content of their jobs, or as causing the lack of promotion. People commonly asked: "Why change if it is working?". Extremely, as Peltu et al. (1996) verify, 'downsizing', I.e. sacking people, is the most obvious 'dark side of BPR'. Thus, to avoid this situation, as we discussed earlier about TQM, many companies try to introduce TQM prior to BPR for the reason of less resistance to change. Stewart (1993) notes that "you cannot do reengineering without an environment of continuous improvement or TQM". BPR can only work when those in the company who have to work with the new design have a role in creating it, and thus support such changes.

Personal View of Change

The people in business – either employees or designers – take a very personal view of the radical change of BPR. For example, the BPR designers design BPR based on their perception, explanation and understanding of the organisation. The solutions they propose for BPR problems are derived from the theories or frameworks of BPR literature, as well as their understanding and attitudes to the organisation context. However each designer or even participant of an organisation has different knowledge and perceives the organisation in different ways – their individual contexts are different. Further, the participants have different experiences, theories and beliefs about why and how a process is operating and organised. We should note that the behaviours of participants generally come from their old contexts. This may cause conflict as newly designed processes can be so radical that they may not fit the participants' old experiences and contexts. Without their participation, BPR designers will not know the cause-effect relationships and how to modify the processes. As Kutschker (1995) reports:

> Obviously, process redesigners ignore this knowledge and repeat the mistakes designers of structural change have already made. The high failure rate of BPR may have its roots in the poor understanding of the deeper structure of organisations.

A way to avoid this is to involve designers and business people in the process, particularly in testing out the design. This is an issue that our participative BPR (to be detailed in section 3.4) emphasises. We believe that participants can be the drivers and a key successful factor of change, but can also act as potential blockers of innovation. The huge investments in IT and highlighting IT systems alone in BPR will not improve organisational performance or create any benefits. Thus even when BPR is enabled by IT, it is essential that BPR designers and IT people understand the human aspects in organisations. To conclude, human factors are much more important than technical factors. The BPR work must obtain the support of people affected by the changes for successful implementation.

3.3 System Development and Business Process Reengineering

In the previous chapter we have discussed the issues of system development. In this section we will try to find out whether the failure of BPR is partly due to the poor design of computer systems, which were initially designed as an enabler of BPR but finally become the main factor of causing BPR failure as they hinder the organisational change and obstruct the innovation.

3.3.1 The Relationship between Information Systems and Organisations

The BPR techniques consist of many different topics, such as system analysis and modelling, operations research, management information systems (MIS), human resource management, etc. Amongst these, in the author's assessment, system development techniques play an important role in BPR in understanding the current systems and in analysing the potential benefits of the redesigned systems, especially when such information systems are the enabler and supporter for the radically redesigned processes. We will discuss the role and impact of information systems on organisations. Then the defects of the conventional approaches of system development for the consideration of BPR, will also be discussed next.

Software Systems, Information Systems and Organisations

The term *information systems* is widely used in business and means the systems inserted into an organisation to support data processing and the decision making within it. In general, information systems are regarded as a kind of structure which reflects the activities of the organisation (Parets and Torres, 1996) and a software system is a part of it. From this viewpoint, information systems represent a kind of abstraction of social systems which may include designed systems (i.e. computer systems) and human activity systems.

Figure 3.1 represents the relationship between the world and a software system. We regard software as a designed abstract system (or conceptual system) because it is a symbolic construct of

human conception. Cook and Daniels (1994) also suggest software as a concept model which partially mimics and abstracts the behaviour of things in the world. However they point out one potential problem: software cannot totally model the real world. This is because the world is unpredictable but software is predictable as its behaviour is preconceived. Also as software is a part of the world, it is impossible to design software on the assumption that its existence will not change the world. However conventional system development is concerned with software systems only and assumes that software will not affect the basic structure of the organisation (Kawalek and Leonard, 1996; Warboys et al., 1999). But now some problems of developing computer systems in this way arise because when new software solutions are provided, new opportunities for business change arise. That is, any designed information system is unlikely to be perfect and suitable for the purpose for which it was designed, and this is why today many organisations have faced *legacy system* problems because their computer systems cannot be changed to serve the changing circumstances of the organisation[9]. Warboys et al. (1999) emphasise this situation:

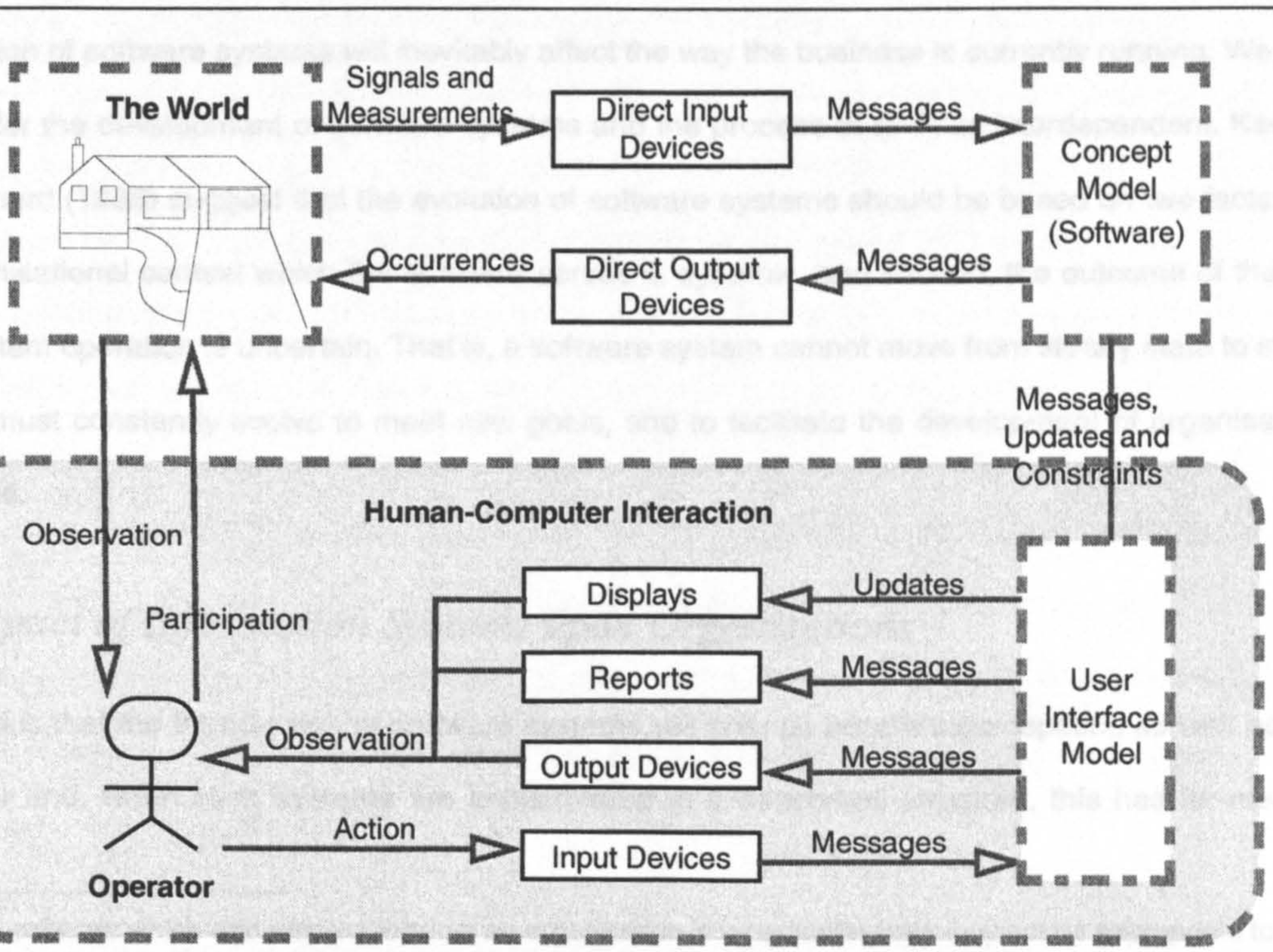


Figure 3.1 Linking the World and Software (adapted from Cook and Daniels, 1994)

> It is not just that software systems are becoming more significant in modern organisations, but the form of the software system is changing. This in turn fuels, and is fuelled by, the drive to new ways of working, new strategies and new products. ... The high costs of developing bespoke software systems leads to a requirement that they be long-lived. This in turn means that their scope, functionality and the assumptions embedded within them are more likely to be breached by new needs which result from broader organisational changes. These changes occur, sometimes with rapidity, as the organisation seeks to survive, to compete and to innovate. (p. 10)

For BPR, it is likely that providing IT to support a process may change the process itself, which leads to further interactions and degrees of support. Thus if IT is adopted in organisations then a change to the organisational environment follows which represents a challenge to which IT may be required to adapt. This suggests that the process of reengineering is *dynamic* and will inevitably lead to further iterations in which the reengineered processes need to be re-assessed and redesigned. For this, we can conclude that business (re-)engineering and software development cannot be independent of each other. The redesign of business processes will decide the requirements of a software system; whereas the introduction of software systems will inevitably affect the way the business is currently running. We need to consider the development of software systems and the process of BPR as interdependent. Kawalek and Leonard (1996) suggest that the evolution of software systems should be based on two facts: first, the organisational context which the software serves is dynamic, and second, the outcome of the software system operation is uncertain. That is, a software system cannot move from steady state to steady state. It must constantly evolve to meet new goals, and to facilitate the development of organisational processes.

The Impact of Information Systems upon Organisations

It is obvious that the introduction of software systems will change people's perceptions as well as their behaviour and, when such systems are implemented in a networked structure, this has far-reaching

9. That is, software which was created to serve an organisation in a particular state may not be appropriate to serve the organisation in a future developed state. Ulrich (1995) claims that the failure of BPR is partly due to the limited ability of organizations to 'retool' existing information systems environments to support process redesign.

impact on the fundamental behaviour of organisations. Our perceptions of organisations and the relationships between IT and organisations may be different depending on which model we use. Harrington (1991) describes three metaphors of organisations: machines, organisms and processes, which represent three waves of organisation theory. The first wave, to perceive an organisation as a *machine,* suggests that IT can be seen as a controllable resource (the tool for management use) which is not part of the organisation and is used to achieve certain objectives. The introduction of IT does not affect the organisational structure but only the relationship between management and workers. The second wave, when an organisation is seen as an *organism*, regards IT as more integrated and less controllable. That is, IT is an element of organisation and also a determinant of organisational structure which cannot be predictable like a resource. The ownership of it is with workers rather than the management. The third wave perceives an organisation as a *process* and IT as a behavioural phenomenon. IT is managed by the users and determines the perceptions of human beings and thus affects their behaviour.

Conventional system development methods, as a reductionist approach, considers an organisation as a machine, whose behaviour is merely determined by the behaviour of individuals within the organisation. They tend to ignore the perceptual aspects of IT and concentrate only on its physical parts (i.e. the system itself) because it is hard to cope with the abstract things such as the perception and interaction of human beings. Such methods are not suitable for the analysis of BPR because they only reflect the situations before and after the implementation of the systems, but do not emphasis the change of people's behaviour affected by it. There is also a similar problem appearing in BPR, as it is common that many BPR methodologies are adopted by consultancies and attempt to reduce the dynamic levels of business processes to a predictable level by management techniques and adopting IT. Crowe et al. (1996) suggest that such methodologies focusing on static definitions of data, role and processes to reflect organisational structures tend to remain relatively fixed for long periods. However the dynamic processes cannot be well dealt with by static (i.e. linear) methods because people are operating in a dynamic manner and prone to error.

On a systems view, an organisation has got more meaning as a whole than just as a sum of parts. We cannot find the characteristics of an organisation merely by identifying its components. Especially

today the essential resource of organisations is from the skill and knowledge of people. Individual knowledge and technology are not only their own but also part of the organisation. Thus the behaviour of organisations can be regarded as a pattern of interactions between people. After the introduction of software systems into organisations, the behaviour can further be understood as a pattern of interactions between people, between people and software systems, and between the software applications. For this, Harrington (1991) suggests that we need the systems approach to analyse the impact of IT upon organisations, because the nature of IT can only be assessed in terms of its total impact. He describes an organisation as a creation of the mind and the perceptions of the people involved within it. Any change in the organisation will have an impact on people's perception of the organisation. When analysing business processes, we find that people interact with each other within the process and are also influenced by that process. How people will react in organisations depends on how they perceive a particular action, and their perception is important in the process of interaction. Today IT plays an important role in organisations because people use their systems to interact with others and the environment. Therefore any changes in IT may cause changes in organisational structure. Harrington comments on the successfulness of using the systems methodology to analyse the impact of IT on organisations because "it not only reflects the way we as individuals tend to aggregate, and thus captures the essence of how we organise, but also allows us a glimpse of a change effect in terms of the whole rather than part".

3.3.2 The Defects of the Conventional System Development in BPR

There are some discussions which identify one of the causes of BPR failure as the failure of BPR implementation by information systems (cf. subsection 3.2.1). For example, Humphries and Dy (1996) conclude that the inability of implementing the newly reengineering business processes is one key reason to consider for BPR failure. Because there exists a gap between BPR and its implementation, and system development is related to the latter as such systems are required to support the redesigned processes. As we mentioned before, conventional 'waterfall' or 'spiral' lifecycle considers a serial rather than a concurrent approach to system development. That is, the requirements, design and implementation

stages are almost completely isolated from each other and normally failures do not show until the implementation phase. This is because these conventional methods concentrate mainly on the technical aspects of design (such as data, data flows or system functions), and the problems come from diverting these technical solutions at complex social and communicational problems. Mingers (1995) argues that such methods are usually either oriented towards computerising existing processes and assuming these processes are effective, or they assume the users know what they want and eliciting their requirements is straightforward. This situation may have been improved today by the greater participation of users in all stages of the development process, or by the development of more flexible systems. But the system development can still result in failure particularly when the system to be designed is very complex, and in BPR which radically transforms the process which is currently running and involves many users and stockholders with different perspectives. Current methods of system development cannot meet the goal of BPR because they concentrate on the initial design of the systems and assume the proper design can lead to a stable solution. Thus the problems of redesign, and design for continuous change, cannot be captured by such methods.

The impact of information systems upon organisations is more than that of replacing paper with computer systems. Because business processes will change, new opportunities may be hindered (due to the circumscribed and programmed processes), some flexibility may be lost and new dependencies arise. Even though current techniques of systems analysis can capture some of the business environment in their models, these models are still built with the [technological] solutions in the minds of designers and they cannot reflect the fundamental business structures which are essential for business reengineering and are needed to ensure the success. Warboys et al. (1999) also point out that the analysis of business today is mainly concerned with facts which directly influence the design of the computer system. It is not concerned with understanding the working of the system and improving the structure of the system by reorganising processes, people, etc.

We have emphasised the importance of understanding the organisation and the business which the information system is built to serve for the success of BPR. And Lewis (1995) reports that gaining an understanding of organisations requires two levels of analysis:

> The first is concerned with the organisational actions and decisions that have observable, substantive outcomes such as inventory levels or the average time taken to satisfy an order. The second is concerned with understanding how such organisational activities are perceived, interpreted and legitimated. (p. 191)

He concludes that conventional system development have concentrated on the former (i..e. the 'hard' systems thinking) but not including the latter (i.e. SSM). This is why most current methods of system development fail to tackle the issue described earlier. The missing focus of current methods is that not just considering the computer systems to be built, but also unexpected change is one of the big problems in the development of computer systems. From the perspective of SSM (and our EM approach), such 'understanding of the served business' should not be restricted to just investigating the existing business processes, and the system designers are today expected to comprehend as well as contribute to what the organisation may or may not aim at, and how it may affect the achievement of objectives in the future. This is why Lewis says that

> The simple, 'hard' systems thinking model that has influenced IS thinking so strongly in the past is that organisations can be understood as goal-seeking, adaptive-regulatory mechanisms. This is now being rejected in favour of more complex systems models of human organisations as purposeful, socio-political systems in which shared meaning and symbolic relationships are maintained and modified through human discourse and interaction. (p. 190)

The activities of BPR and its implementation (which includes system development) focus on different views and need different techniques or skills. What we need is to bridge these two activities otherwise the outputs of BPR may not be useful or achievable in the implementation stage or the systems developed may not support the newly redesigned business processes. Humphries and Dy (1996) propose an integrated conceptual framework to bridge this gap as they claim the only way is conceptual integrity, i.e. the BPR and system development methodologies in BPR implementation must implement a common set of shared concepts[10]. Through this the concepts used in BPR can be mapped directly to the

10.Their framework is based on the premise that "a business process is equivalent to the lifecycle of a business object". Further detailed for their framework can be found at http://www.intranetsys.com (20 March 2001).

concepts in system development. In this thesis, by using EM, we propose another method to bridge the gap by observation and experience, which provides a natural and participative way to link these different stages and activities. We will detail our approach in the rest of the thesis.

3.4 EM Approach to BPR: Preliminary

In earlier sections we have surveyed some issues and problems causing the high rates of BPR failure, especially in respect of human factors and technology exploitation. In this section we preview the broad approach of EM technique to business processes, and the requirements of a BPR model and how EM is well suited for application to BPR. How we understand the relation of business processes to information (software) systems is crucial for understanding the potential contribution of EM to BPR. We suggest to widen our focus from developing information systems to include modelling business processes and persons. Our focus should include the participants – and their views, knowledge, capabilities and perceptions – behind business processes.

3.4.1 Participative BPR

It is evident that no BPR will be successful without the support and active participation of the people. Even after all persons agree to go with BPR, it is still a hard task for everyone to carry on. As Peltu et al. (1996) describe:

> The BPR process is a 'walk in the fog' because of the difficulty involved in reaching agreement among many stakeholders about the current situation and future needs.

Scarbrough (1996) emphasises that an important question in all programmes for change is "what is required to bring about changes in how people relate to each other?". This suggests a reason for the high rate of failure of BPR, as it is not possible to change relationships without working within them. The IT tools and techniques chosen for BPR can only be the starting point. But the change will not be successful without people's learning, participation and adaptation – in order to understand the require-

ments and processes and then take responsibilities for such change. According to Sherwood-Smith (1994), business processes are purposeful processes, in the sense that they are people-controlled and subject to human behaviour. Hutchison (1994) and Lundeberg (1994) have a similar viewpoint. Each business process has some inputs and outputs. They identify the outputs as a combination of people outcomes and task outcomes, and the inputs as a combination of people preconditions and task preconditions. Thus business processes rely on the interaction of their participants. We argue that BPR is people-centred and driven by business needs, because it is the people who decide the value of the redesigned process according to their understanding and objectives. As Sherwood-Smith advocates:

> Administrative systems involving people should not be reengineered, they should be participatively re-designed.

Such a participative approach respects the culture and social context of an organisation because one important thing of BPR is to ensure the redesigned processes fit the organisational context and are acceptable to their people. This demands a high degree of communication and evaluation.

People Participating in BPR

One of the early problems in IT development, as Lundeberg (1994) points out, is to know whether you are building the information systems that the users need or not. Chen et al. (2000a) argue that available software resources are often accepted as given and their limitations go uncommented although it is, we suspect, precisely their profound limitations that are a significant factor in the 'failure' of some BPR efforts. Many existing applications are designed to be offered in a 'take it or leave it' fashion (i.e. with no commitment to subsequently negotiating or adapting its functionality) which is inappropriate to a rapidly changing business environment. Further, as each person in an organisation has his knowledge and experience in every organisational activity, the members of the organisation should have some representation and voice in the BPR process. This is why Peltu et al. (1996) emphasise that, when initiating a 'clean sheet' radical change, it is necessary from a human perspective to complete the past before moving ahead. We suggest *participative BPR* both as a means to fully understanding what the people need, and of evaluating existing processes and the effects upon the organisation.

Participative BPR – which is different from a traditional top-down approach to BPR – encourages all levels (including managers and workers) to get together to redesign business processes. This is similar to the definition of the approach by Sherwood-Smith (1994):

> A ... participative approach to BPR ... includes the radical innovative vision by management, the innovative use of IT and then takes on board the innovative insights of the staff to the system of organisation and operation.

Participative BPR is similar to one main theme of TQM: *employee involvement.* This aims to involve persons from all levels of an organisation in problem solving techniques. The difference is the improvements of TQM are generated bottom-up (whereas BPR is commonly viewed as a top-down solution imposed by management). We believe participative BPR – which combines both top-down and bottom-up – provide a comprehensive and shared understanding of current processes. Lundeberg, in his argument for focusing on business processes, cites one advantage of participative BPR as "an increased closeness between you and your knowledge of what you are achieving and the consequences of this for your own work situation". This is a natural way to progress from the present situation to meet the demands of the future.

The primary objective of our EM model is to enable the participants to develop and share a common understanding of the current state of business processes. Perhaps the main contribution of EM to BPR, as well as the way we consider software development, is to emphasise and focus on the *interactive* processes in which we participate. The EM model is used to identify possible changes in business processes which help the analysis of designers. This process in EM combines both analysis and design activities because the development is an interactive process where participants can establish a better understanding of the current business situation as well as the future one. Further as we describe in (Chen et al., 2000a), with frequent mergers and outsourcing of activities, businesses need computing environments which support unforeseen changes in needs and can exploit opportunities as they arise. The EM approach creates such environments and they support collaborative working.

3.4.2 Modelling Business Process

Organisations and business processes are complex, especially when they rely on the interaction of people. How deep an understanding of current processes is necessary before redesign to gain the best results is a controversial issue in the BPR literature. Some researchers argue the change in BPR being radical rather than incremental is "to avoid being trapped by the way things are currently done" (Vidgen et al., 1994). Hammer and Champy (1993) also argue that a very detailed analysis of the current process is not needed because the goal of the reengineering effort is not to improve the existing process but to design a 'totally new and superior design'. However we do not believe that there will be much gained from ignoring or paying less attention to the past. Pettigrew (1985) comments on the importance of this view:

> No observer of life or form begins with his mind a blank, to be gradually filled with evidence. ... The more we look at present-day events the easier it is to identify change; the longer we stay with an emergent process and further back we go to disentangle its originals, the more we can identify continuities. Empirically and theoretically, change and continuity need one another ...

The trend of BPR, as reported by Corrigan (1996) and Meester and Post (1998), now is to change the business processes in a less radical way:

> While 59% of the organisations ... claimed to be planning or doing BPR, most were improving existing processes, rather than 'reengineering' those processes identified as a result of a radical rethink of how the organisation needed to be reconfigured, and managed. ... It was therefore felt that the concept of 'reengineering' was too restrictive and was not felt to be an accurate reflection of the current reality of managing business processes. (Corrigan, 1996)

> ... the expectations of BPR as a radical change instrument were high but now we see a certain reluctance towards its application, given that the chances of failure are high, ... clever marketeers sell these products (BPR tools) as a new development in BPR, but in fact it is a way of mechanising an existing process without radically changing it. (Meester and Post, 1998)

In the subsection 3.3.1, we emphasised that business processes have to be understood to enable the participants to (re)design a new process. That is, it is important to understand the existing processes (the old culture, the legacy of the past, etc) both to help in the change to a new culture and different business processes, after BPR, and to avoid some unpredicted things emerging -- especially when discovered after BPR effort -- which sometimes turn out to be a disaster. It is not always possible to predict outcomes: "'Big bang' change is inherently risky!" (Scarbrough, 1996). We note some researchers, such as Jacobson et al. (1995), Schader and Korthaus (1998), Galliers (1998) and Vogel (1994), who support our view. As Vogel asserts: "BPR promotes a comprehensive and shared understanding of current processes, encapsulation of a vision, and a means and path of migration to progress from the present situation to meet the demands of the future". The following description highlights the nature of the BPR process:

> The [BPR] process is a learning experience requiring on-going assessment and review, which emphasises the need to develop and communicate a shared vision, leading to a review of existing structures and processes, but which takes account of customer requirements and the values, experiences and viewpoints of those whose job it will be to implement the change. (Galliers, 1998)

Jacobson et al. (1995) also address this theme in one chapter ('Reversing the Existing Business') in which they write:

> We do believe that you need a good picture of your current organization before you can finally decide on the best way to change it. If the reengineers understand the business as it is today, they will be able to avoid making unfeasible change proposals. (p. 153)

Before commencing BPR, we have to examine the existing business processes deeply in order to redefine and redesign their nature. To gain a wider picture of existing processes we need to consider the following questions: "How does the process under development fit into the whole business picture?", "Who are the participants [or the 'actors' in EM terms] in the process?", "What are the requirements of the process?" and "What is the final objective of this process?". As concluded by Arnott and O'Donnell

(1994), the effect of BPR activities "may be new, it may be revolutionary, it may even be useful; but the seeds of its decline are there to be found – the past".

3.4.3 Requirements for BPR Models

We have justified in (Chen et al., 2000a) that how we understand the relation of business processes to software systems is crucial for understanding the potential contribution of EM to BPR. Adopting the viewpoint of EM, we claim that the issues involved in modelling a business organisation and modelling a software system are not so very different. In the remainder of this subsection we summarise the requirements of modelling techniques to support BPR and explain why we believe that EM is suitable as a modelling technique for BPR.

Artefact

The BPR models should serve as a medium for communication between the participants involved in the redesign process. An artefact that can be held in common is needed to enable all participants to understand and discuss the process to be reengineered and gain a shared understanding of problems. In the BPR literature it is emphasised that reengineering should be achieved by active participation of people and not only be done by designers, managers or IT experts. To enable active participation, the model should be easily interpreted by participants.

Inevitably individual participants in the redesign process will have different insights and expertise, and their requirements may 'drift' throughout the process of interactions and experiences (cf. section 2.3 and Loomes and Nehaniv, 2001). Thus there exists a problem of requirements drift and inconsistency. How to merge the changes of multiple modellers and integrate views from the results of experiment which might cause such a problem need to be considered in the construction of the model. One benefit of interacting with the EM model is that it makes individual insights and shared understanding 'visible' and 'communicable'. We shall further discuss this issue in chapter 6.

Creativity

Simsion (1994) points out that BPR is recognised as a design process, requiring creativity, innovation and radical thinking. He further identifies *creativity* as necessary in view of the following characteristics of requirements engineering:

- Requirements as presented are never so complete or precise as to restrict the designer to a single option;
- There is no mechanical way of proceeding directly from requirements to the best design (or even a sound design);
- Proposed designs may include desirable features not specified in the original requirements. As a result, it may be necessary to modify the requirements to include these features;
- There is no way of assessing whether a given design is the best possible.

Traditional phases of system development – specification, design, implementation and testing – may be one barrier to creativity because of the lack of freedom in how to accomplish a task. EM has no such separate phases – it involves continuous development – in an open-ended and situated manner – throughout the evolution of the models. This is so-called 'cultivating requirements' in building an ISM (interactive situation model), where the seed-ISM being elaborated is refined and optimised into the final useful system. With the SPORE (situated process of requirements engineering) framework, people participating in the reengineering process can gain a sharing understanding and thus apply (or cultivate) their own creativity through collaborative interaction with each other (Sun et al., 1999; Sun, 1999). We will give further details in chapter 6.

Design-In Change

Business often changes in unpredictable ways and what used to be a standard process may not be sufficient for later situations. That is, in the dynamic business environment, there is no such thing as the right or standard solution. Pettigrew (1998) describes this change as a journey:

> In the complex and ever changing situation of business today it is very unlikely that any fixed standards of customer satisfaction, quality, efficiency or competitiveness are likely themselves to be sustainable for long. Hence the dictum that changes represent *journeys* and not *destinations*.

In these situations there is a need for a quick adoption of the processes associated with the new solutions. For modelling business processes, Warboys et al. (1999) point out that traditionally the focus in the information systems domain has been to *design for each change* and to improve maintenance processes, rather than *design-in change*. But we find that, even if we adopt the design-in change philosophy, the designing, testing and use of the resulting artefacts in conventional methodologies will also change human activities in such a way that the resulting system will conceptually differ from the one which was intended. Similarly, the changes in the dynamic organisational environment will also influence the human conceptions of relevant information. In the EM approach, we regard organisations as open systems which must constantly adapt in an integrated way in order to improve their processes and services.

The EM principles and tools directly support the modelling of a collection of agents, which perform structured activities, in an open environment where unforeseen changes may occur at any time. This maps to the business environment more naturally than to that of software systems. Further it is reasonable to view software systems as associated with the circumscribed cases of business processes. Thus when applying EM to BPR we are not immediately confronted with the usual mismatch that arises when the informal problem world (of business) meets the formal solution world (of programming). The SPORE framework we propose for participative BPR supports iterative and incremental modelling which means that the ISMs are built by adding new portions to the whole model. This delivery of small increments allows continuous feedback and evaluation of the progress achieved.

3.5 Concluding Remarks

In this chapter we have explored the questions: "where does BPR come from?", "what is involved in BPR?", and considered some factors affecting its success and failure, especially the human factors and the role of IT in reengineering process. We also discussed the relationship between system development and BPR, and described the defects of conventional system development under the climate of BPR thinking. In the latter part of the chapter we gave the background to our approach to BPR and described briefly how EM principles fit into the thinking and practice of BPR.

In conclusion, the application of EM requires us to widen our focus from an intended computer system to include the entire business processes (the environment) and the people involved (the human factors). This means our focus should include participants – and their views, knowledge, capabilities and perceptions – behind the business processes. It also means shifting our first attention from software requirements to business requirements. The former should be a result of the latter. The scope for creativity in EM, through collaborative interaction of user participation, will be one important aspect of BPR.

CHAPTER FOUR

Empirical Modelling Principles

Empirical Modelling (EM) is an approach to human-centred and situated computer-based modelling which has been developed at the University of Warwick since 1983. The principal aim of this thesis is to investigate the suitability of EM as a framework for software development to meet the need of BPR. As the research of this thesis is based on the framework of EM, this chapter will give an overview of EM, illustrate the fundamental principles and concepts of EM, and show how our approach differs from the modelling methods traditionally used in software system development.

4.1 Introduction

Empirical Modelling is a situated computer-based approach to modelling. EM is based on an unconventional approach to analysing phenomena which is *observation-oriented* rather than object-oriented in nature. This analysis is directed at accounting for phenomena in terms of the state-changing activities of *agents*. Actions of agents and interactions among agents are expressed as changes to *observables*. The way in which a change to one observable affects changes to other observables in a manner that is conceptually indivisible is modelled as a *dependency*.

The concepts of agent, observable and dependency are fundamental to EM. To some degree, these concepts are represented in current computing practices, for instance in 'agent-based systems', 'observers in the Java package `java.util`' and in 'spreadsheets'. What differentiates EM from the ad

hoc use of concepts relating to agency, observation and dependency is the special nature of the relationships between the modeller, the computer model (as artefact) and its referent. Specifically, the artefact stands in a special relation to the modeller's understanding of the referent (his 'construal') as informed by observation and experiment.

The nature of EM invests the concepts of agent, observable and dependency with an ontological status that is unrecognised, if not unrepresented, in their use in other contexts. In this thesis, it will be convenient to regard modelling methods other than EM as essentially framed in a 'classical' ontological framework, and to refer to them as 'conventional modelling methods', even though they may make use of agency, observation and dependency in some form. This classification is uncontroversial when standard modelling techniques for software system development are concerned, since these do not lead to the identification and embodiment of agency and dependency in the manner characteristic of EM.

The term 'empirical' reflects the fact that the EM principles are based on observation and experiment. The emphasis of our approach is on modelling (parts of) the world through experience, i.e. modelling our experience (that of the subject domain) by means of other experiences (of the computer model), in contrast to conventional approaches of system development and modelling which model systems by preconceived abstraction. Actually the term 'empirical modelling' was adopted in 1995 to make such a distinction between conventional mathematical modelling and our experiential models based on definitive scripts.

When describing the concept of circumscription in previous chapter, we mentioned that traditional modelling approaches try to formalise the development process, as it was presumed that good solutions can be obtained by following rigid processes and abstract rules. But the activities under such circumscription, for example when the designer follows a formal method and follows such rigid protocols, are quite different from the activities in our everyday life – the *situated activities* (cf. Suchman, 1987; Hutchins, 1995; Varela, 1979; Varela et al., 1991). This situated activity is difficult to prescribe as it involves the human's dynamic interaction – or 'social interaction' following Goguen (1994; 1996) – with

the external environment in which some unpredictable events may appear and humans may deal with the same situation in different ways according to their knowledge, experience and tools available at the time. Such the activities involve social *situatedness* – they can only be understood in relation to the particular and concrete situation in which they occur; they acquire meaning through interpretation in that situation (Goguen, 1996). That is to say, the activities of our everyday life are context-dependent and human-centred. Thus the formalised activities in conventional modelling (in the closed-world paradigm) can only reflect the real-world activities to some extent, as they are dominated by algorithms and are independent of the context. For example, when we solve a problem in everyday life, we are conceiving the solutions based on our knowledge about the problem situation and our past experience, rather than following a general law. Sometimes we may make a plan for this, but such a plan is not acting in the same way as an algorithm, as Suchman (1987) describes, most plans are simply used as a 'resource' rather than a source of control. This is a motivation for EM: to provide another approach to modelling through the use of a computer as an open-ended artefact. The computer is used in EM as a instrument to facilitate cognitive activity. Through interacting with the computer model, the correspondences between the mental model of the modeller, the computer model and the referent can be established. Through this process the modeller's understanding of the subject can be enhanced and a computer-based model that can serve as a rapid prototype of the proposed system will be created based on the modeller's understanding of the subject rather than his imagination.

To give an overview of our modelling approach: EM is observation-oriented and state-based, that is, all variables in the model are considered to represent the current state of the observed referent. Also EM provides a visual metaphor with interfaces for direct manipulation by the modeller to represent the observables of the real-world subject. That is, the model is constructed as a metaphorical representation of the subject which is consistent with the modeller's construal. EM is also definition-based: it uses spreadsheet-like definitions to express the dependencies between observables, which cannot be achieved by conventionally circumscribed modelling.

4.2 The Essential Empirical Modelling Concepts

The focus of EM is on the construction of artefacts (the computer-based models) for interaction and experiment in a domain in which the user may not understand the subject well. It is observation- and experience-based as we can analyse the domain and the subject through the identification of the structure of our experience in that domain. Also the analysing and modelling of EM are experienced rather than preconceived which brings the computer modelling closer to users and also preserves the openness in the artefacts that reflects our experience in the real world (Russ, 1997). In this section the details of what EM is and how it works will be described.

4.2.1 General Concepts and Principles

There are three key concepts in EM: observables, dependencies between observables, and agents which are acting through introducing or changing observables and dependencies. The definitions and characteristics of these concepts can be summarised as:

- An *observable* represents a element of the domain which can be perceived and identified, and whose current value in the model corresponds to the state of the referent. An observable can be either physical or abstract. The observable can only have meaning when referring to the perception and interaction of an agent with a feature of the subject or the model (Ness, 1997).
- A *dependency* is a relationship between observables which may have different interpretations through different viewpoint of agents. Any change to the value of a particular observable will cause changes to the values of other observables (called the *dependants* of that observable) in a predictable way. The changes to the observable and its dependants are indivisible.
- An *agent* here is a state-changing entity which is a family of observables and is capable of changing the states of observables and dependencies[1]. The agent in EM refers not only to the human

1. Generally the change to the values of observables is due to the actions of agents.

being but also to any component of the subject which is deemed responsible for changes of state. The *agency* represents the agent's responsibility or privilege for the state change.

The first step of the EM modelling process is to analyse the domain and identify the observations (i.e. the observables) which are relevant to the modeller. It is essential that the observations here not only mean visual observations but also the class of scientific observations which can in principle be made (Farkas et al., 1993). For situated modelling, the presence of observables can also be intermittent rather than persistent (Sun, 1999), that is, the observables can appear or disappear at any time in response to the current situation. After analysing the observables, we can recognise the dependencies between them and define the agents associated with these observables. We group the observables around the agents which act as sources of change in those observables. The appropriate classification of observables into agents is essential in the modelling process as it decides the behaviour of agents and the overall effect upon the state of changing any observable.

The shaping of change in the observables is through dependencies which are expressed by definitions. The dependencies in EM reflect how the change of the values of one observable will predictably and indivisibly change the values of its dependants. Chen et al. (2000a) identify characteristic EM dependencies as 'law-like' dependencies which operate either like the physical constraints such as Hooke's Law or Newton's Law or the abstract conventions of when a game has been won. Further the dependencies between observables can also invoke a *hierarchical* kind of state change as they can 'propagate' the effect of changing one observable to all relevant observables directly and indirectly dependent on it (Sun, 1999). All the observables, dependencies and agents identified are provisional and subjective as they reflect the viewpoint of the modeller. That is, the identification of agency and dependency is based on previous knowledge and past experience, as well as new observation and experiment.

Definitive Representation of States

The key theme of EM is to construct a computer-based artefact to represent a subject-domain based on the concepts mentioned above in a situated and open-ended manner. In EM the definitive notation as a simple programming notation provides the support to do this. The process of definitive representation of states includes representing observables by *variables* and introducing *definitions* to express the dependencies between observables. A definition takes the form:

```
q is f(x, y, z)
```

where `f(x,y,z)` is a formula similar to the formula used to assign a value to a variable in conventional programming languages. Here `q`, `x`, `y` and `z` are variables which represent observables in the situation. The variables `x`, `y` and `z` are defined elsewhere in the script and `f` is a operation (usually a user-defined function) which reflects the perceived dependency of `q` on `x`, `y`, `z`. The definition acts as a one-way dependency rather than a constraint. The values of variables 'metaphorically' represent the observed situation from the viewpoint of the modeller, rather than the mathematical abstractions of numbers in the conventional sense. The current values of variables should conform to the current observations of the corresponding observables in the real world. The value of the variable `q` (the dependant) is always equal to the evaluation of the function `f` with the current values of the variables `x`, `y` and `z` (the dependees)[2]. Any change of the value in a dependee will lead to a re-evaluation of the value of its dependant. The artefact constructed in an EM framework will have a collection of observables, and each of these observables has its own definition to represent the dependency. A collection of these definitions – a *definitive script* – corresponds to a single state of the model. Any state of the model should correspond to a state of its *referent* in the situation. The order of definitions in a script is not important as any redefinition of a variable will automatically re-evaluate all its dependents and bring them to a new state. This feature allows the modeller to be more relaxed and have more freedom in planning and developing the script than when writing a conventional program. The structure of the EM model based on the definitive representation of state can be summarised in Figure 4.1.

2. The '`is`' in the definition indicates that the dependency is maintained automatically.

The semantics of a definitive script is similar to that in a spreadsheet in which a user identifies the relationships between cells and any change to the value of one cell will automatically cause other dependent cells to update. It represents one state in the real world which can be immediately experienced (Beynon and Russ, 1994). Modelling such immediate experience in the EM manner makes our approach different from other programming methods as the latter are based on more abstract ways of describing the circumscribed behaviour of the proposed system. The main idea of spreadsheets – state change through dependency and agency – has not been made central in conventional approaches to programming (Beynon et al., 2002). In EM, the typical interaction of a modeller is to modify a definitive script by redefining a variable. The activity of on-line 're-programming' the model resembles 'what-if' activity in the spreadsheet. Also, as the values of variables in the script represent the observations of the referent in a particular state, our model acts in a similar role to that of an engineering prototype. Thus our approach is 'experience-based' because the models are continuously open for the modeller to re-program based on his new experience of the world as well as his experience of the current state of

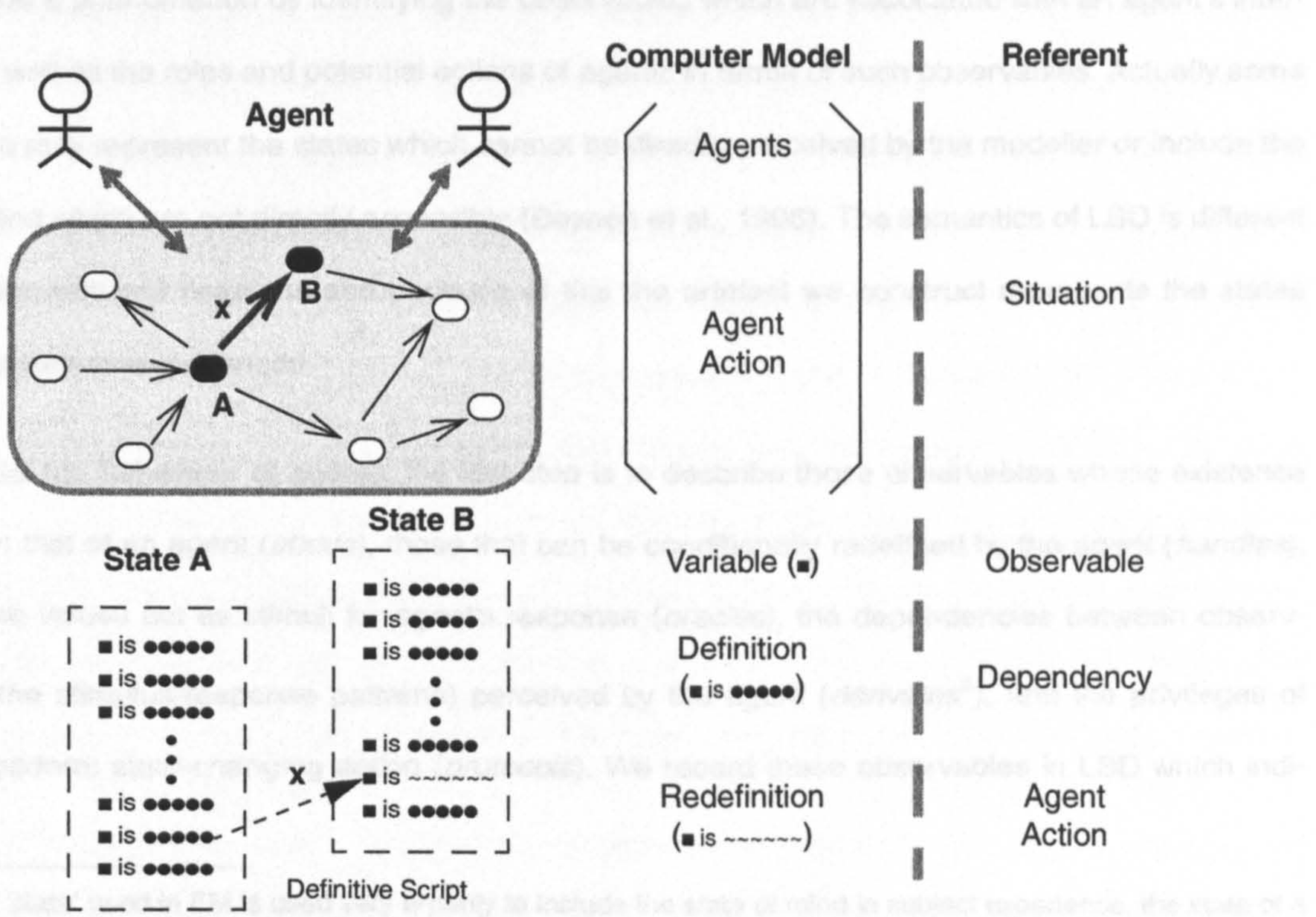


Figure 4.1 The Framework for the Modelling

the model. This can explain the primary focus of EM on state[3] in the sense of *state-as-perceived-by-agent* or *state-as-experienced* rather than the more conventionally abstract and limited sense (Chen et al., 2000a).

4.2.2 Software Tools for Empirical Modelling

The aim of EM is to enable the modeller to extend and experience his mental model of the real-world situation through the interaction with the computer-based model. The EM concepts mentioned above are supported by several tools which have been developed by this group.

The construction of EM models is based on the concepts of observables, dependencies and agents which are used to represent the modeller's viewpoints and interpretation of the domain, i.e. his experience of the domain. The modeller's beliefs and perceptions are then recorded and organised using a open-ended notation: *LSD* (Language for Specification and Description). The purpose of LSD account is to describe a phenomenon by identifying the observables which are associated with an agent's interactions, as well as the roles and potential actions of agents in terms of such observables. Actually some observables may represent the states which cannot be directly perceived by the modeller or include the states of mind which are not directly accessible (Beynon et al., 1996). The semantics of LSD is different to that of conventional notations and because of this the artefact we construct represents the states which are states-as-experienced.

To model the behaviour of agents, the first step is to describe those observables whose existence depends on that of an agent (*states*), those that can be conditionally redefined by the agent (*handles*), those whose values act as stimuli for agent's response (*oracles*), the dependencies between observables (i.e. the stimulus-response patterns) perceived by the agent (*derivates*[4]), and the privileges of agents to perform state-changing action (*protocols*). We record these observables in LSD which indi-

3. The term 'state' used in EM is used very broadly to include the state of mind in subject experience, the state of a computer artefact and the state of its real-world referent.
4. Sometimes a derivate may also represent an idealised relationship.

cates both the internal perception of agents as well as the external perspective of the modeller based on his personal construal of subjects. The identification and classification of such agents, agency and observables in LSD reflect the modeller's observation of both the model and its referent. Further, an LSD account generally allows different operational interpretations, which correspond to different presumptions about the environment for the interaction of agents, and the nature and reliability of their stimuli-response actions (Ness, 1997). The notation of LSD is informal but structured which is helpful throughout the construction process (like the role of use case models in the OOSE modelling process). LSD serves different purpose from conventional specifications as its interpretation is personal and its establishes a 'soft' boundary to be continuously developed, whereas the conventional specification is the 'culmination' of earlier drafts that establishes a hard boundary which is not intended for revision (Beynon et al., 2002).

The LSD accounts can be animated through the use of the Abstract Definitive Machine (*ADM*). As the redefinitions in a definitive script can represent the actions of different agents, in ADM the state transitions are represented by the parallel redefinition of variables in such a script. The ADM is the environment for the modeller to act as a superagent (in the role of arbitrator) to add appropriate scenarios for the actions and interactions of agents. In this sense the degree of agent autonomy in the simulation depends on the extent to which the ADM program is driven by human intervention rather than the automatic controls of perceived protocols (Beynon, 1994). Generally the modeller manually transforms an LSD account into an executable ADM program. Through the animation the interaction between the LSD agents can have operational meaning and the modeller has an unrestricted power to redefine the existing definitions or add new definitions. This means the patterns of interaction between agents within the artefact is entirely directed by the modeller (the superagent). Through this on-the-fly interaction with the model, the modeller can improve his understanding, and ensure the system behaviour modelled in terms of corporate behaviour of a set of agents is meaningful with reference to his external observation.

The tool *EDEN* (the Evaluator for Definitive Notations) is another interactive tool for realising and exploring the state changes consistent with the definitive script. EDEN is developed on the basis of the principles of EM and has so far been found to be the most successful tool for EM. Within EDEN there is

a window-based interface (based on the Tcl/Tk interpreter) which provides an interactive environment through which the modeller can at any time (1) create a new variable or modify the value of an existing one; (2) create dependencies by introducing new definitions or redefining existing ones; (3) observe the current values of variables and their definitions as well as the influence due to the modeller's intervention. There are two main features of EDEN: the scripts of definitions on variables, and stimulus-response actions triggered by changes to the values of variables in the scripts. There is a close correspondence between the LSD account and the EDEN animation: the derivates determine the definitive script and the script defines the state of the model; and actions are used to implement protocols. The visualisation of the model is realised using the definitive notations *DoNaLD*[5] (for the two-dimensional line drawing) and *Scout*[6] (for window layout) using the **`tkeden`** interpreter. There are three main windows in the screen display of EDEN (Beynon et al., 2000c):

- The EDEN *input* window: Through this the values of variables and their definitions can be modified and new definitions, functions and actions can be created to change the current state of the interactive situation model (ISM). The modeller can interact with the ISM by typing definitions line-by-line which represents the introduction of new observables and the redefinition of existing ones.
- The *interface* window: This window provides an interface for observations, visual feedback and interaction by the mouse. It can serve different functions according to the configuration of the script.
- The *commentary* window: This shows information about the current state of the ISM and the outputs of EDEN actions or requests by the modeller.

Within EDEN, **`tkeden`** is the interpreter to maintain the dependencies between all kinds of the internal data values as well as the dependencies between the values and other elements (geometric, textual or iconic) on the screen. As in spreadsheets, any change to the value of a variable will cause **`tkeden`** to automatically change all values that depend on that variable and re-evaluate their values. Through this

5. The Definitive Notation for Line Drawing.
6. The notation for SCreen LayOUT.

the dependencies are maintained as indivisible and propagated state changes which reflect certain state changes of the referent. The change to the values of variables can be achieved by either textual inputs through the input window or graphical inputs via the interface window. The construction of an ISM via the EDEN notation is situated rather than preconceived as no one can predict what definitions will be added or redefined by the user.

The environment of EDEN offers some limited assistance to the user for the management and debugging of scripts (Chen et al., 2000a). The notations of EDEN are mainly a tool for research purposes and are sufficient to explore the principles of EM. In this situation EDEN lacks the features of robustness, efficiency and consistency needed for large-scale applications for commerce or industry.

4.3 Modelling Activities in EM

EM is a form of interactive modelling which allows the modeller to construct an artefact through the use of computers. In the construction of, and interaction with, the EM model, the modeller can 'embed' the knowledge about observables, dependencies and agencies in such a computer-based model (Beynon et al., 1996; Beynon et al., 2000c; Ness, 1997; Sun, 1999) and gain a better understanding about the domain in which the system will be developed as well as the real-world situation.

4.3.1 The Modelling Process under the Definitive Notations

The modelling of EM and the construction of an ISM is summarised in Figure 4.2. The modeller must firstly build up a correspondence between the EM model (i.e. the ISM) and its referent. This process typically involves the identification of dependencies and the introduction of definitions into an ISM. The modeller here is broad including the role of an system user. In constructing the correspondence, the modeller initially identifies relevant agents, observations and protocols to form the partial LSD account. The LSD account can then be animated by the ADM or EDEN tool in which the definitions in the definitive script can be evaluated. The visualisation of the model can be established by DoNaLD and Scout

which represents the current state of such model. The modeller can intervene in the current behaviour of an ISM by modifying the values of variables or the dependencies and immediately experience the result. During the construction process, the modeller can make experiments and observations on both the model and its referent.

The association between the model and its referent consists of the correspondence between the states of the ISM and the states of the real-world referent (i.e. between the variables of an ISM and the observables of the subject). This correspondence is established by two correspondences: the one connects the modeller's mental model of the current situation (as expressed by knowledge of agency and

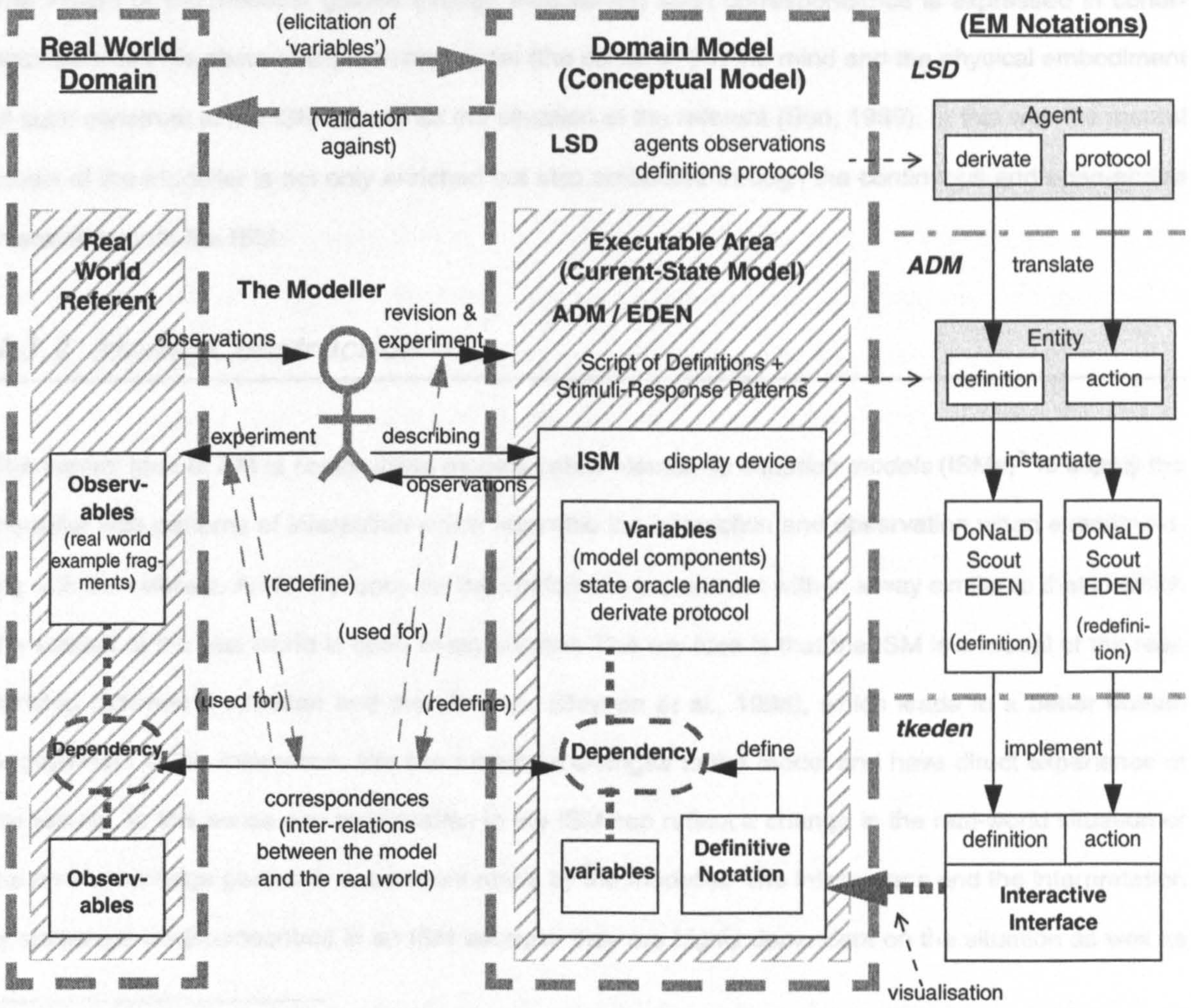


Figure 4.2 Empirical Modelling and its Tools and Notations

dependency, and expectations about interaction) with the ISM and the other one connects the mental model with its referent. These two correspondences are established by interactive activities such as the observations and experiments by the modeller. Any changes occurring in the model or the referent may affect the mental model of the modeller. The modeller can simulate different types of interactions with the real world by interacting with the ISM. Through modifying the dependencies in the model either by adding new definitions or refining the existing ones, the modeller can ensure and maintain the correspondence between the dependency in the model and the dependency in the external referent. The process of EM modelling enables the modeller to experiment with the effect of adding new elements into the system, experience parts of the system, and interact with the system in a unrestricted manner. The insight of the modeller gained through establishing such correspondence is expressed in coherence between his abstract explanatory model (the construal) in the mind and the physical embodiment of such construal in the ISM as well as the situation of the referent (Sun, 1999). In this way the mental model of the modeller is not only enriched but also embodied through the continuous and open-ended interactions with the ISM.

4.3.2 Model Construction

The central idea of EM is constructing models called *interactive situation models* (ISMs)[7] to supply the modeller with patterns of interaction which resemble the interaction and observation when experimenting with the referent. An ISM is open for the modeller to experiment with in a way similar to that in which the subject of the real world is open to experiment. The key idea is that the ISM is a model of the relationship between a situation and the observer (Beynon et al., 1998), which leads to a better human engagement in the interaction. We can introduce changes to the model and have direct experience of the results. In this sense any modification in the ISM can reflect a change in the real-world situation or the new knowledge gained or experiment made by the modeller. The interactions and the interpretation of states are uncircumscribed in an ISM because they are highly dependent on the situation as well as

7. The term 'interactive situation model' is introduced in this group as a synonym for 'model generated using EM principles and tools' (Beynon et al., 1998).

the knowledge and motivation of the modeller. That is, the construction of, and interaction with, an ISM take place in a situated manner rather than in the abstract manner of conventional approaches. The word 'situation' in 'interactive situation model' refers to this fact that the model is rooted in a concrete context that affects the modeller's expectation and interpretation (Chen et al., 2000a). The character of an ISM in this respect is similar to a spreadsheet where we may: (1) update the cells to reflect changes in the external situation; or (2) redefine the formula to reflect a new insight in the situation; or (3) introduce new cells and formulae for additional information or expectation about the situation. Both the EM model and conventional spreadsheet models are characterised by Chen et al. (2000a) as examples of ISMs because they incorporate the agency and dependency which are revealed during the analysis and construction processes. At the same time, the correspondence between the variables in a definitive script and the observables perceived by the modeller is maintained. To establish and maintain a correspondence between the real-world observables and the variables defined in the model is the essential principle of EM. This is achieved through the modeller's interaction with both the model and its referent in parallel. The process of establishing such a correspondence is iterative which involves continuously refining the model and making observations and experiments within the model and the referent. The aim of the correspondence setting is "to achieve consistency between the way in which sets of observables are indivisibly linked in change in the subject and the way in which the corresponding sets of observables are indivisibly linked in change in the model" (Ness, 1997).

The changes of state in ISMs occur through either the introduction of a new definition or the modification of an existing definition. A definitive script contains many definitions which represent indivisible dependencies. From a single redefinition may arise hundreds of automatic updates to other variables due to the relevant definitions. In the ISM, the state transitions are associated with agent action which is typically represented as a group of redefinitions to be executed in parallel. The modeller can act in the role of super-agent who can always intervene in any state by introducing definitions in an unrestricted manner. For this reason, the human and the automated activities in the ISM can be closely integrated.

There is no fixed and preconceived process of analysing a domain and constructing ISMs. That is, the modelling activity is open-ended and the modeller has flexible control over the redefinitions of the

model. Generally the changes of state in the ISM should reflect the expectations of the modeller but in reality observations may agree or disagree with expectations and therefore affect further construction. Usually an ISM presents only a partial and provisional artefact to the modeller (Chen et al., 2000a). This means the construction of EM models occurs in an experimental and incremental manner which reflects the modeller's knowledge of the domain. This process is also situated because the model reflects the personal experience and subjective viewpoint of the modeller by comparisons between experience with the domain and experience of interaction with the model. The distinctive characteristics of ISMs can be summarised as follows (Beynon et al., 1998):

- An ISM captures knowledge that is not task-specific. That is, the purpose of an ISM is to provide computer support for the modeller's conception of the referent and capture the knowledge about the referent by some metaphorical representation. Such knowledge here can broadly refer to anything the modeller can observe through interaction with the referent. It can be either the modeller's naïve observation about the current state of the referent or the modeller's expectation about its responses.
- An ISM represents subjective knowledge. That is, an ISM reflects the perception and experience of a particular person to a particular referent within a particular situation. Different people may have different viewpoints of the same model and sometimes these viewpoints are difficult to combine in system development due to their different concerns. For example, the user, analyst and implementer of a system have different application-specific knowledge. How an ISM can be the general framework for the representation and use of these differing perceptions is detailed in (Beynon and Russ, 1994).
- An ISM represents knowledge about interaction in particular contexts. That is, an ISM generates a direct metaphorical representation of a particular state of the referent. By interacting with an ISM the modeller can get immediate feedback of the consequences of some state-changing actions. This differs from conventional computer models as they represent objective knowledge about the intended functionality of the system and modes of user interaction for a specific goal.

Computer-Based Model as Artefact

The essence of an artefact is that it should imitate the interaction with the referent in the real world. Simon (1996) gives the definition of artefact in terms of form (inner environment), context (outer environment) and purpose:

> An artefact can be thought of as a meeting point – an 'interface' in today's terms – between an 'inner' environment, the substance and organisation of the artefact itself, and an 'outer' environment, the surroundings in which it operates. If the inner environment is appropriate to the outer environment, or vice versa, the artefact will serve its intended purpose. (p. 6)

According to this definition, an artefact is a kind of abstract boundary between the inner and outer environments, i.e. the form and the context. However such an abstract interface does not exist in the real world, it is represented in our mind or by artefacts which are constructed to represent knowledge (Ness, 1997). In EM we broadly define an artefact (or cognitive artefact) as an object or environment which is constructed for interaction which imitates interaction with the real-world system (Beynon and Cartwright, 1995). This means that an artefact should exhibit different states which correspond to the external states in the real world. Also it should provide an environment for modeller to interact in an open-ended manner and explore the correspondence between the artefact and its referent. Thus two kinds of artefacts in the design process are (Beynon and Cartwright, 1995; Cartwright, 1998):

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

One theme of our research is to find out how the computer can best be exploited in the construction of artefacts. Following a convention that is often useful in EM, the term 'computer' here refers to any reliable, interpretable, state-changing device (Beynon et al., 2002). The environment in which a computer operates is important for the interpretation of computer behaviour. This is the reason we refer to the construction of artefacts as *modelling* rather than conventional programming as the process includes

not only the development of algorithms and scripts but also the configuration of human agents and other resources within the system. The computer in the EM modelling process serves as a physical instrument with which the modeller interacts. This differs from the conventional way in which the computer acts as a means to implement an abstract algorithm or computation. In this respect, Beynon et al. (2001) suggest that the experience of constructing ISMs is close to the characteristic of an *instrument* as 'maintaining a relationship between aspects of states'. This is in contrast to the role of the computer in traditional software development as a *tool* which is developed to serve a particular purpose. The characteristic difference between instruments and tools also reveals the different emphasis of artefact use: in the tool-like use of artefacts, the concern is with the efficient and reliable progress to the goals; whereas in instrument-like use of artefacts, the focus is on the experience rather than the achievement of the goal. For this reason, constructing an ISM can be regarded as using an interactive instrument, as the construction is a situated activity that can be developed in an open-ended manner which involves exploration and experiment.

Ness (1997) shows in his thesis that the characters of EM and conventional system development are determined by the nature of artefacts used. That is, the nature of modelling activities may be different according to the kind of artefacts used by modellers. The following summarises some aspects of modelling activity that may be influenced by the nature of artefacts.

Artefacts The artefacts are used in system development to represent the structure and functionality of the proposed systems. The artefacts of EM correspond to the subject; but in conventional methods the artefacts correspond to abstract conceptual models of the subject. For example in OOSE (Jacobson et al., 1992), the use case model is constructed to specify the functionality of the system. The analysis model and design model are both used to represent the structure of the system: the former focuses on the logical structure of the system (under the ideal conditions) in terms of objects, whereas the latter refines and formalises the analysis model to meet the actual implementation environment. In contrast, the LSD account and the animation of an ISM resulting from `tkeden` interpreting the EDEN and DoNaLD scripts form the artefact during the modelling process of EM. The LSD reflects the observables and agents which are perceived by the modeller in the subject. Such representation ena-

bles the modeller to construct the model in an intuitive way without detailed knowledge of the structure and function of the proposed system (Ness, 1997). The modeller can observe and experience the function of the system by interacting with the model. The animation of an ISM represents the function of the systems and the users can animate the artefact by adding new definitions or redefining the definitions in the scripts in an experimental way.

Subjects It is argued by Ness (1997) that whether the subject (i.e. the system to be modelled) is novel or familiar to the modeller will essentially influence the suitability of a particular kind of artefact for representing the subject as well as the activities in the modelling process. The ISM seems to be more suitable for representing subjects which are novel to the modeller. As the ISM is a computer-based model of situations, it can be used to deal with many different subjects and Sun (1999) infers that such subjects are typically an 'intelligence-intensive' process (or a soft process) such as software system development or requirements understanding. By way of contrast, the artefacts in conventional modelling methodologies are more successful in representing the subjects more familiar to the designer.

Actions The actions in conventional system development are transformational in nature, whereas in EM the actions are generative and exploratory in nature (Ness, 1997). For example in object-oriented methods, the designers begin by identifying the roles of the proposed system and the correspondence between models and the subject is preconceived through imagination. It is transformational because the construction of models is transformed from the completion of the models in previous stages, for example, transforming the nouns or verbs in the requirements specification (or the use cases) into the analysis model and the transition from OOA to OOD. But as we showed in section 2.2, there are difficulties when integrating these models by applying the concept of objects. In addition, the designer may need to spend more and more time to modify the models whenever inadequacies are found in the artefacts. The actions in EM are generative and exploratory because the modeller typically generates the artefact firstly and then evaluates the artefact with the subject. Ness (1997) describes how the refinements (i.e. adding new definitions or redefining existing ones in a definitive script) reflect a better understanding of the subject by the modeller. The validity of the model can be ensured as its representation can directly correspond to the subject perceived by the modeller through observation and experiment.

Knowledge We have summarised two processes involved in the knowledge manipulation for system development in section 2.3: knowledge construction and knowledge representation. Conventionally the knowledge used by designers in system development comes from either their long-term memory or the experience of interacting with the subject. Thus most modelling methodologies focus on knowledge representation and their artefacts are established mainly to record such abstract conceptual mental models in a 'context-free' manner. In contrast, the focus of EM is knowledge construction[8] in which the modeller constructs their knowledge in a situated manner. The ISM is an interactive and open-ended artefact which not only represents but also enriches the modeller's knowledge (his construal) by knowing-by-doing.

4.3.3 The Comparison with Other Modelling Methods

The nature of modelling activities may be different according to the nature of the artefacts used. This is because the way the modeller interacts with the state of the model in EM is different from how this happens in conventional modelling methods. It is common in many conventional mathematical models that features of the application domain are represented early in the process by abstract objects and operations. This means it is necessary to preconceive the possible future states for the model and the proposed system. For example using object-oriented methods, the objects, containing attributes and methods, need to be defined and carefully planned in advance before used in program. These attributes form the *data* for manipulation in the computer model and such data will determine the complete state space (i.e. the operational domain) of the system and all possible interactions with it. Such a program can be edited and re-compiled but it still remains an inherent feature of the paradigm that the program forms a 'boundary' within which the state and interaction must be preconceived (Chen et al., 2000a). The mathematical model expressed in the programming cannot be formulated without knowledge of this boundary. The human interpretation of program state is therefore restricted by this boundary and involves an association between real-world observations and the abstractions of this program. It is obvi-

8. Thus the main modelling activity of EM is the construction of the model (the ISM).

ous that the computer's manipulation of data is affected by such interpretation and major problems may occur when exceptions and errors appear.

In EM the description of state is by 'observables' through which the state of the EM model is presented to the modeller by a perceptual process as the way in which we typically perceive and interact with the real world. It is suggested by Chen et al. (2000a) that there exists a *comparability* and *connectedness* between observations of the model and observations of its referent. Such comparability is absent in conventional modelling methods because there "the 'observation' of the model is the reading of an abstracted value, or the preconceived interpretation of a pre-programmed display". That is, even though observations can be performed in conventional modelling, such observations are intended to identify elements (such as objects or classes in object-orientation) whose nature is strongly associated with a particular situation (the ideal situation). These elements are preconceived and context-independent (i.e. isolated from the modelling process), and observation is described in terms of them. In contrast, the observation in EM is a subjective and situated experience which refers to the modeller's ability to apprehend the features of a particular situation directly (Sun, 1999). Without this key feature of observation, the modelling activities in EM will not be differentiated from conventional modelling, where the modeller relies on imagination or conjecture whilst lacking both comparability and connectedness.

The Key Features of EM Modelling Process

The activity of observation plays a significant role in the EM modelling process as through observation, the modeller can not only construct the current situation but also enrich his knowledge and understanding for dealing with future situations. We argue that there exist comparability and connectedness between the observation of the EM model and the observation of the referent. There are three main features of the EM model that can illustrate this issue. Firstly, the correspondence between the observable of the referent and the variable in a definition is direct and simple. By this, the referent can be metaphorically represented by the model. This is in contrast to many conventional programs in which the interpretation of variables and their states can be highly abstract and problematic. However the variables in a definitive script still have the 'abstracted' quality connected with conventional programs. But we view

this as a 'limited abstraction' in which the connection between the concrete and the abstract is deliberate and familiar just like the use of natural language for describing the world (Chen et al., 2000a). Of course from the modeller's viewpoint, a definition is functioning as recording a dependency between the observables. But in the other hand, from a computational perspective such a definition has an abstract semantics similar to a formula in a spreadsheet.

The second feature is that the observation of the model can be performed through a visualisation implemented in definitive notations which is indivisibly linked to the script of the model. This visualisation is implemented using the **tkeden** interpreter which maintains the dependencies in spreadsheet-like manner. This ensures the visualisation of the model metaphorically represents its subject and provides a better virtual environment for interaction and rapid prototyping. This is so-called 'readiness-to-hand'[9] by Winograd and Flores (1987) – because such definitive scripts are within the 'definitive world' – which provides the right coupling between the modeller and his action in the relevant domain. Chen et al. (2000a) describe the nature of such visualisation as implicit: "just as the existence of a physical edge in the field of vision of a person with normal faculties is automatically accompanied by its perception".

The third feature, which combines the directness of the first two features, is the quality of interaction offered by the EM model. There is no preconceived constraint for the modeller to interact with an EM model and revisions can be made at any time. Like a experimenter, the modeller may not be aware of, or predict, what action will affect the states of the subject. Also there are some consequent changes which may automatically arise due to dependencies rather than from human actions. The interactions of a modeller with the referent may be different because his knowledge of the observables and their dependencies can be changed. In system development, we find that the essential need for interaction between users and the system makes it difficult to develop a system under a closed-world paradigm. This is because experience is involved in the interaction between users and the systems. A user may want to interact with the system in ways beyond the scope prescribed by its designers. This is similar to the situation of driving a vehicle when the purpose of the interaction is to explore the engine's full poten-

9. 'Readiness-to-hand' means one object is a part of the background which is taken for granted without explicit recognition as an object. They give an example that to a person doing hammering, the hammer does not exist because it is a part of the hammerer's world.

tial or experiment with new driving skills. Prescribing the functionality of the system, or the interactions between users and the system, will inevitably restrict the experimental opportunities and openness which is available for the users. As the EM model acts as an instrument, the interactions offered in an EM model allow the modeller to introduce and test new scenarios on-the-fly and support 'what-if' experimentation. When several modellers involved, this guarantees a close and continuous engagement among participants.

Where interaction with the state of models is concerned, one fundamental difference between EM and conventional modelling is the way the changes of state take place. With a conventional approach, the modeller must plan and preconceive change methods, i.e. the inputs and outputs before model construction in order to prescribe the process. That is, the control of change is 'handed over' to actions within the program boundary (Chen et al., 2000a). When such actions are complex or additional system features (new inputs, outputs, etc) are required, it may be hard to make the appropriate changes and the process of revision and redesign will be costly. In the EM model the change of state is only achieved through automatic dependency updates or the direct action of an agent. Only when a pattern of state change has been experimented with and shows to operate reliabily, they can be delegated to an 'automated' agent action. This represents the progression from a subjective to an objective view of the system, starting from 1-agent modelling (corresponding to the modeller's view of system), then n-agent modelling (patterns of state change are identified and methods of communication are developed), and finally an 0-agent system is established whose meaning is independent of the modeller and the subject (Ness, 1997). This is why we choose the term 'empirical' to reflect the characteristic of our approach. Without such experimentation, the modelling process will become the reliable patterns of imagined state change as in conventional modelling.

The patterns of state change in observables are identified in EM through observation and experiment, which are also subject to revision within the subsequent observation. The observations of the EM model resemble the ones of the real-world situation. This is due to the interface of our model being for the purpose of the experimental interaction by the modeller in order to experience the development process, rather than just representation that the modeller views as a snapshot of the referent. Thus the

process of EM modelling involves incremental construction of artefact and modeller's knowledge which can be done by evolving the knowledge about the properties of the referent. The EM interface includes the EDEN input window in which new definitions can be entered, so the interface window (visualised by DoNaLD and Scout) can be used to animate the results or create an effect by introducing new definitions.

Conventionally the process of system development consists of three essential stages: requirements, design and implementation. However as discussed in chapter 2, this paradigm considers a serial rather than a concurrent approach to system development. All these stages are isolated from each other and normally failures will not appear until the implementation phase. The modelling process of EM provides an integrated environment for these stages in a development. That is, the EM modelling is a single 'process' rather than the traditional lifecycle and the same model can be used for understanding requirements of a system and its subsequent development. Thus in EM there is no requirements phase at the beginning of the development because such conceptual modelling is embodied in a script with a visualisation and can be experimented with by the modeller. Also, EM allows the on-line intervention for the modeller to experiment and establish reliable components which in some way resembles the testing task in conventional modelling. But such 'testing' in EM can be done earlier in advance of the whole application with the minimum of additional work and enable us to respond rapidly to the future evolution.

Finally, the interface of EM models serves the role of rapid prototyping which offers a 'mock-up' of a interface to the future system at an early stage (Chen et al., 2000a). This is similar to the use of an engineering prototype which is a physical object with characteristics similar to those of the actual system to be constructed. In other words, the process of EM modelling is the construction of a 'virtual prototype' (Beynon et al., 1996; Cartwright, 1998). The use-case driven approach OOSE (Jacobson et al., 1992) also has a similar user-interface prototype before developing the object model for the system (cf. p. 351), where the use case serves as an interface description to identify what information should be held and how it should be used. Typically in EM the design of interface is left until the purpose and requirements of the system have been clarified through the construction and exploration of the EM model.

4.4 Concluding Remarks

In this chapter we have described EM as a novel approach to modelling in a situated and open-ended manner. Based on the concepts of observable, dependency and agency, the process of EM modelling is the construction of an artefact, or a computer-based model, in a way similar to the way in which humans form a mental model of the real world subject. The primary focus of EM is the use of computer-based interactive situation models (ISMs) to represent the way in which systems are constructed in terms of these concepts. Thus far EM has been applied in artificial intelligence (Beynon, 1998), educational technology (Beynon, 1997), concurrent engineering (Adzhiev et al., 1994), software system development (Ness, 1997; Sun, 1999; Beynon et al., 1998; Beynon et al., 1999), geometric design (Beynon et al., 1996; Cartwright, 1998) and requirements engineering (Beynon and Russ, 1994; Sun et al., 1999; Sun, 1999). The ongoing research of applying EM includes decision support systems (Beynon et al., 2002), program comprehension (Beynon and Sun, 1998; Beynon et al., 1999) and BPR (the AMORE project).

To sum up, conventional modelling which formalises the development process and preconceives the possible states in the operational domain may be highly efficient at developing systems. But it may also restrict the flexibility of modellers in choosing alternatives and their openness to new concerns. In contrast, EM modelling aims to achieve *effectiveness* – measured by the overall outcomes of such a process rather than the speed or immediate results during the process – for the whole development. Further when broadly considering deploying computer systems into organisations, it is suggested by Winograd and Flores (1987) that the resulting design of systems, if concentrating on its efficiency, can erode the effectiveness of the organisation.

The potential application of EM to BPR is the current focus of this thesis and of work elsewhere (Chen et al., 2000a; Chen et al., 2000b). There are other methodologies for BPR, such as the process approach by Warboys et al. (1999) and the Object Advantage by Jacobson et al. (1995). Such methodologies have some characteristics similar to our EM approach, but there are some potential problems in

modelling business within them. As EM has been shown as a suitable approach to software system development and a tool for decision support and business application, we shall make a comparison between EM and other approaches for BPR in the next chapter to establish the link between system development and BPR. Based on this background, the framework of applying EM to BPR can be developed and will be detailed in the next few chapters.

CHAPTER FIVE

Empirical Modelling, Object-Orientation and Use Cases

Over the last few years, use cases have become popular as one of the fundamental techniques of object-oriented analysis. This is partly because they capture the system requirements in a simple (informal) and readable format and such a technique is easy to learn. Also use cases describe the requirements from users' perspectives and define who is interacting with the system. There are some similarities between the use case approach and the EM approach, for example, both of them are human-centred and the issue of scenarios is considered within both approaches. We have illustrated the fundamental principles and concepts of EM in chapter 4. In this chapter the details of the use case approach to system development (OOSE) and business process reengineering (Object Advantage)[1] will be discussed and we will show how the EM approach differs from the use case approach.

5.1 Introduction

Object-oriented programming and design have been around since mid-1960s but their realisation in both industry and business only started from the mid-1980s. The notation UML (Unified Modelling Language) – a third-generation object-oriented modelling language – was developed to combine the main object-oriented methodologies and is today accepted as a standard methodology in the OO society[2]. It

1. The concept of use cases is originated from Jacobson's *Objectory*, the process of his use case approach. The use of his approach is detailed in his three books: *Object-Oriented Software Engineering* (Jacobson et al., 1992); *The Object Advantage* (Jacobson et al., 1995); and *Software Reuse* (Jacobson et al., 1997).

combines the efforts from the main OO methodologists (Grady Booch, Jim Rumbaugh, Ivar Jacobson, etc) to construct a common means of using OO concepts for describing and modelling systems. However, UML is the language for software design and is used by developers. Its early version was not well suited for communication between designers and users. For requirements engineering, it is essential that both the users and developers, or the analysts and the domain experts, can communicate with each other and users can be involved throughout the process of system development. This is the reason that the concept of use cases became popular as it addresses such issues and it also bridges the gap between OOA and OOD[3]. Thus most major OO methods added the concept of use cases and recently Jacobson's use case approach has been included in UML to capture the higher level of user functional requirements of the system.

Ivar Jacobson is regarded as the inventor of use cases. In his use case approaches (e.g. Jacobson et al., 1992 and 1995), the Use Case Model created in the analysis process serves as a basis for the models of the system developed subsequently. In this chapter the general concepts of OOSE and Object Advantage are firstly introduced. The potential problems of applying use cases in system development are also explored. Then the comparison between EM and OOSE will be discussed.

5.2 Object-Oriented Software Engineering (OOSE)

The original technique of Jacobson's use case approach is *Objectory*, a process of system development[4]. It was first used in 1987[5] and its fundamental ideas are described in OOSE (Jacobson et al.,

2. The UML was originally developed in response to the Object Management Group's (OMG) call for a proposal for a standardised object modelling language. The OMG web site can be found at http://www.omg.org (20 December 2001).
3. Another reason, according to Dano et al. (1996) and Firesmith (1998), may be that use cases corrected the overemphasis of previous structured methods and early OO methods on the static architecture but ignored the dynamic behaviour issues during the requirements analysis.
4. Objectory was also the name of a company found by Jacobson in Sweden, which was merged with Rational Software Corporation in 1995.
5. At the 1986 Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '86), Jacobson presented his ideas of use cases on the design of large, real-time systems. And in the 1987 OOPSLA conference, Jacobson described his Objectory as a technique and use cases as a tool used within that technique (Firesmith, 1998).

1992). As claimed by Jacobson et al., use cases are an effective way of capturing both system requirements and business processes. OOSE consists of a number of processes which result in different models, each of which captures a specific aspect of the system. We shall summarise these processes and define the models created during different processes.

5.2.1 General Concepts and Modelling Process

The lifecycle of OOSE consists of three main processes: Analysis, Construction and Testing. The aim in the Analysis process is to create a conceptual model of the proposed system in order to understand the system and enable the customer to communicate the essential aspects of the system. The Construction process includes both design and implementation which results in a complete system. The activities in the Testing process include integrating and verifying the system. The activities in each process produce models: Analysis results in the Requirements Model and Analysis Model, Construction in the Design Model and Implementation Model, and Testing in a Testing Model. The processes and models in OOSE are summarised in Figure 5.1.

Requirements Analysis

The aim of the requirements analysis process is to analyse, specify and define the system to be built. As it is essential for the clients to ensure the system to be built is what they want, the models built in this process intended to make it easier for them to understand the system. The first model to be developed is the *Requirements Model*, which consists of a Use Case Model with interface descriptions, and a Domain Object Model. The *Use Case Model* is based on actors and use cases. It specifies the complete functional behaviour of the system by defining what exists outside the system (actors) and what should be performed by the system (use cases). A *use case* consists of structured textual description of a specific way of using the system (a case of usage). Each use case contains a complete sequence of related transactions performed by some actors and the system. The collection of use cases are the complete functional requirements of the proposed system. The *Domain Object Model* – which consists of objects representing entities from the problem domain – is used to communicate with the potential

users and to ensure a stable ground for the use case description[6]. It is essential in OOSE that the use cases are the core through all modelling processes, helping in the construction of other models and maintaining the focus on the problem.

Robustness Analysis

The Requirements Model describes the functional requirements specification based on the needs of the users. Once the Requirements Model is developed and has been approved by the users, we then start with the *Analysis Model*. This model defines the 'logical structure' of the system independent from the implementation, i.e. assuming an ideal world for the system. It is developed during the robustness analysis to be oriented to the application and not the implementation technology. Thus the Analysis Model is

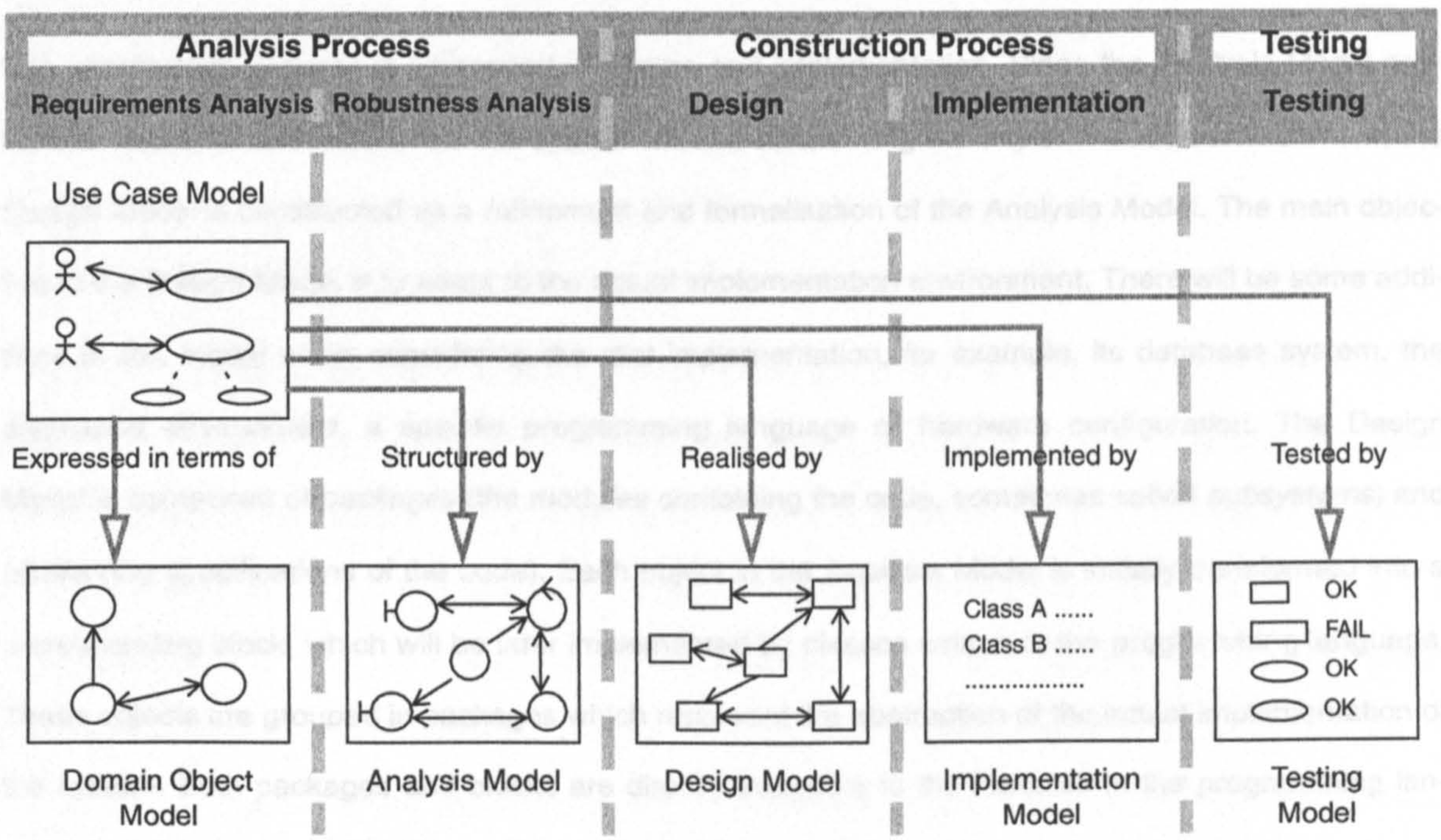


Figure 5.1 The Process and Models in the Use Case Driven Development

6. Jacobson et al. (1992) differentiate between OOSE and other OO methods by observing that many object-oriented methods focus entirely on such problem domain models. In these methods this model forms a base for the actual implementation, i.e. the objects are directly mapped onto classes during implementation. But in OOSE, the Analysis Model is claimed to be more robust and maintainable for the future changes, rather than serving the problem domain model as the base for design and implementation.

the architectural model (a conceptual configuration of the system) used for analysis of robustness[7]. Three kinds of objects are used to separate the control of use cases from their interface behaviour and from entities in the area of the application:

- The *entity objects* are used to model information which the system will handle for a period of time.
- The *Interface objects* model the behaviour and information which is directly dependent on the environment of the system. The task of interface objects is to translate the actor's actions with the system into events in the system, and vice versa.
- In some complex cases, there exists behaviour which cannot be placed in either interface objects or entity objects. Such complex behaviour is placed in *control objects*[8].

Design

The construction process is composed of design and implementation. Since the Analysis Model contains a functional specification of the system without considering the implementation environment, the *Design Model* is constructed as a refinement and formalisation of the Analysis Model. The main objective of the Design Model is to adapt to the actual implementation environment. There will be some additions in this model when considering the real implementation, for example, its database system, the distributed environment, a specific programming language or hardware configuration. The Design Model is composed of *packages* (the modules containing the code, sometimes called *subsystems*) and *blocks* (the specifications of the code). Each object in the Analysis Model is initially transformed into a corresponding block, which will be later implemented by classes written in the programming language. These objects are grouped in packages which represent the abstraction of the actual implementation of the system. Both packages and blocks are directly traceable to the modules in the programming lan-

7. The reason for developing the Analysis Model, according to Jacobson et al. (1992), is to ensure that even though many future changes are coming from the changes in the implementation, these changes will not affect this logical structure, thus making this model more robust and maintainable throughout the development lifecycle.
8. Such behaviour is typically transaction-related behaviour or control sequences specific to one or a few use cases, which normally consists of operating on several entity objects, doing some computations, and returning the result to an interface object.

guage, and the Interaction Diagrams – which describe how the use case is realised through the interaction of blocks (through sending/receiving stimuli) – can be traced back to the use cases directly.

Implementation and Testing

Finally, in the implementation process, the code is written from the blocks in the Design Model. The Design Model forms the basis for implementation as it specifies the interface of each package and block and describes their behaviour behind the interface[9]. The Testing Model is developed during the Testing Process. Normally the testing activities include two main works: *verification*: checks whether the results meets with the requirements specification ("are we building the system correctly?"); and *validation*: checks whether the result is what the customers want ("are we building the correct system?"). There are three levels of testing during this process: in the *unit testing* only one unit is tested; the *integration testing* is to verify that these units are working together properly; and finally in the *system testing* the whole system or application is tested.

5.2.2 The Use Case Concepts

The main concepts in Use Case Models are actors and use cases. A use case may have different alternative courses depending on different circumstances of using the system. A single sequence of events which represents the realisation of the use case is called a *scenario*. Hence a use case is a set (or class) of related scenarios. The content of a use case typically can be divided into a basic course and some alternative courses. The basic course, as Jacobson et al. (1992) suggest, is the course which gives the best understanding of the system, but is not necessarily the one which is executed most often. Use cases are related by two associations, *uses* (or *Includes* in UML 1.3) and *extends*. The 'extends' association specifies how one use case description inserts (extends) itself into another use case description which is independent and unknowing of the former one. So 'extends' can be viewed as a kind of 'inheritance' between use cases. The 'uses' association can be viewed as a kind of 'aggregation'

9. Normally a block corresponds to one class, which makes it easy to map the block interface to a specific class interface in the code.

because it captures how more use case descriptions incorporate the common description of another use case[10].

Jacobson et al. (1992; 1995) stress that use cases should be used throughout the system development lifecycle and point out that use cases can be an effective tool for business process modelling and BPR. They define each use case as a specific way of using the system and each execution of the use case is viewed as an instance of that use case. In general, the motivation for using the use case technique may include gaining an understanding of the problem and the proposed solution, and identifying the candidate objects for the analysis and design. The use case technique is claimed to help to manage complexity because it decomposes the problem into some major functions (i.e. use cases) and focuses on one specific aspect of usage at one time. It is different to other techniques for capturing requirements as use cases treat the proposed system as a black box and it enables designers and users to see the implementation decision easily during the requirements analysis. McBreen (1998) describes the importance of not considering the internal structure of the system during the requirements analysis because specifying the internals will put extra constraints on the designers. Without these constraints, the designer can have more freedom to develop a system which correctly implements the external behaviour, and some 'breakthrough' solutions may arise from this.

5.2.3 The Object Advantage

The Object Advantage is a work by Jacobson et al. (1995) for business modelling which uses the same type of techniques as in OOSE. The *Use Case Model* is used as an external model which describes a company and its environment. It describes the processes in that company which satisfy the customers and also includes other relevant interests outside the company. As an external model, the Use Case Model only shows how the company is perceived by the people who will use it but not the structure of

10.As people are always confused about these two associations (cf. Simons, 1999), Firesmith (1998) compares these with an example: "if A *extends* B, then extended B *contains* A; whereas if A *uses* B, then A *calls* B".

that company. This is why Jacobson et al. (1995) say the Use Case Model is a 'what model' whereas the Object Model[11] is a 'how model'.

The company is made up of relevant business processes. These processes are modelled as *use cases*. The *actors* in the Use Case Model represent the environment, which includes the customers and other relevant entities such as suppliers who take part in or interact with the processes (the use cases). That is, actors – which can be human or mechanical – are used to represent everything in the environment which interacts with the company. Through the Use Case Model, we can find out how the external environment interacts with the company and how actors communicate with use cases in the company.

The *Object Model* serves as an internal model which describes each business process as defined in each use case. It shows how the processes are built up of different tasks and things, as well as the resources used during the process or the products after the processes. There are also two types of Object Model: Ideal Object Model and Real Object Model. The former is the model of the company which does not consider the implementing factors in practice, for example the human factors or the environment; whereas the latter will take these factors into account. Three types of objects are used in the Object Models: the entity objects (represents products and things handled in the business), the interface objects and the control objects (both represent the tasks in the business). The *interface objects* represent the tasks which are performed by one resource (including the human being) and that involve communicating with the external environment[12]. The *control objects* represent the tasks which are performed without directly contacting with the environment.

Reverse Business Engineering and Forward Business Engineering

The work of reverse engineering involves the design of a model of an existing company. This model aims to serve as a basis for the reengineering work in the forward engineering of the company. Two models are developed during the reverse engineering: the Use Case Model and the Object Model. The

11.The Object Model in the Object Advantage corresponds to the Analysis Model and Design Model in OOSE.

12.Thus the part of the business which has direct contact with the outside world is visible in interface objects; whereas entity and control objects are more independent of the environment.

Use Case Model describes the processes of the existing business in terms of actors and use cases. It is used as the basis for prioritising the processes to be reengineered. The Object Model gives the internal view of how the existing business operates. Most companies are structured in a functional or hierarchical way, which can be modelled by a subsystem for each functional unit or department. The reasons for reverse engineering, as described by Jacobson et al. (1995), may include: (1) to enable the designers to obtain a common understanding of what is not working well in the business and thus identify the areas to be reengineered; (2) to understand how to proceed in order to change the existing business so the objectives and requirements can be satisfied; (3) to describe to the employees why the existing processes are not adequate and why a new way of working is needed; (4) the models can be used later to evaluate the benefits of the proposed changes. The analysis of the models of existing business can help to find the way to achieve a more effective reengineering of the company.

After identifying the existing processes which need to be reengineered during reverse engineering, the next step is to find out how the new redesigned processes will affect the existing business and determine how to carry out the improvement and reengineering. Three models are developed during this forward business engineering: the Use Case Model, the Ideal Object Model and the Real Object Model. The Use Case Model represents the external view of the new company. The Ideal Object Model, based on the use cases, shows the inside view of the new company in terms of objects and the way they interact with each other. This model contains only the objects needed to perform the use cases. The Real Object Model adapts the Ideal Object Model to the real environment with modifications. Some issues or restrictions are considered in the model, for example, the human factors or the structure of the company (distributed or centralised). According to Jacobson et al., the work of developing the supporting information systems for the new business can be done in parallel with the modelling activities and, as they depend on each other, both (the model of the business and the model of the systems) should be completed at the same time.

Object Advantage and OOSE

Both the OOSE and the Object Advantage are introduced by Jacobson et al. (1992 and 1995) for modelling computer systems and business systems. As IT is the main enabler for BPR, the business process models and computer systems models will be highly dependent on each other. It is essential to emphasise the relationship between BPR and system development in order to develop the support system to meet the need of BPR. Jacobson et al. claim that in their methodology for BPR it is easier to express such relationships because they are using the same type of techniques.

In OOSE, the computer system to be built is modelled as a system which contains use cases (its functionalities) and has some actors outside which interact with it. But in business modelling, software development is viewed as a business process in a company. Thus it is represented as a use case in the business model. The actors (of the use case model in OOSE) of such a system are thus modelled as objects (interface object or control object) in the business model which means the users or resources of the system. Each model in OOSE, such as the Use Case Model or Analysis Model, is represented by an object (entity object) in the business model. Each stage in the development lifecycle, such as Requirements Analysis, Design, or Implementation, is modelled as a subsystem. Each of the subsystems represents the sub-flow of the use case Software Development. More details of how to define objects in the business model which correspond to the actors, use cases and objects in OOSE models can be found in chapter 9 in (Jacobson et al., 1995).

5.2.4 Some Potential Problems of Applying Use Case Approach

Recently use cases have become a fundamental part in many OO methods. However like many system development methodologies, there are some problems arising when using use case techniques in developing a system. This subsection provides an overview of these problems.

Functionality Orientation vs Object-Orientation

The Use Case Model in the requirements analysis is developed from a system perspective because it defines and captures the major functionality of the proposed system. Thus use cases are system-oriented (or functionality oriented) rather than object-oriented. Jacobson (1994) also admits that:

> It should be clear that use case modelling is a technique that is quite independent of object modelling. Use case modelling can be applied to any methodology structured or object-oriented. It is a discipline of its own, orthogonal to object modelling.

The functionality orientation of use cases can enable users and designers to define or understand the system (or the business when applied in BPR) easier by using functional descriptions than using data-centred descriptions such as object models. However this can cause some problems when applying two different paradigms in system development and business modelling. For system development, this kind of functional decomposition into use cases may compromise the benefits of object-orientation and cause the problems which OO was intended to avoid. Also each use case represents one functional partition which may involve different features of objects or classes; and individual objects or classes may be involved in multiple use cases. Firesmith (1998) describes how the decomposition based on use cases scatters the features of objects and classes among the use cases. In this situation, the designers may have to check all the partitions to make sure the objects are designed accurately. This can also result in problems in system integration, because different use cases (partitions) may be implemented by different designers or teams. The multiple or redundant objects and classes that will be designed decrease the productivity and reuse of software. Even the same objects in different parts of the system will have different implementations. This will result in significant effort and time in redesign.

Another potential problem is the mapping from the Use Case Model to the Object Model. In Jacobson's OOSE the use cases are directly used to create the Analysis Model which describes the structure of the system in terms of objects. As we can see the Use Case Model and the Object Model belong to different paradigms and thus use different concepts, techniques and notations. This process may be problematic because we present a set of functional requirements but aim to produce an OO solution.

Further the structure of the Use Case Model does not clearly map to the structure of the Object Model and the requirements trace from the use case to the objects or classes is not one-to-one. Such mapping in OOSE is informal and arbitrary, with little guidance as to the identification of objects and their interactions.

No Clear Definition

It is argued that the use cases are poorly defined and the lack of precise definition has led to many researchers and institutes calling their approaches use-case based[13] (cf. Graham, 1996a; Dano et al., 1996; Cockburn and Fowler, 1998; Regnell, 1999). Even the definitions of use cases by Jacobson are a bit vague: a use case is "a behaviourally related sequence of transactions *in a dialogue with the system*" (Jacobson et al., 1992) and "a sequence of transactions *in a system* whose task is to yield a result of measurable value" (Jacobson et al., 1995). Further a use case can be a sentence or an essay or as Jacobson says 'as long as you like'. It is not clear what kind of events we should include in use cases. For example, should we include only external stimuli-responses or internal system activities as well, because they are all about the functionalities of the proposed system. The translation of use cases into design and implementation is based on the informal understanding of designers on what must be done. As there is no accepted definition of use cases, most companies as well as academic institutes have re-invented their own version of use cases, for example, Regnell (1999), Dano et al. (1996) and Firesmith (1998).

The lack of clear definition of use cases may result in the problem of the explosion of use cases. For example when modelling a business, the use case does not only address a single system but all of the systems used by the business. In this kind of situation there may be hundreds or thousands of use cases because they are many exceptions in the business processes. Graham (1996b) says that these exceptions may not be errors and they may be important and business critical. For system develop-

13.Graham (1996b) criticises that "the ugly phase 'use case' itself, possibly being a transliteration from Swedish, can also cause problems with English prose". Bruce Anderson (in Cockburn and Fowler, 1998) also points out that the name is a hindrance for business people. Use cases have no clear link to a business process model and are offered as such a model in their own right, which they feel does not make good sense.

ment, the use case is restricted to a single system. But the problem can still happen when the use of concurrency and distributed architectures means the order of the interactions between the system and its environment may be infinite (Firesmith, 1998). In some big projects the use cases are described by different designers or teams. There may exist inconsistencies between these use cases. Regnell (1999) also points out that use cases might be contradictory when they express goals of different actors.

Restricted to User Interface

Jacobson et al. (1992) state that Use Case Models are restricted to the user interface. That is, the intention of use cases is to focus on *what* the system does for its users, rather than *how* it does it. But in some domains this is not adequate. For example when applying use case techniques in business process modelling, users often have a mental model of the internal states of the system being described (Graham, 1996b). Further when developing a system which aims to replace or automate the paper-flow processes, the development team consisting of users and developers will have some expectations about *how* the system will operate initially. Graham (1996a) emphasises that such expectations are often the basis of insights into business processes and opportunities for reengineering them. We will illustrate this situation in the case study of the warehouse management system in chapter 7.

For system development, in some applications there may not be a clear actor who will get the benefit from interacting with the system or who directly or actively participates in the use cases. Systems of this kind, such as embedded systems, do not have actors because they can perform functions without user inputs. Thus use case modelling seems inappropriate for such cases. Even for other applications in which actors are clearly defined in use case modelling, there could be many different actors and use cases can only capture the needs of some actors and sometimes ignore others.

5.3 The Comparison between EM and OOSE

The aim of this section is to compare the approaches to software development of EM and of Jacobson's OOSE. The comparison is based on two concerns: the first is on their different development processes,

involving the construction of the models; and the second is on the similarities and differences in the structure of their models.

5.3.1 Comparison between the Development Processes

In subsection 2.1.2 we explained that generally there are some essential stages in the software development process: requirements, design, implementation, testing and validation (cf. Figure 2.2). The process of the OOSE approach also consists of these prescribed stages. Figure 5.2 shows the lifecycle of OOSE and its models developed and used during each process. It is obvious that the focus of the requirements process is to define the functionality of the system, I.e. the external perspective of the system as observed by the user. In OOSE the Use Case Model describes the specific ways of using the system and the Domain Object Model represents a logical view of the system in which the objects have a direct counterpart in the application environment. Following the requirements process, the Analysis Model and Design Model define the structure of the system, I.e. the internal view of the system architecture. The former is the ideal model which is independent of the implementation whereas the latter is the

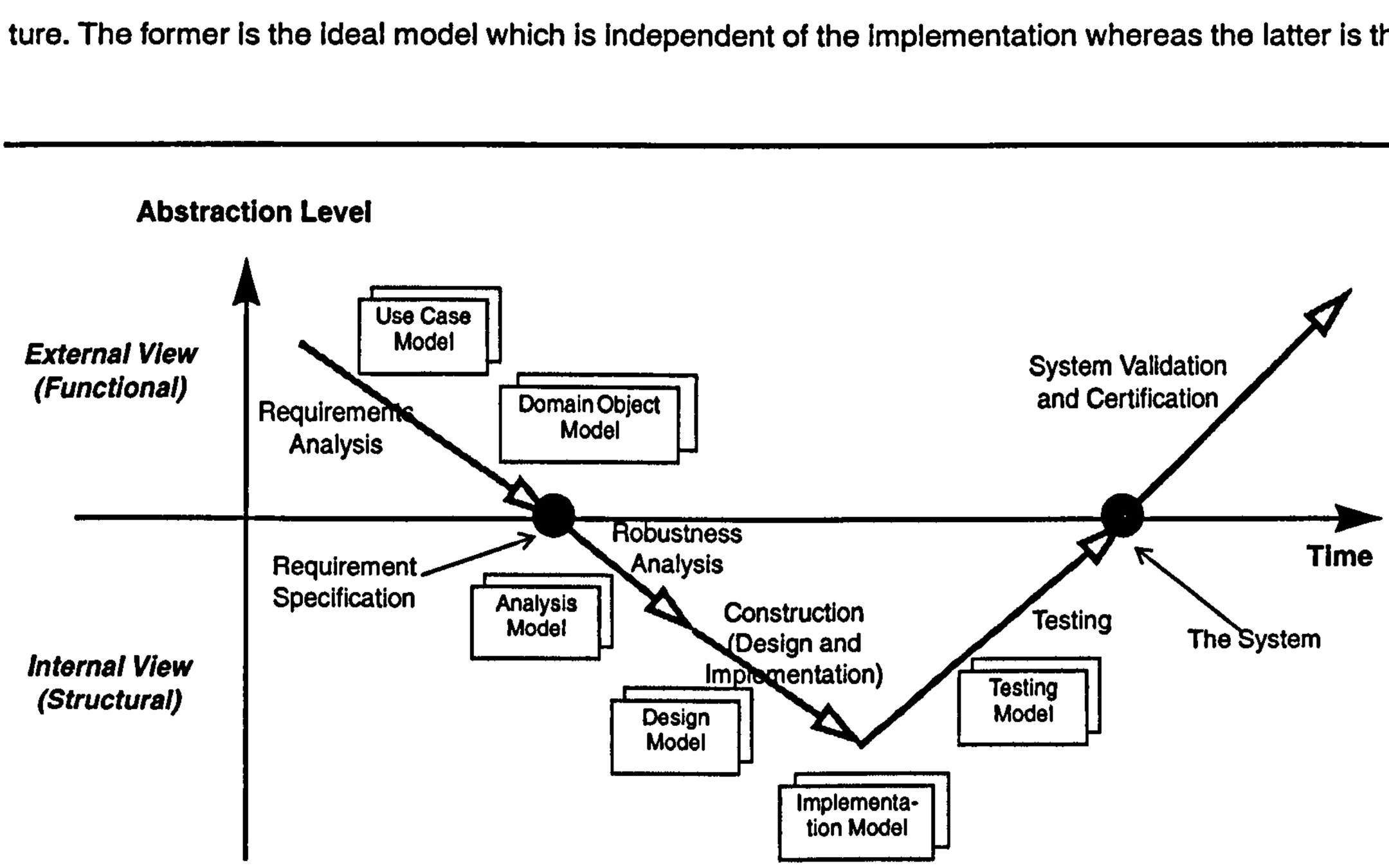


Figure 5.2 The Lifecycle of the Use Case Driven Development

real model which considers the implementation environment. The system is then implemented and tested and will be released after its external functionality is validated.

In comparison, EM has no such distinct sequential stages for development, that is, it is a process of model construction and system identification, but alongside which activities related to requirements analysis, design, implementation and even testing are also done. These activities are to be performed iteratively and do not follow the strict sequence of conventional methods, i.e. analysis then design then implementation. The modellers can perform any activity at any time for the best result of the development. For instance, when a design for the system emerge, the modeller can refine the model to address implementation issues or revisit the analysis activity behind the design. That is to say, the design decisions do not need to be laid down at an early stage and the modeller can change the variables and dependencies at any time to ensure the system meets the requirements of its users. In OOSE or other methods such design decisions have to be made at the early stages. For example, the functionalities of the proposed system should be decided and described in the Use Case Model; or the interfaces of objects or classes have to be defined while developing the Object Model (the Analysis Model or Design Model). The following processes may not be started without such activities having been done. We can also find that to make such decisions, or to describe the requirements of the proposed system, needs a kind of prediction and foresight of the future situation and this is sometimes difficult as we can not have complete knowledge of the system at the early stages of the system development. Furthermore, many decisions may need to be taken along the line as the design progresses, even though these decisions could ideally be taken at a later stage. Thus it is important that the users are continually involved throughout the design process, and this issue will be further detailed in chapter 6.

Figure 5.3 illustrates the procedure of EM modelling for system development. The development procedure may start from the existing or new model which represents the elements of real-world states related to the problems to be addressed or the system to be developed. This starting point is in the form of a *seed*-ISM. The aspects of the real-world states may be represented by different seed-ISMs. Alongside the process of EM modelling, the activities such as requirements analysis, design, implementation

and testing are also achieved but in different forms from the conventional methodologies. By having observation and experience with the ISM and its referent, the modeller identifies the relevant agents, observations and protocols which inform the modeller's conceptual model (or construal). After this, the modeller can have the initial idea about the new inputs or the requirements for the new system[14], for example, adding new agents and definitions or modifying the existing definitions. These activities are mainly carried out in relation to the visualisation of the ISM and the modeller's interaction with the artefact by which the external behaviour of the system is observed. The processes and the modeller's activities mentioned above are mainly conducted from an external view of the model, that is, with the reference to the EDEN visualisation. At the same time, the modeller can intervene in the current behaviour of the ISM by modifying the values of variables or the dependencies and immediately experience the result. After this modification to the ISM, the correspondence between the states of the ISM and the

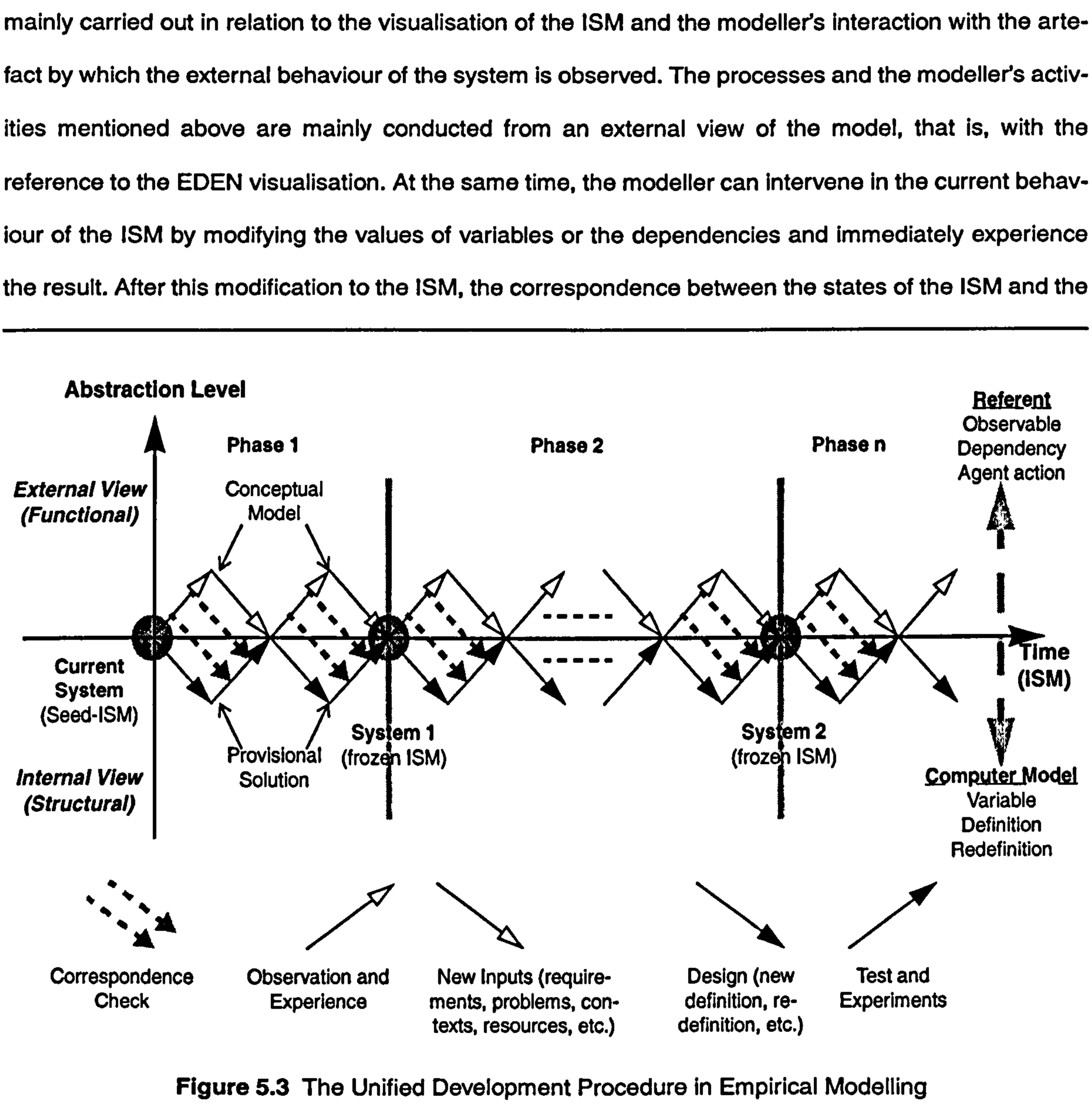


Figure 5.3 The Unified Development Procedure in Empirical Modelling

14.The 'system' in this paragraph refers to the software system, which is regarded in this thesis as a computer model. The details of how ISMs support the system development can be found in Beynon et al. (1998).

states of the referent is checked. This resembles the testing process in the conventional development lifecycle. These processes focus on the internal view of the model (i.e. the structure of the definitive script itself), in which the modeller modifies the state of the ISM through the EDEN input window. Also the commentary window shows information about the current status of the ISM and the outputs of the modeller's interaction.

All processes of EM modelling are iterative and through the continuous and open-ended interactions with the ISM, the modeller can ensure and maintain the correspondence between dependencies in the model and dependencies in the referent. When the result, i.e. the change of states in the ISM, contradicts the modeller's expectations, then the modeller can determine whether the model is constructed in an inappropriate way or any unsuspected behaviour is identified. Conversely, when the result is consistent with the requirements of the users or the modeller's expectations, the ISM can be 'frozen' as the final 'system' after the modelling phase. After the 'system' is released, when the requirements change due to the dissatisfaction of the users or changes in the environment, the 'system' developed during the previous phase will become the seed-ISM of the new development phase and a new modelling activity can be commenced.

Figure 5.4 gives a general concept of the development processes in both OOSE and EM. The aim and the final product of these two processes is a useful system. The lifecycle of OOSE shown in Figure 5.2 is summarised in the left-hand side of Figure 5.4. The arrow in the right-hand side represents the unified development procedure of EM modelling in Figure 5.3. By comparing these three figures (Figures 5.2, 5.3, 5.4), we can conclude that the models in OOSE can be viewed as phased-based models as the development of such models is through a kind of rigid sequence of prescribed phases. Each phase or stage reflects a practice done by prescribed and proven methods, techniques or tools. This enables the designers to develop the systems in a systematic way. But it may only be of limited use in today's rapidly changing domains of application. This is the main issue of circumscription which is emphasised in section 2.3. OOSE can be said to fit the culture of 'closed world paradigm' because it offers a preconceived framework for system development and once the models are created, they are no longer in direct connection with their referent. For EM there are no such prescribed phases and specific

methods or techniques during the development process. It fits the culture of 'open development paradigm' as the focus of EM is on the open-ended interaction between actors and the models and the referent, not following a fixed pattern of activities. We will furthermore discuss the comparison between EM and OOSE, focusing on the two main activities of requirements analysis: the understanding of the domain and functional requirements description.

Figure 5.5 depicts the Requirements Process of OOSE and the relationships among the models created during this process. This can be compared with Figure 5.6 (reproduced from Figure 4.2 in chapter 4) to show the differences between OOSE and EM in requirements-oriented activity. The main difference between these two approaches is in the correspondences between the real world system and the models. In OOSE, such correspondence is established through the developer's pre-understanding of the problem domain as well as the negotiations with the customers and the future users. This can be

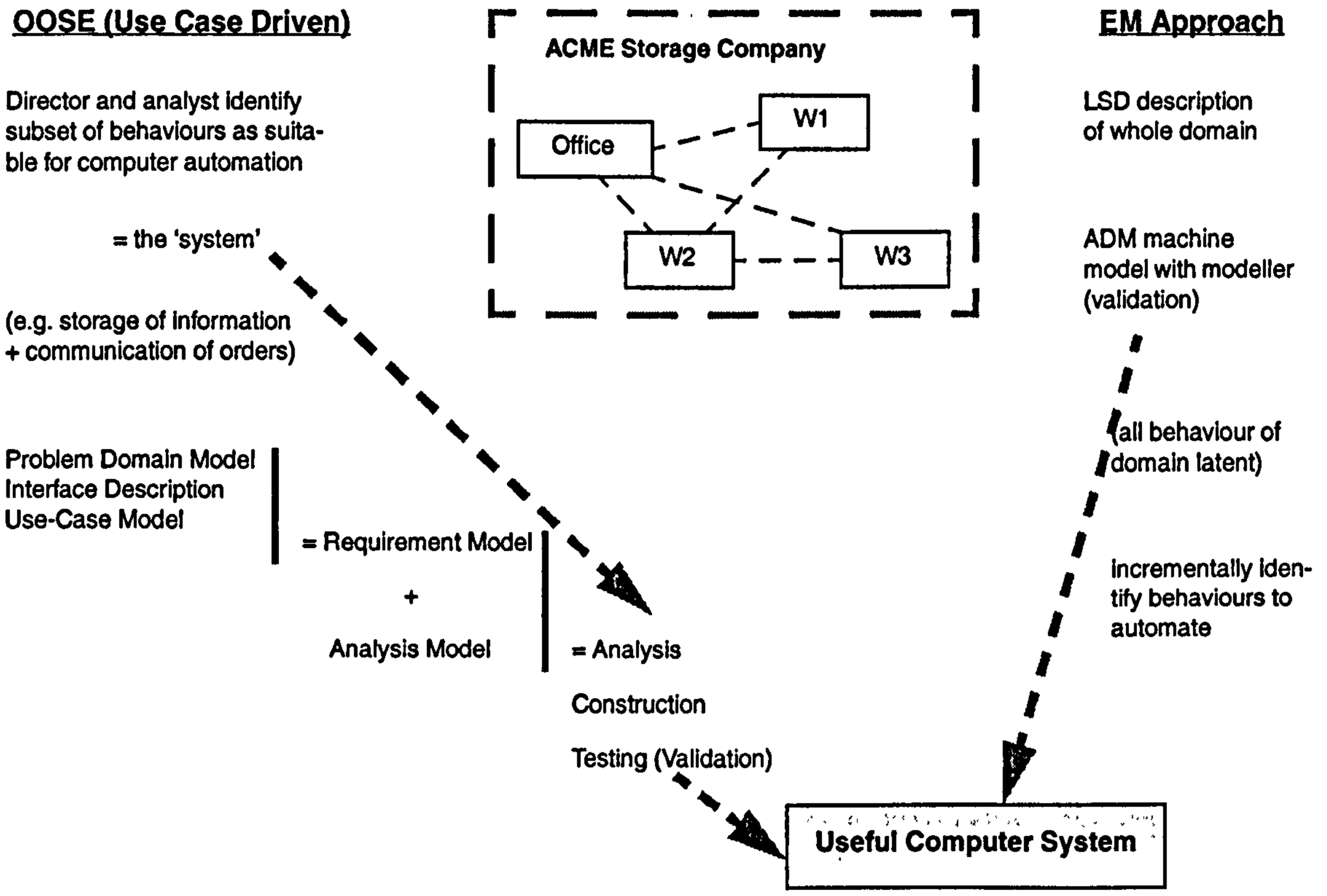


Figure 5.4 Contrast between OOSE and EM Approaches to System Development

viewed as a textual correspondence because it is achieved in the form of text descriptions, I.e. the use cases, to define the desired functionalities of the proposed system. By way of contrast, the correspondence in EM is one which links the states of the model and the states of its referent. This correspondence is established through the modeller's mental model of the current situations with both the ISM and the referent (which in turn arises through his observation and experiment), rather than the prediction of all the possible states and the behaviour of the future system. There is also a difference between the 'actor' in OOSE and the 'modeller' in EM. In OOSE the actor is a generic term which means any external entity, for example the users, developers or other systems. But each of them has a specific role in

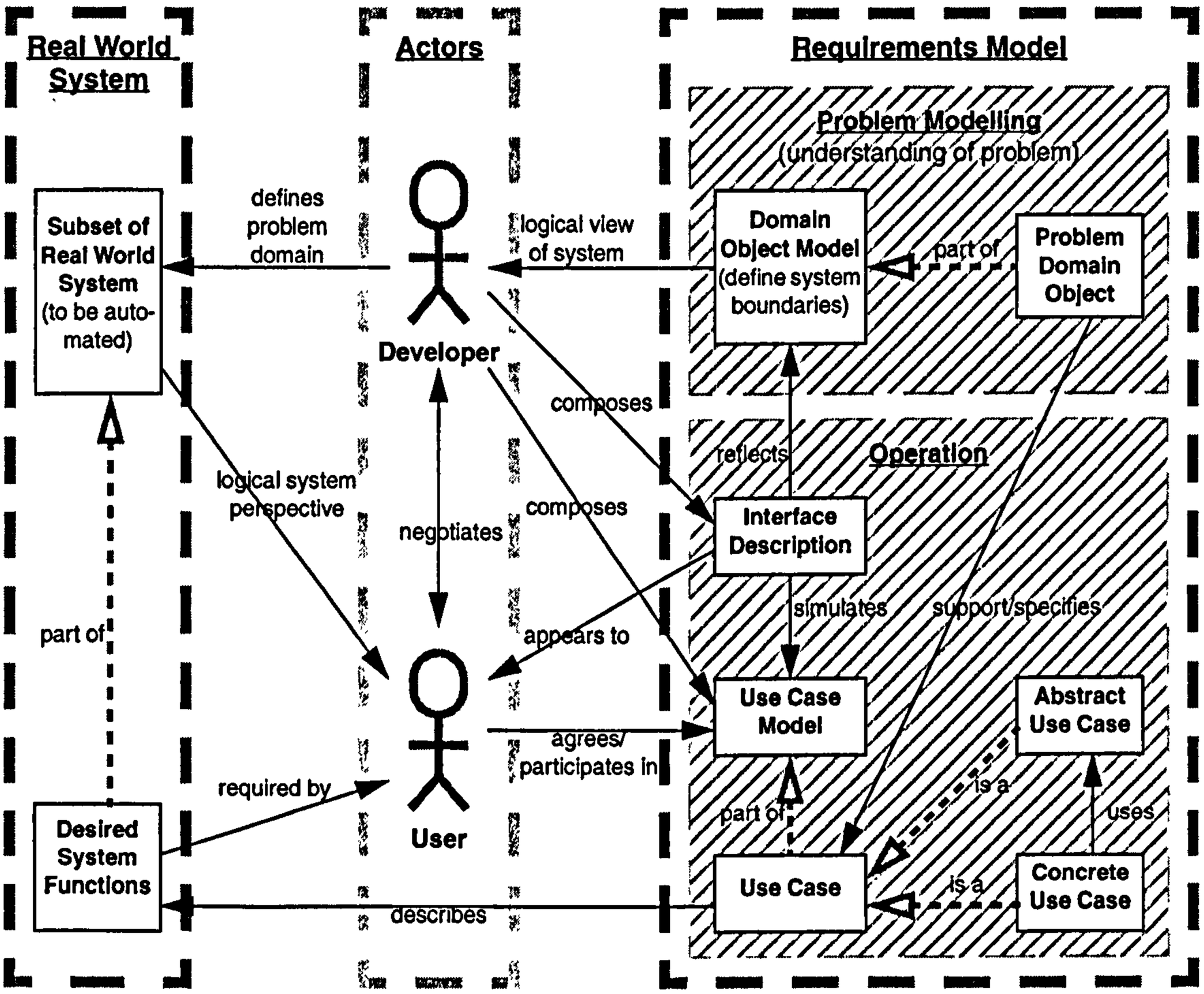


Figure 5.5 The Requirements Process and Models in OOSE

interacting with the system. For EM the modeller can be a designer or user or any other participant. There is no specific role for the modeller as he can interact with the ISM in a flexible and open-ended manner. Through this, different types of interactions with the real world can be simulated by interacting with the ISM. The right-hand block in Figure 5.5 representing the Requirements Model of OOSE can only reflect the inner block of the executable area in Figure 5.6. This means the domain of OOSE is mainly the 'system' itself, whereas the domain of EM involves the real world situation which could be outside our prediction of the system states. It is also interesting to see that in OOSE the activity of understanding of the problem domain (the upper inner block) and the activity of requirements/function-

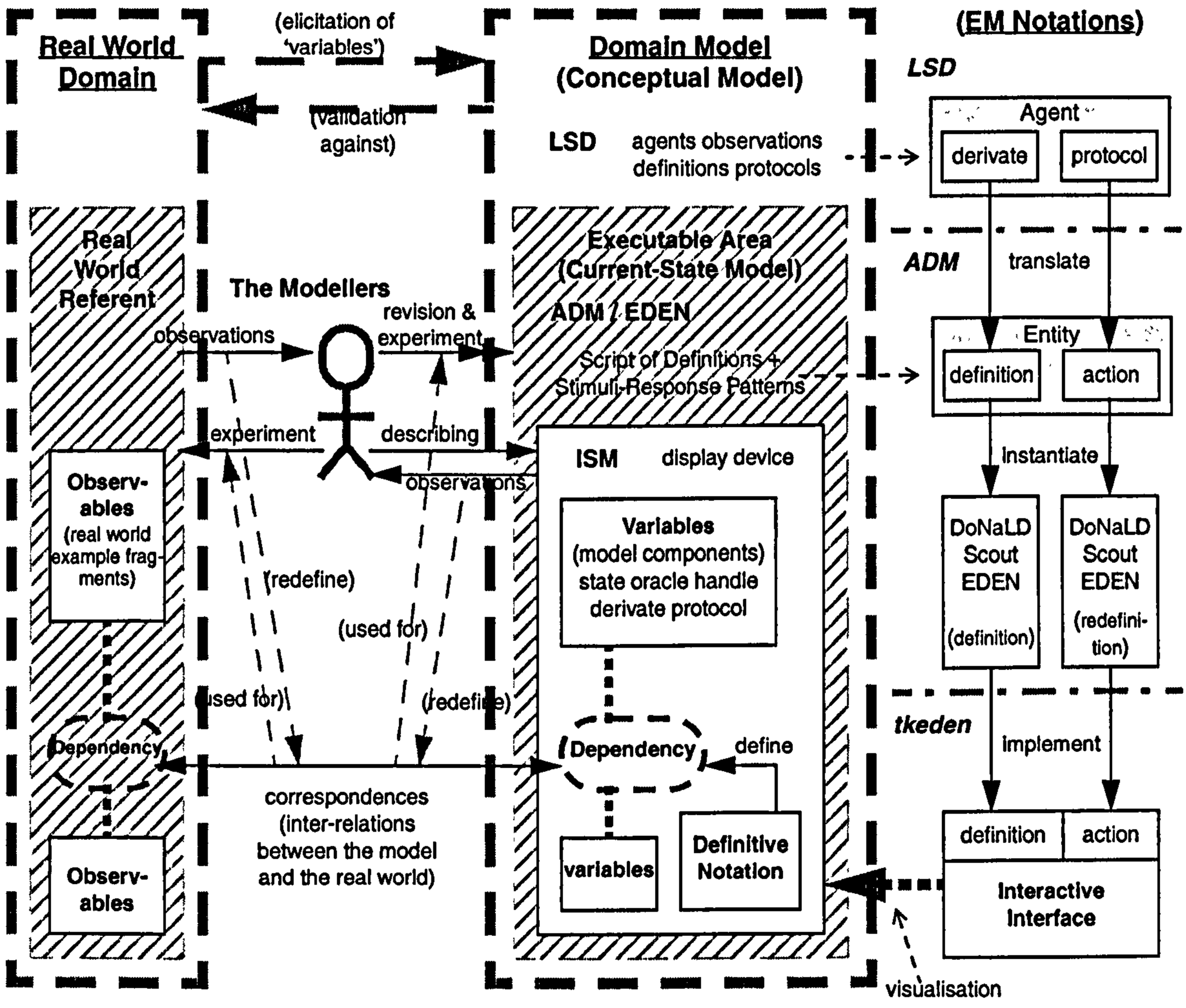


Figure 5.6 Empirical Modelling and its Tools and Notations (copy of Figure 4.2)

ality description (the lower inner block) are separate whereas in EM these two activities are 'fused'. (These issues will be further discussed in the next subsection.)

Finally, it remains to discuss the user interface. In OOSE, the user interface description is created before the actual system is developed. It serves as a prototype and many use cases here will often need the description of such an interface. This kind of prototype differs from the 'prototype' of the EM model. In OOSE the interface is static and passive, and the actor's interactions with the interface are represented by single arrows. The interface of the EM model represents a 'virtual prototype' but one which is integrated as part of the whole model and is as open to interactive modification as any other part of the model.

Understanding of the Domain

The activities of understanding of the domain in the requirements analysis are done by the Domain Object Model in OOSE and the LSD account in EM. The Domain Object Model mainly serves as a logical view of the proposed system. It consists of *problem domain objects* which represents the entities from the problem domain which the system must know. Through this model some concepts which the system should work with can be defined. In EM the LSD account records the modeller's conception of the situation. This account captures the observables and dependencies which are significant or relevant, the agents which are present and the classification of observables based on the view of these agents. We can compare the Domain Object Model and LSD in the following ways:

Objective vs Subjective The Domain Object Model, as well as other conventional models, is objective because it is constructed to acquire the objective knowledge about the system behaviour in order to achieve a specified goal. As Jacobson et al. point out, the Domain Object Model is to enable the users to recognise all the concepts needed when defining the functionalities of the system. It is also the tool for users and designers to communicate about the system. This model is developed through discussion among these people and aims to form an objective base for understanding of the domain and the development of the Use Case Model. In contrast, the LSD account is subjective as it represents

only the modeller's perception and experience of a particular subject in a particular situation. It is subjective because in EM, the definitive script or the activity of constructing an ISM is tentative and provisional and is open to change and extension in order to meet changing requirements as well as the changing environment. In this sense, Beynon et al. (2001) characterise the computer as an instrument (or interactive instrument) whereas in conventional approaches the computer is a tool. In other words, the instrument-like use of artefacts is more subjective because the emphasis of using instruments is on exercising personal skills; and the tool-like use of artefacts is more objective because the use of tools is to perform a specific function or goal in a organised and public pattern of interactions. Within the EM context, the LSD account records the modeller's conception of the referent which may be different to the objective knowledge of the referent or the intended functionality and interaction with that referent. This characteristic differs from those methodologies which seek to form a formal specification. Ness (1997) classifies the conventional (object-oriented) methodologies as being like an agentless or 0-agent system in EM because of their absence of considering the role of the modeller in the process.

Domain The problem domain of OOSE is mainly the 'system' itself, I.e. the aim of the OOSE development process is to build a new system to replace the existing one. Thus the domain defined in OOSE is a part of the real world in which the activities or processes can be replaced or automated by the new system. For this situation it is essential to define the boundary of the system at an early stage in order to develop a system which is meaningful and useful for its users. In comparison, the domain of EM involves the real-world situation. That is, the domain of EM involves the situations which reflect to some extent the way in which the modeller perceives, and interacts with, the world. Even the past experience of the modeller or the legacy of other people's experience can be drawn on here. The states in the EM model are a metaphorical representation of the relevant situations in the real world. The possible transitions between states are recorded within the LSD account in terms of agents, observables and dependencies. This LSD account is consistent with the modeller's construal of the referent.

System Behaviour The system behaviours in OOSE models are preconceived because these behaviours have to be defined at an early stage in order to construct the subsequent models. In the Use Case Model, each functionality, or usage, of the proposed systems is defined by a use case and its

potential users are defined by actors. Normally such functionalities or usages defined in use cases are based on the frameworks which have been identified earlier to be reliable or predictable. This model serves as an agreement between the customers and the designers in which the clear description of the system functionalities is included. Thus we can say the focus of OOSE methods is on the representation of the system behaviour. In contrast, the system modelled by EM exhibits the same behaviours which may be appropriate to the applications but, in the process of development, they are not preconceived. Such behaviour is represented in the EM model as immediately experienced rather than circumscribed. This is because the changes in the states of the EM model are not anticipated and thus cannot be derived in advance. However, it is still possible to circumscribe the system behaviour within the EM model just as in other models, and the modeller can introduce agents or dependencies in the order which is appropriate.

Ontology vs Epistemology An important issue, relevant to the discussion above, is whether the philosophical orientation of a system development approach is towards *ontology* or *epistemology*. The philosophy of object-orientation is generally viewed as concerned with ontology[15] – the common way of thinking of 'object' is as a representation of a real-world entity, i.e. any real-world entity is an object (cf. section 2.2). Thus the OO models are objective in character and their artefact use is tool-like, and their domain is limited to the proposed system itself. In contrast, the philosophy of EM modelling is more related to epistemology because we are emphasising experiment and long-established practice during the process of modelling. This is similar to the outlook of Checkland's SSM and Lehman's E-type program that were described in chapter 2. Lewis (1995) comments that the philosophical orientation may not be significant when OO is applied at the implementation level for a problem domain which is well-defined. But if applied in less well-defined situations, then "the assumption of one-to-one correspondences existing between the objects in the data model and parts of the external [real-world] reality is more dangerous and blinkering"[16]. And McGinnes (1992) has the same view:

15.Lewis (1993 and 1995) has a different viewpoint. He mentions that "the recent interest in object-oriented analysis has led to re-interpretations of the nature of data analysis and at least some recognition that a data model might be an epistemological device, a coherent means of investigating the problem domain rather than being a description of the real world" (Lewis, 1993). We will further continue this discussion later in this paragraph.

> The idea that object-oriented systems are a 'natural' representation of the world is a seductive but dangerous over-simplification. In reality, the fact that the object model seems so close to reality makes it far easier to misuse than other modelling techniques which do not propose to represent the world so directly. (p. 13)

Many researchers (Winograd and Flores, 1987; Stowell, 1995b; Mingers, 1995; Lewis, 1995; Checkland and Scholes, 1990) argue that OO and many traditional methodologies which are based on an objectivist viewpoint are inappropriate for designing information systems as they are a part of the whole process of human communication. Instead they argue that human interaction involves understanding and meaning[17], and thus our actions cannot simply be described objectively by an external observer. Thus Mingers (1995) describes the task of system analysts as not simply the objective description of particular information flows or data structures, but as the interpretation and elicitation of the socially constructed patterns of meaning which generate observed behaviour. For EM we do not aim to model the proposed system directly through one-to-one mapping in a preconceived manner, but rather focus on the process of human agent's perception and experiment with the real-world reality. This is consistent with the view of Lewis (1995), who suggests that OO analysis, as well as other models or database, is not modelling the real-world reality, but modelling the way the reality is understood by people – they are the models of user's models of the world[18]. Thus the analysis leads to models which do not reflect any objective reality but a particular group's knowledge and perceptions of a problem situation. Thus we need ways to investigate and model that knowledge and perception, especially ways "that are sympathetic to the complex manner in which social realities are created". The EM approach can in this sense complement other analysis approaches, because, as Lewis says, with current methods, analysis "con-

16.He gives an example of whether a 'country' is a tangible item (an object) in the real world or a perception shared by many individuals: "Did there exist a real-world thing called Slovenia in December 1991 when some inhabitants of Yugoslavia had declared its independent existence? Did it exist in January 1992 when the European Community (but not the United Nations or the governments of Yugoslavia) recognised its existence? The answers, of course, depend on how we define 'a country', and this is socially and politically decided: it is not an absolute characteristic of something existing in an independent 'real-world' ". (pp. 194-195)

17.This is why Checkland and Scholes (1990) say "information equals data plus meaning", and Galliers (1995) contrasts *information* systems (which inform) from *data-processing* systems (which automate the operational tasks). Goguen (1994 and 1996) comments that information is *situated* – it is highly dependent on its social context for interpretation and we have to consider how it is produced rather than merely how it is represented.

18.Thus the question for evaluating models should not be "Does this design accurately represent the real world?", but should be "Does this design accurately represent the user's model of his environment?".

tains no model of human sense-making that explains it, and none of the techniques of data modelling assists by making the subjectivity explicit and open to debate".

Functional Requirements Description

In OOSE the functionalities of the proposed system are defined in the Requirements Model. The first step in the requirements process is to create a simple prototype of the system which helps to describe the boundary and the actors of the system. Then the functional behaviours of that system are described by the informal text descriptions in use cases. Jacobson et al. (1992) calls the content of the use cases *interface descriptions* (or *use case specifications*) because they define the interface between the users and the system itself. The use case specification consists of both textual and graphic descriptions. It also consists of various phases each representing a particular set of behaviours in that use case.

In EM the ADM or EDEN is used to animate the LSD account. Through the animation the interaction between agents in LSD will be visible and meaningful and the modeller can intervene in the model by adding or redefining definitions. Through this kind of interaction the modeller can ensure whether the system behaviour meets the requirements with reference to the external observation. The following is a comparison of the techniques of functional requirements description between OOSE and EM.

Verbal (Text-Based) vs Machine-Support Use cases are written in natural language to describe the dynamic process of using the system. According to Jacobson et al., use cases are easy to understand and provide a better way for communicating with customers and users. However Dano et al. (1996) point out that although the description of the use cases makes the involvement of domain experts easier, it is still not entirely satisfactory because the designers have to make sure no requirement is misunderstood as well as ensure the completeness and consistency. Further, such a kind of communication is still passive and Sun et al. (1999) argue that the shared understanding of all participants within this kind of text-based model is invisible and incommunicable. Inevitably the visibility and communicability of shared understanding is restricted to the boundaries of language description or comprehension. Also it is hard to keep the requirements specifications synchronous with changing

requirements as the people involved gain a better understanding of the problem and may change their points of view or focus. Maintenance may also be a problem because a change in the requirements may affect many places in the text of the specification. In other words, there can be many versions of a use case. This is especially problematic while developing a complex system as too many or too complex scenarios cannot be fully described by text-based models. But if we can animate these versions and then test them, we will get a kind of feedback through the visual artefact during the analysis and design. And once we have the feedback, we will be able to improve the requirements specification and generate a new version of a use case easily. This is what we have been doing in our EM research. In EM the interaction with the ISMs can make the modeller's insights and the shared understanding with other participants visible and communicable. Pictures are more informative than the textual descriptions: "A graphic is worth a thousand words". In EM the computer is used to generate the metaphorical representation of particular states and to animate the modeller's expectations about the results after state-changing actions. Interaction with the computer model will lead to the evolution of the modeller's insights as well as a shared understanding. Conventionally, much knowledge (especially tacit knowledge) is kept in the participants but cannot reach the corporate knowledge. But in the EM modelling process, we are trying to make such tacit knowledge and other trial-and-error knowledge accessible to the decision process. The knowledge through the interaction with ISMs can solve the text-based problem in the use cases and other methodologies. For this reason, EM can be regarded as an alternative approach for creating use case specifications. The difference between the different versions of use cases is mainly from the change of responsibilities of objects or the addition of new objects. EM allows such kinds of change to be made 'on-the-fly' by redefining or adding definitions in the script. Sun et al. describe this synchronisation between the evolution of computer models and modeller's insights allowing the participants to 'see' the viewpoints of others and to 'communicate' with them by interacting with versions of their artefact. Another advantage of the EM models is, it can capture the richer reality of the real-world situation and make the revision of the model easier and more faithful to reality.

Scenarios In OOSE, a use case in its normal course and alternative courses can be regarded as a collection of the possible scenarios between the actors and the system. That is, a use case describes

the possible scenarios of user's interaction with the system and represents the steps to accomplish the result.

Figure 5.7 illustrates some different types of scenarios in system development. The first type of scenario is the system internal scenario which consists of the interactions of the internal components within the proposed system. This kind of scenario is described by the use case specifications and is modelled by the following Analysis Model, in which the external context of the system is not considered. The second type of scenario is the interactions between the system and a part of its external environment, which only includes the actors and other systems directly interacting with the proposed system. Such scenarios are defined in the Use Case Model. The third type of scenario includes the wider context of the external environment. In addition to the direct interactions between the system and its context, it also includes the information about the context of that system. We argue that the scenarios in the EM model are categorised into this type because the state of the referent is metaphorically represented in the EM model and the consequent results of the state-changing actions by the modeller may or may not meet his expectation. Obviously the EM technique can be applied to model all these types of scenarios.

For example, the perspective of an external observer, as described in 'Scenario 1'[19] in Beynon (1997),

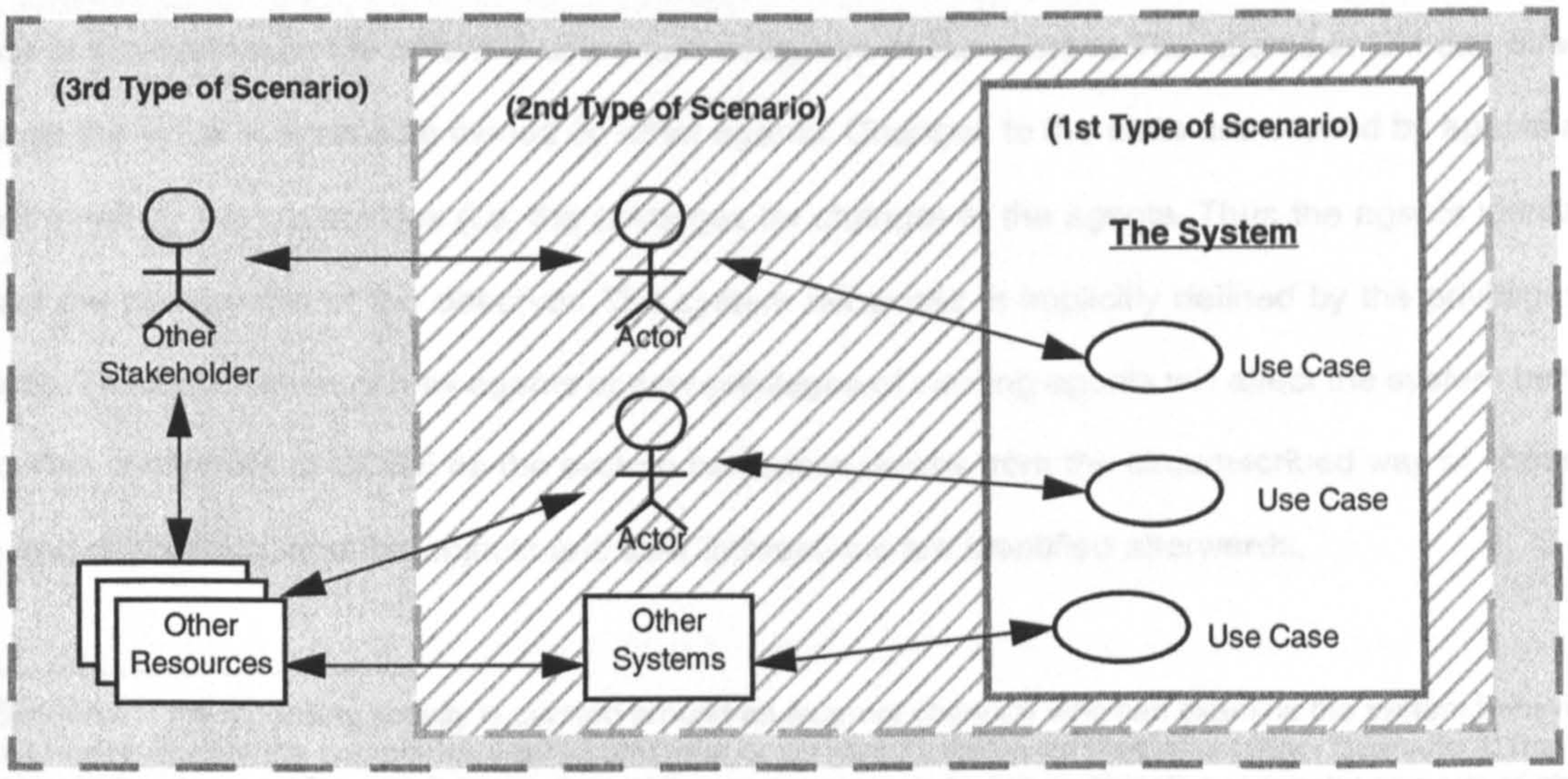


Figure 5.7 The Scenarios in System Development

is similar to – but even broader than – that of the designer in OOSE, as the functionality of the proposed system and possible interaction of users have to be identified in the Use Case Model before the objects and their interactions are modelled. The first type of scenario can be represented through the interaction of internal agents in the EM model. However, according to systems thinking, each agent (or object) can be specified, but the whole system behaviour made up of the cooperation of their interactions may sometimes lead to conflict or incoherence. This explains why sometimes existing methods fail to develop a system to meet the user's actual needs as they did not take the third (or even the second) type of scenario into consideration when developing the system.

Systems-Oriented vs Agent-Oriented The use case is systems-oriented because it defines and captures the major functionality of the proposed system. It is user-centred because the aim of use cases is to ensure the system is developed to meet the requirements from the user's viewpoint. A use case may involve many actors, but only one actor initiates the use case. However the actors are beyond the scope of the system considered, the use case modelling does not consider the interactions between actors. In contrast, EM is agent-oriented because its process involves the identification of agents as well as the observables and dependencies in the subject. However the concept of agent-orientation is different to the one in the AI field. In EM the agents are state-changing agents which refer to entities related to the representation and transformation of the state, for example the individual components of the system or the potential users and designers of the system. The agents in EM can directly change the value of a variable owned by other agents. Changes to the state are caused by agents and constrained by the capabilities (i.e. the privileges for change) of the agents. Thus the agents identified reflect the perspective of the observer. The system behaviour is implicitly defined by the privileges of agents. The introduction of new agents or new privileges of existing agents will affect the system behaviour. This is different to OOSE as the system behaviour comes from the circumscribed way of observation and description, and the objects and their interactions are identified afterwards.

19.Scenario 1: the modelling activity is centred around an external observer who can examine the system behaviour, but has to identify the component agents and infer or construct profiles for their interaction; Scenario 2: the system can be observed from the perspectives of its component agents, but an objective viewpoint or mode of observation to account for the corporate effect in their interaction is to be identified (Beynon, 1997).

Programming through Modelling vs Modelling through Programming

Beynon and Joy (1994) describe the primary motivation of the EM paradigm as a philosophy of *programming as modelling* or perhaps more clearly *programming through modelling*. That is, using modelling as a means to program a computer. This idea differs from the concerns of object-orientation and other modelling methodologies which can be characterised as *modelling through programming* as we program the computer in order to model a system.

By 'programming through modelling', the emphasis of EM is to model what we observe of the whole system and its environment in terms of interacting agents, rather than constructing the models independently of the real world situation. One advantage of the EM approach is that we can specify the synchronised changes which may cut across the object boundaries. Further, according to Beynon and Joy, the development process of the EM models is the process of formulating relationships (i.e. the dependencies) between observables which are provisional. This means that the EM models can be viewed as a specification of a program only *after* the majority of the modelling process has been done, i.e. the reliability of the relevant observations have been validated. By way of contrast, in OO methods the specification of the system is decided *before* the construction of models. For EM, the emphasis is on software construction to establish a *prototype* of the real-world system within the computer; whereas in OO the emphasis is on developing a *finished model* based on the requirements which are in turn based on the assumptions and initial experience of the real-world system (Rawles, 1997).

5.3.2 Comparison between the Models

This subsection will compare the similarities and differences between the EM model and the models of object-orientation (and OOSE). For EM we are concerned with matters of state in ISMs and comparing this with the states of OO models in terms of the different characteristics of agents and objects. We also consider the issues of modularity and reusability, which are fundamental features of OO, to find out the different concept used in the EM model.

The Concept of State

EM differs from conventional programming in that the focus of EM modelling is on *state* rather than the structure of control or the representation of behaviour. The 'state' here broadly includes the state of mind in the subjective experience of the modeller as well as the states of the artefact and its referent. The values of variables in the EM model at a time correspond to a particular state in the real world. The change of state is only achieved through the automatic update of dependencies or the explicit action of agents, that is, through redefinitions or the addition of new definitions. Thus change can be made at any time during the animation. For EM the concept of state, which differs from the public sense, is primarily *state-as-perceived-by-agent* or *state-as-experienced*. That is, the state represents what the person currently perceives/experiences of the environment and is subject to continuous change. In most conventional approaches, such a subjective state is not directly represented. In order to prescribe the process, the inputs and outputs have to be preconceived before the construction of the model. The OOSE method is an example of emphasis on prescribing the system behaviour in use cases before constructing the model. If any new inputs and outputs and additional features of systems need to be added, the revision and redesign of the whole process may cost time and effort. Also the EM model seems to be more robust than the OO model: because EM enables the user to experience an unfamiliar situation, which includes unexpected situations or even abnormal conditions.

Generally in object-oriented programming, the representation of states is through the identification of the state of each object. Thus we can identify the difference between agents of EM and objects in OO in modelling state-transitions. In object-orientation, each object is responsible for changing its state through performing its internal action or method. Other objects can send a message to an object to invoke its method. The message passing is the only way in which these objects interact and thus change their internal variables and states. This is different from the agents of EM in which the actions of one agent can directly affect the changes of state in other agents, and sometimes result in an indivisible propagation of state changes which involve several agents. The agents in EM can be either active or passive. The active agents can perform autonomous actions to change the state of the model; whereas the passive agents serve to record state information but cannot perform actions (Beynon et al., 1990).

The character of passive agents may be close to that of objects in OO. The system state represented in OO models is more tricky than in an EM model, because the designers need additional information about the message sending, receiving and responding for objects in order to understand the system behaviour.

When BPR is concerned, the mathematical models used in conventional methods are not suitable for modelling states in the business environment. It is essential to identify the factors which affect the manager's or user's perception of the state of the business which they are in. Such factors cannot be appropriately represented in the mathematical model. As Beynon et al. (2000a) describe, modelling these factors involves consultation with other related parties, the development of physical artefacts or prototype products, and computational models based around databases and spreadsheets. In the next chapter we will illustrate an ISM which encompasses models of this nature and is well-suited to modelling business processes.

Issues of Modularity and Reusability

Modularity is one of the fundamental properties of the object-oriented methodology which has made OO popular and makes system development more effective. The central theme of object-orientation is to synthesise the software system from modules which can be developed independently. But the decomposition into objects in the OO paradigm means a commitment and knowledge about the problem domain is needed before defining the behaviour and attributes of objects. From the viewpoint of the object-oriented paradigm, it is essential to minimise the dependencies between modules for the reason of maintenance and reusability. In the construction of large systems, the division into modules sometimes can be a difficult task.

In EM the emphasis is on observation rather than the object. That is, our emphasis is on modelling the observation of the whole system with interacting agents but not constructing objects in isolation and then integrating them by means of message passing. The observables and variables in the EM model are global. The indivisible propagation of state changes in the EM model will result in state changes

across the object boundaries. This makes EM more suitable than object-orientation in the synchronisation design of system development, due to the synchronised updates of variables through definitions. In OO the protocols between objects are not made explicit and thus the synchronisation of messages cannot be specified.

On the other hand, as there is no kind of modularity in the EM model, the issue of reuse cannot be coped with well within EM. For object-orientation, each object can be easily reused due to the features of encapsulation and information hiding. Some techniques, for example isolating the agents and observables, can make some kind of modularity in the definitive scripts, it is still hard to achieve modularity in the EM model because of the rich dependencies in the definitive scripts. Rawles (1997) suggests two structures for applying the idea of modularity in the EM paradigm. The first is through the construction of scripts, each of which models an individual element of the system. But this does not make it different to normal EM modelling because the system is still developed as a whole. This only makes the modeller construct the scripts more conveniently and easily, i.e. splitting the scripts into some conceptual chunks. In this case the scripts may be reusable. But the modeller will still make more effort to reuse such scripts than to reuse objects in OO in which only the change of class interface is needed. The second is using agents as modular units. However even if we modularise the agents, they are still with global character due to their observables and the dependencies between them. Further work such as localising these dependencies may be needed. This indicates the need for further investigation into the status of modular design within the EM framework.

Even though software reusability has been regarded as a claim for object-orientation, Gammack (1995) points out that such software reusability still has been a massive problem for the industry. Together with other researchers, he concludes that

> hardware reuse benefits from the fact that standard functions are easy to identify owing to their close mapping into the real-world model of hardware systems, and if software reusability is to become a reality languages must support a more direct mapping from the model of the real world functions to the implementation. (p. 165)

This argument is closely related to the discussion of relationships between software (or the software system) and the real world (its business environment) in section 3.3. Such discussion may provide another direction for research into software reusability within the EM framework.

5.4 Concluding Remarks

The concept of use cases has become more and more popular in the object-oriented community and has been established as one of the fundamental techniques of the OO methodology. It provides some advantages in the process of requirements elicitation which the conventional object-oriented methods failed to achieve. But like many other development methods, there are still some problems when applying the use case technique in developing the system. This chapter summarises the general concepts of the use case technique and the processes of applying this technique in both system development (OOSE) and BPR (Object Advantage). We also gave an overview of the drawbacks associated with use cases.

The concepts of the EM approach have been described in chapter 4. Several EM projects have identified issues of common interest in the EM approach and object-oriented approaches (including the use case approach). Related work on this theme can be found in Ness (1997), Rawles (1997), Beynon and Joy (1994) and other research reports of the EM and AMORE projects. In this chapter we have compared these two approaches mainly in regard to software development. The comparison is based on two concerns: first, their processes of system development which involve the construction of the models; and second, the structures and characteristics of their models.

The paradigm of object-orientation has been long established and been widely accepted for software development. But EM is a novel approach to modelling which has been developed for just over 15 years. The research of EM projects shows that EM can bring a new approach to development in some different fields, for example, business modelling. Through this comparison, it is hoped that we gain a better understanding of how the EM approach can relate to existing methodologies of system development as well as an appreciation of how this approach can itself be used in system development.

CHAPTER SIX

Empirical Modelling for Participative BPR

As business environments continue to change rapidly, it becomes important that the developers of information systems should have a good understanding of the business environment which the systems are designed to support. The importance of linking information technology to business goals and objectives is addressed by the concept of BPR. In response to competitive pressures and changing conditions, many companies are rethinking the way they are doing business. For this, information systems are expected to provide 'innovative' solutions to business problems, rather than merely implementing the usual or known solutions (i.e. simply automating the established processes).

Many methodologies have been proposed for the BPR analysis[1]. However most of them do not address the implementation issues of the development of information systems and the combination of this with the implementation of BPR in detail. That is, the work of business modelling and supporting system development are normally implemented separately by different teams and at different times. As, in BPR, the organisation is regarded as a complex business system, we need an effective method to analyse the dynamic behaviour of the organisation and to evaluate the alternative choices of solutions to problems. Also, as computer systems and applications are currently viewed as transforming the tasks and practices in business processes, we need an artefact for representing, analysing and planning how

1. The Business Processes Resource Centre (BPRC) at the University of Warwick has provided the access to BPR research and the enabling technology providers for BPR analysis and implementation (the hardware, software, or consultancy). The collection of methodologies and tools for BPR analysis can be found under 'Best Practice' at http://bprc.warwick.ac.uk (20 December 2001).

the users' activities and experiences are influenced by the introduction of computer systems or the changes in business processes. In this chapter, we introduce the concept of participative process modelling and participative BPR, because we view system development and BPR as cooperative activities which involve different groups of people – system designers, business experts, end-users or customers – with different competencies, viewpoints and requirements. As no one group can fully understand the practices and meanings of all the others, what we need is to bring their experiences together with the aim of creating a framework for a modelling process which involves all these groups. In this chapter, we will describe a framework, based on the principles and tools of EM, that aims to give a comprehensive – or at any rate representative – view of the real-world situations and systems so as to allow all participants to experience what it is, or would be like, to work within such situations and therefore draw on their tacit and non-explicit knowledge and experience. We propose the application of EM to provide a practical way of implementing participative BPR. This is presented here, and illustrated in chapter 7, as a major part of the original contribution of this thesis.

6.1 Introduction

The role of information technology as the enabler for organisational rethinking has been emphasised in much BPR literature. However when coming to the actual implementation of the BPR project, the implementation issues of IT are usually ignored or addressed by just picking an off-the-shelf application and changing the business processes to fit it. Furthermore, these approaches only attend to the automation of business processes after the BPR implementation; they do not mention the actual procedure of the reengineering work. In chapter 3 we have pointed out that businesses and business processes are such complex systems that the developers and users require appropriate models to understand the behaviour of such systems whether in order to design new systems (or processes) or to improve the existing ones. Most of the BPR methodologies so far have their own systematic phase-based techniques to model business processes. Such 'scientific' techniques may be efficient and necessary if fully automated processes are going to be modelled or reengineered. But it is important that we should take account of the fact that businesses or organisations are not just systems like technical systems; instead

such systems include humans participating within them. Thus any attempt at radical modification of the current structure and operations of a business should consider the human factors and enable the participation of all involved if we want to make sure the results are satisfactory and acceptable. This is necessary if we are to avoid a common problem – that newly designed structures or processes may appear to be suitable for business performance 'theoretically' but not be applicable in practice due to some factors which were not considered or expected during modelling work.

In defining the requirements for BPR and its support systems, it is essential to have a broad understanding of the organisational environment in order to make appropriate decisions about what changes to make or which parts to retain. There is a problem in requirements analysis, namely, that it is difficult to mitigate the bias of individual participants towards either the current or the future context. It is well-recognised that people have difficulty in envisioning the impact of a proposed system in the future. There are also biases when dealing with uncertainties and risks, especially when multiple agents are involved and multiple changes are happening simultaneously.

In this chapter, we will investigate the potential of computer models as a suitable technique for business modelling. We start by introducing of the concepts of participative process modelling and participative BPR. After the presentation of our framework for cultivating requirements, we will assess its potential as a medium for participation in BPR.

6.2 Participative Process Modelling and Participative BPR

BPR calls for a radical change that involves focusing on the analysis of business processes rather than the functions of a organisation, and reengineering these processes in order to maximise the performance. But the major issue in the BPR exercise is that the scope of the factors concerned is exceedingly broad, and many of these factors cannot be captured in the abstract process described in conventional mathematical models. For example, there are practical considerations which may affect the execution of the process such as the capabilities of the personnel or the environmental conditions and the equipment used. Further, in order to describe and model processes, we must have knowledge not only of

their static structure, but also of their dynamic behaviour. Thus good approaches to BPR should focus on the analysis of processes and help to identify the problematic situations and new business opportunities. Besides, as we argued in chapter 3, BPR has more of an organisational focus than a technical one. The effort of BPR is directed at changing people's thinking and thus has to take into account the expectations and viewpoints of people in the organisation, which potentially involves conflict. The concept of participative BPR is thus proposed to answer the high rate of failure of BPR due to the adoption of modelling procedures that do not involve the participation of organisation members. As we cannot reengineer processes which have not been engineered before, we need adequate models for analysing the operations of proposed processes if we want the redesigned processes to perform successfully. We have given a preliminary introduction to our BPR requirements models in section 3.4. We will give further details of how EM has potential for BPR later in this chapter.

6.2.1 Participative Process Modelling

Participative process modelling can be interpreted in two different ways: *modelling of participative processes* and *participative modelling of processes* (Chen et al., 2000b). The term 'process' used here broadly refers to a generic sequence of state changes which follow a reliable pattern[2]. This broad definition is similar to that of Warboys et al. (1999) as they describe in their work that:

> In its loosest sense *process* relates to flux in the real world, to observable progressive changes of state of the world. ... A process is structured change, i.e. there is a pattern of events which an observer may recognise across different actual examples (or occurrences) of the process, or which may be made manifest, or implemented, in many different occurrences. (p. 32)

When we think about how processes are conceived in the business area (i.e. the business processes), there are two kinds of actions involved: the manual actions by human being and the automated actions

2. The processes of changing the state may or may not be necessary to achieve a specific purpose. For example, many natural processes exist and always have existed. But the business processes exist for a specific purpose to change the state of the real world to suit human needs.

by systems. These can be described in terms of two kinds of agency. The first is associated with the *internal* agents which enact the process and which are responsible for the state change. Their interaction can be referred to as a 'participative process'. The interactions of the internal agents may be governed by strict rules and reflect different degrees of autonomy. The second is associated with the external agents whose observation and comprehension of the interactions within a process rely on the integration of many agent viewpoints. Their interaction can be referred to 'participative modelling'. The external agents are typically responsible for designing and managing the process, but may also act simply as observers. As discussed elsewhere in this thesis (see sections 5.3 and 6.4.2), both kinds of agency can be supported in EM (cf. Beynon (1997)).

By way of illustration (cf. Figure 5.7), when a use case is a means of describing a system functionality or a business process, the objects in that use case are internal agents whereas the actors, other systems or other stakeholders, etc. are the external agents for that process. The distinction between internal and external agents can help to refine our thinking about the issue of BPR. The basic tenet of BPR is to gain competitive advantages by rethinking business processes and the use of IT for the redesigned processes. Such rethinking can be made from two directions. External rethinking is made from the external observers (generally from the customers' viewpoint) of the business system and its products or services. The aim is to make the business more responsive and effective. Internal rethinking is made from the viewpoints of component agents of the process. The aim is to find the barriers in that process or any difficulty these members may meet, and thus look for new ways to make the process more efficient. The work of BPR thus should combine both kinds of rethinking: to provide the personalised empowered environment for individuals and to develop the supporting technology for the automation of business processes and the collaborative environment.

But in most BPR contexts, as mentioned in chapter 3, the creation or modification of computer systems to support the newly designed business processes is not suitable or even has a negative impact on the business. This is because the complex issues of human factors are not taken sufficiently into consideration during the BPR work. This is why Warboys (1994) says that, "the general BPR approach could be said to suffer from a tendency to treat organisations as machines rather than ecosystems".

Such rethinking is mainly based on the external viewpoints and the redesigned processes are framed in terms of preconceived interactions to serve particular purposes. So the models are designed for describing 'what' the business process looks like, but cannot express 'why' the process has a certain form, and the motivation, intents, etc. behind the activities. This reflects the potential problem of circumscription in conventional modelling methods which has been described in chapter 2. That is to say, it may be necessary to pre-define personal responsibilities which constrain the individual and define the means of individual cooperation with others to ensure the business system can function appropriately. So we can imagine – as in the age of information technology a large amount of work is performed by individuals at computer terminals – that the constraints, or even the rules, are defined by the capabilities of the information systems which provide the means of interaction among individuals within the organisation. But ignoring such human factors when modelling business processes may be dangerous. Most business processes involve many participants with complex relationships among them, and the results of changing the current process or employing new IT systems are subject to the complexities from these relationships. The human factors associated with these relationships and the insights of individuals should be considered because each of them is concerned about opportunities and seeks to protect or further their interest in the attempt at redesigning the process. Understanding the situations and the usage of systems requires that we understand factors other than the frequency of use for specific features or the patterns of use expected of some users. Without considering or understanding these factors, it will be difficult to integrate existing systems (the problem of legacy systems) or to design new systems to meet the changing needs. When considering the relationship between system development and BPR, Warboys points out the need to consider the following:

- The business is managed as a set of business processes;
- These business processes are modelled in software;
- The IT strategy should be integrated with the business strategy;
- Designing the software is just part of designing the business.

Although the design of the software system does radically impact the design of the business, Warboys further points out two important points ignored by conventional BPR approaches:

- The software system is still a software system with all the old problems of versioning, integration, complexity, etc;
- The rules governing good architectural practices in software are not the same as the rules governing good business structures.

These problems may be even more tricky for the development of business applications, especially when such programs are parallel and asynchronous. Another prime difficulty of designing software systems for business is to achieve a balance between providing flexible and adaptive systems and maintaining and promoting group working (Warboys et al., 1999). This is important for two reasons: because designing support systems is not simply to assemble all the tools and applications and provide the users with an executable access; because there exist dependencies between the group members and each member in the group can achieve more through the cooperative actions than he could do alone. Yu and Mylopoulos (1994) describe the human beings in organisations as social actors who are *intentional* – have motivation, wants and beliefs, and *strategic* – they evaluate their relationships to each other in terms of opportunities and vulnerabilities. Thus when modelling business processes, we should not only describe the entities and the activities (to be automated) within processes, but also the concepts about how, or why, the actors are performing such activities, for example, their goals, abilities or commitments. What we need is a flexible modelling approach that can not only describe the context that the individual members need in order to handle the tasks as freely as possible, but also informs other members in the organisation about how each individual output will affect them. It is important when BPR is seeking new ways for the operation of a organisation, that the goals of each member's interactions are understood by those who participate in them. Without considering such factors, the efforts and results of BPR may simply become using computers to automate existing or outdated processes and will neither increase efficiency nor realise the true potential offered by IT.

6.2.2 Participative BPR

The importance of stakeholders' active participation in the process of BPR has been described earlier in this section and in section 3.4. The purpose of BPR is to employ IT systems to achieve radical change in the business processes and to obtain maximum improvement and performance, and the main potential advantage of our EM approach over conventional methods is in helping the management to find an optimal solution to the many variants of the business problem. As showing in Figures 6.4 and 6.6, conventional approaches involve an abstraction phase to analyse and abstract the problem domain and preconceive new solutions (or systems) in terms of abstract representations such as objects. Under this closed-world paradigm, the end-users are not involved in this abstraction phase, and any further changes to the final product due of their being unsatisfactory will result in another costly development lifecycle. Timetabling can be taken as an illustrative example. Computer-based support for timetabling may be directed at replacing manual processes by algorithms which aim to perform a labour-saving task and reduce the human effort as far as possible. However in the manual timetabling process, human judgements play an essential role which is usually influenced by various environmental factors explicitly or implicitly. Human timetablers explore different possible temporary solutions based on resources available and make qualitative judgements. The judgements they make usually stem from their tacit knowledge which may emerge during or after the construction of the timetable. This tacit knowledge is the kind of knowledge which is used by individuals and is difficult if not impossible to transfer or represent. It is argued by Baets (1998) that if we try to transfer tacit knowledge, then it will become explicit and hence lose its distinctive character. He further concludes that transferring tacit knowledge can only be done via a process of joint learning. That is to say, if people live together through the same experiences, they can learn from each others without making their mutual knowledge explicit. Thus the aim of the EM approach is to provide an environment for participants to communicate and 'visualise' their thinking and knowledge through their interaction with the computer model to achieve the purpose of 'organisational learning' (Baets, 1998). Baets's argument also highlights the potential problem of conventional methods in preconceiving manual processes and explicitly representing human knowledge about timetabling during the abstraction phase. The result is that the human timetabler has to conceive

all the possibilities of the timetabling activity during that phase. Through this development the tacit knowledge of human timetabler cannot normally be captured or can only be captured in a limited way.

Some of the considerations that apply to timetabling also apply to BPR. For BPR, any person in a business process has his own perception of that process, and knowledge (include tacit knowledge) about business processes that is closely related to these perceptions. Thus it is important that BPR should take account of this kind of knowledge, and this knowledge can only be considered in the real-world context rather than in the abstract. In this connection, Stowell (1995b) observes that "the incorporation of computing power is more likely to be successful if undertaken by a person with knowledge of the problem than by a person with knowledge of the system". We also find that the trend in information system development has been to develop systems with languages and interfaces increasingly closer to the forms of human communication and mental representation. For example, by the mid-1990s as Crowe et al. (1996) observe, many facilities for implementation were at the application level and the level of tools and operations were meaningful to end-users. This can explain why today many businesses and industries are increasingly appreciating the importance of involving users in the development process.

We propose the concept of participative BPR which means that people (especially the end-users) participate in the process of BPR in such a way that their individual actions can form an input into the BPR process and also lead to an organisational interaction (and learning) with the environment. All these participants can also get feedback, both from their interactions with other members and from the global responses of the environment, by which they can learn and update their beliefs about the relationships of cause and effect. Because business processes and the process of their changes (during reengineering) are dynamic, it is not sufficient to describe and model these using the abstract (context-free) notations. What we need is a flexible framework to model the continuously changing situations and allow people with different perceptions in the business process to have access to their own environment in the modelling process. The characteristic of our participative concept is that we are aiming to enable the modeller to capture partial solutions, and to explore 'nearby' solutions, and opportunistic uses of them, in an experimental manner. This contrasts with the representation of partial solutions in a conven-

tional program which merely traces a path in the construction of a solution from the point of defining the problem to the point of proposed solution in the problem domain. In particular, we are focusing on *situation* rather than *process*, as a situation relates to immediate experience whereas a process is concerned with sequences of situations following a particular pattern.

The aim of this research is to adopt an alternative perspective on computer use which is primarily concerned with constructing computer-based models to support the task of BPR. For this the central issue is the relation between actions which are circumscribed and situated actions where, as mentioned in Chen et al. (2000b), "we have to address unprecedented problems, engage in interactions of a serendipitous and creative nature, and isolate the cues for process interaction from a world where sensation can be overwhelming and action unconstrained". This issue plays a centre role in participative BPR: we have to consider the subjective nature of an individual participant's perception as well as their capability. The main consideration is to disclose the tacit knowledge of participants which plays an essential role in the successful execution of business processes.

In the warehouse case study to be discussed in the next chapter, the external objective description of the processes can be represented by the interactions of actors with filling and distributing paper forms. One advantage illustrated in the warehouse case study is that the participants are human, thus we can hope to understand the perceptions and responses of the participants to some degree. The form processing, which abstractly represents the real world activity of the warehouse, involves the physical movement of items, the observation of their location, and the monitoring of their status. Such activity is essentially situated, and is associated with the complex perception of individual participants of many aspects of the current state. The internal insight into the warehouse operation requires an account of each participant's activity which is based on their direct personal experience of the environment. What we need is a kind of computer-based modelling of processes for understanding and reengineering via participative modelling. In which the aim is to develop a suitable computer-based representation for the state-as-experienced for each internal and external participant.

6.3 The SPORE Framework

SPORE (Situated Process of Requirements Engineering) is a human-centred framework which was initially proposed by Sun (Sun, 1999; Sun et al., 1999). It is problem-oriented because the requirements in this framework are viewed as the solutions to the problems identified in the application domain. The requirements are developed in an open-ended and situated manner. It is open-ended because such requirements cannot be completely specified in advance; and it is situated because the context presented in SPORE is closely connected to the referent in the real-world domain. Within this framework, people participating in the requirements process can cultivate requirements through collaborative interaction with each other and aim to solve the identified problems rather than search for requirements from the 'jungle' of user's needs (Sun et al., 1999).

The SPORE framework for building situated models for the process of requirements engineering is depicted in Figure 6.1. Three kinds of inputs of SPORE are:

- *Key problems* of the domain which are identified by the participants in seeking to address the functional and non-functional requirements of the proposed system. Such identification of problems can occur at any time during the requirements process and is never regarded as completed.
- *Relevant contexts*, such as the organisation's goals and policy, as well as the relationships between participants, which act as motives and constraints for the participants in creating the outputs.
- *Available resources*, such as documents, technology and past experiences of participants, which are used by participants to facilitate the creation of SPORE outputs.

There are also four kinds of outputs from SPORE. The main one is *provisional solutions* to the identified problems which are developed by participants on the basis of the available resources and the current contexts. The other three outputs, which include *new contexts*, *new resources* and *new problems*, combine with their earlier versions and in turn form the new inputs to the model for creating the next outputs. That is, all these contexts, resources and problems are modifiable and extensible in SPORE. Thus par-

ticipants can develop requirements in a situated manner to respond to the rapid change in the contexts, resources as well as the problems themselves. This framework addresses the concern raised in chapter 2 that requirements are changing all the time and can never be regarded as complete.

6.3.1 The SPORE Framework for Cultivating Requirements

There is no specific activity nor fixed ordering of activities in the SPORE framework, since the problems to be identified during the requirements process are various. For example, some may be difficult to solve given the specific context and the resources available; some are interdependent and need to be solved concurrently. In describing the drawbacks of the closed-world paradigm in system development, we have already concluded that no rule or algorithm can be applied to take all the various factors into account in advance. Different problems need solutions by different methods depending on different situations. That is why the SPORE framework enables the participants to take the current context and available resources into consideration to cope with the diverse and unexpected issues raised during the requirements process.

As mentioned earlier, the main activity of participants in the SPORE framework is to cultivate requirements by interacting with each other as well as with the referent and the external environment to

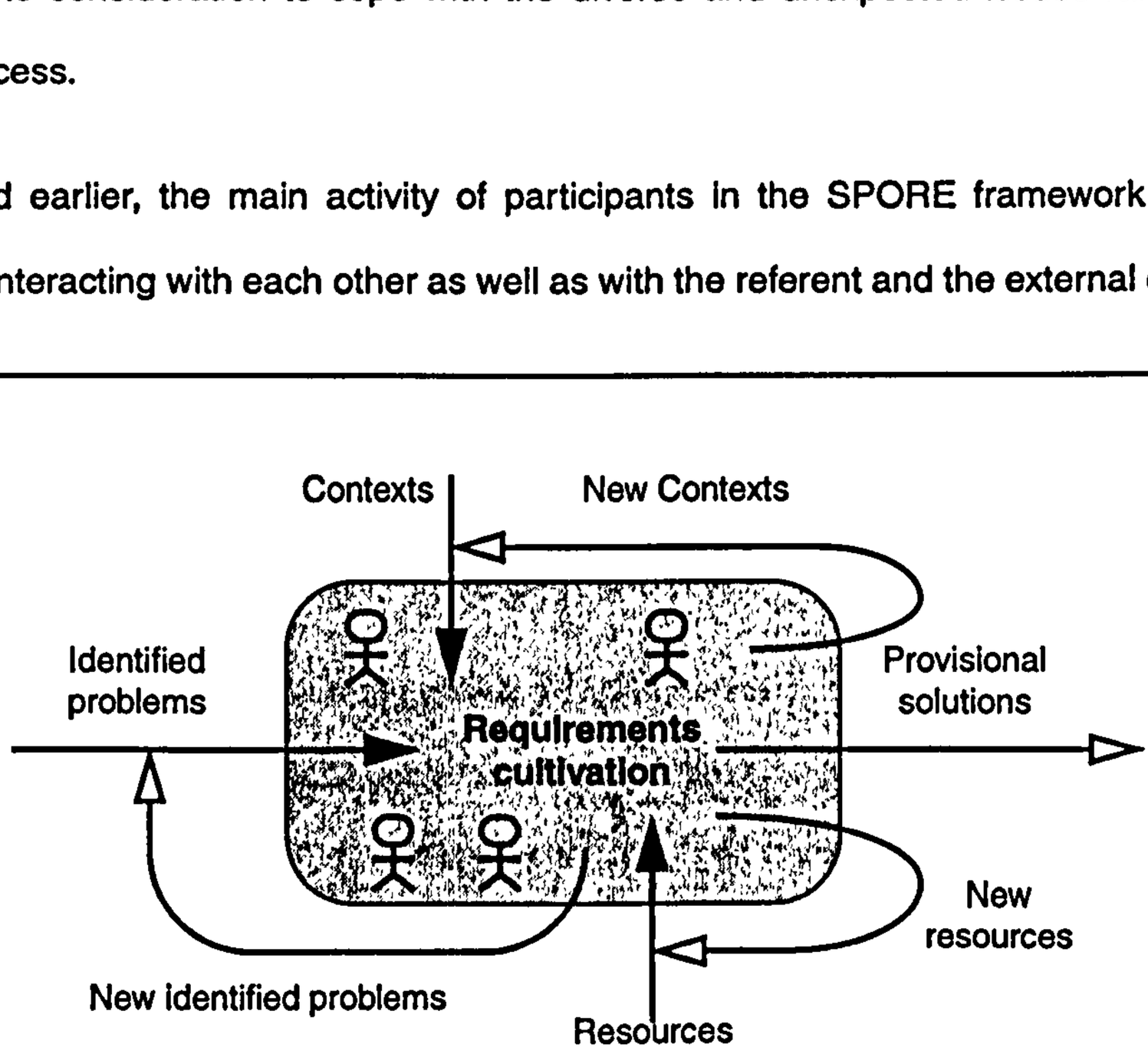


Figure 6.1 The SPORE Framework

develop the requirements (i.e. the solutions to identified problems). Sun et al. (1999) say the term 'cultivation' is used to express the idea that, "requirements (like plants) should grow gradually rather than be conjectured from their initially fragmentary, chaotic and rapidly changing states". Extending the metaphor, Sun et al. make the complementary assumption that the requirements are pre-existent but hidden in the sources just like the grown plants in a huge jungle. So an additional purpose of the SPORE model is to elicit the appropriate requirements just like searching for the right plants from the jungle. The elicitation of appropriate requirements is a difficult and challenging task because there exist various and unexpected factors and elements during the process of requirements, just like searching the right plants from the jungle with a mixture of numerous kinds of elements.

In SPORE the requirements are cultivated through the collaborative interaction among the participants. The focus of the process is neither on the problem domain nor the solution domain, but on the interaction by which the problems are solved on the basis of the current context and the resources available. The participants, who can be designers or end users, are working together to solve problems. In this way the solutions are not just in an individual's mind but distributed among all participants. In addition, the solutions are developed in an iterative and incremental manner through the participants' interaction with each other. Thus the requirements are cultivated through various activities with different purposes. This can also rectify the bias of the insight of an individual participant, which is always limited by his context, expertise, and the resources available at the specific time.

6.3.2 Applying EM to SPORE

The models that feature in a requirements specification in conventional software development are usually too abstract for it to be possible to get a detailed understanding of the model by simply reading it. This situation may be even worse if the application domain is a complex system. Therefore having a way for the participants to experience and visualise the behaviours defined by the model is essential for system development. The SPORE framework involves constructing computer-based artefacts to be used to explore and integrate the insights of individual participants in an interactive manner. The arte-

facts created are ISMs and are based on the principles and tools of EM. That is to say, within the iterative and incremental framework of SPORE, the ISMs are built in a structured development cycle, and new elements are added into the whole model during each phase (cf. subsection 5.3.1). The insight of the individual participants can be reflected through the consistency between the construal in their mind, the context of the models, and the situation in the external environment. Progress can be achieved through the continuous feedback and evaluation gained by the participants.

In EM the knowledge of participants is constructed in an experimental and not a declarative manner. That is, the insight of the participant is expressed by the coherence between what he expects in his mind and his experiments with the ISMs and the external referents. His insight can also be extended through 'what-if' experiments. Any introduction of new definitions, or redefinition of existing definitions, will evoke changes of state in the models and these in turn will affect the state of his mind through the visual interface. These results of experiments not only change the participant's individual insight but also are stored in his memory, which forms an important resource for the participants to take situated actions. This experimental interaction of EM is particularly powerful in SPORE because the participants can interact with each other as well as with the computer model. Figure 6.2 illustrates how the participant takes situated actions to explore his individual insight by interacting with the computer model as well as the external environment, based on the current context and resources available. Through the distributed EM tools the interaction of an individual participant (i.e. adding or redefining definitions) can be propagated to other artefacts and thus affect the views and insights of other participants. In this way the participants can collaboratively interact with each other through their artefacts.

The distributed version of EM tool – **dtkeden** – which was developed by Sun (1999) provides a computer-based distributed modelling environment in which the interaction between modellers is computer-mediated. As the networked computer-based models act as a communication medium for the modellers' interaction, modellers do not need to look at each other or use verbal language for interaction (through this may still be desirable where it is impossible). The visualisation of these models represent the construals of modellers. The 'communication' is achieved through changing the computer models by modellers, and such change is passed through the network to affect other computer models

and thus 'tell' other modellers (the listeners) what they are thinking. Four interaction modes have been implemented in **dtkeden**:

- The *broadcast* mode: This mode provides a broadcasting function by which any message sent to the server will be propagated to all other clients. In this mode the server acts in the role of message-transferring centre[3]. This supplies an open environment for multi-agent modelling, such as is needed for a multi-user game.
- The *private* mode: In this mode, each client has a private communication channel to the server. In contrast to the broadcast mode, the message sent by a client will not be propagated to the other clients. Because it is possible for more than one private channel to exist in parallel, this private mode is suitable for environments such as computer-supported classroom teaching where many-to-one communication is required.
- The *interference* mode: This mode allows the superagent (in the server model) to directly interfere with the interactions between agents (the clients), or between each client and the server. That is, each request sent from clients is displayed on the server's input window. The superagent can then have discretion over how the suspended request is processed, e.g. to perform 'what-if' experiments.

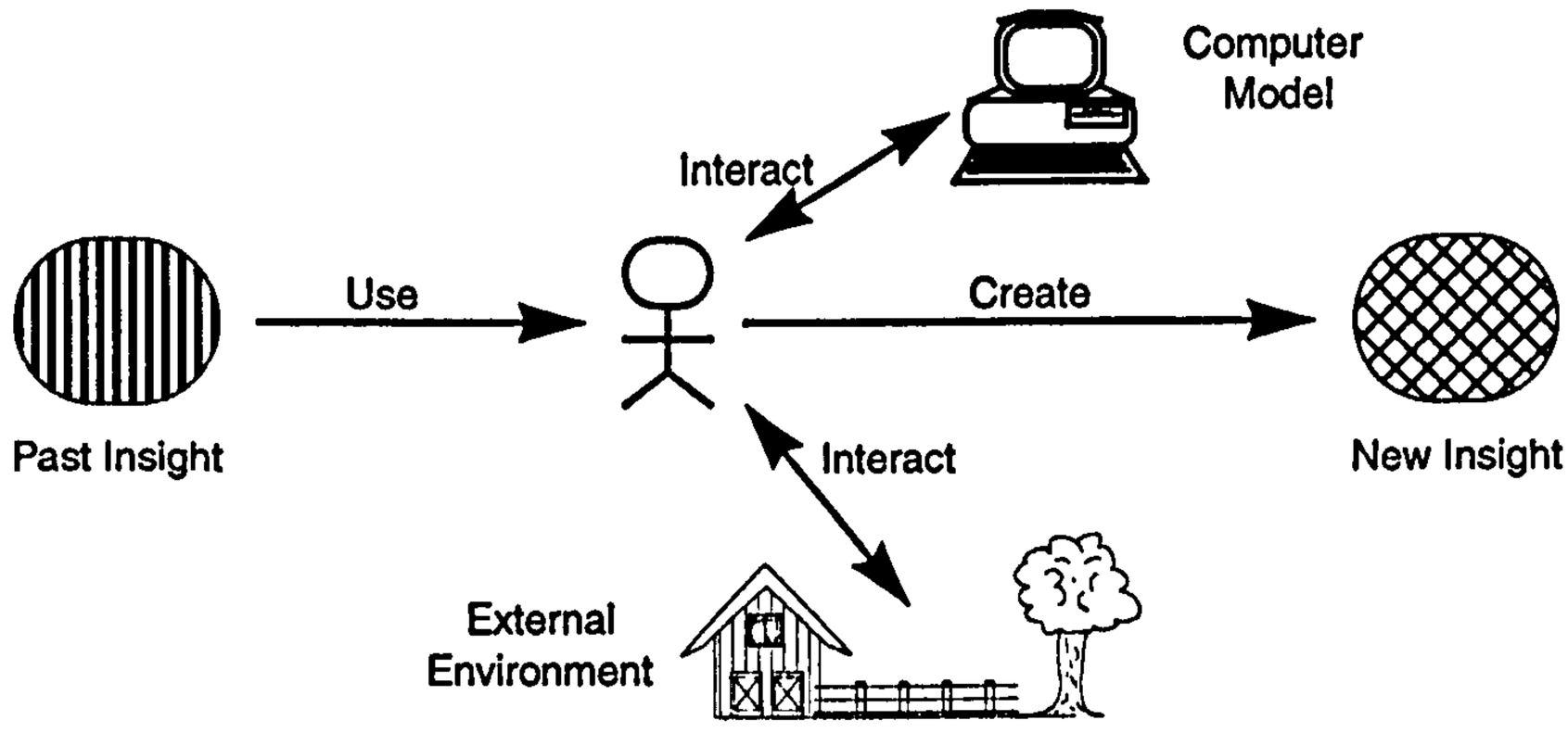


Figure 6.2 The Experimental Interaction of a Participant (adapted from Sun et al., 1999)

3. That is, each message sent from a client will firstly come to the server, and will be automatically broadcast to all other clients and consequently change the visualisation of the server's and clients' computer models.

- The *normal* mode (default mode): In this mode, the interaction between clients is mediated by the computer with reference to specified privileges of modellers to access observables and change the definitions. Any request will be accepted by the server, subject to the current privileges of that agent[4]. This mode is good for hierarchical modelling environments where the relationships between agents are known. The LSD notation supported in **dtkeden** can be used to set up the privileges.

This distributed EM tool supplies the framework for the collaborative environment in which the shared understanding of the key problems and their solutions (i.e. the requirements) can be established (cf. Figure 6.3). The requirements are cultivated through the interaction between participants and the exploration and integration of their individual insights. The greater the consistency between the interactions of the participants in the distributed ISMs, the better their mutual understanding and the more appropriate are the requirements that have been developed. Furthermore, the participants can continually refine their interaction in order to achieve greater coherence and consistency.

The four modes of interaction provided by **dtkeden** address the needs of modellers to participate in a cooperative modelling environment under different situations, or cultivate requirements for different applications. For example, with the broadcast mode of interaction, the modelling environment is very similar to that of an electronic group meeting without video and audio facilities. All the participants share all messages from each member and typically interact with each other in an iterative manner to reach consensus. According to Sun (1999), this mode can be used to develop a system such as a multi-user game which requires a shared environment for supporting the interaction between its users. The private mode is suitable for a many-to-one modelling environment such as the classroom project in which a teacher (at the server) can monitor the learning process of each student. The interference mode enables the server as a superagent to do 'what-if' experiment to explore different scenarios, to present significant situations to clients and result conflicts between their viewpoints. For example, where two

4. The access privileges of modellers are given so as to reflect the management hierarchy and social relationships between modellers (Sun, 1999). For example in a system design process, users are not allowed to change the observables associated with implementation; even designers from different groups may need to be restricted to access different observables.

designers propose different values for the same observable, the superagent can arbitrate and suggest or impose a compromise value. The interference mode is particularly useful in modelling the openness of real-world phenomena; the introduction of an unexpected context by a superagent can enrich the understanding of all the modellers involved in the process. It is especially important in representing the perspectives of external observers (in what we have described as participative process modelling in section 6.2) for the process of system development and BPR.

One of the most important benefits of interacting with the ISM, as discussed above, is that it makes the individual insights and the shared understanding visible and communicable. This overcomes the disadvantage of invisibility and incommunicability of shared understanding based on conventionally test-based models. This experimental interaction can also keep the requirements synchronised with the shared understanding among participants, which evolves faster than textual specifications. That is, the way in which the evolution of computer models and the individual insights are synchronised allows the

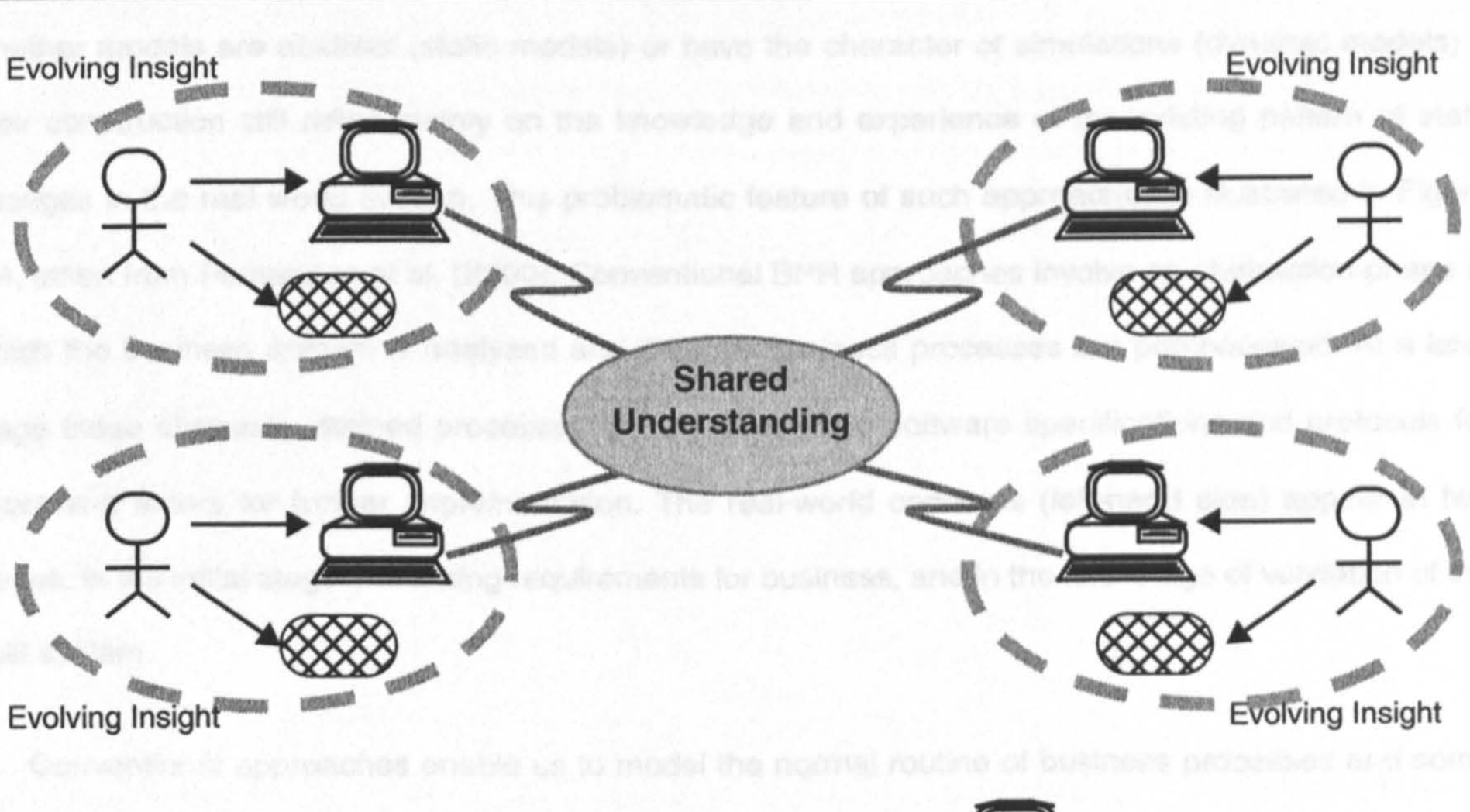


Figure 6.3 A Collaborative Working Environment for Cultivating Requirements (adapted from Sun et al., 1999)

participants to 'see' other participants' viewpoints and to 'communicate' with them by interacting with their own artefact (Sun et al., 1999).

6.4 Applying EM to System Development and BPR

The conventional approach to business process modelling is first to make an in-depth analysis of the objectives of that organisation and business needs, then to construct the specification of the requirements, and subsequently to build models based on this specification. Under this paradigm, the modellers have to preconceive all the normal situations as well as the potential error situations or exceptions. This activity is guided only by discussion with the management and sometimes with the personnel who have experience in the practical execution of processes. This is insufficient for business process modelling, because any minor change in the business context which is considered insignificant during the construction of abstractly mathematical models may become the crucial factor for the success or failure of the real execution of newly designed processes. In conventional approaches to process modelling – whether models are abstract (static models) or have the character of simulations (dynamic models) – their construction still relies mainly on the knowledge and experience of the existing pattern of state changes in the real world system. This problematic feature of such approaches is illustrated in Figure 6.4, taken from Fernandes et al. (2000). Conventional BPR approaches involve an abstraction phase in which the business domain is analysed and the new business processes are preconceived. At a later stage these abstractly defined processes are translated into software specifications and protocols for users and actors for further implementation. The real-world concerns (left-hand side) appear in two places: in the initial stage of defining requirements for business, and in the final stage of validation of the built system.

Conventional approaches enable us to model the normal routine of business processes and some exceptional situations to some degree. However, we cannot preconceive in advance all the circumstances which may arise or the hidden factors which may be essential for the successful BPR. The context or the system which can be modelled in the traditional paradigm is the application of the modeller's

pre-existing knowledge about the business process to be modelled. Normally, conventional modelling techniques focus on the activities which can be captured in systematic processes. But, as Evans et al. (2001) point out, such techniques cannot take account of the human observation and assumptions about the context which are tacit in normal successful execution. They cannot also be integrated with the human intervention which is necessary to deal with unforeseen or unpredicted situations. The potential problem in conventional approaches is that the users are not able to give concrete expression of their ideas – in the technical notations with which they are unfamiliar – without being able to intervene and negotiate in the development process. Thus there may be implicit 'politics' which are against the interests of users, and inhibit the capturing of user's expertise in the domain.

In the SPORE approach, the models are intended to deal with the knowledge which is discovered by the participants of the process through observation and interaction in the real-world environment. Our approach focuses on modelling the states as well as the potential agency in the environment where the process to be modelled is enacted. This differs from conventional approaches in which the modelling describes the preconceived patterns of state changes and transitions which define the process in an abstract and implicit manner. No matter how we choose observables to describe the process, there

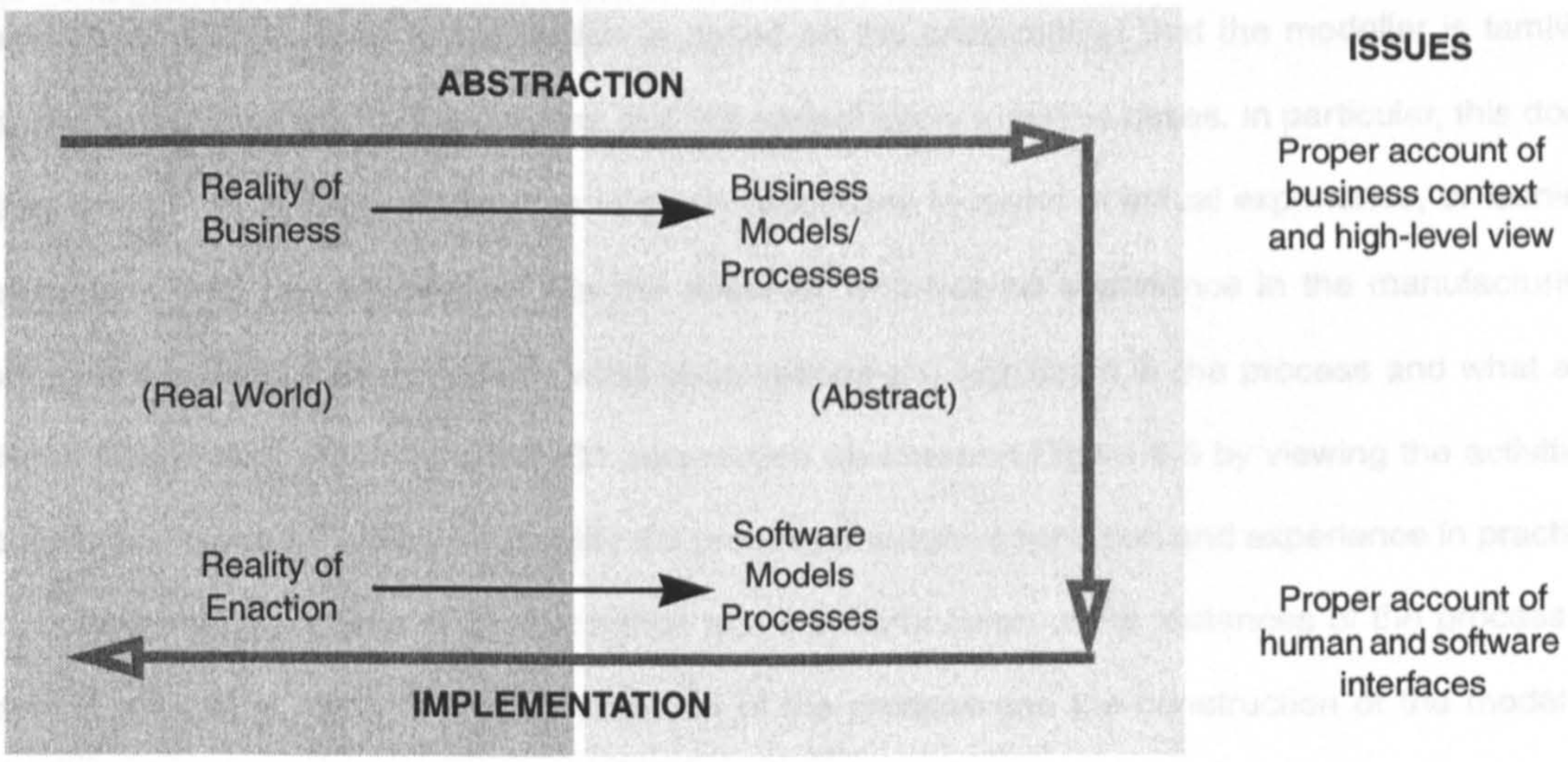


Figure 6.4 The Challenge of Realistic System Engineering (adapted from Fernandes et al., 2000)

will always be observables which will not yet be taken into account. In EM, the states of the model are associated with 'state-as-experienced', and only represent implicitly the particular patterns of state change enacted by the modeller. Through the construction of the computer model within EM the modeller can trace the process, and take account of any environment factors which are considered to be relevant, in an open-ended manner. That is to say, the definitive script cannot include all the observables associated with the states to which it refers, but the modeller can interact with the model at any stage and this "allows the choice of observables to be opportunistic rather than constrained by prior commitment" (Evans et al., 2001). This relates to the issue of circumscription which we have discussed in chapter 2.

Following the discussion in Evans at al., we can illustrate the difference between the EM approach and other conventional approaches in process modelling with reference to the context for rework[5] in the manufacturing process. Figure 6.5 shows the three main activities involved in a manufacturing process: the standard work within the normal scope of the process, the routine rework where the nature of the rework problem is familiar, and the exceptional rework where unprecedented or ill-understood problems are encountered. Conventional approaches to process modelling entail viewing the activities from the inner to outer, i.e. normal operation of the process is considered first, then the routine rework, and then the exceptional rework. Such interpretation is based on the presumption that the modeller is familiar with, or has experience within, the process, but this cannot apply to all the cases. In particular, this does not apply when – as in EM – we wish to interpret processes in terms of actual experience, or fashion new processes from raw experience. For the observer who has no experience in the manufacturing environment, it is difficult to distinguish what observations are significant in the process and what are associated with rework. Thus from the EM perspective we interpret Figure 6.5 by viewing the activities from outermost inwards. That is, we identify the process through observation and experience in practice which involves a long process of familiarisation and the observation of the instances of the process in operation (Evans et al., 2001). The identification of the process and the construction of the model is

5. 'Rework' refers to the standard processes for recording and/or remedying flaws in manufacturing products, where these have been anticipated.

made by the modeller step by step in an interactive manner which resembles the process of observation and experiment in an unfamiliar and ill-defined environment. This is a natural way of modelling as our conception and evolution of the process are subject to the identification of reliable patterns of interaction from the experience of participants. This evolution and the growing familiarity from participants' experience have the characteristic of ongoing understanding and reengineering activities which is closely associated with the agenda of BPR. In the next chapter we will show how to apply EM principles and tools to a warehouse management system, and how the computer-based modelling can support the execution of processes and the development of a useful system.

6.4.1 Understanding System Environment with LSD

When we redesign business processes and in turn redesign computer systems to meet the new requirements, we usually need to have a broad understanding of the organisational environment in order to make decisions about what changes to make and which parts/components should remain. The reason is that when we define the requirements for the business or its support system, the main work is to investigate the future vision and the impact of the changes. We have described the concept of systems thinking in chapter 2 and emphasised its focus on 'emergent properties' (i.e the ways in which the whole is greater than the sum of the parts). This means that the implementation of system development

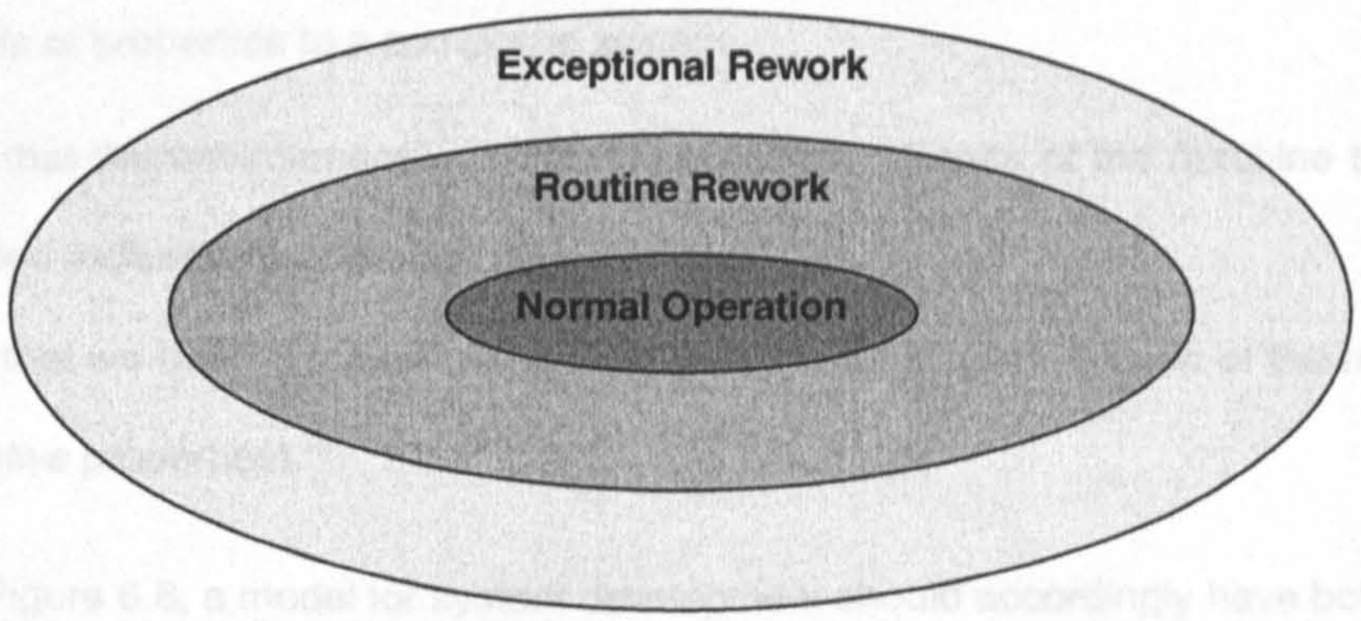


Figure 6.5 Normal Operation, Routine Rework and Exceptional Rework (adapted from Evans et al., 2001)

or business modelling should also refer to the higher level (e.g. on purposes or the future vision) when focusing on relationships at any level. Section 2.1 also mentions the difference between systems thinking and other approaches (e.g. reductionism and holism) that may indicate why conventional methods for system development have failed to meet the actual needs for a business and its users, and why we need the systems approach to analyse the impact of the system upon the business (cf. section 3.3).

The move towards to the future vision must consider the context which is shaped by the past. As Checkland and Scholes (1990) remark of SSM: "it was found useful to think of an intervention in a problem situation as itself being problematical". This means that for modelling we need to understand the structure of the environment in which we are involved, and our first concern should be to ascertain what our intervention is intended to achieve. Davenport (1993) puts forward similar arguments in respect to BPR. For example, in the framework he proposes for BPR implementation, one step is the understanding of the way existing processes are performed. A view of the existing processes is necessary in order for problems and pitfalls to be identified and eliminated. Moreover, especially in the case of complex and specialised business processes, the reengineering team might need to get familiar with the nature of the current work before designing the new processes. Modelling and understanding existing processes will provide a basis for comparing the impact of new processes on specific metrics.

As Jackson (1995) describes, the pre-existing components of a composite system are the elements of the environment into which a new machine or system is to be introduced. In this respect, Jackson defines two kinds of properties to a composite system:

- properties that the environment intrinsically possesses, in spite of the machine to be introduced (the so-called *indicative* properties); and
- properties that we hope the environment will satisfy after the introduction of the machine (the so-called *optative* properties).

As depicted in Figure 6.6, a model for system development should accordingly have both kinds of properties: the indicative ones, which refer to properties already in the environment, and the optative ones, which refer to properties of the environment that the proposed system will include. In Figure 6.4, we see

that there are two kinds of 'world' in system development: the real world (the left-hand side) and abstract world (the right-hand side). The real world includes the real-life context (the upper block) for our daily work and the business activities, and the technical world (the world 'inside' the computer systems) which comprises both the hardware and software. The two kinds of properties, the indicative and the optative, feature in the real world (i.e. the business domain) in which the computer systems are used. The abstract world is mainly shaped by the conceptual analysis and design. It is abstract because many of the complex technical details are 'hidden' in order to enable the designers to control the development more easily and to search for solutions at the logical level. For instance, because of the presumption that as much as possible of the human process will be automated (i.e. become computerised), the workflows (by means of information flows) between human agents are replaced by data flows within the computer-based system.

Conventional approaches to system development and BPR mainly focus on the activities in the abstract world and thus various techniques and tools are developed for the tasks and procedures in this

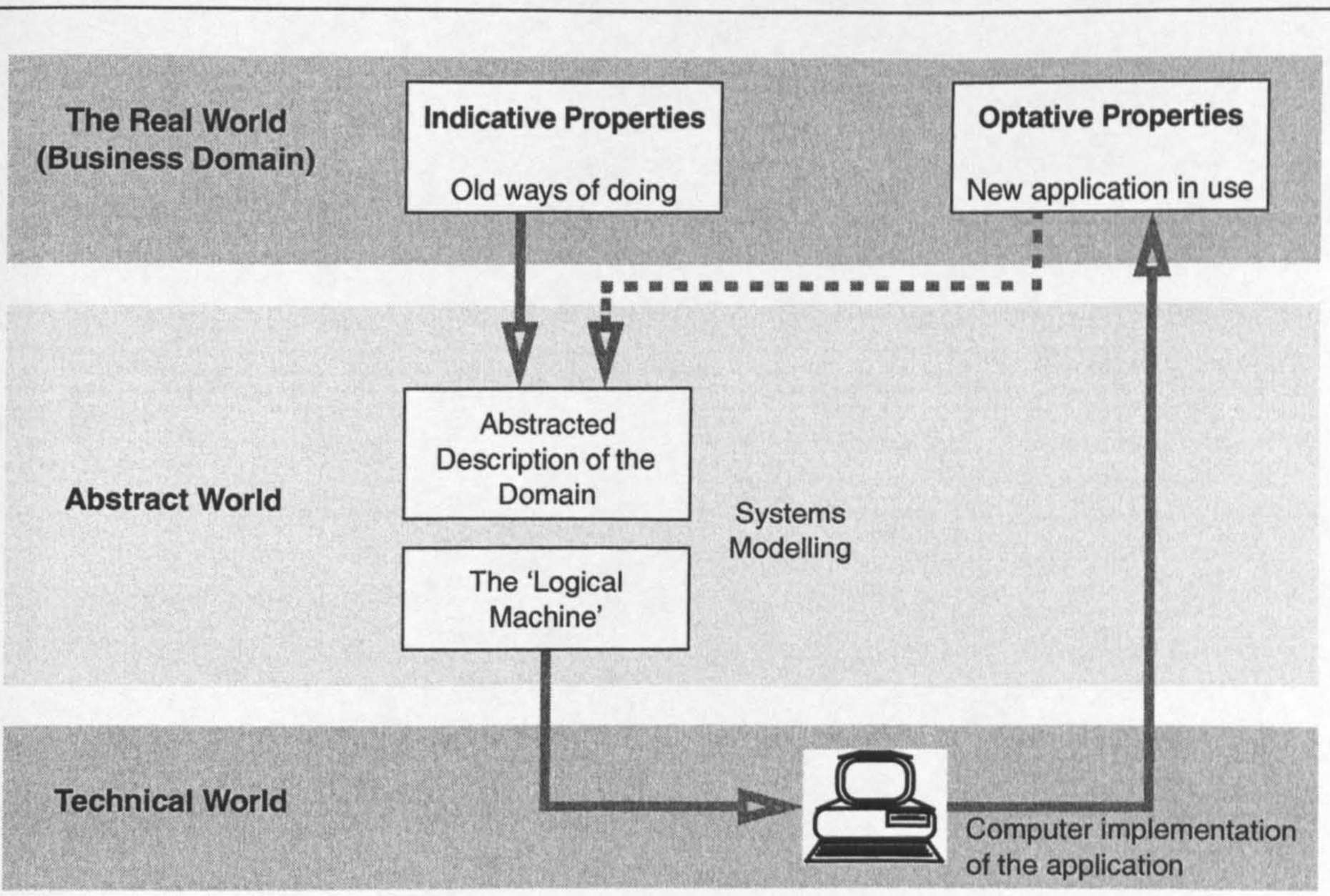


Figure 6.6 The Real World and Abstract World in System Development

world (cf. Figure 6.4). Generally, these approaches can be regarded as product-oriented because they abstract concepts from the characteristics of the systems and consider the usage to be fixed, thus allowing the requirements to be predescribed. But as Figure 6.6 indicates, the business processes in the real world will always be influenced by the use of computer systems and thus change the situations. It is difficult to explicitly define our situations, or the business processes, in a preconceived way.

Many commentators highlight the difficulty on making the transition from the real to abstract world. Crowe et al. (1996) criticise the replacement of human workflows by computer dataflows in the process of automation as too reductionist[6], but also caution against too close an identification of human and machine perspectives that might stem from more holistic view. In this context, the observation of Stowell (1995b):

> It seems clear that the method by which we develop technology-based information systems, and the manner in which we use them, will have a significant effect upon the way the systems behave and also upon the actions of those who access the data held. (p. 121)

and Winograd (1995):

> The overall environment of computer use is a constant co-evolution in which new tools lead to new practices and ways of doing business, which in turn creates problems and possibilities for technical innovation. (p. 73)

are also relevant.

In EM, we prefer to view the development of the computer system and its software associated with human learning and communication that should take place in an evolving world with changing needs. In this view, the business processes should impact on both system development and their use. The focus should be on a better understanding of users' needs – in the real world business domain – and such needs can only be revealed from the business domain in which the systems are going to be used. The

6. And too holistic when with respect to the customer and the organisational environment (treating them as things).

aim of EM is not to provide another efficient system development method by prescribing some technique (i.e. in the abstract world). Rather it is to make the understanding of the business domain (the problem area), open the area for concern for human activities, and the elaboration of the business solutions proceed in parallel with the development of models. For the success of BPR, we need to understand the way the business works. EM not only assists business comprehension but meets the need identified by Stowell (1995b) for analysis and design methods take account of subjective as well as objective viewpoints:

> Explanation is sought from the perspective of the observed rather than the observer, and social reality is considered to be a function of an individual's self-consciousness plus assumptions, together with the shared meanings and beliefs between individuals and groups. (p. 125)

In the rest of this section we will give details of how such business-comprehension activities can be carried out using an EM approach.

The LSD account provides the framework for describing the system environment in terms of agents and their interactions. Agents perform the state-changing actions with different privileges which are constrained by protocols. In LSD, agents are modelled as depending on each other through the observables to which they can response, and which they can redefine. The identification and classification of the agency and observables reflect the modeller's observation of both the model and its referent. The LSD account records the observables which indicate the external view of a modeller about how the agents interact with, and depend on, each other, based on his personal construal of the subjects. It assists the modeller to gain a deeper understanding about the system environment and, through the animation by ADM or EDEN, helps to explore other alternative and sometimes unpredicted patterns of state-change.

LSD encourages the understanding of business situations by focusing on the observation and interaction of agents, which differs from the usual understanding based on activities or entity flows. Modelling the business processes in terms of dependencies and agency within the LSD account can provide

a higher level of description that does not presume fully preconceived activity, by which open-ended situations (such as exceptions) can be accommodated more easily. LSD can be regarded as a conceptual model and the semantics used in LSD (such as states, handles, oracles, derivates and protocols) are abstractly representing the concepts of the agent's ability and commitment in the situation. Unlike conventional approaches to business modelling, LSD has the potential advantage of managing large amounts of knowledge and dealing with large scale real-world situations in an open-ended fashion. For this purpose, the LSD account, as well as the EM models, are usually considered as of their essence incomplete. Any new agents or observables which are relevant can be included in the model at any stage.

From Process Redesign to System Development

As explained above, the LSD account frames an understanding of the system environment in a business domain. The next concern is to specify the systems requirements in order to develop the system to support the business. The transaction from business modelling to specifying system requirements is typically not straightforward. That is, we need to go back and forth between business models and system requirements because new issues discovered in one model will require us to go back and look at the other. Throughout the requirements process, the analyst needs to go back and forth between system requirements and organisational requirements in order to deal with the impact of one on the other. Thus the main issue here is how to combine the processes of these two modelling efforts. In EM, the conception of the interactions of internal agents, as represented in the evolving LSD account, can give a good understanding of the system environment. (This is the modelling of a 'participative process' in the sense of subsection 6.2.1). In parallel with this, the system requirements can be cultivated through interaction with the computer model and the associated exploration and integration of individual insights of the participants. (This is the 'participative modelling' of the process as carried out by the external agents in the sense of subsection 6.2.1). We now elaborate on how the EM approach can help BPR and system development by exploiting these two kinds or levels of modelling.

For understanding the system environment (the first level of modelling), we need a model which can capture situations in such a way that the actions and interactions of agents are not circumscribed. This means that they may violate the constraints which are pre-imposed in a conventional approach, so that the state-changes in the model are similar to those in the real business environment, and cannot be totally predicted. For successful BPR, we shall need to analyse such exceptional situations and the behaviours and insights of individual participants. The LSD model is mainly descriptive. It helps the modeller to understand the existing or proposed situations, as well as the conditions, in an organisation and can be seen as serving a strategic function. The LSD account can provide a strategic view of the reconfiguration of relationships associated with adding new dependencies (which typically means introducing new systems or processes, or new arrangements in the work environment in an organisation) in EDEN. Such changes may further alter the scenarios (paths) of business processes which the modeller or management deems to be possible, or impossible. The discovery of such new opportunities is an essential factor for successful BPR. The LSD account is viewed as incomplete because new strategic factors can be added and analysed as required.

The cultivation of the requirements of support systems (the second level of modelling), exploits the distributed variant of EDEN. Agents have states and actions, and their interactions are represented in terms of what they can observe and what they can change (i.e. the 'oracle' and 'handle' observables). The ISMs offer a higher level view than conventional approaches for requirements analysis and specification through the use of concepts of agency and observation. We have compared our EM approach with object-oriented approaches to system development in section 5.3. We will describe the use of EM in this respect in more detail in the next section.

We use LSD to evaluate the organisational environment and make more permanent revisions; whereas the analysis of support system requirements can be done by temporary redefinitions made directly to the ISM. As the organisational requirements change, they need to be reflected in the requirements of its system. The elaboration of the system requirements will further reveal more issues of the business which need to be addressed. We have described three categories of system boundary (i.e. hard, semi-hard and soft) in subsection 2.1.2 and we can conclude that the boundary of information

system development for business should be categorised as the 'soft' system. This means the process of business modelling is iterative, as both the organisational requirements and system requirements affect each other and accordingly need to be refined. Thus we need a unified approach to examine the issues of both the organisation and system development rather than different frameworks to be used independently.

6.4.2 EM for System Development

In today's business environment, information systems are becoming more interconnected to each other and are increasingly involved in complex business processes. That is, the systems used in an organisation (which are broadly viewed as information systems) need to cooperate with human beings to achieve the organisational goals. These systems engage with the business processes that involve customers, staff, etc, which together form the organisational configuration to achieve goals or services. So the information system development occurs in the context of legacy systems and business processes, which involves the issues of systems comprehension and BPR. For this, we can anticipate that the focus of system development will shift towards the understanding of organisational environment and needs, and conventionally focusing on the technical design and implementation in the development of single system will not be sufficient for the needs of today's business. In determining the requirements of information systems, it is necessary to understand the organisational environment so that the proposed systems (and the existing systems) can work well together with human beings. What we need is an open-ended and flexible approach to modelling the organisational environment and the behaviour of the actors and the support system. In EM, the agents, and how they relate to each other, are characterised in terms of observations. Through the construction of the computer model, the understanding and analysis of agency can be made concrete and amenable.

LSD and ADM in System Development

As discussed earlier, there is a need to have contextual knowledge in system development and BPR. In EM, contextual models operating at two levels are proposed for this purpose: the LSD account is served

to represent the organisational environment at an abstract level; whereas animation in the ADM supplies a more concrete model for representing such the information relating to the interactions captured in the LSD account and the interactions of the modeller. Both models enable the modeller to express his contextual knowledge about the real-world situations and thus support the understanding of interactions within the situations.

In the EM approach, the LSD account describes the contextual information about the organisational environment through observation and agency, which represents the modeller's viewpoint and interpretation of the domain. That is, the contextual knowledge of the modeller is represented by defining the actions and relationships between agents by means of observables and protocols. The LSD specifications can be animated through the use of the ADM, which represents the organisational environment on the lower and concrete level. By animating and interacting with the ADM, the modeller can use 'what-if' experiment in order to understand how a certain interaction, or a set of interactions, are performed.

Requirements Elicitation and Validation

As mentioned in subsection 6.4.1, a model should have both the indicative properties referring to properties of the existing environment and the optative properties referring to properties of the future environment in which the proposed system is to operate. We have described the developing procedure characteristic of EM in a previous chapter (cf. Figure 5.3). In the procedure, the elicitation activity (i.e. the arrow of new input) identifies the indicative properties of the existing system (the ISM) and its environment from observation of the real-world situations. It provides the support for producing the conceptual model (the modeller's construal) and generating goals from the interactions and experience of the observed system and model. Through this procedure, the interrelations between the abstract requirements (and goals) and the artefact of real-world referent can be established, and these can be used for the elicitation of system requirements.

Since the conceptual models of participants are too complicated and difficult to represent in notations which all participants can understand, we propose the cooperative validation of the conceptual

models within the SPORE framework. This is carried out by participants with different roles who can elaborate different behaviours through ISMs in an interactive experiment. During the experiment, different patterns of behaviour can be explored.

In EM, the construction of ISMs is closely linked to comprehension. Thus the interactions in the real-world domain and the development and validation of ISMs are interdependent. During the validation process (i.e. the arrow of test and experiments), the indicative factors identified during the elicitation activity will be extended and adapted to fulfil the additional optative properties. This validation is mainly achieved by correspondence checks. Once a reliable correspondence between the states of ISMs and those observed in the referent is established, the interaction with ISMs can serve as a representation of the understanding of real-world domain. Thus, through the correspondence checks, the modellers will gain insights into both the existing reality (the indicative properties) and the new requirements or new goals (the optative properties) which in turn form the new inputs for the next phase of system development. With reference to the SPORE framework depicted in Figure 6.1, we find that the 'inputs' to the ISMs depicted in Figure 5.3 (i.e. the arrow of new input (the elicitation activity) and the arrow of test and experiments (the validation process)), correspond to the inputs of the SPORE framework (the key problems, relevant contexts and available resources). Similarly, the 'outputs' of ISMs in Figure 5.3 correspond to the outputs that provide the provisional solutions in the SPORE framework which form the inputs in the cooperative validation of the conceptual models. We regard the development procedure of EM as a continuously evolving process, so that the model is rarely completed. For example, during the elicitation activity the modeller may want to experiment on parts of the model to check and improve its accuracy. Also, during the validation activity, the modeller may need to revisit the situations considered in the elicitation process so as to have a better understanding of the environment. The open-ended characteristic of ISMs provides a way of integrating the elicitation and validation. Prior to defining and establishing the reliable patterns of state change, the interactions with ISMs, similar to activities in our everyday life, have an experimental character and are the primary means to improve our understanding of the domain. Thus the correspondence between the particular patterns of agency, dependency and

observation embodied in ISMs and our expectations of state changes in the referent can be established in a natural way.

The development procedure of our proposed approach to building a new system originates from a 'seed-ISM' based on considering the current system. The reasons (and advantages) are that the new system, whether we propose to automate or replace the existing processes, will to some degree provide the functionality of the old system. Also, the modeller already has knowledge of problems in the existing system and thus can avoid making the same mistakes again. This strategy for initiating BPR activities does not rule out the possibility that the ISM will be enriched at some stage by knowledge from sources independent of the current system. This will certainly be necessary if the BPR is to involve a radical change.

There are various potential advantages of using the EM approach for requirements elicitation and validation. For example, the artefact provides the concrete explanation for the conceptual model of the modellers. That is, during the procedure the modeller will make a decision about what action to perform and see its impact. Also it allows the modeller to check how accurate and consistent are the correspondences between the computer model and its referent, and between the computer model and his construal. Furthermore, he can check and discuss his construal with other participants and at the same time animate the model. Whilst the development of the conceptual model can draw on 'direct access' to real-world examples, it can additionally allow exploration of different variants which may not be predictable or preconceived. Correspondence checks can be used to validate the accuracy and completeness of the model, and lead to a focused elicitation of missing knowledge which in turn form the new inputs to the next development 'phase'. In this manner the requirements of the proposed system can be driven by several elicitation and validation cycles for evolutionary improvement.

When talking about BPR, there are many disciplines which are considered to be relevant to the issue of BPR. Among these, there are four main fields which we think to be most relevant to BPR: human-computer interaction (HCI), process modelling, requirements engineering, and decision making. For HCI, the ISMs can be used for interface construction; for the development of processes from the

ISMs, an observation-oriented analysis and an associated simulation of behaviour can be provided during the construction of the model; for requirements engineering, the ISMs serve as a prototype which help in understanding the current problems and visualising the reality of the future system; for decision making the ISMs can be used to explore a set of alternative solutions for a problem and their consequences.

The Construction of ISMs as Scenario-Based Design

The construction of ISMs in EM is similar to the paradigm of scenario-based design for human-computer interaction (HCI). The 'scenarios', as described by Carroll (1995), are not just the traditional activity logs of human factors, but refer broadly to the artefacts which are meaningful and discussible by users that "are couched at the level at which people understand and experience their own behaviour". The researchers of scenario-based design[7] argue that computer systems should be viewed as agents that transform user tasks and their supporting social practices. As Caroll asserts:

> When we design systems and applications, we are, most essentially, designing scenarios of interaction. (p. v)

Thus scenarios (especially the user-interaction scenarios) are the appropriate medium for representing and analysing how the computer system might impact on the users' activities and experiences. Galliers (1995) has described the key features of the scenario-based approach and proposed a way to build up scenarios[8]. The scenarios can be represented in various forms, such as textual narrative (as for example, in use cases), video mockups or computer simulations. On this basis, we can regard the construction of ISMs as (amongst other things) a kind of scenario-based design, since ISMs can serve as a working design representation of the users' experience with, and reaction to, the system functionality.

7. These include: (1) the ESPRIT research project CREWS (Cooperative Requirements Engineering With Scenarios). Further details can be found at http://sunsite.informatik.rwth-aachen.de/CREWS/ (20 December 2001). (2) The various projects published in *IEEE Transactions on Software Engineering – Special Issue on Scenarios Management*, Vol. 24, No. 12, December 1998. (3) Professor Bob Galliers who was in Warwick Business School and is now in the Department of Information Systems in LSE (cf. Galliers (1991; 1992; 1993; 1995)).
8. He suggests that scenarios can be built up by reviewing (1) those key elements that are expected to remain *constant* during the planning period; (2) those trends that appear likely to continue; and (3) those *issues* over which there is some debate.

Within EM, what the users can do with the existing or new systems, as well as the consequences of their structure, can be analysed prior to the detailed designs of the system functions and features which enable that use. Interacting with the computer models enables the designers to reflect the concrete circumstances and experiences of users throughout the design process.

The ISMs can support the needs of scenario-based design by making it possible to suspend commitment and by supporting concrete progress (Carroll, 1995). The analysis of tasks can be done vividly by the modeller with reference to direct experience of the ISMs, and the form and roles of the system components can be justified by showing what they are used for in a experimental way. The ISMs can also be used to demonstrate design alternatives and expose the analysis behind a particular choice of design. The ISMs are regarded as incomplete and the analysis of the modellers exploits the principles of 'what-if' reasoning. Furthermore, ISMs are an exceptionally promising medium for participative design. They allow the designers and users to communicate in a common medium. As mentioned in the previous section, users may have difficulty in describing their needs in conventional modelling approaches, due to the language barrier associated with the descriptive notations used in functional specifications.

Figure 6.7 illustrates how the development procedure of EM depicted in Figure 5.3 is influenced by the observation of scenarios both in the model and its referent and can be regarded as a scenario-based design. The requirements side in Figure 6.7, represents the observation and experiments of situations (scenarios) of both the model and the referent that can help to identify issues, or criteria, which are then validated against further observations. This corresponds to the arrow of 'new inputs' in Figure 5.3.

The specification of requirements generates the needs, design problems, resources available, etc. for the design stage. Once a prototype with its associated user interface (a 'provisional solution' in Figure 5.3) has been built, the work of evaluation can proceed to validate the design and requirements. The evaluation is done by observation and experiment with the computer model whose focus is on the internal (structural) view of the system. This corresponds to the 'test and experiments' arrow in Figure

5.3. In participative process modelling, as the participants' understanding of the situation increases, their perception of previous activity directed at requirements, design or evaluation may change, and necessitate a return or review of past decisions. Thus the process in Figure 5.3 is sometimes never-ending.

Process Modelling with ISMs

Since the principal concern of BPR is to facilitate radical change, it targets business processes rather than organisational functions alone. Thus the concepts of process and process modelling are the main issues in the area of BPR. However process modelling by itself is not sufficient to define the whole domain of business. Most research on process modelling has concentrated on how to represent processes and the relationships between them by means of structural architectures such as flow diagrams or activity diagrams. But no particular notation for process description is suitable for the needs of all applications.

We have emphasised the process view of systems thinking which regards the organisation as a process (cf. section 3.3), and have also described the contribution that this systems approach to the understanding of how such systems can be developed earlier in this section. In section 2.3 we introduced the concept of E-type software/systems (i.e. system embedded in the real world). Lehman's research shows that even if the requirements of a software system are developed and preconceived by

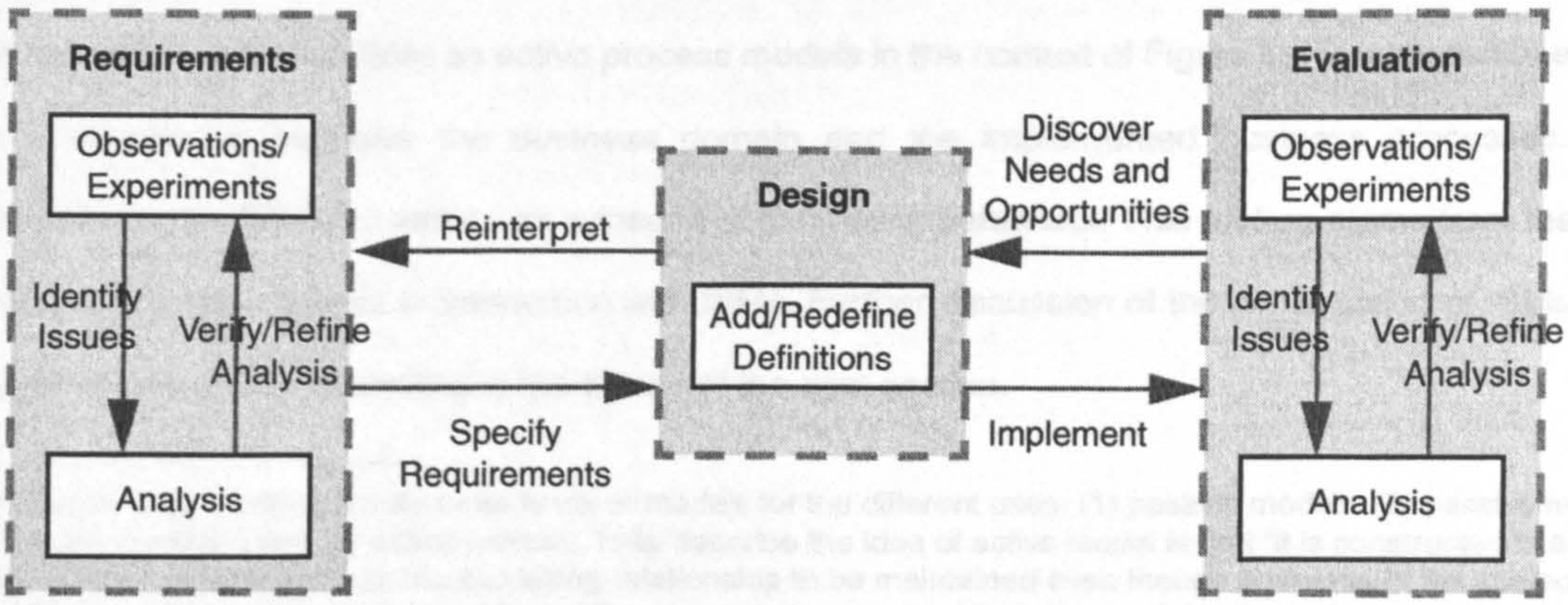


Figure 6.7 The EM Development Procedure as a Scenario-Based Design

carefully exploring the views and needs of the users, the installation of the system will still change the world in which the system and the users operate. As such changes are unpredictable, the users will find their perceptions of the world also change in the new context. Thus the system comes to embody a set of requirements which the user did not identify. This is why the operation domain of the E-type system is unbounded, and its acceptability is determined by its 'consequences' (cf. Table 2.3).

The philosophy of the concepts of participative process modelling and participative BPR that we introduced in section 6.2 is similar to the process-view of SSM and the E-type paradigm. As far as proof-of-concept is concerned, the processes represented in ISMs have been thoroughly validated through experiments and long-established practice. What we are representing in our models are the situated activities rather than abstraction of the process involving 'coordinatising' all possible situations with reference to selected aspects of state. Where such abstraction is used, the model can only represent the modeller's understanding at a particular time plus an abstract representation of part of a possible future (cf. Figure 6.6 'The description of the domain'). Warboys et al. (1999) suggest that four things are necessary for a model to exist: (1) the part of reality that is the subject of the model; (2) the model itself; (3) the relationship between them; (4) an observer, user, or creator of the model. The EM modelling process described in section 4.3 and the illustration in Figure 4.2 have given the general idea of how our modelling emphasises all these four things and integrates them within the modelling process in a natural way. The key idea is that ISMs serve as an *active model* in the sense defined by Warboys et al., that is, as a model that it is more closely linked to its real-world referent than the conventional passive model[9]. The use of ISMs as active process models in the context of Figure 6.4 is intended to establish a connection between the business domain and the implemented business processes. The development of ISMs also serves as a means of simulating behaviour. This feature stems from the participation of human agents in interaction with ISMs. Further discussion of the construction of ISMs as a process of simulation modelling is the theme of the next section.

---

9. Warboys et al. (1999) classify three kinds of models for the different uses: (1) passive models; (2) passive models that are dynamic; and (3) active models. They describe the idea of active model is that "it is constructed in a modelling medium which allows the modelling relationship to be maintained even though elements of the subject may change. ... The active model and its subject are synchronised so that the model reflects the current state of its subject". (p. 43)

The ISM as a Rapid Prototype

In system development, it is difficult to capture the right, or appropriate, requirements due to the different viewpoints of the many interested individuals and the long period of the development life cycle. For example, in the use case model, the development process may generate large sets or versions of use cases. Each use case is associated with a collection of possible scenarios involving the actors and the system, and maintaining this large set of complex scenarios poses a problem. What is more, such text-based descriptions rely on the designer's imagination to fill in the details of the system design and thus only allow the design to be evaluated at an abstract-level. As Crowe et al. (1996) point out, it is because of this that software engineers have turned to prototyping to enable users to experience and experiment with incomplete implementations of a software system before the final system is developed.

In addressing the issue of viewpoint on requirements in EM, the ISMs are the tools for understanding and communication. In the application of EM to BPR, the role of ISMs is similar to that of rapid prototypes in engineering design: they are constructed to inform the design of a complex system and can reduce the uncertainty about organisational or technical requirements[10]. For example, the visual display of ISMs can be the 'mock-ups' that exist on papers or computer screens. Such mock-ups can be used for heuristic evaluation or some testing. Like other business process models, the construction of ISMs does not need to be complete but still helps to give an overall picture to the modeller. The characteristics of an ISM are in some respects different from those of an engineering prototype, which restricts the users to performing actions that are prespecified. The ISMs resemble the artefacts that provide specific information associated with a representative experimental context relating to one or more aspects of a complex system design (cf. the experimental rig that might be used to investigate the effect of temperature on a material to be used in an engineering structure). By constructing ISMs and interacting with them, the modeller focuses on critical issues, and the differences between the states of the model and the states of the referent, without requiring a complete description of the current system or the new sys-

10.Generally the prototype in engineering design works by reducing the complexity of the real systems, either by eliminating functionality or by eliminating the coverage of some parts of the interface (Johnson et al., 1995). In an ISM, the effect of eliminating functionality is achieved (somewhat paradoxically) by enhancing the model through the addition of a restricted interface.

tem. Through this interaction, some aspects of the proposed system which are considered important by designers or users can be highlighted from a subjective viewpoint. This can speed up the interaction between designers and customers in requirements elicitation and advance the system implementation and validation. This endorses Winograd's (1995) view of prototypes as vehicles for communication "both in the designer's interactions with the materials and the user's interaction with the designer". The significance of ISMs is that the interactions made by the modeller are intrinsic to the domain and may only be loosely related to the functionality of the whole system. We can say that ISMs provide an intermediate artefact between declarative and operational types of specification for system development. In applying EM to BPR, the ISMs which are developed in the early stages of understanding business processes serve mainly to capture the open-ended localised interactions with real-world situations, rather than the circumscribed behaviour framed by human agency or agency automated by support systems which are associated with traditional processes or objects. This is consistent with Winograd's view of the organisational context for software systems:

> structures and practices of business are not taken as a fixed environment to be analysed and adjusted to, but as a domain of potential change and new design. (p. 73)

The advantages of interacting with ISMs for supporting design decisions and cultivating requirements can be summarised as the following:

- They can help the evaluation of system requirements in the presence of incompleteness and inconsistency.
- The construction process as prototyping is in the sense of participative modelling in which the user can actively participate in the process of defining the requirements as well as in the evolution of the design. For this reason the system is developed based on the aspects of the requirements which are understood and agreed with the users.
- ISMs as rapid prototypes can help the modeller to estimate the potential effects or costs of the specific requirements, such as unintended side effects of proposed solutions.

- The framework of SPORE can help to define the roles and responsibilities of participants in the process. This can help the project leader in making design decisions (e.g. about the division of work), and help the modeller in validating functions from a user perspective.
- The open-ended and interactive experiments with ISMs can enable the requirements for a proposed adaptation to be evaluated at the early stages. Through various kinds of interactions and analysis, some dynamic aspects of the system requirements can also be explored.

ISMs for Decision Making Support

In the BPR, it is essential that the directions for reorganisation are defined by both the management and designers. To identify these directions, the management has to make decisions about changes which are purposeful but also novel (the tenet of BPR) under uncertain conditions. Such decisions may result in high impact outcomes for the organisation. Thus the models for BPR should also provide support for the management to make decisions. As has been explored in parallel research on EM, the construction of ISMs has some characteristics in common with the development of decision support systems (DSSs) (see e.g. Beynon et al. (2002), Beynon et al. (2000b), Rasmequan and Russ (2000), Rasmequan et al. (2000)). The links between ISMs and DSSs, as they relate to BPR, will be further explored in this section.

As we described in chapter 3, although BPR and DSS have much common with each other, there exists a significant difference between BPR and DSS in their scope of analysis. That is, BPR focuses on the whole organisation (embracing the business processes) whereas DSS focuses on individual decisions (within a business process). With reference to our characterisations of participative process modelling in section 6.2, we find that DSS focuses on the perspectives of the internal agents whereas BPR focuses on those of the external agents. Both perspectives can be taken into account when employing the distributed EM tools. Under the *normal* mode of interaction, modellers can interact with each other in the roles of the internal agents. Under the *interference* mode, they can act as external agents in a context where conflict resolution and the exploration of different situations is orchestrated by an external observer acting in the role of superagent at the server. The EM artefact is in a narrow sense represent-

ing the current situations but also can simulate the future situations of an organisation. In a broader sense, it embodies assumptions about processes and actions, the models and procedures for determining the elements of requirements, and the modeller's decisions, situations and interpretations.

Experimental interactions with ISMs offer the modeller some insights into future situations which help in defining and implementing stages of the system development and BPR. This can give the modeller or the management a clearer idea about the directions for exploration and development, and clarify the actions which need to be taken in order to get closer to desired future states. Open-ended interactions with ISMs provide a flexibility for the decision maker to choose plans. They also enable the modeller to alter the decision factors quickly and assess their impacts. Whether the change of state in an ISM is consistent with, or confounds, the modeller's expectation, it offers insight that is important for making decisions. Interactions that are consistent with expectations can provide the basis for a step-by-step approach to achieve a desired result. Those that confound expectations can identify the indicators which need to be monitored, and inform the modification of plans that may be needed to correct unexpected behaviour. Recognising the unexpected changes of state which require further assessment of the work is the most important ingredient in this decision making. Another by-product of constructing ISMs is protection against errors of judgement.

As the business environment and the technology is constantly changing, no-one can forecast the future and preconceive the most appropriate solutions for business problems. Thus continuous reviews and corrections are integrated into the EM modelling process in order to build models corresponding to the real-world situations. The models are reviewed and assessed through observations and interactions to determine whether the current plan is appropriate or needs to be modified. The main issue is that the analysis, revision and modification can be made in an efficient and responsive manner within EM. To sum up, the potential advantages of the EM modelling for system development are:

- In the perspective of analysis: to uncover some hidden requirements; to envision future system usage; to make requirements behaviour more concrete; to enrich the contextual information which

helps to uncover risks and other organisational (global) problems; and to help communication between participants.

- In the perspective of design: to illustrate the trade-offs between proposed solutions; to validate the design solutions; to understand and resolve the conflicting requirements defined in earlier stages.

6.4.3 Using SPORE for Participative BPR

As described earlier, there is usually some conflict of interest between different levels of hierarchy within an organisation, or even within one level. For example, within a level there are multiple viewpoints between departments or between different user groups. There may also be conflicts between the organisational goals (profit, policy, etc) and the purposes of users (flexibility, convenience, etc). Thus any framework for business modelling should make these different viewpoints and relationships explicit. In the previous section, we have emphasised the importance of gaining a shared understanding of problems in order for participants to negotiate and reach consensus about solutions. Most organisations and most methodologies of system development are also aiming for such understanding. In chapter 5, we have described the problems and disadvantages of conventional models based on natural language and diagrams, which themselves form a boundary restricting the communication between participants and inhibit shared understanding among these participants. Sometimes the models need to be read or understood by non-computer scientists. Especially for BPR, the purpose is to 'enlighten' management; so the models need to be accessible to management and to express the process in a helpful way. With the help of EM we propose computer-based models as a new means for making the shared understanding and knowledge visible and communicable. They support the communication amongst people in the organisation that provides the basis for the design process. Under the SPORE framework this leads to the development of techniques which allow information arising from users and the organisation to be used in modelling. The participants (as agents), their activities (the agencies) and other environmental factors form the inputs into SPORE in formulating the requirements. This results in the shared understanding of existing situations through the collection and communication of information about the participants' insights and tasks. The scenarios shown in ISMs metaphorically represent how these

tasks are to be carried out without any further commitment to the detailed implementation of the artefact. Through the development of the artefact, the framework also provides the representation of future situations which are expected to result from the further detailed design of the artefact. Thus we see that our modelling is expressed in terms of agents, agencies and observations, and makes use of some techniques such as role playing, observation and direct participation.

Our research work in EM and the SPORE framework proposes a computer-based modelling medium for the participative work of constructing the artefacts which can faithfully and flexibly represent referents in the real world. It enables the modeller to ensure that people can directly understand how the artefacts resulting from the design can be used. In their account of the merits of rapid prototyping, Johnson et al. (1995) mention their design philosophy in the development of computer systems:

- Good systems design and business modelling should be done through having increased involvement, communication and participation of users and designers in the modelling process.
- The increased involvement and communication are occurring through the greater understanding of users and the greater knowledge of usage of the artefact.

Our approach to participative BPR not only endorses these principles, but also provides a practical technique for the involvement and communication of users participating in the modelling process. Through the process of artefact construction, the systems can be developed in a open-ended and interactive evolution, from the original conception of that systems, with immediate feedback. The feedback may be at various levels of abstraction, for example, via direct experiential knowledge, or conventional mathematical representations.

The Shifting Focus of Participative BPR

The goal of our framework is to facilitate the direct involvement of users in the work of BPR and the design of systems which will influence their working lives. That is to say, people who are affected by decisions should have a voice or opinions in the decision making. The participative process modelling should take place during the entire lifecycle of the business modelling and system development. The

various activities in the development procedure, for example from analysis through design to assessment, involves a shift in the conceptual focus. For example, in the analysis activities, the focus is on the existing business environment and the user's working process. This includes the activities of understanding the structure of the business by means of understanding the system environment. This analysis is done through the description of the LSD account. Next, the focus of the design work is concerned with how specific tasks can be carried out under some specific conditions, for example with the technology or resources available. The flexible interactions of participants with ISMs can help the users to keep their focus on the real-world situations and be aware of the constraints from different insights of other participants or from the implementation environment. Finally, the focus of the validation activity is to determine whether a particular design artefact is meeting the user's needs. Normally this testing is done by the developers. But within the participative framework of EM the assessment can be re-situated in the user's working context and enable the users to be involved in this activity. The shift in focus identified above are not interpreted as associated with a sharply defined phases in the modelling activity. In particular, the validation is not a separate activity but instead is the central feature of our process that pervades the iterative participatory design, and is reflected in the interaction of the designers or users with the models at every stage. The computer-based models make it easy for participants to experiment with them and use the validation activity as an opportunity for redesign.

The movement of attention during the participative modelling process is broadly from the business process to the domain of the support systems to be developed. That is to say, the focus is shifting from being process-oriented to being application-oriented. Conventional approaches are not well-suited for the integration of the implementation of business modelling and the implementation of system development. In particular, traditional business models which are process-oriented cannot be easily translated to the domain for software development, because this involves a shift from the concrete to the abstract, whilst the focus of the software development often narrows to a concern for how specific tasks can be done under specific conditions. Another issue is that the abstract notations of system development cannot be well understood by the business stakeholders or other non-computer professionals (cf. Figure

6.6). In the light of the above discussion, the way in which the use of ISMs allows the focus in BPR to shift throughout the development process is one of the most significant potential benefits of EM.

The Characteristics of EM in BPR

From our previous discussion, we infer that EM has great potential for system development and BPR. Through the modelling process within the EM framework, we can have a better understanding of the behaviour of the existing business environment and system, and more clearly identify the tasks, processes, and problematic elements. We can also find, and experiment with, the alternatives more easily. The following summarises the characteristics of experimental interactions with ISMs, as these apply to business process modelling:

- As described in chapter 4, EM is an unified activity rather than the traditional development lifecycle. It can potentially address BPR from the perspectives of both the internal and external agents, and empower and allow participants to "consider the meaning of the task at hand rather than becoming embroiled in peculiarities of its implementation" (Crowe et al., 1996).
- Interacting with ISMs within the SPORE framework allows participants to experiment with any entity of the business system. This means that the behaviour of both computer systems and human components can be identified and thus incorporated in a natural way. Thus the various enablers of BPR, such as information systems or human resource management strategies, can be investigated in essentially the same experimental manner.
- Under the 'open development' paradigm, participants can set up problematic scenarios, realise them visually and correct them easily and inexpensively using a 'what-if' strategy. Furthermore, the computer-based character of ISMs renders the changes and updates to the existing system apparent and makes the model maintainable and reusable.
- EM helps the participants to communicate their ideas and assess the impact of proposed changes/ alternatives immediately. Participants can be guided towards a shared understanding and consensus for decision-making through the continuous evaluation, communication and checks for consistency in the distributed ISMs.

- The visibility and communicability of the interaction within the SPORE framework increases the participants' understanding of roles and relationships amongst others, as well as the effects of individual activities. This lets the participants gain feedback from the results of experiments which is used not only as a basis for comparing the alternative solutions but also as a means of evaluating their validity (through investigating the results of corresponding experiments with the real world referent).
- EM allows the participants to obtain a 'global' view of the effects of 'local' changes made by individual artefacts. This assists the identification of implicit dependencies between parts of the business system.

We have mentioned in the previous chapter that the modellers have no specific role when interacting with ISMs. A family of ISMs built in the SPORE framework can be regarded, according to the different patterns of interaction, as (1) a *requirement*: when we interact in the roles of particular users; or (2) a *system*: when we interact in the role of key components (or agents in EM) and thus explore the internal structure of the model; or (3) a *business process*: when we interact in the roles of workers, markets, suppliers, etc (Chen et al., 2000a). Such flexibility of interpretation according to the style of interaction allows the SPORE framework to be applied to BPR.

We can also identify higher-level functions that a family of ISMs can offer when considering the integration of the IT system with a business system: (1) If we want to integrate the IT strategy with the business strategy, we need a business process model to support. In this case the ISMs can serve as the representation of the business process. (2) If we want to develop software applications to support the business processes (either the existing ones or the new ones proposed from the BPR work), the ISMs in this case can serve as the media for the integration of the business system with the IT system. (3) If we wish to provide a cooperative environment within an organisation, the ISMs can be the means of representing the coordination among the relevant people.

In chapter 3, we proposed that the reengineering work should start by modelling the existing business processes. Under the SPORE framework, this activity can be carried out by simulating the proc-

esses by getting participants familiar with the existing process to interact with the computer model and communicate with each other. In the sections on participative BPR (sections 3.4 and 6.2), we have stressed the importance of the key problem with much conventional BPR: the difficulties – but necessity – of involving all relevant people in the reengineering process. We argue that EM is appropriate for participative BPR as it is a human-centred approach and has tools to support the distributed working environment with various patterns of communication and interaction. The aim of EM modelling is not to provide the final answers to problems, but rather to help the participants to make inferences about the different aspects of the system within their individual viewpoints. That is to say, during the construction of ISMs, and through interaction with them, the identified problems and their solutions can be structured, whilst the participants also gain some visual and quantitative information as feedback which is essential for decision-making. The benefits of applying EM to the cultivation of requirements in a participative and distributed manner, as described in section 6.3, can also apply to other BPR applications. Within EM some features of business process modelling, such as making dynamic behaviour explicit or being able to communicate and analyse such behaviour, can be addressed.

The result of experiments with ISMs (the what-if modelling) giving feedback to participants is critical to the success of BPR, as the proposed solutions or alternative business processes can be validated through comparison with the real world system. This validation of the model can indicate the errors of the proposed processes and thus new solutions can be developed based on the feedback. On the other hand, if the proposed processes are validated and accepted by the management, the BPR project can proceed by actual implementation in the organisation. The aim of our framework is to enable participants to gain shared understanding of the problems and to identify solutions which are acceptable to both the management and the end-users and that, at the same time, will not contradict the existing organisational structure nor be subject to the restrictions of the existing systems.

It is clear from the foregoing paragraphs that the openness and flexibility of EM can be exploited to allow participants to develop new versions of models in response to new requirements or changes in the system environment. In an industrial or commercial setting such new versions correspond to the 'frozen' stages, or systems, as described in subsection 5.3.1. For medium or large scale applications, it

is possible that parts of the model, such as frequently used agent actions, be translated into a conventional language for the sake of speed and efficiency. Then if the whole model needed revision after such a process of optimisation, the developer would revert to the 'pure' EM version while revision to a new version took place. The optimisation could then be re-applied when appropriate. In a large scale application open to many participants creating new versions there is potentially a major problem of version control and management. This could be addressed to some extent by using the version control tools available on a given platform (e.g. SCCS[11] on UNIX) but this is an area for further work.

6.5 Concluding Remarks

Many processes in science and engineering are precisely prescribed by theories and equations, but for business processes it is difficult to consider the process and its associated real-world factors (including human factors) in an abstract and unified way. That is, the correspondence between the states of the process and the states in the real world is hard to represent precisely. Furthermore, business processes cannot be fully prescribed and it is impossible to give a full account of the roles of human beings within them. In contrast to problem solving in science and engineering, we cannot (and never) understand the human activities in enough detail to be able to list the attributes for the proposed system/process and apply modelling approaches which only consider the results of experiments and techniques in a predetermined way. Our best line of attack is to develop the rich and flexible concepts and models to directly involve the potential users and incorporate their activities into the analysis and design of the new system or business. In this context, subject to the modeller having an adequate construal and sufficient understanding of the situation, EM has the advantage of allowing the states of the real world to be modelled in an open-ended fashion so that any new factors considered relevant can be taken into account. This kind of experimental interaction with the computer model seems to be the appropriate technique for the purpose of business process modelling. It can also be used for experimentation purposes and to help decision making during the modelling procedure. By the EM approach singular conditions can be

11.Source code control system.

explicitly modelled and human intervention, which is essential when modelling the scenarios in business, is possible throughout the modelling process.

The role of ISMs in business modelling and system development is as an 'active model' which is similar to the construction of a spreadsheet. The construction of ISMs is guided by the semantic relation between the ISM and its external referent. Referring to Figure 6.4, the understanding of the business domain and the elaboration of the business solutions proceed in parallel with the development of an ISM. It is an essential and distinctive feature of EM that the shaping of the semantic relation between the model and its referent does not shift the focus from reality to abstraction, but rather is established through experimental interactions as the computer model is interpreted as situated in the world. For BPR, the ISM can encompass both the necessary knowledge of business behaviours and provide the prototype of human and automatic agencies in the process.

As the configuration of information systems and human beings in organisations are increasingly becoming both cooperative and distributed, it becomes more important to analyse and model the complex interlinked relationships within business systems. The development of distributed ISMs in the SPORE framework (Sun, 1999) offers an approach that is suited to this purpose.

Through open-ended experimental interaction with the computer model, the adoption of the EM approach to process modelling and reengineering can connect both the business process modelling and system development. Although it may be argued that EM does not handle all parts of the BPR exercise, the techniques we describe in this chapter are aimed at integrating the implementation of the supporting system with the BPR implementation and participative modelling during BPR. Furthermore, there is a potential for integrating EM with other tools used to carry out the modelling and analysis process of BPR. For example, as described in chapter 5, EM can be used as an alternative approach for creating use case specifications that may help to address the problem of managing many variants of use cases expressed in a text-based form.

Where the conception and application of BPR is concerned, we do not regard EM as a methodology or so-called 'scientific method' because its primary concern is for exploratory activity rather than

systematic problem-solving procedures. That is, EM is not aimed at developing the final answers/products or detailed specific guidance for designers. Like SSM, EM is concerned with the exploration of different people's systems of meaning but not with describing the 'objective' reality (Mingers, 1995). We are developing a rich and flexible modelling approach and concept to directly incorporate the descriptions of the modellers, as well as potential users, by involving them in the design process. This gives priority to generating a rich understanding of the relevant situation before exploring potential improvement. As Mingers suggests, it also involves considering notional activities which may, or may not, already exist, and may lead to BPR before the information system design is considered. EM principles are based on the presumption that there are always experimental factors in the real-world system of which the modeller is unaware or that they cannot predict. Such factors may play a critical role in the success of system development or business modelling and this can explain why conventional methodologies sometimes fail to achieve their preconceived goals. This is especially true for BPR as we need to take account of other human factors such as knowledge, perceptions and skills. With EM, the construction of computer models can lead to a way to disclose hidden conditions and investigate the matters of agency and observation such as what knowledge or perceptions are associated with a specific role.

CHAPTER SEVEN

Case Studies

This chapter provides the practical case studies to help to understand – through knowing-by-doing – the principles of EM and the issues discussed in the last two chapters, such as the EM approach for system development, participative process modelling and participative BPR. The purpose of these case studies, developed during the research, is to illustrate the ways in which EM can be applied to software development and to participative BPR which have already been described theoretically in chapter 5 and chapter 6.

7.1 Introduction

This chapter aims to exemplify the features of the EM approach discussed in previous chapters through practical research. There are two case studies included in this chapter.

The first case study (in section 7.2) illustrates an ISM for a digital watch model (or *Timepiece* artefacts). In this model, three artefacts represent three aspects of timepieces: a digital watch, an analogue clock and a statechart. This modelling case study is to demonstrate the EM process for system development which was described in chapter 5 and chapter 6. This digital watch model was mainly developed by Dr Beynon and Dr Cartwright in 1992 and 1994 respectively, and is still being improved and modified today in the EM group. It is helpful to refer to Figure 4.2 and Figure 5.3 to have a general view of how the EM concepts and modelling process are applied in this case study.

Another case study (in section 7.3) involves modelling a business process of a warehouse (the manual distribution within a warehouse or between different warehouses). This model was developed by this author during his research. The purpose of this modelling is to show how to apply the SPORE framework for cultivating requirements of the warehouse management system and how the participative process modelling can be achieved through the distributed environment provided by **dtkeden**. For this, several computer models are constructed as artefacts in a distributed modelling environment in order to shape the action of agents associated with the manual distribution process within the warehouse. These agents are interacting with the models at the client nodes and the window at the server model shows the current state of the manual redistribution process in the warehouse environment (cf. Figure 7.6). Modelling within the server node provides a potential opportunity for modelling the system or business process from the perspectives of an external observer. Individual participants in the modelling process can 'communicate' with each other and have the 'visualised' insights of other modellers through different modes of interactions provided by the distributed EM tools. Further details of the features of distributed EM (DEM) and these modes of interactions can be found in the research work and thesis by Sun (1999).

7.2 The Digital Watch

In this case study, we will use the artefacts of the digital watch to illustrate the characteristics of the EM approach in system development. There are three artefacts in Figure 7.1: the model of a digital watch and an analogue clock, and a statechart which represents the functionality of the digital watch display mechanism. The clock face is the artefact for communicating time and the statechart is the artefact to tell the user or designer how the display functions of the digital watch are affected by the pressing of watch buttons as well as the level of the battery energy. There are four buttons surrounding the digital watch at each corner which respond to the mouse clicks. These buttons are linked by definitions to the statechart so the current internal state of the watch can be shown by the boxes with solid lines (rather than dotted lines). In the analogue clock face, the hands of the clock are represented by a graphical line drawing. The positions of these hands are linked to the time on the digital watch. The movement of the

hands metaphorically represents the mechanical coupling of the hands of a real analogue clock, e.g. when the second hand moves, the minute hand and, in turn, the hour hand move with it.

7.2.1 The ISM for a Digital Watch

In constructing the ISM for a digital watch, the modeller was working in a situated manner and with reference mainly to Harel's statechart but also having a real watch (the referent) in mind. The relevant observables for the ISM include the characteristics of the actual watch which can be reliably identified, through an interaction with the watch during a period of time. Examples of observables may include the appearance of the watch, the digits appearing on the LCD display, the layout of buttons, the presence and absence of the power supply, and the current mode of display. Of course there is no constraint on

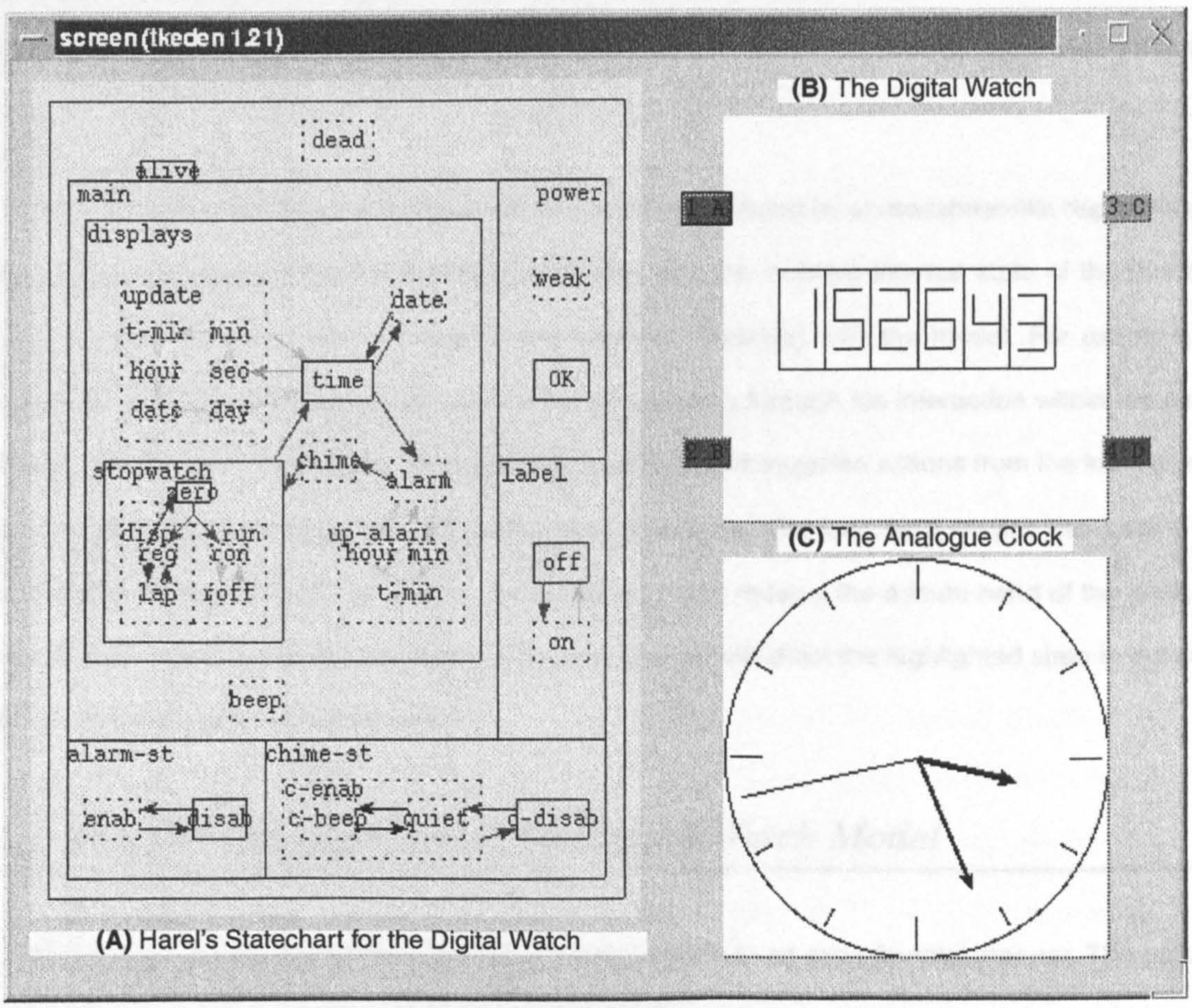


Figure 7.1 The Digital Watch

the choice of observables to suit a particular goal, and new observables can be added to the model when necessary. Adding new observables to the model can, as Beynon et al. (1998) suggest, "be viewed as recruiting more observers in the spirit of 'subject-oriented' programming". The only constraint may be how far the conception of the watch in the minds of new observers can be consistent with that of the original modeller.

The ISM for a digital watch metaphorically represents the observables of the actual watch (its referent). This model only depicts the rectangular shape of the watch, with LCD display and four buttons at each corner. There is also an analogue clock synchronised to the digital watch. The statechart expresses the significance of the current display by highlighting a particular region. The statechart was originally from Harel (1992) for the software development of reactive systems. The statechart in this case study is serving as a cognitive artefact (a term introduced in Norman (1991)) because some aspects of the state of the actual watch shown in the statechart cannot be directly observed by the modeller.

Different types of agency in the Digital Watch model are shaped by spreadsheet-like dependencies. Some of these dependencies link the visual elements with the invisible internal state of the model so they reflect the interaction of the modeller (the external observer) with the model. For example, the modeller can simulate the user's behaviour of button pressing through his interaction within the simple interface. He can extend or modify the definitions, functions and triggered actions from the input window at any time during the development. Other dependencies in this ISM include: changing the current time will affect the displays on LCD as well as the analogue clock; moving the minute hand of the analogue clock will move the hour hand; changing the mode of display will affect the highlighted state in the statechart and the display in the digital watch.

7.2.2 The EM Development of the Digital Watch Model

The association between the artefact and its referent is made in an experimental manner. The patterns of state changes in observables are identified through the modeller's observation and experiment, and

these identified patterns are subject to revision after subsequent observation. Referring to Figure 5.3 (the unified development procedure in EM), the process of developing an artefact involves incremental construction of a model which is guided by the evolving knowledge of the modeller (through his experience and experiment during the process) about the properties and situation of the referent. In this process, it is essential that the modeller has an insight of the current status of this model, and can revise this to consider new information and perspectives on the referent.

The simulation of this digital watch was firstly developed by Dr Beynon in November 1992. During that development the model was incrementally constructed being given intermittent attention while running as a background process on a workstation (Beynon and Cartwright, 1995). Based on this original model (the seed-ISM), the subsequent development process typically involved the addition of groups of definitions to add new visual components to the model (or redefining the existing definitions to modify them) as well as the addition of procedures (I.e. actions) in EDEN to simulate the newly introduced agents. When the seed-ISM has been constructed, the tasks of adding new components or behaviours are usually done by developing portions of the definitive script or tracing problems by extracting pieces of the script and testing them in isolation. For example, the statechart in Figure 7.1 (a) was firstly developed by describing the disposition and interaction of boxes abstractly without giving coordinate information. This coordinate information was then added by imposing the grid from Harel's statechart. The process of tracing abnormal or unpredicted behaviour of the model (I.e. the test and experiments arrow in Figure 5.3) was assisted by the close correspondence between the variables in the model and observables of the watch (the referent). The modeller's experiment with both the model and its referent can help to determine whether the simulation meets its intended behaviour.

The above work can be regarded as phase 1 of the development procedure of a digital watch. The further details of the statechart, and the button interface constituted phase 2 and were added by Richard Cartwright in December 1994. The model after phase 1 was 'frozen' and became a 'system' because it had satisfied what we needed at the time[1]. But at later stages the functionalities of the system may not be appropriate or not enough for the needs of users due to the change of the environment

or technologies. For example, in the model of an analogue clock in phase 1, the motions of both the second hand and the minute hand are defined independently via the `gettime()` function (but are synchronised by updating the position of the minute hand at regular intervals):

```
%eden
tn = gettime();          /* Get the current real-world time from terminal */
tnsecs is abs_time(tn);  /* The variable tnsecs records the number of seconds elapsed in real-time
                            from an arbitrary initial time */
t is tnsecs / 60;        /* The variable t representing time elapsed in minutes */
%donald
within clock {
    secAngle = (pi div 2.0) - float (tnsecs % 60)* (pi div 30.0)
                         /* "tnsecs % 60" determine how many seconds elapsed since last whole minute
                            time */
    minAngle = (pi div 2.0) - float (t % 60) * (pi div 30.0)
    hourAngle = (pi div 2.0) - float (t % 720) * (pi div 360.0)
    secHand = [center + {size_secHand * radius @ secAngle}, center]
    minHand = [center + {size_minHand * radius @ minAngle}, center]
    hourHand = [center + {size_hourHand * radius @ hourAngle}, center]
}
```

This might not cause a big inconvenience for the use in our everyday life if this analogue clock is simply used for interpreting the time. But we may need to improve this situation for a more accurate use in other environments. In 1994 Cartwright enhanced the model to include a chess clock which requires greater accuracy in the movement of the minute hand. This was achieved by binding the motions of the second and minute hands together indivisibly thus:

```
%eden
tn = gettime();
tnsecs is abs_time(tn);
%donald
within clock {
    secAngle = (pi div 2.0) - float (tnsecs % 60) * (pi div 30.0)
    minAngle = (secAngle / 2 * pi) * (pi / 6)
    hourAngle = (minAngle / 2 * pi) * (pi / 6)
    secHand = [center + {size_secHand * radius @ secAngle}, center]
```

1. For example, this watch artefact was 'frozen' at this stage as it was sufficient for the demonstration of EM principles to MSc students, i.e. how the characteristics of the statechart can relate to the presence of LSD agents and the roles they play.

```
    minHand = [center + {size_minHand * radius @ minAngle}, center]
    hourHand = [center + {size_hourHand * radius @ hourAngle}, center]
}
```

This modification makes the clock reflect more closely the real-world referent in which the minute and hour hands are mechanically coupled with the second hand. The principle of a traditional spreadsheet was also added to the model at this stage to relate the power consumption of the LCD display to the pattern of use.

Following Beynon and Cartwright, two other EM modellers have contributed to building the watch artefact. The functionality of the watch was completed by Carlos Fischer in 1999, which we regard as the phase 3 of the development. He added new definitions and procedural actions to the model, and buttons for updating time and date and for switching the clock on through the interface. This resembles the design and test/experiments activities (i.e. the internal view of the model) in Figure 5.3. But during the same time the interaction with the evolving watch artefact (i.e. the external view) can also be achieved which, as described by the modeller (cf. Fischer and Beynon, 2001), played a far more significant role in the modelling activity than communication with the contributors. He also commented that the model itself is a more useful repository of knowledge about its construction than the memories of previous modellers.

The development of the watch artefact has been continued further by Chris Roe in this group which can be viewed as phase 4. Roe has modified the analogue and digital visualisation of the watch with buttons which correspond to the ones on his real watch (a sports watch). In this phase, a new visualisation of a mental model, based on the statechart in Figure 7.1 (a), has also been developed which the user may construct when learning to use the watch artefact (cf. Figure 7.2). It shows the state transitions which are familiar to the user and, as the user gains experience of interacting with the watch, the number of familiar states increases and they are added to the visualisation during the interaction process[2]. For example, through Figure 7.2 we can deduce that the user has explored changing the time of

2. The further details of the new digital watch with visualisation can be found in (Roe et al., 2001).

the main clock and altering the alarm, but not yet explored the other features. The reason for developing the new visualisation instead of the statechart is that though the statechart is a good mental model of a watch that users already understand – as it represents a complete view of a situation – it is not a good mental model of a watch when users understand only partially. The new visualisation in Figure 7.2 aims to represent the knowledge the user has of the watch at the present time. In this phase Roe also added the artefact of two runners who are completing a race and the aim is to use the watch to record their times when they complete the race. The visualisation of the runner artefact takes a very simple form: the progress of each runner is depicted by a dynamically extending line.

The development history of the digital watch illustrates the unified development procedure of EM (Figure 5.3) in which the EM model is always open to revision and extension. These modifications also in some ways indicate the potential for reuse in EM.

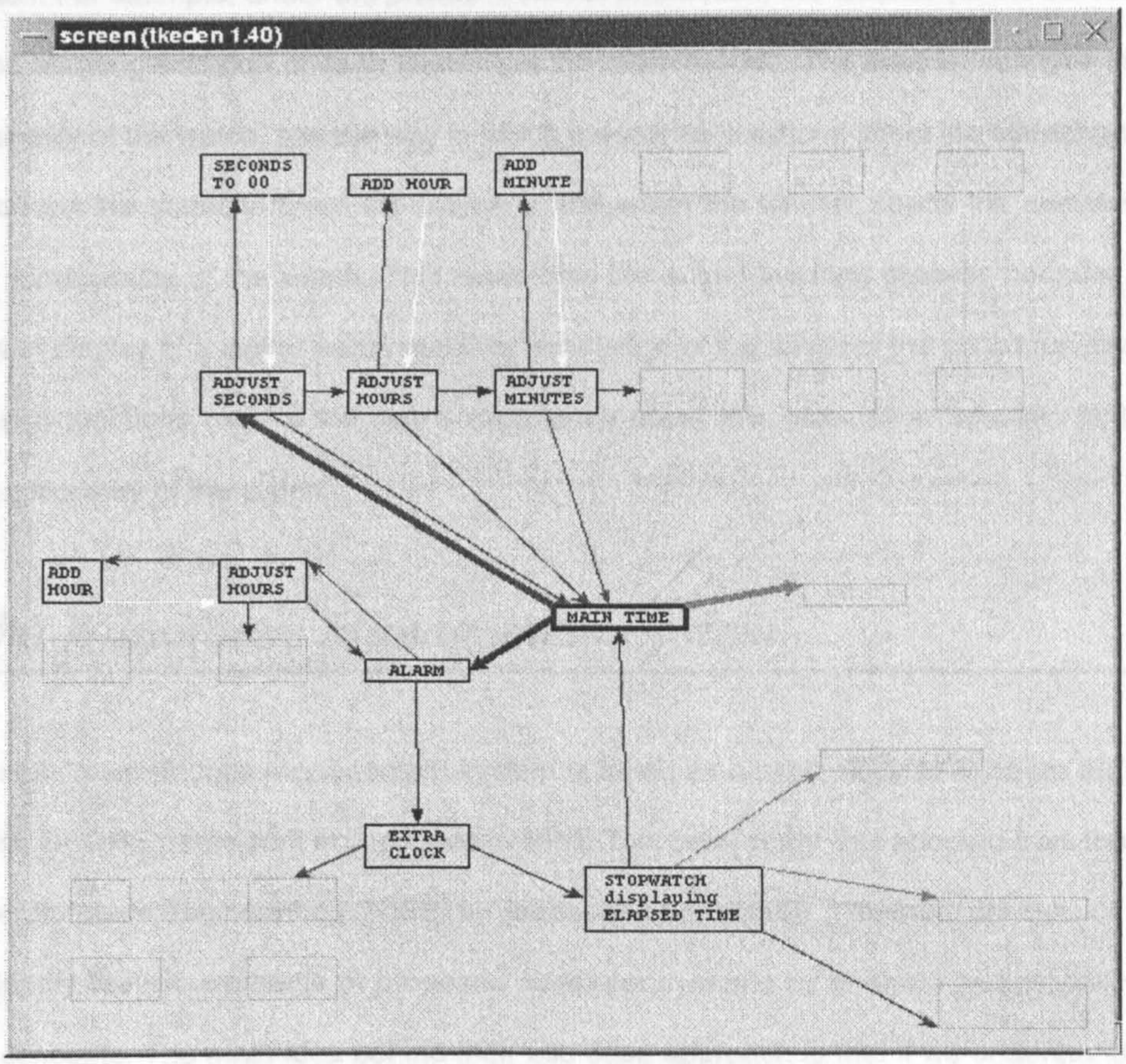


Figure 7.2 The Visualisation of Mental Model of the Digital Watch

Participative Process Modelling for the Digital Watch Model

The choice of observables for the digital watch is subjective and reflects the knowledge and perception of the human agents who interact with it. For example, interpreting aspects of the appearance of a watch can be subjective and will sometimes depend on different environments. The aim of EM is to integrate these different perspectives of the human agents as well as the automated agents within the modelling process. The distributed EM tools (**dtkeden**) can be used for this purpose. Several models can be run concurrently in the **dtkeden** configuration, and each of these models provides an artefact for the human participants to interact with. The synchronisation between the evolution of the models and the individual insights can visualise the participants' viewpoints and show them to each other. It also provides a channel of communication between these participants interacting with their own artefact. This distributed configuration of **dtkeden** can also serve as an environment for learning how to use a digital watch. For example, under the *private model* of interaction, the teacher (at the server model) can monitor the learning situation of each learner (at the client model). The teacher can give instruction on the functionality of the watch, see the way in which the learner's actions affect the statechart, and sometimes introduce the statechart into the learner's ISM when the learner needs the relevant knowledge about the functionality of the watch. This resembles the actual learning process because recognising the modes of display of a digital watch requires knowledge of the range of the watch functions. Familiarity with these functions require the user's experience about the association between button pressing and the functionality of the watch.

7.3 The Warehouse Management System

In this section, a warehouse management system is taken as a case study to illustrate the potential for applying the SPORE framework in participative BPR. This case study was adopted from the text *Object-Orientation Software Engineering (OOSE)* by Jacobson et al. (1992). The main concern of Jacobson et al. is to identify the requirements of proposed computer systems by analysis and modelling based on use case concepts. The main idea behind their use case approach is that if we understand the roles of

the users who will have access to the proposed system, then we can identify some of the essentials of the system usage and from which the requirements can be elicited. As described in chapter 5, each use case is associated with a particular kind of interaction between actors and the system and as such the interaction might be directed towards one of the required functions of the system, for example, the use case of manual redistribution between warehouses in the case study. In their other book *The Object Advantage* (Jacobson et al., 1995), they extend the use case approach to modelling business processes by introducing the concept of 'business use cases', and propose a method for object-oriented business engineering (OOBE). Here the use case model serves as a process model of the existing business (the external view of the organisation), which is used as the basis for prioritising the processes to be reengineered (cf. subsection 5.2.3).

As mentioned in previous chapters, we regard it as important to address BPR within a broader context of developing a business model. This means widening our focus to include the real world (i.e. the environment and human factors) rather than the computer system alone. When adopting this perspective, the role of EM is to develop a computer-based model which can be used to explore all the charac-

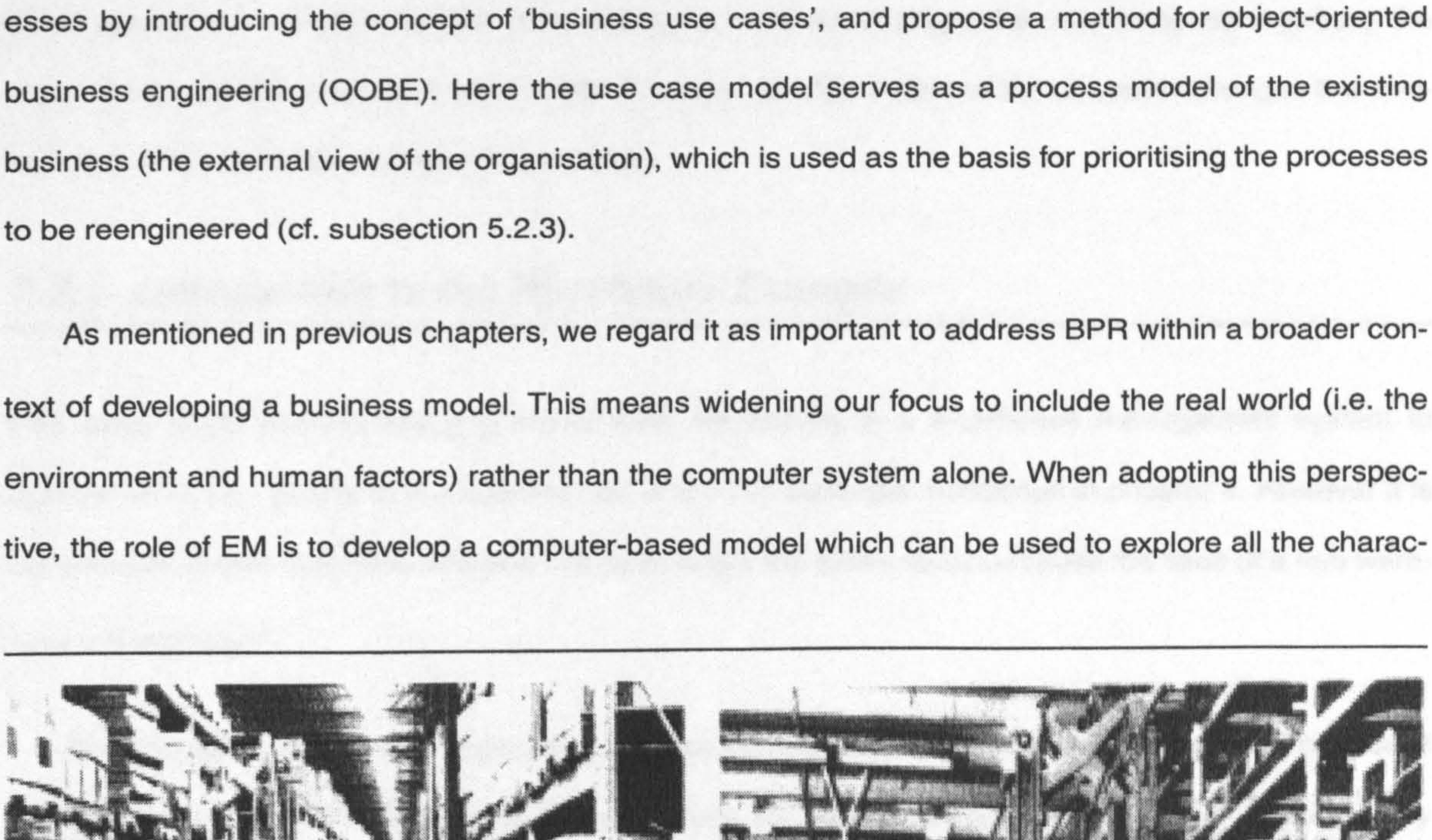


Figure 7.3 The Warehouse (from Magad and Amos, 1995)

teristic transactions of the warehouse. The character of our framework is through-and-through agent-oriented, so that the warehouse activity is conceived with reference to the state-changing protocols for both the human agents and automated components within the system. Because the human actions are constrained by the business process and follow some reliable patterns, it is possible to regard their cooperative activity as a form of computation (in the same way as we might say "the users are programmed"). The human users and the computer-based components can then be viewed as computational agents in a complex system. This agency is mediated through the user-computer interface: the input of the user influences the state of the computer, and the output of the computer changes the environment of the user (Beynon and Russ, 1994).

7.3.1 Introduction to the Warehouse Example

This case study involves applying the SPORE framework to a warehouse management system to achieve BPR. Our goal is to illustrate the use of the EM concepts mentioned in chapter 4. However it is not possible to give complete details and fully illustrate the entire study because the case of a real warehouse is too large[3].

The proposed system is to support warehouse management. The main function of a warehouse is to provide its customers with space to store goods for varying amounts of time, during the journey between the plants of production and the retail outlets; or to support companies that need local warehouses but may not have local offices (cf. Figure 7.3). The operations of the warehouse also include storing different kinds of items and using trucks (or any other forms of transportation such as railway) for the shipments and redistribution of items[4] (Figure 7.4). The aim of introducing computer systems into the warehouse is to give automatic support to the storage and redistribution. These processes involve keeping track of the locations and status of items, differentiating between kinds of items (for example

3. Jacobson et al. (1992) describe in their warehouse case study that the complete documentation alone would cover more pages than their text book (which is more than 500 pages).
4. Sometimes the term 'distribution centre' is synonymous with warehouse since most goods in a warehouse are in somebody's distribution system (Johnson and Wood, 1996). Thus in the distribution channels, warehouses are intermediate storage points between the manufacturer and the retailers.

they may be perishable or flammable, or some items must not come to contact with other items[5]), maintaining security and integrity checks, and managing storage, retrieval and relocation[6]. The people in the warehouse who will use this system may include: the *foreman* responsible for that warehouse; the *warehouse worker* who is responsible for loading and unloading; the *forklift operator* who drives a forklift in the warehouse (Figure 7.5); the *truck driver* who drives a truck between different warehouses; the *office personnel* who receive orders and requests from customers, arrange the truck routines, and keep records of all warehouses; and the *customers* who own the items in the warehouse and give instructions about their items, such as where and when they want the items.

In this case study, one process of a warehouse is presented. The process, 'manual redistribution within a warehouse and between different warehouses', is proposed based on a use case of the warehouse case study in Jacobson's OOSE book. Appendix B illustrates the possible details of the manual distribution process prior to the introduction of a computer system. That is, before automatising parts of

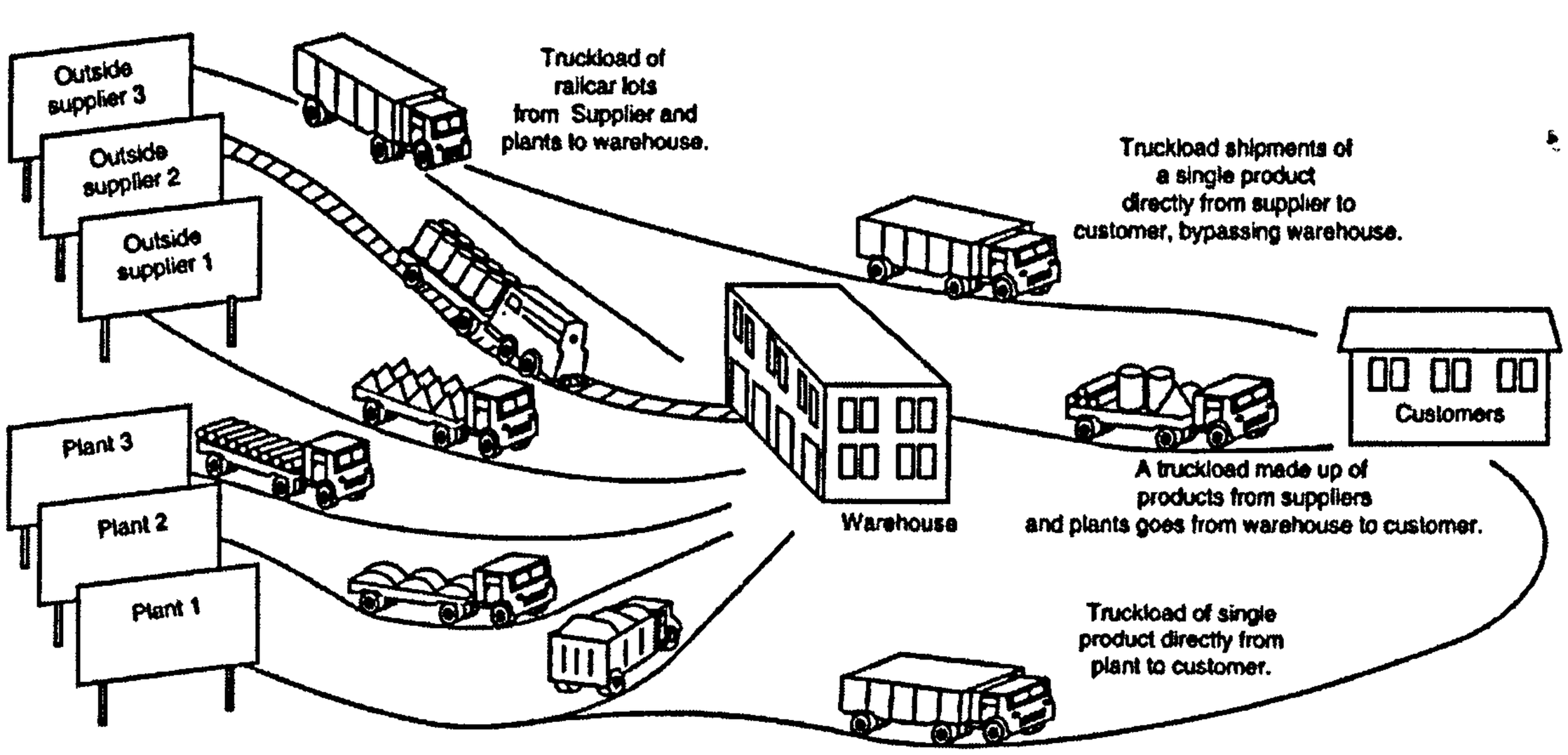


Figure 7.4 The Overview of a Warehouse (adapted from Johnson and Wood, 1996)

5. Such as industrial chemicals and foods.
6. The possibility that some of the functions of the warehouse associated with the physical storage and retrieval of items might also be automated using robots is not beyond the scope of our approach, but this is not directly addressed here.

the process by computer systems, the information needed for manual distribution tasks would be done through filling in and distributing different kinds of forms as well as the use of public inventory boards. Other methods of communication may be used such as telephones, but the delivery and completion of paper-based forms would still be the main method for the sake of authorisation and auditing. Another reason we chose form delivery to abstractly represent the process is that the forms, and the relevant information recorded on the forms, are obvious observables to identify the current situation of the warehouse. Figure 7.6 shows a snapshot of the process appearing on the window of the ISM.

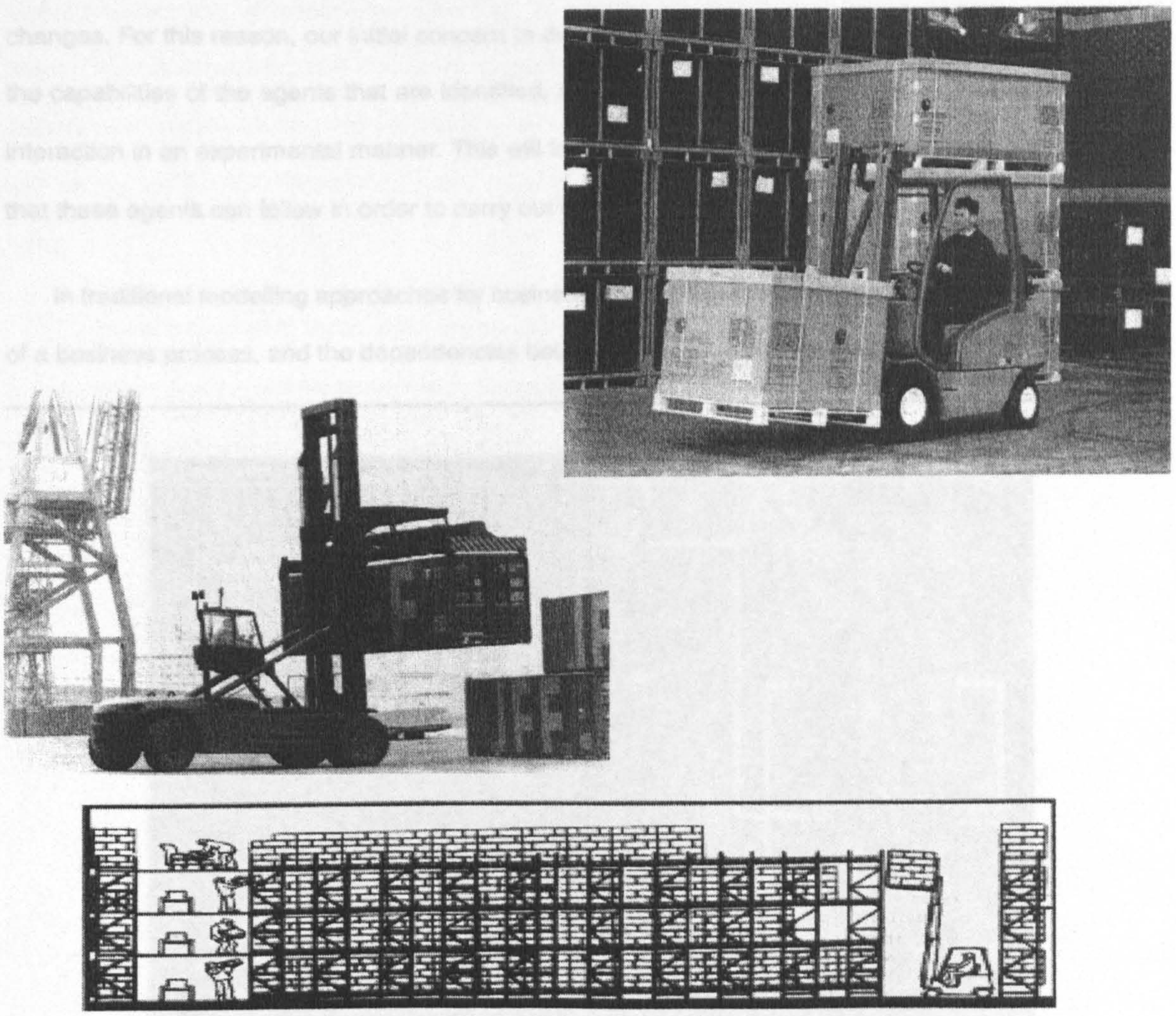


Figure 7.5 Forklift Truck (from Rushton et al., 1989; Johnson and Wood, 1996)

7.3.2 Business Process Model for Warehouse

In applying the use case approach, the first step is to create a simple picture of a *system* which describes the system boundaries and the actors (i.e. the users) of the system (cf. Figure 7.7). The significant difference between the EM approach and the use case approach is that the boundary of the system is not preconceived but grows with the increasing understanding of the modeller. In conventional system development, because the boundary is defined in advance, the modeller focuses on those interactions which respect the functionality that will be imposed in the proposed system. In the EM approach, the warehouse operation is conceived in terms of each agent's perception of states and state changes. For this reason, our initial concern in developing the business process model is with studying the capabilities of the agents that are identified, and examining the possibilities for their unconstrained interaction in an experimental manner. This will form the basis for subsequently exploring the protocols that these agents can follow in order to carry out the characteristic transactions of the warehouse.

In traditional modelling approaches for business, the issues of how agents relate to the current state of a business process, and the dependencies between entities, is not explicitly addressed. In object-ori-

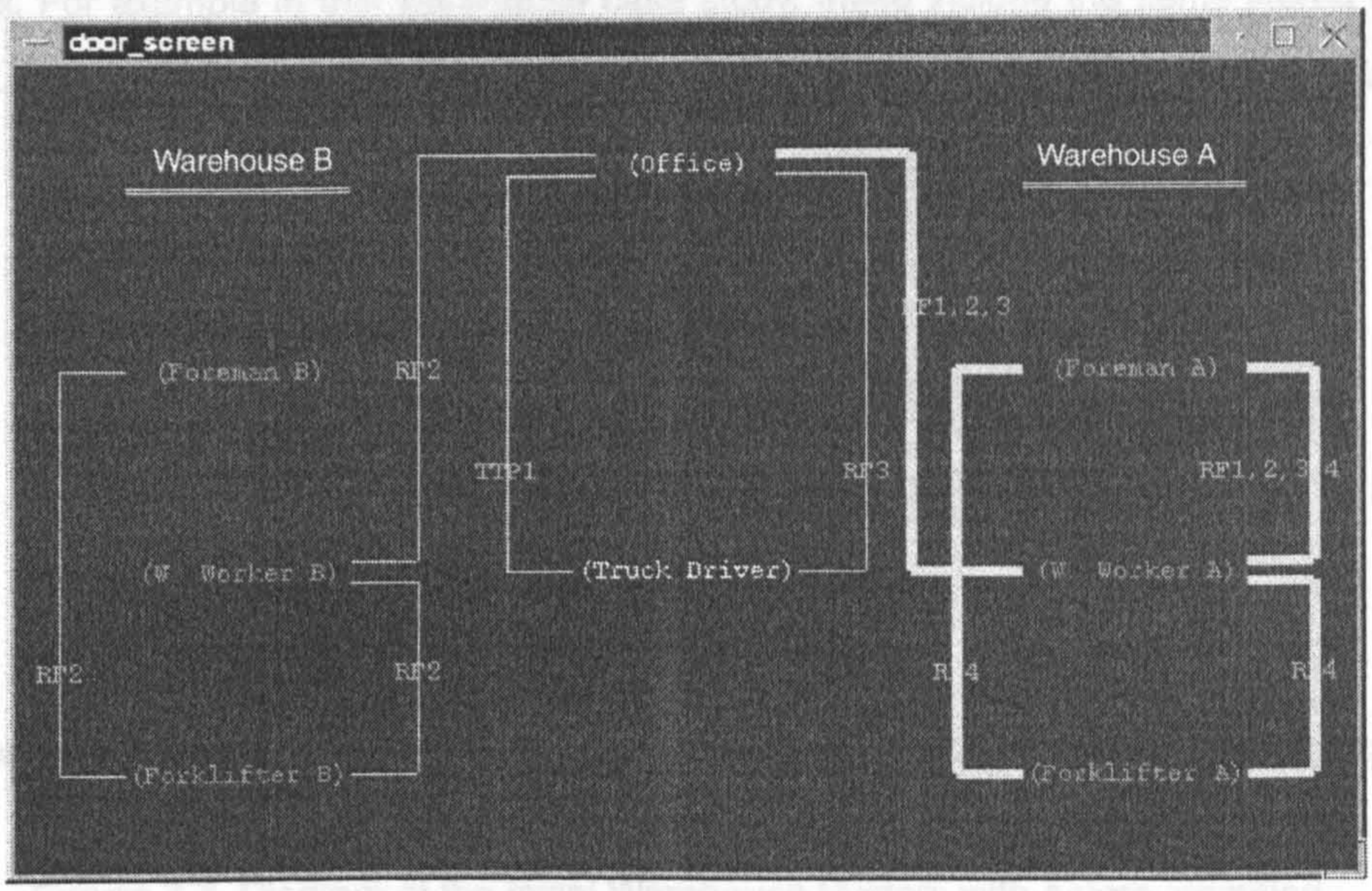


Figure 7.6 Snapshot of the Warehouse Process

ented approaches, for example, such relationships and dependencies are implicitly expressed through messages passed from one object to other objects which trigger the calling of methods with the objects to update their state. It may be easy to implement such kinds of dependency within the object-oriented paradigm, but it requires all the dependencies represented in the model to be preconceived prior to implementation and the message passing used to control the update requires specified ordering. Further, if the messages are not sent at the correct time to trigger the update, then the state of the model will become inconsistent.

The development of process models from our EM approach has two aspects: an observation-oriented analysis and an associated simulation of behaviour. The latter is based on the participation of human agents interacting with the computer model, and represents a shift in emphasis from exploratory activity and model extension to the study of the intended process operation (Chen et al., 2000b). The key issues are the identification of stable protocols for interaction and suitable stimulus-response patterns.

To model a business, there exists some key observables that are recognised to be strongly related to both the daily work of personnel within the business and the phases in the preconceived transactions in a process. For example in this warehouse case study, these include the items stored in the ware-

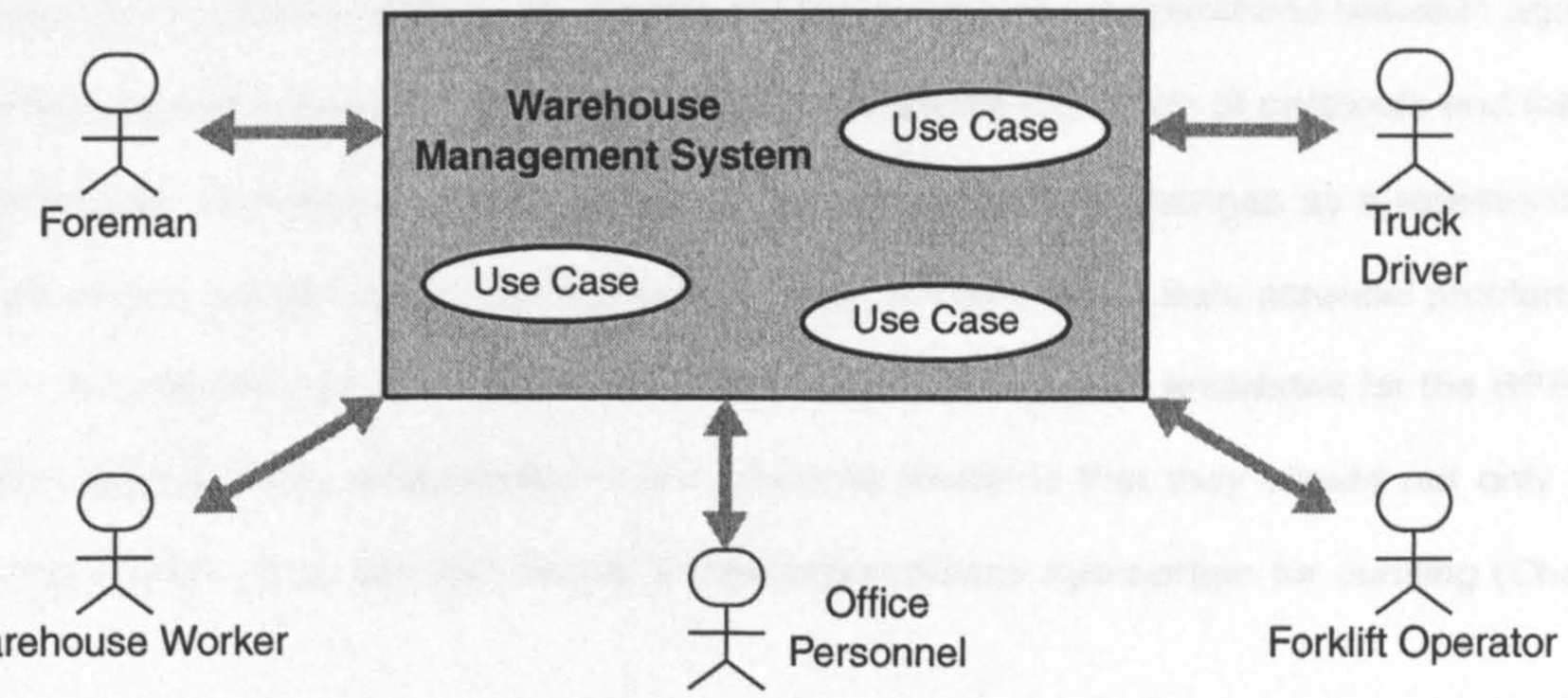


Figure 7.7 Diagram of the Initial Warehouse System with Actors Identified (adapted from Jacobson et al., 1992)

house and their locations. Our EM approach pays more attention to the true character of the real-world observation. As a result, the role of observations in the business process differs from a conventional approach in three important respects (Chen et al., 2000a):

- Firstly, the way in which states and events are observed by an agent is considered to be significantly important. It is not simply the fact that an item is at a location or that a truck has arrived at the warehouse that is regarded significant; it also matters how and by whom the presence of an item or the arrival of a truck is, or can, be observed.
- Secondly, the fact that agents are aware of the abstract stage that has been reached in the business process is taken into account in identifying their observables. For example, the office personnel will distinguish the abstract status of a redistribution process according to whether a group of items is still at the warehouse, in transit or has been successfully relocated.
- Thirdly, there has to be a means by which agents can interpret physical observation of the real-world state as disclosing the status of abstract business transactions. For example, there must be some concrete indication which points out that an item has now officially left the warehouse and that a new phase in the redistribution process has begun.

Our EM business model is built with reference to state change of an abstract nature that is associated with observation of a process. In the warehouse example, the relevant observables are related to the current status of the communicating information about the warehouse operations between agents. The corresponding state changes are concerned with the systematic execution of protocols and the associated transition from one phase to the next phase. By using the state changes as a representation for business processes, we can easily identify the existing processes and – from potential problems or dissatisfactions of customers or employees – can also find those that are candidates for the BPR activity. An important aspect of the observables in our business model is that they should not only serve to determine the current state, but also supply a transaction history appropriate for auditing (Chen et al., 2000a).

In order to understand the existing business processes properly, the ISM we develop to represent the business process model is modelled on the practices that would have been used in the operation of the warehouse prior to the introduction of computer systems. This is consistent with the emphasis of Jacobson et al. (1995) on the benefits of modelling existing practice[7]. In this context, forms and paper delivery serve as records of the operation of the model. This kind of manual data entry following systematic processes of form delivery can represent both the current status of all transactions such as which items were in transit, and the history of transactions. The aim of BPR is to automate these transactions by introducing computer systems, and to try to find alternative ones which will reduce the work-hours of personnel and achieve a more efficient process for business.

From our viewpoint, the forms can be interpreted as a *paper-based* ISM for the business process. In carrying out a particular transaction, specified procedures are followed in filling forms and transferring them between warehouse agents. For example, as depicted in Figure 7.8, when a manual redistribution between warehouse is initiated, four copies of redistribution forms (RFs) are transferred from the foreman to the warehouse worker. The manual activities of processing forms can effectively identify which agents have the roles in the transaction, which are currently active in any phase, and how their interaction is synchronised (cf. Figures 7.8 and 7.9). The current status of any transaction is determined by what sections of the forms are currently completed and who currently holds the forms.

7. They emphasise the significance of analysing and modelling the existing business in a chapter ("Reversing the existing business") in their book (Jacobson et al., 1995).

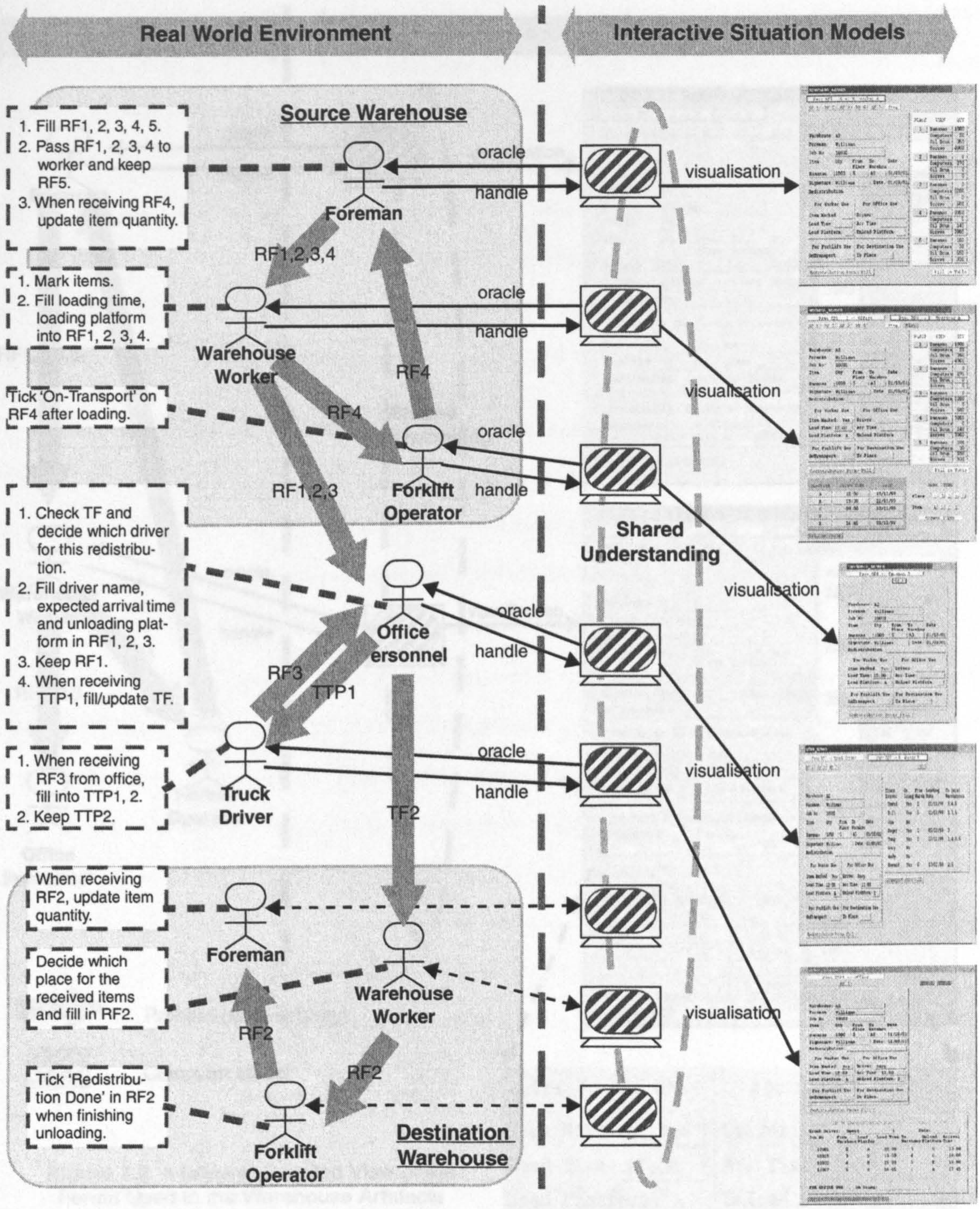


Figure 7.8 A Collaborative Working Environment for Manual Redistribution Between Warehouses

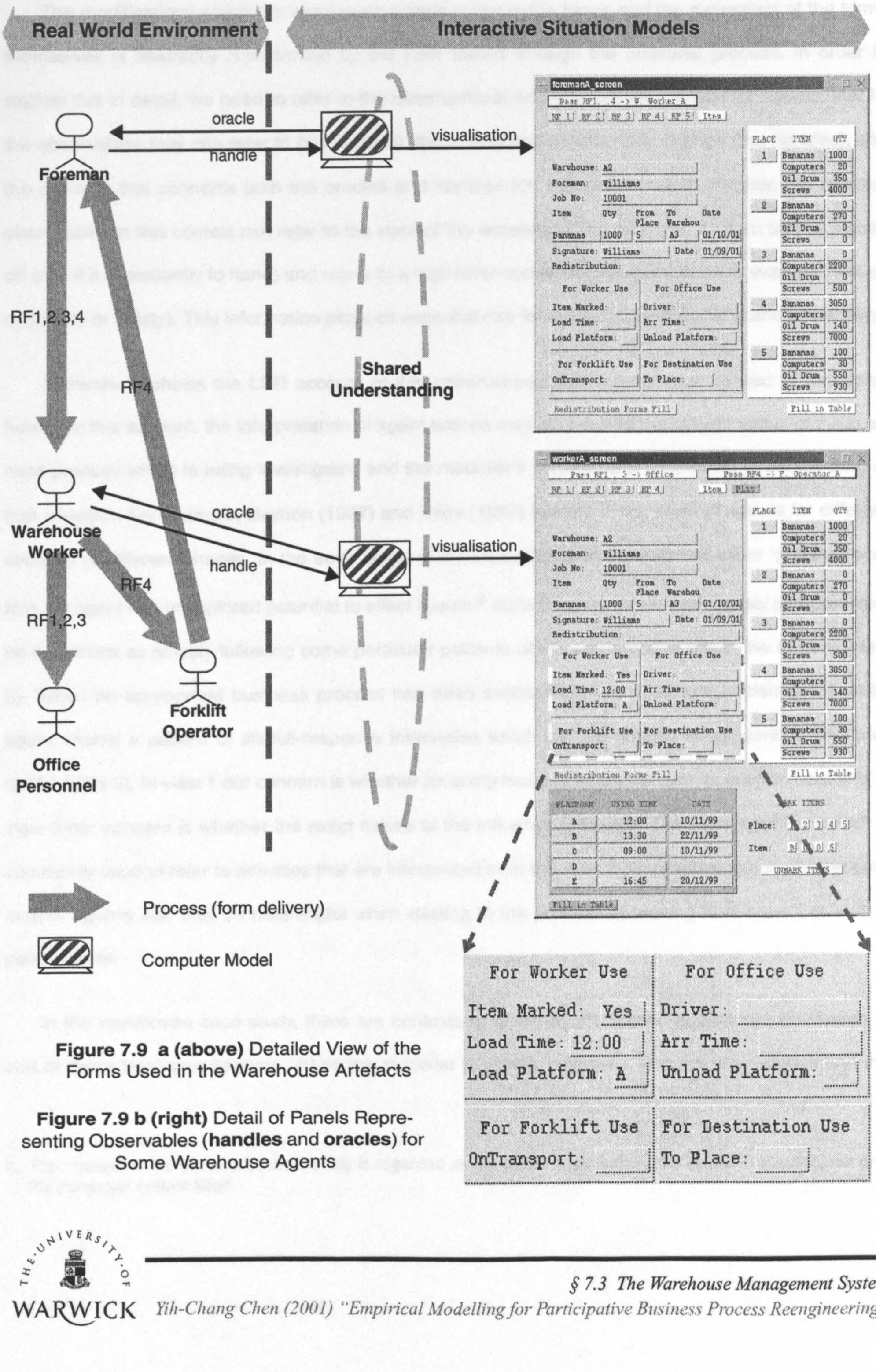


Figure 7.9 a (above) Detailed View of the Forms Used in the Warehouse Artefacts

Figure 7.9 b (right) Detail of Panels Representing Observables (**handles** and **oracles**) for Some Warehouse Agents

The modifications which the warehouse agents make to the forms and the movement of the forms themselves is abstractly represented by the path traced through the business process. In order to explain this in detail, we need to refer to the observational and interactional context for agents, that is, the observables they can refer to (the *oracles*), those they can conditionally change (the *handles*), and the protocol that connects both the oracles and handles (cf. subsection 4.2.2). Further, the relevant observables in this context can refer to the state of the warehouse (for example, an item can be signed off only if it is presently to hand) and relate to a high-level context for interpretation (for example, issues of legality or safety). This information plays an essential role for the purpose of auditing and traceability.

Appendix A shows the LSD account of this observational and interactional context of the warehouse. In this account, the interpretation of agent actions may vary due to the current status of the business process which is being investigated and the modeller's current (and subjective) understanding of that situation. For example, Beynon (1997) and Ness (1997) identify three views of agents for different contexts at different phases. In the early stage of familiarisation with an environment or 'putative' system, an agent has unexplored potential to affect system[8] state (view 1). At the later stage, an agent may be construed as reliably following some particular patterns of stimuli-response within the system (view 2). When an appropriate business process has been successfully identified and implemented, each agent enacts a pattern of stimuli-response interaction which can be entirely circumscribed and predicted (view 3). In view 1 our concern is whether an entity has any influence over its environment and in view 3 our concern is whether the exact nature of the influence is known. The term 'agent-oriented' is commonly used to refer to activities that are interpreted from the view 3 perspective, but in EM the concept of agency can only be meaningful when relating to the process of moving from view 1 to view 3 perspectives.

In the warehouse case study, there are contexts in which all the human agents can be viewed in one of these three perspectives. When the modeller is initially unfamiliar with the environment and the

8. The 'system' in our warehouse case study is regarded as the whole organisation (the business system), not only the computer system itself.

processes of the business, the personnel will represent the example of view 1 agents whose interaction with the warehouse environment and the operation of the business is unexplored. When the roles of a particular employee, such as the foreman, have been more clearly identified, they can be regarded as the view 2 agents as whose pattern of stimuli-response can be in some levels clearly identified. The aim of our modelling process is to fully understand the whole business process and attribute automated agency to the view 3 agents whose pattern of stimuli-response is entirely predictable. As described in Chen et al. (2000a), this accords with our argument that the business model is a form of generalised program, and the activity of business process reengineering closely resembles the capture of the program requirements.

The LSD account can be viewed as identifying and classifying the observables that reflect the modeller's current understanding and subjective viewpoint on the warehouse operation. When the construction of the model through the modeller's experimental interaction follows the transition from the view 1 to view 3 perspective, the LSD account can then be regarded as specifying the observables that describe the stimulus-response patterns in the organisation. We have described in chapter 4 the various kinds of observables in an LSD account. The observables which are attached to an agent are referred to as *state* observable. In general, the values of these observables can be directly changed by other agents. For example, the agent **`warehouseWorker`** can change the status of an item to 'moving pending' by manipulating the **`rf_moving_pending`** observable which is a state for the agent **`rf`**. The *oracles* are the observables to which an agent responds. For example, **`rf_item`** and **`rf_quantity`** in the agent **`warehouseWorker`** are examples of oracles for the warehouse worker, who has to know the identity and quantity of items to be redistributed before changing their status (cf. Figure 7.9 and Appendix A). The *handles* for an agent are those observables that are conditionally under its control. The observable **`rf_moving_pending`** is an example of a handle for the agent **`warehouseWorker`**.

The stimulus-response patterns for an LSD agent are modelled in two ways. The *derivates* are used to represent stimulus-response relationships that are indivisibly coupled. For example, the observable **`transportationError`**, which indicates whether there is a truck available for the specific time at which the foreman intends to make redistribution, is a *state* for the agent **`environment`** but also a *deri-*

vate for the agent **foreman**. That is to say, any change in the status of truck availability will also simultaneously change this observable. Looser coupling of stimulus and response is modelled in *protocols*, which consist of a set of guarded actions, each of which takes the form of an enabling condition and an associated sequence of possible redefinitions of observables. Each 'guarded action' can be regarded as a privilege to act. That is, if an enabling condition is met, a particular action *may* be performed. As an example of this principle, the agent **warehouseWorker** receives redistribution forms from the foreman (the enabling condition), then decides the loading time and platform, and passes the forms to both the office personnel and forklift operator (the guarded action).

Appendices B and C give the details of the warehouse process and Figure 7.10 shows a diagrammatic summary derived from the LSD account of the interactions between all the agents participating in the warehouse processes. The tables (which represent the information shown in the paper-based forms or on the boards) detailed in Appendix C represent all the artefacts which are necessary for the completion of redistribution among warehouses for a specific day. And Appendix B gives the details about the actions of different agents in terms of oracles and handles (i.e. the observables in the tables) as well as the dependencies between these observables. The diagram depicted in Figure 7.10 represents an abstract level of the warehouse process by means of form (tables) delivery[9] and the access to, or observation of, forms or boards by different agents. For example in Figure 7.10, the oracles to the warehouse worker are represented in the associated family of tables, where Tables F and G feature in Figure 7.12 below.

7.3.3 The ISMs for Representing the State in the Warehouse

One of the consequences of using EM principles is that the construction of computer-based artefacts has an explanatory role in that construction process. The state of each artefact is specified by a definitive script and is intended to represent the current state of its referent as viewed by a particular internal or external agent (cf. Figure 4.2). For the warehouse example, the relevant internal agents include fore-

9. The concrete details of form delivery have been depicted in Figures 7.8. and 7.9.

men, warehouse workers, forklift operators, office personal and truck drivers. The possible external agent can be an 'omniscient' global observer who is able to see all the movement and interaction within the warehouse (like the role of a God-view), or an auditor whose task is to trace transactions. By networking these artefacts and establishing dependencies and interactions among them we can represent the state of the relevant domain which reflects the related views of agents.

For interpreting the business process model and ensuring that the abstract phases of the model can be appropriately embodied in an agent's perception and action, we need to take account of the explicit and physical observables associated with the operation of the warehouse. In order to construct the business model under the SPORE framework in a situated manner, the ISM is used to incorporate

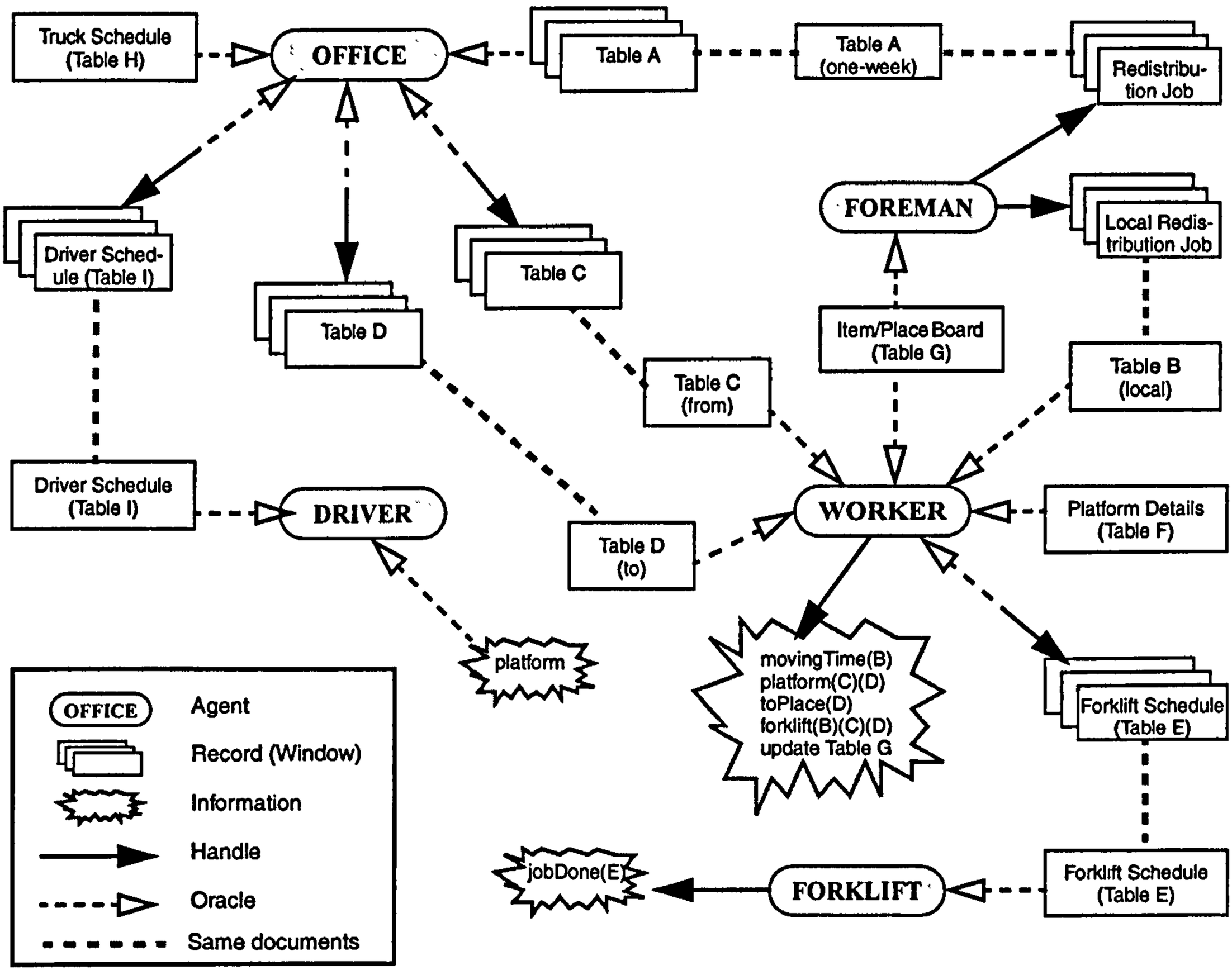
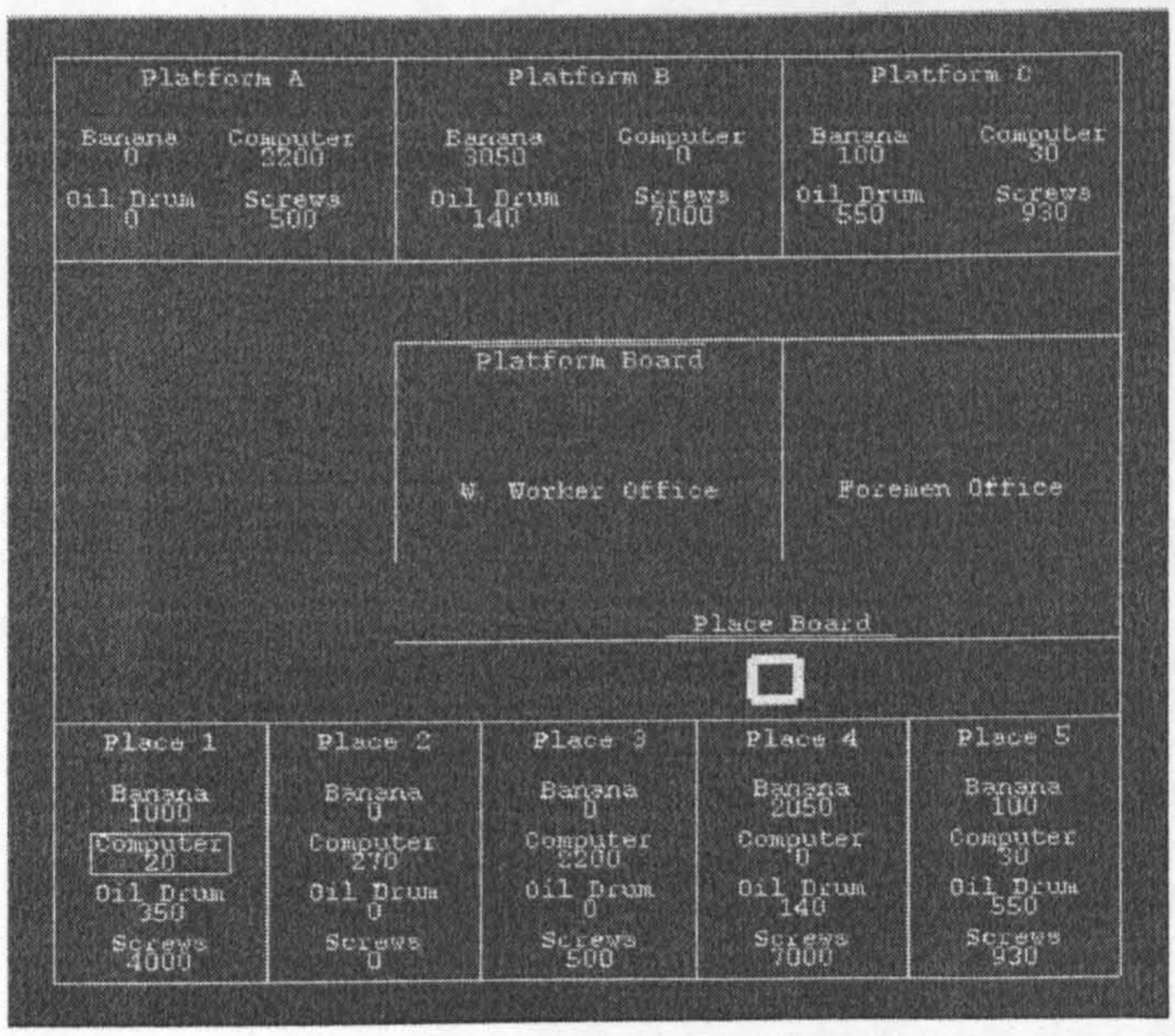


Figure 7.10 A Diagrammatic Summary of an LSD Account of Warehouse Processes

the 'matter-of-fact' observations of the current state of the warehouse. As mentioned earlier, the typical observables which may be significant are the items and the locations in the warehouse, as well as the inventory which records (connects) both the items and locations. The ISM we developed which represents these observables can supply the visual representations for items and locations, and also display the status of the inventory. Figures 7.11 and 7.12 are screenshots from artefacts created to represent several aspects of state in the warehouse situation, as seen by a warehouse worker. The left-hand side of Figure 7.11 depicts the physical state of the warehouse, showing the layout of the storage places and transportation platforms, and the location of items and information boards. The right-hand side depicts a form (the redistribution form) relating to the status of individual items in a redistribution process. Figure 7.12 depicts two of the many tables of information that together give the warehouse worker a more comprehensive view of the current status of items in the warehouse. All these tables have been described in Appendix C.

Such a representation of the current warehouse state is complemented by informal actions, for example the relocation of items, item looking-up in the inventory or the reception of a new item for stor-



Redistribution Form Copy: 1

Warwhouse: A2
Foreman: Williams
Job No: 10001

Item	Qty	From Place	To Warehou	Date
Bananas	1000	5	A3	01/10/01

Signature: Williams Date: 01/09/01
Redistribution:

For Worker Use	For Office Use
Item Marked: Yes	Driver:
Load Time: 12:00	Arr Time:
Load Platform: A	Unload Platform:
For Forklift Use	**For Destination Use**
OnTransport:	To Place:

Figure 7.11 (left) Physical State of Warehouse and **(right)** Form Showing Individual Item Status

age. And these different aspects of state play their part in shaping the role of particular workers in the warehouse. For example, the warehouse worker will be aware of all these aspects when making decisions, bearing in mind the physical locations when dealing with items (Figure 7.11 left); knowing the current status of the transaction through examining forms (Figure 7.11 right); and consulting the work schedules and timetables when planning the movement of items (Figure 7.12). In some contexts, this will motivate visualisations representing the intermediate states in the warehouse operation, for example the items in transit, or items located via the inventory but having not been retrieved from the warehouse.

Of course there is much more to the state of the warehouse than those aspects of state mentioned above can express. For example, checking the status of items will typically involve communication with other personnel. Also the architecture of the warehouse, the locations of information boards and the organisation of items all have a significant impact on the activities of the warehouse process. There are

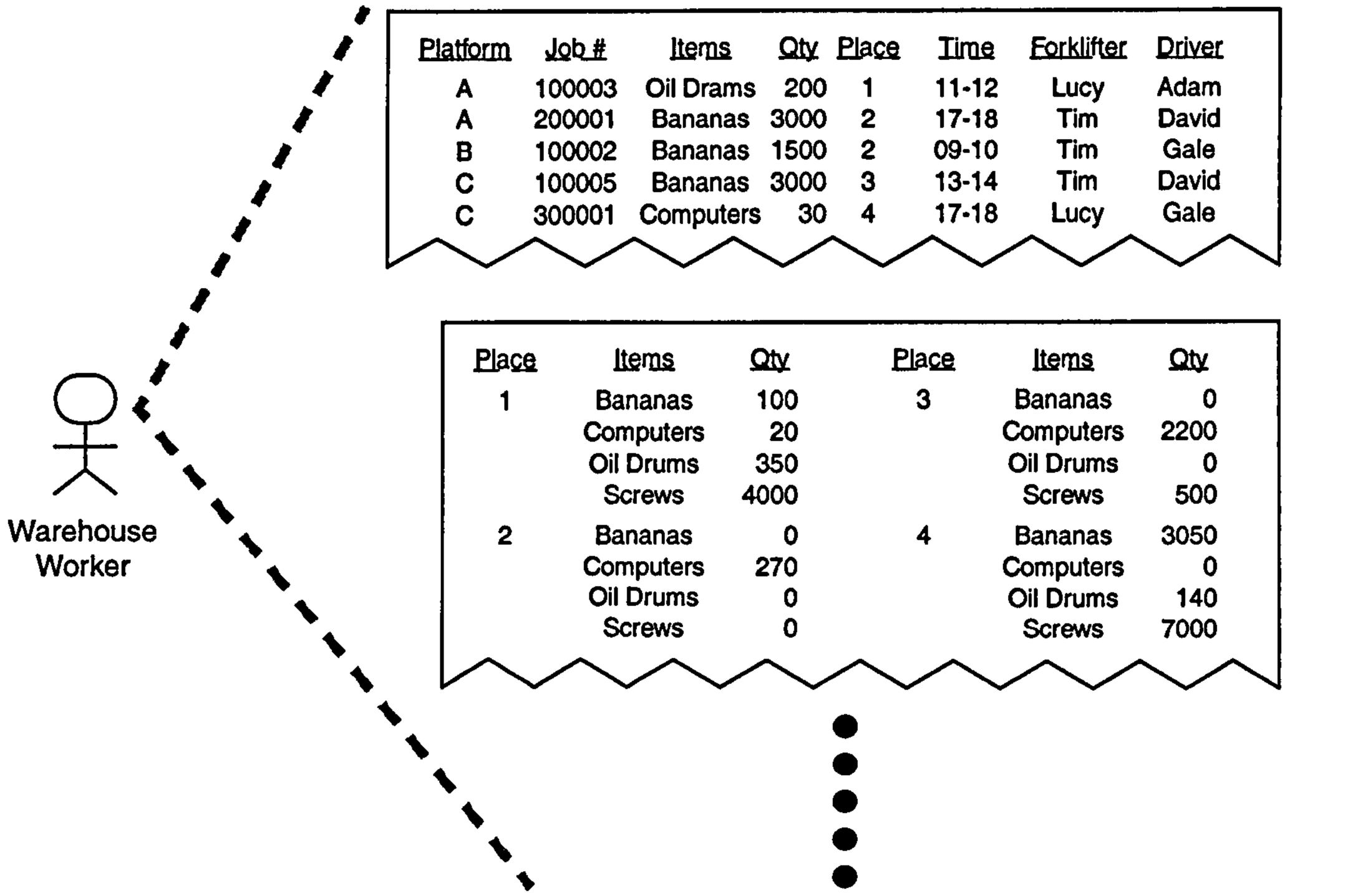


Figure 7.12 Tables Showing Comprehensive Item Status

also other issues about the way information is stored and transmitted on the paper-based forms. For example, there are security issues: generally the paper forms are physically inaccessible and are hard to modify without detection and can be audited. There are many additional significant aspects of states which are not represented above, for example the geographic location of trucks, drivers, other warehouses, and other factors such as traffic conditions.

From our viewpoint, a model of the warehouse should incorporate as much as possible of such aspects of state and state changes in order to correspond faithfully to its referent. It should also provide a context to experience the behaviours which are unpredictable and outside the scope of normal operation. Relevant observables for this purpose in the warehouse case study may include the issues such as the loss of items or warehouse locations, the concept of items being mislaid, or of items being perishable. The significance of our approach to the construction of models is that there is flexibility to adapt state interactively through cooperative activity in a distributed environment. This is achieved through the EM paradigm for state representation, which not only allows the state of the artefact to be changed by redefining variables[10], but is also open to allow the participants to *reinterpret* this state both through a shift in an individual participant's perception and through negotiation among these participants.

The interpretation of the LSD account (Appendix A) is closely tied up with the experimental interaction and investigation of the artefacts which are depicted in Figures 7.11 and 7.12. In the manual warehouse system, the dependency between contexts of tables represented in the LSD account might be maintained by directly copying from one table to another, or by the use of carbon copying. In an automated environment with supporting information systems, this dependency can be achieved by the use of relational databases with regular updates. We have suggested in subsection 6.4.1 that the organisational relationships in a business can be understood from the LSD account. That is, the LSD account can be viewed as a conceptualisation of agency and an expression of the agent's abstract perception of state. Provided that this is consistent with the physical and perceptual situations, such a LSD account

10.The redefinition of variables in this context can be interpreted as a state-transition, a correction, or extension of the artefact.

can be used as the basis for system redesign and implementation. For example, how the oracles of one agent are linked to the handles of another agent in Figure 7.10 indicates how this LSD account supplies a framework for defining the warehouse processes. The protocol, oracles and handles in the LSD account of an agent such as the warehouse worker indicates how the interfaces required to carry out the actions of its protocol are defined.

7.3.4 The Development of Warehouse Management System

This subsection will introduce the final system for warehouse management developed by this author and describe the process of its development which aims to link this practical work with the theoretical framework developed in chapter 6. As a definitive database management system (the EDDI) has been used in the development of the final system, this subsection will give a brief introduction to this database interpreter before describing the warehouse management system.

Introduction to EDDI

EDDI (Eden Definition Database Interpreter) is a definitive database notation which is implemented by the EM tool as a database management system. The notation of EDDI was developed to illustrate how the definitive scripts can be used to set up and query a relational database with the characteristics of the models by Codd (1970; 1979) and Todd (1976)[11]. Within EDDI, the database and the concept of dependency can be combined and this demonstrates a powerful character by applying the EM principles to a database structure. As implemented by the EM tools, the tables defined under EDDI are themselves made with definitions and thus they will automatically update when the contents of tables change. A database table under EDDI can be defined through a definition which maintains the dependencies in the script. This means that any change in the script will automatically update ('propagate' to) all the relevant observables which directly or indirectly dependent on it. This situation can be illustrated

11.Codd's relational model allows attributes to be organised in a systematic way according to the relationship between them. Tood implements a system to maintain relationships between data (based on Codd's relational model) by using dependency.

through a running script depicted in Figure 7.13. The tables shown in Figure 7.13 are created with the EDDI commands (in the EDEN input window), for example:

```
%eddi
Coventry (JobNo int, FromHouse char, Foreman char, Items char, Qty int, FromPlace int, ToWarehouse
      char, FinishedBy char);
Coventry << [100001,"Coventry","Bill","Screws",550, 4,"London","30/11/01"];
Coventry << [100002,"Coventry","Susan","Bananas",1500,2,"London","10/11/01"];
Coventry << [100003,"Coventry","Bill","Oil Drums",200,1,"Manchester","20/11/01"];
Coventry << [100004,"Coventry","Susan","Computers",120,5,"London","20/11/01"];
```

Terminal<4>

File Options Help

❶

```
?Coventry
```

JobNo	FromHouse	Foreman	Items	Qty	FromPlace	ToWarehouse	FinishedBy
100001	Coventry	Bill	Screws	550	4	London	30/11/01
100002	Coventry	Susan	Bananas	1500	2	London	10/11/01
100003	Coventry	Bill	Oil Drums	200	1	Manchester	20/11/01
100004	Coventry	Susan	Computers	120	5	London	20/11/01

```
(4 rows)
?London
```

JobNo	FromHouse	Foreman	Items	Qty	FromPlace	ToWarehouse	FinishedBy
200001	London	Nick	Bananas	3000	3	Coventry	10/11/01
200002	London	Nick	Bananas	1500	2	Manchester	20/11/01
200003	London	Buck	Oil Drums	700	5	Manchetser	20/11/01

```
(3 rows)
?Manchester
```

JobNo	FromHouse	Foreman	Items	Qty	FromPlace	ToWarehouse	FinishedBy
300001	Manchester	Chris	Computers	30	1	Coventry	30/11/01
300002	Manchester	Alex	Oil Drums	950	3	London	20/11/01
300003	Manchester	Alex	Oil Drums	40	4	London	30/11/01

❷

```
(3 rows)
ToLondon is ((Coventry+London+Manchester): ToWarehouse == "London");
?ToLondon
```

JobNo	FromHouse	Foreman	Items	Qty	FromPlace	ToWarehouse	FinishedBy
100001	Coventry	Bill	Screws	550	4	London	30/11/01
100002	Coventry	Susan	Bananas	1500	2	London	10/11/01
100004	Coventry	Susan	Computers	120	5	London	20/11/01
300002	Manchester	Alex	Oil Drums	950	3	London	20/11/01
300003	Manchester	Alex	Oil Drums	40	4	London	30/11/01

❸

```
(5 rows)
Coventry << [100005,"Coventry","Susan","Bananas",3000,3,"London","30/11/01"];
?Coventry
```

JobNo	FromHouse	Foreman	Items	Qty	FromPlace	ToWarehouse	FinishedBy
100001	Coventry	Bill	Screws	550	4	London	30/11/01
100002	Coventry	Susan	Bananas	1500	2	London	10/11/01
100003	Coventry	Bill	Oil Drums	200	1	Manchester	20/11/01
100004	Coventry	Susan	Computers	120	5	London	20/11/01
100005	Coventry	Susan	Bananas	3000	3	London	30/11/01

```
(5 rows)
?ToLondon
```

JobNo	FromHouse	Foreman	Items	Qty	FromPlace	ToWarehouse	FinishedBy
100001	Coventry	Bill	Screws	550	4	London	30/11/01
100002	Coventry	Susan	Bananas	1500	2	London	10/11/01
100004	Coventry	Susan	Computers	120	5	London	20/11/01
100005	Coventry	Susan	Bananas	3000	3	London	30/11/01
300002	Manchester	Alex	Oil Drums	950	3	London	20/11/01
300003	Manchester	Alex	Oil Drums	40	4	London	30/11/01

(6 rows)

Figure 7.13 A Snapshot of EDDI Script

will create the table `Coventry` and insert four tuples with typed fields into it (marked ❶ in Figure 7.13). The *insert* command '<<' followed by a tuple separated by commas enables the modeller to insert tuples into a table at any stage (cf. mark ❸). Tuples can also be deleted from a table by *delete* command '!!' such as

```
Coventry !! [100004,"Coventry","Susan","Computers",120,5,"London","20/11/01"];
```

The query command '?' followed by a specific table name enables the modeller to view that specific table, especially to inspect the result of a relational algebra expression or the effect of any change in the script due to the dependencies. For example, running '`?Coventry`' in the EDEN input window will display the contents of the table Coventry (cf. mark ❶).

The EDDI interpreter also provides other commands for data manipulation and data definition. There are several operators of relational algebra which, as identified by Codd (1979), are used to specify the definitions for existing tables or refer to views: *union* (+), *difference* (-), *intersection* (.), *projection* (%), *selection* (:) and *join* (*)[12]. The definition can be defined at any time in the script. For example in Figure 7.13, the specification of a definition (the line marked ❷)

```
ToLondon is ((Coventry+London+Manchester): ToWarehouse == "London");
```

will define a definition and create a new table `ToLondon` which will consist of the tuples found in either table `Coventry`, `London` or `Manchester` (under the *union* operand '+') and whose destination warehouse is London (by the *selection* operand ':' to compare the tuples which satisfy the specific condition). The word '`is`', as described in subsection 4.2.1, indicates that the dependency is maintained automatically. So the value of left-hand side (`ToLondon`) is maintained to be the up-to-date value of the relational expression on the right-hand side. We can illustrate the effect of the definition within EDDI by the same script in Figure 7.13. Before adding the new tuple into the table `Coventry` (i.e. mark ❸), the table `ToLondon` consists of all the relevant tuples which are to be found in the three existing tables.

12.These operators are in increasing order of precedence. It is only possible to construct the *union*, *difference* and *intersection* of two relation tables if they have the same type.

After inserting a new tuple into the table `Coventry` (i.e. the job number: 100005), not only this new inserted tuple appears in the table `Coventry`, it also appears in the table `ToLondon` (the dependant). This is because it depends on the table `Coventry`, and also the new inserted tuple has satisfied the condition set in that definition.

Introduction to the Useful System

In this warehouse case study we assume a simple situation in which the warehouse management company has a main office (as its headquarters) and three warehouses located in Coventry, London and Manchester. The following people will be using the system: the office personnel and truck drivers, and in each warehouse the foremen, warehouse workers and forklift operators. The system used is formed by several computer-based models (the ISMs) running in a distributed environment. There are three main artefacts in each model presented to the human agent: the user interface window for giving a command and inputting/modifying data; the terminal displaying the relevant information, in the format of EDDI database, to respond to the human action; and the EDEN input window which enables the human agent to send messages to other models and, provided such that human agent has an appropriate privilege, to add or modify the definitions in the script.

In Figure 7.14 the upper part (A and B) shows the window management system for the foreman in the Coventry warehouse, the lower part (C, D and E) shows the window management system for the office personnel. Figure 7.14 (A) is the window presented to the foreman in which he can give a command for redistribution between warehouses, either by adding new redistribution jobs or modifying existing ones[13]. Figure 7.14 (B) is the terminal windows which shows two tables in the EDDI format: one shows the current status of items in the warehouse[14] (when the foreman clicks on the button marked ❶ in (A)), and the other shows all the redistribution jobs which have been planned in the warehouse Coventry[15] (i.e. the button and table marked ❷).

13.The foreman can input the job number of an existing job in the window and modify its contents.
14.Compare with the Table G in Appendix C and in Figure 7.10.
15.Compare with the Table A-1 in Appendix C and Table A in Figure 7.10.

Figure 7.14 (C), (D), (E) show the windows for the office personnel as well as the relevant EDDI tables on the terminal[16]. As the main work of office personnel is to plan a truck drivers' schedule and the date for a specific distribution job, there are several information formats which the office personnel may find useful and helpful for their decision making:

- The office personnel may like to view all the distribution jobs from any specific warehouse (for example they can view all the jobs made from the warehouse Coventry (mark ❶) or from Manchester (mark ❷)). Alternatively, they can choose to view all the distribution jobs whose destinations are the same warehouse (for example, all distribution jobs to the warehouse London (mark ❸)). The information represented here can help the office personnel to decide the most economical truck schedule (for example, allocating the same truck for two redistribution jobs from Manchester to London and from Coventry to London) and avoid the allocation of two trucks between the same two warehouses on the same day.
- The office personnel may sometimes need to view all the redistribution jobs which need to be finished by a specific date (mark ❹).
- When deciding a truck for a redistribution job, the office personnel also need to check the time schedule of all drivers (mark ❺). They can either view the individual schedule of a driver[17], or they can input the date in the window to view all the time schedules for drivers who have been allocated jobs on that specific date[18].

Note that the table showing all the redistribution jobs to London (i.e. mark ❸) is the same table of `ToLondon` described in Figure 7.13 (but using different definition with different attributes listed after *projection* '%'). In reality, all the tables represented here are defined under EDDI and use different definitions which have been described in the previous subsection. For this reason, inserting any new tuples or modifying the existing ones in one table will automatically update the contents of its dependent

16.Figure 7.14 (D) and (E) are representing the same information but running at different stages. We will use these two artefacts to illustrate the dependency feature in EM in this system.

17.Compare with Table I in Appendix C and in Figure 7.10.

18.Compare with Table H in Appendix C and in Figure 7.10.

tables. Using the same example illustrated in Figure 7.13, when the foreman in the warehouse Coventry gives a new command for redistribution (cf. Figure 7.14 (A)), this new job information (the tuple newly inserted into table `Coventry`) will appear in the relevant tables in the office model (cf. Figure 7.14 (E)) as such tables depend on the table `Coventry` and this newly inserted tuple satisfies the conditions set in their definitions (e.g. `FromWarehouse == "Coventry"` (Figure 7.14 (E) ❶) and `ToWarehouse == "London"` (Figure 7.14 (E) ❸)).

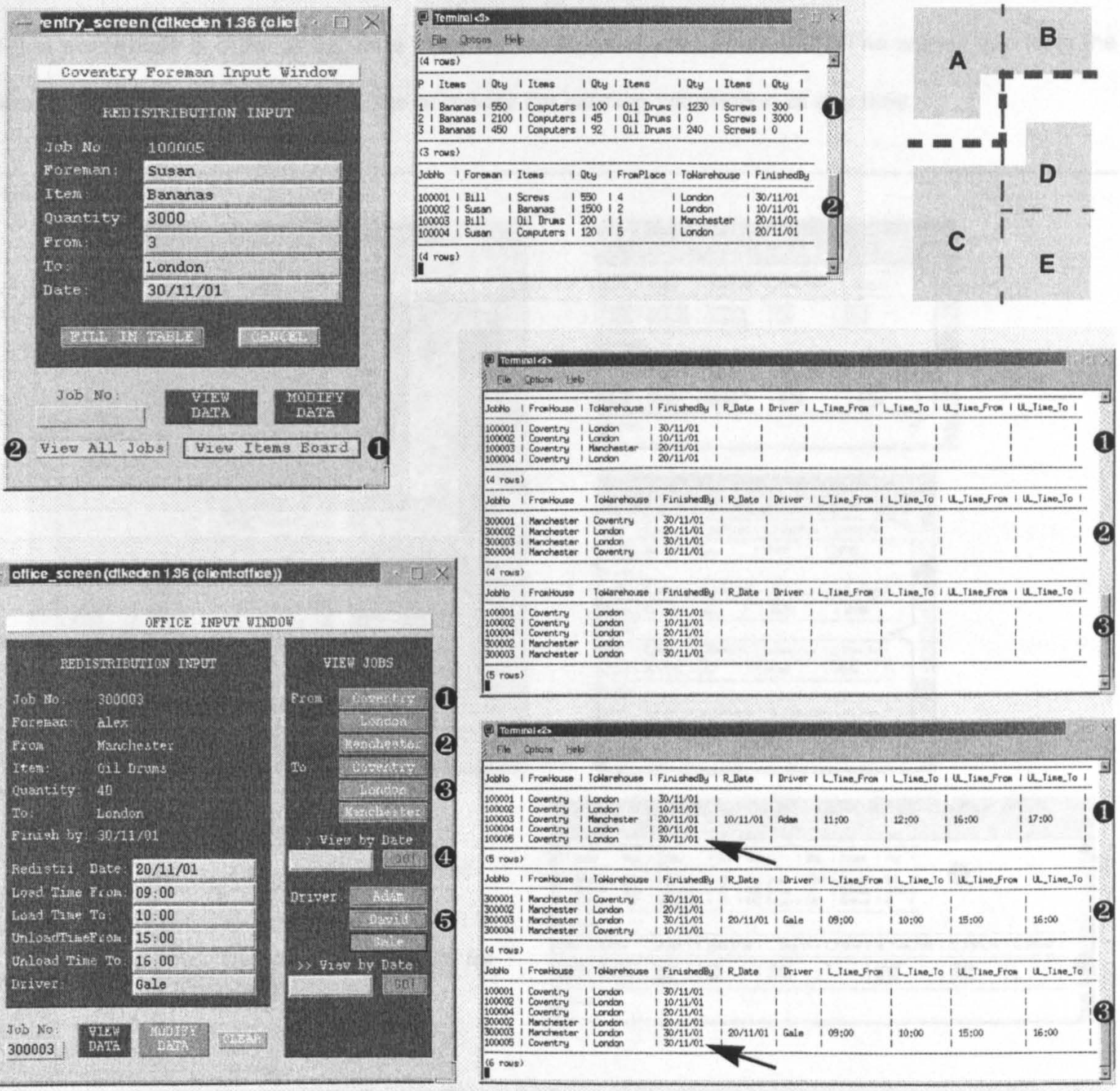


Figure 7.14 The Window Interfaces for Foreman and Office Personnel

Figure 7.15 depicts the window interface for the warehouse worker. Similar to the discussion above, there are also several information formats which the worker may find useful for their work, e.g. deciding a forklift operator for a redistribution jobs, allocating a loading or unloading platform, and finding out a suitable storage place for the items received. The worker may firstly need to know all the relevant jobs redistributed from, or sent to, his warehouse (mark ❶ in Figure 7.15). The worker needs to check the time schedule of each forklift operator in order to allocate the jobs[19] (mark ❷ and ❸). He also need to check the usage of platforms at a specific date[20] (mark ❺), and view the current status of item storage in that warehouse in order to decide a place for the items received (mark ❹). The worker can fill in the relevant data through his input window or modify the existing information at any time.

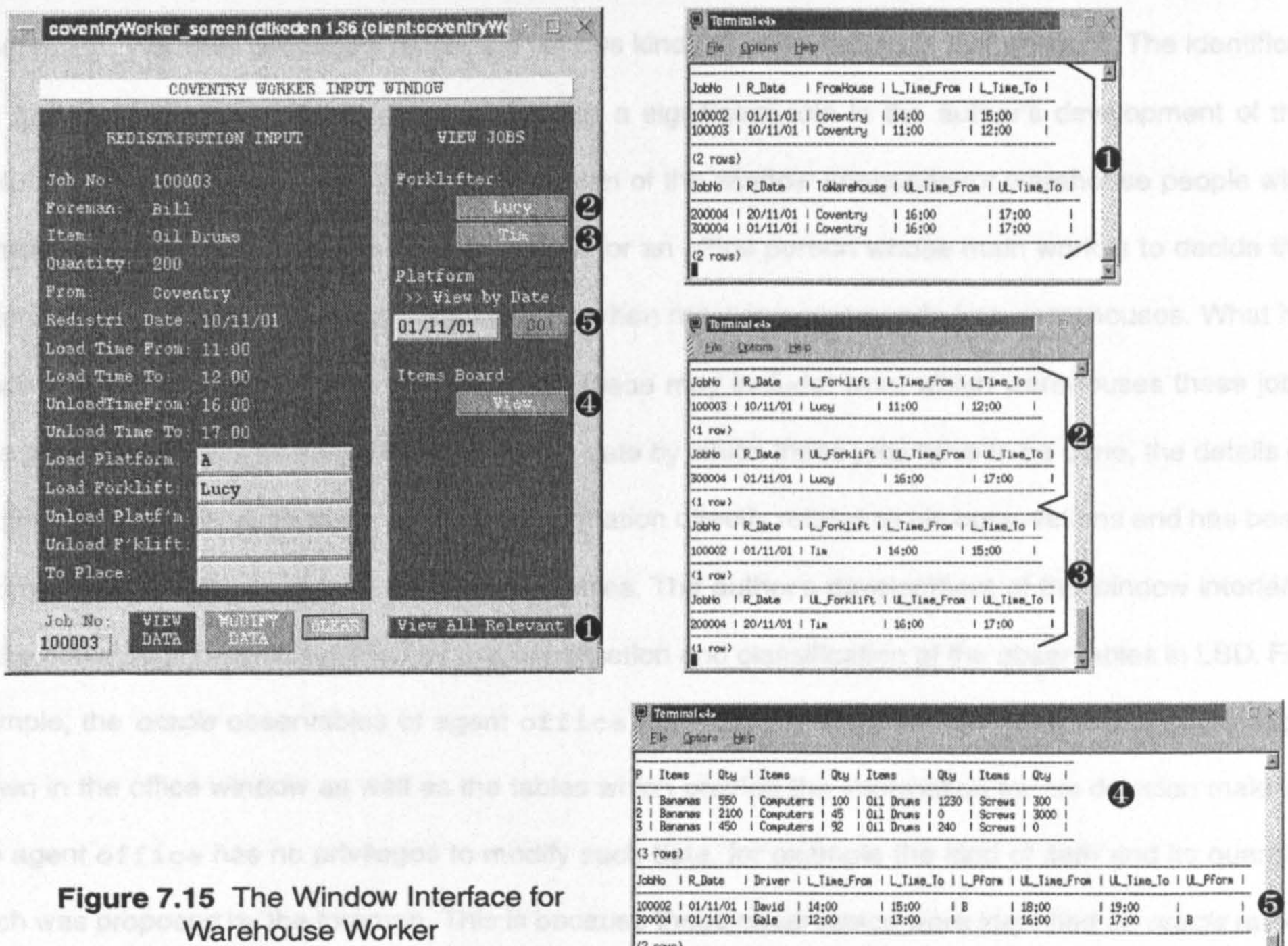


Figure 7.15 The Window Interface for Warehouse Worker

19.Compare with Table E in Appendix C and in Figure 7.10.
20.Compare with Table F in Appendix C and in Figure 7.10.

All the tables represented in the worker's window interface are also automatically updated whenever the contents of tables (the dependees) in office or foreman window change. For example, when the office personnel have input the driver and redistribution time for a job whose source or destination warehouse is Coventry, this job information will immediately be updated in the table within the Coventry worker's window due to the dependencies maintained by EDDI interpreter.

The Development Process of the Useful System

As described in subsection 6.4.2, there are two levels of contextual models in EM which enable the modeller to have contextual knowledge about the system to be developed or reengineered: the LSD account and the EM model. The LSD account in Appendix A records this author's perception about the observational and interactional context of a warehouse management system in terms of agency and observation. We have described earlier the various kinds of observables in that account. The identification and classification of these observables play a significant role in the author's development of the useful warehouse system, especially in the design of the window interfaces for warehouse people with different roles and responsibilities. For example, for an office person whose main work is to decide the redistribution date and allocate a truck for a job when receiving commands from warehouses. What he needs to know in order to make a decision for these may include: from which warehouses these jobs were proposed as well as their destinations, the date by which these jobs have to be done, the details of the time schedule for each truck, etc. Such information directly relates to his observations and has been recorded in the LSD account as *oracle* observables. The author's development of the window interface for the office personnel was guided by the identification and classification of the observables in LSD. For example, the *oracle* observables of agent `office` were directly mapped into the relevant information shown in the office window as well as the tables which provide the information for his decision making. The agent `office` has no privileges to modify such data, for example the kind of item and its quantity which was proposed by the foreman. This is because these observables were identified as *oracle* rather than *handle* in the LSD account. The *handle* observables, in contrast, gave the guidance when defining the input data within the office window such as the truck driver, the date and loading/unloading times for

a redistribution job. Also, as the *derivates* and *protocols* in LSD reflect the actual behaviour and action of human agents, referring to these helped this author to decide the appropriate buttons in the windows as well as define their functions.

Through the practical development of this system, the author found that the LSD account not only served as a contextual model for understanding the system environment, it also provided clues and guidance for choosing and defining the relevant data displayed in the interface and avoiding any omissions or redundancy. For example, the office personnel does not need to know in which place the item is currently stored, and a warehouse worker does not need to know which truck driver is allocated for a redistribution job or which warehouse is the destination when he decides the forklift and the platform for that job. Because these observables were not classified as *oracle* observables for such agents, through referring to the LSD account, they do not appear in the developed window interfaces. By analysis of this nature, possible omissions and redundancy can be avoided.

The second level of the contextual model is the EM model which embodies the contextual information described in LSD and enables the modeller to interact and have experience with it in an open-ended manner. The author has developed a business process model for the warehouse which has been described in subsection 7.3.2. This model (the paper-based ISM) can be regarded as the seed-ISMs in the unified development procedure in Figure 5.3. By interacting with this model, the author, as well as other participants playing various roles in the warehouse example, have gained a deeper insight and experience of the current redistribution process between warehouses, and understood how and why certain interactions were performed (cf. Figures 7.6 and 7.8). The visual displays of the paper-based ISMs are the mock-ups of the actual forms used in the warehouse (cf. Figures 7.9 and 7.11). Metaphorically 'filling in' and 'delivering' these forms in the ISM represents the current status of the process. The user windows of the final system were developed mostly based on the interface of these forms. The layout of the windows, as well as the definitions in the script, of the final system were modified mainly from the ones in the seed-ISM (through the advantage of the 'on-line' redefinition under the EM tools) to meet the actual need of relevant users. In this case, the ISM in the process of development plays a role as a rapid prototype which we described in section 6.4. The warehouse process prior to the introduction

of the system, i.e. the paper-based form delivery as depicted in Figure 7.6 and Figure 7.8, has been replaced (automated) by the dependencies maintained in the EDDI database. For example, the original delivery process of three redistribution forms (i.e. RF1,2,3 from the foreman to warehouse worker then to the office) has been replaced by the dependency between two tables: one in the foreman's interface (Figure 7.14 (B) ❷) and the other in the office interface (Figure 7.14 (D) and (E) ❶). Through the construction of, and interaction with, the ISMs the author can evaluate the system requirements to find out whether they were incomplete or inconsistent. For example, the system is required to immediately update whenever new data has been input. For this, it is necessary to check whether a new tuple has been inserted in the relevant table in the office interface when a foreman gives a new command through his input window.

The development process of the final system also reflects the essence of participative process modelling under SPORE. By interacting with the ISMs, the various roles of human agents (e.g. foremen, warehouse workers and office personnel) were participating and thus directly involved in the process of system development. The system was built under the distributed EM tool (the **dtkeden**) which formed a collaborative working environment as depicted in Figure 6.3 and Figure 7.8. Four interaction modes provided by **dtkeden** (cf. subsection 6.3.2) enable the participants to communicate with each other and 'visualise' their viewpoints in order to gain better consistency between their insights. For example under the *normal* mode, the foreman (probably with some specified privileges) might wish to express his idea of new requirements by modifying the query for an EDDI database in the warehouse window interface. The agent in the server model is able to see all the movement and interaction during the process (the God-view). This provides an opportunity for the auditor to be involved in the process or enables the senior management as well as the BPR designers to have a global view of the effect of any local change in a part of the system or the process. Decision making support and BPR for the warehouse management can also be achieved increasingly under this framework. As described in section 6.4, DSS and BPR are different mainly in their scopes of analysis, and DSS focuses on the perspective of internal agents whereas BPR focuses on that of external agents. In this final system, the office personnel or warehouse worker might wish to propose a protocol in their scripts in order to make their decision making more effi-

cient and accurate. For DSS, they can try their proposed protocol by adding or redefining the definitions in the script and test this in an interactive manner (the 'what-if' experiment). For example the warehouse worker can propose an procedure which will search for a forklift operator whose daily schedule is available for the redistribution jobs received and automatically fill the operator name in. From the perspective of BPR, the external agent in the server model can intervene in the process or give different privileges to the internal agents to view the global effect under the *interference* mode.

Participative BPR in the Warehouse Case Study

We emphasise in this thesis the importance of stakeholders' active participation in the process of BPR, and the distributed EM tool has provided a collaborative working environment for participative BPR. In this warehouse case study, besides the designers (including both the business designers and system designers) and the warehouse personnel mentioned earlier (foremen, officer personnel, warehouse workers, etc), the participants in the process may also include the senior managers and their customers. It is essential that senior managers are involved in the process because a critical factor to the success of BPR is support from top management; and customers must be involved as BPR is radically customer-oriented. And we include different roles of personnel in the process because they have different perceptions (and knowledge) about the business process they are currently using in as well as the newly designed ones. The following cases indicate some possible scenarios during the process of participative BPR, which include consideration of an individual viewpoint about the BPR plan proposed and the problems encountered.

Case 1 After using the management system, the foreman has experienced that it is easy to add and modify redistribution data through the interface, but it is difficult to identify the right place in which the items are currently stored. For example as Figure 7.14 (A) depicted, the foreman would like to move 3,000 boxes of bananas from Place 3 to Warehouse London. But actually there are not sufficient bananas in that place (i.e. only 450 boxes of bananas in Place 3 as depicted in Figure 7.14 (B)) for the command he proposed. During the process of BPR, the foreman can -- perhaps with assistance by the

designer – add a new definition (in his ISM input window) to automatically display only the places and quantities of the item the foreman input in the interface. For example:

```
%eden
Proc showDetail_click : showDetail_mouse_1 {
   if (showDetail_mouse_1[2]==4) {
      itemName = itemText_getText();
      execute("%eddi\n ?PlaceDetailsCoventry: Items == \"" // str(itemName) //
         "\" %Items, P, Qty;");
   };
};
```

This will show an EDDI table in the foreman terminal which gives the information about the current storage places and quantities of the item the foreman inputs in his window interface. And the foreman can modify the existing process `addButton_click` (cf. Appendix D for the original script) to:

```
proc addButton_click : addButton_mouse_1 {
   if (addButton_mouse_1[2]==4) {
      changeData();
      if (detectError()) {
         writeln("The input of the item's quantity is not correct!");
         break;
      } else {
         execute(%eddi\n  Coventry << [\"Coventry\"," // str(jobNumberC)
            // ",\"" // c2 // "\",\"" // c3 // "\"," // c4 // "," // c5 // ",\"" // c6 // "\,\""
            // c7 // "\,\"\"",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\"];";
      jobNumberC++;
      };
   };
};
func detectError {
   for (i = 2; i <= PlaceDetailsCoventry#; i++) {
      if ((PlaceDetailsCoventry[i][2]==str(itemText_getText())) ||   /* Item Name */
         (PlaceDetailsCoventry[i][1]==(fromplaceText_getText()) ||   /* Place */
         (PlaceDetailsCoventry[i][3]<(quantityText_getText())) {     /* Quantity */
         return TRUE;
         break;
      } else {
         return FALSE;
      };
   };
};
```

Such a redefinition will enable the foreman to add or modify procedures/functions in the definitive script in an open-ended manner, i.e without rerunning or compiling the script. As these changes are localised within the foreman's ISM, other participants may all agree to this modification and thus include this in the new process.

Case 2 The office personnel find that the system makes their work easier and more effective than doing the same procedures by paper-form delivery. However they may find that the foremen in some warehouses send their redistribution commands to the office very late, and this always causes inconvenience for their daily work. During the process of BPR, they can experience a new situation by adding a new definition which will reject the foreman's command if the finish-by date is less than three days prior to today. Of course, such a change will meet resistance from the foreman when he sometimes receives the rejected commands from the office. With several 'communications' by redefining the definition, they may, for example, agree that the commands will be rejected if they are less than seven days prior to today.

Case 3 The truck drivers are sometimes annoyed by the situation that the redistributed items are not ready on the loading platform when the truck arrives. This usually delays the truck and thus causes further inconvenience to the workers in the following destination warehouses. The office has sometimes received complaints from other warehouses when such a delay in loading means that the arrival times are different from the ones in the truck time-schedule made by the office. For this reason, the office may wish to make a new procedure in the script which enables the truck drivers to record the actual loading and unloading times. For example, a new EDDI table is created for Driver `Gale` to record the actual time of loading and unloading:

```
%eddi
Gale_actualTime (JobNo int, Date char, FromWarehouse char, L_Time_From char, L_Time_To char, ToWare-
      house char, UL_Time_From char, UL_Time_To char);
```

Then a new procedure is created for the office personnel to check and compare the difference between the actual loading/unloading time and the planned ones[21]:

```
%eden
proc check_click : check_mouse_1 {
   if (check_mouse_1[2] == 4) {
      execute("writeln(\"Gale's Proposed Time Schedule\");");
      execute("%eddi\n ?Gale %JobNo, R_Date, FromHouse, L_Time_From, L_Time_To, ToWarehouse,
        UL_Time_From, UL_Time_To");
      execute("writeln(\"Gale's Actual Time Schedule\");");
      execute("%eddi\n ?Gale_actualTime");
   };
};
```

This will display these two tables in the office window interface. For example through the two tables shown in Figure 7.16, the office personnel may find out that the delay of the truck was caused by the delay of both the unloading work at Warehouse London and the loading work at Warehouse Manchester. The auditor may find that this is useful for him to check the daily performance of each warehouse. The manager will also probably support such changes as he can observe the performance of each part of the process in more detail.

Case 4 The customer sometimes complains about the service provided by the warehouse, such as it takes too long time to deal with his requests or sometimes the items are sent to the wrong warehouse. In order to improve this situation, the manager may decide to introduce a bar-code system for identification of items or a system to enable the customers to directly give their requests on-line through

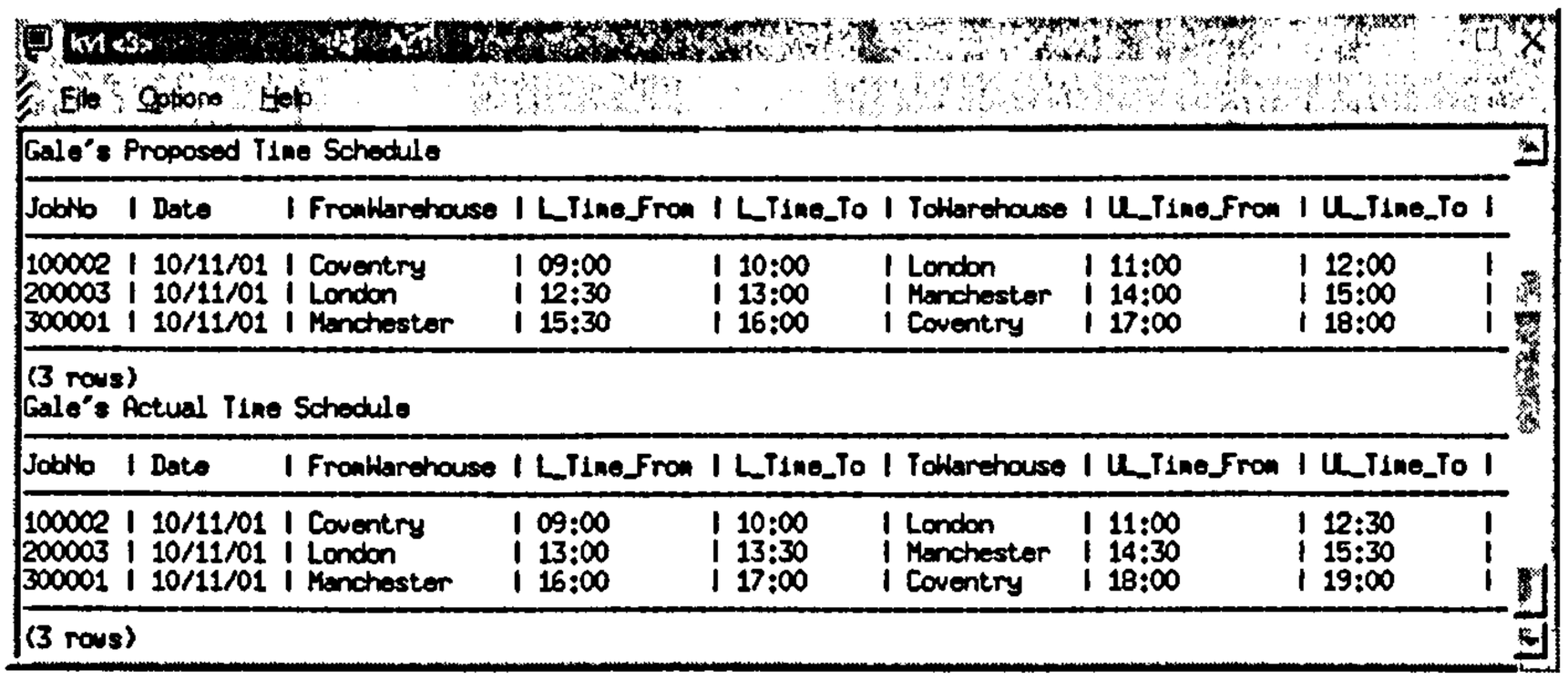


Figure 7.16 Two Tables for the Comparison by the Office Personnel

21.The `execute (string)` function executes a string as Eden statements, where the string must be the valid and complete Eden statement.

the internet. The introduction of such change may narrow the role of the foremen, i.e from that of an active character (i.e. making the redistribution commands) to a passive character (i.e. checking the data received from customers). The involvement of customers in the process of BPR can let them experience the changes and evaluate them. At the same time other personnel in the warehouse may resist such changes because some of them may even lose their jobs if their daily work is automated by the system. The manager in this case can observe their different perceptions about the change and find out the optimal solution for the conflicts.

7.3.5 Concluding Remarks

There is no single ISM which can represent all the aspects of the warehouse state. Typically the state of the warehouse is represented by different ISMs according to what problems are focused on in the SPORE framework. The ISM we construct here incorporates the seed-ISMs for the warehouse: the form-based abstractions that captured the state of the business process model and the activities of the agents; the storage, retrieval and distribution of items; and additional observations such as the one associated with the wider significance of the warehouse operation (e.g. concerned with the legality and the integrity of the business process). The main activities involved in the construction of ISMs in the warehouse case study have been summarised in Sun et al. (1999) that:

- the identification of agents (e.g. foreman, warehouse worker, etc);
- the conception for the roles for these agents corresponding to their characteristic skills;
- the apportioning of responsibilities for particular phases within a given transaction;
- the refinement and formalisation of their precise observables and protocols.

The potential framework for BPR established by applying SPORE can be illustrated by the transformation from a paper-based to a computer-based ISM.

Business process activity can also be viewed from perspectives other than those of the participants within it (i.e. the internal agents), such as the perspectives of an auditor ("What has been going on?" or

"Does it conform to regulations and standards?") and of an analyst ("What could be improved?" or "How could processes be more efficient and stable?"). The distributed version of EDEN (**dtkeden**, see Sun, 1999) enables us to separate the viewpoints of the agents within the model and to complement these with an external observer's interpretation. Figure 7.9 shows how computer-based forms are used to represent the environment for each agent's interaction. In this way, the distributed ISM can serve as a medium by which we can identify and enact appropriate transactions, and refine these through the collaborative interaction between participants. Many possible issues in requirements can be addressed by SPORE in this way such as (Chen et al., 2000a):

- Through experimentation at different workstations (the models), we can identify issues which are problematic from the viewpoint of particular agents: for instance, "how does the office know which drivers are available?", "how does the office determine whether a transaction is completed?".
- Through the elaboration of different seed-ISMs, we can address some additional issues, such as transportation costs, perishable goods, security and trust concerns.
- Through modifying dependencies and communication strategies, we can consider the effects of different technologies, such as are associated with the use of mobile communications, the Internet, optical bar code readers, or electronic locking agents.
- Through collaboration and synthesis of views, we can distinguish between subjective and objective perceptions of state, e.g. to contrast "I remember doing X" with "I have some record of doing X" with "There is an official record of X", or to model misconceptions on the part of an agent.
- Through intervention in the role of superagent, it is possible to examine the consequences of singular conditions which arise from opportunistic interaction or Acts-of-God, and to assess activities outside the scope of normal operation such as are associated with fraud, or manual back-up to automated procedures.

CHAPTER EIGHT

Conclusions and Further Work

This final chapter brings the thesis to a close by returning to the agenda which was established in chapter 1. It summarises the focus of the EM approach on both system development and BPR, and draws conclusions from the discussions and results in the previous chapters. In the final section of this chapter we also discuss briefly some thoughts and open issues which should become the avenues to be explored for future developments about and around EM and BPR as well as other areas of business information systems.

8.1 Research Summary

At the start of the thesis it was stated that the aim of this research is to propose an innovative approach to business process reengineering and the development of associated information systems. The preliminaries about the challenges and potential problems faced in these subjects were firstly summarised (chapter 1). Following this there was an investigation into system development and the current works of BPR (chapter 2 and chapter 3). This included a survey and discussion of two main problems met during the BPR projects: the role of information systems in BPR and the human factors affecting the success of BPR. The former discussion linked the relationship between BPR and system development; whereas the latter discussion concluded the significance of user participation in the process of BPR as well as system development. The principles and concepts of Empirical Modelling were introduced (chapter 4) and a comparison between this approach and object-orientation was also given (chapter 5). Through

this comparison, focusing on both their different development processes and the artefacts used, the characteristics of EM as a human-centred and situated computer-based approach was thus emphasised. Finally how EM might be used as an alternative approach to these fields and how this approach could contribute to the solution of problems discussed earlier were considered (chapter 6) and some case studies to illustrate these features were given following this discussion (chapter 7). The following subsections summarise the main work of this research.

System Development and BPR

System development has a close linkage with business process reengineering for several reasons. Not only is one tenet of BPR to exploit IT to support radical change and to view IT as the central enabler of BPR, but there are also many similarities between the process of system development and the process of BPR[1]. But the most significant reason is the impact of information systems upon organisations. As we have concluded in chapter 3, information systems are part of the business environment (the social systems) and their design and use cannot be specified solely in just technical terms. The lack of this consideration has contributed to the high rates of failure for BPR projects. Also it was found in this research that information systems developed in conventional paradigms were unable to implement the newly reengineered processes of BPR or even the existing business processes. The latter has raised the problem of legacy systems which has been increasedly addressed in both academia and business.

Another common problem faced by both system development and BPR is the role of human factors in their processes. BPR fails partly because the approaches put much emphasis on the scale of change (through IT) but fail to consider such change in relation to people or to consider other complex issues such as cultural issues. That is, these BPR projects did not provide the channel for the relevant people (employees, end-users, etc) to participate in the process of BPR. Conventional system development has also faced similar problems. The process of conventional system development was context-free, i.e. independent of the real world environment in which the software system is intended to operate. It was

1. For example, OOSE by Jacobson et al. (1992) and OOBE by Jacobson et al. (1995); or many other object-oriented methodologies for software engineering and for business engineering.

also abstract and phase-based and started from the informal step of eliciting user's requirements. It assumed that users can explicitly describe their requirement. But even if users can describe what they are currently doing, they still cannot describe fully or visualise how they might use the system or how their task might be reengineered. Under this paradigm the users who are (or will be) using the developed systems do not participate in the development process and are not even allowed to change the system in response to their evolving requirements. The systems developed in this manner are not appropriate to the rapidly changing and radically competitive real world, and consequently the organisation will face the legacy system problem mentioned earlier.

The aim of this research has been to find an innovative way to rethink these problems and put them in a different prospective through modelling processes. It has been suggested in this work that EM is an appropriate candidate due to the character of its situated and computer-based models (ISMs). Through the networked computer-based models, the users and developers are enabled to interact with each other in an open-ended manner. This interpersonal interaction supported by EM makes the model reflect closely to the real-world environment and enables the modeller (and all the participants) to take the systems approach to analysing the proposed system and its impact on the real world (the organisation), compared to the conventional methods, like a reductionist approach, which emphasises the individual component behaviours but not the interactions among them.

Review of Empirical Modelling

Empirical Modelling is a new and radically different approach to complex systems design and business modelling. The primary focus of EM is on the comprehension and on the use of computer-based interactive situation models (ISMs) that represent the way in which the aspects of systems behaviour are constructed in terms of agencies, observables and dependencies. On this EM view, computer-based models of business processes can be built in a way similar to that in which human beings make conceptual models of such processes. We can then specialise and circumscribe our models to derive software systems. And in this way EM can offer both cognitive and operational support to BPR from the very early and conceptual stages of modelling (Chen et al., 2000a).

To sum up, EM serves as an open development approach with the following characteristics:

- **Situatedness** A model built using EM is situated because it is the model of the relationship between a situation and the observer. The modeller can directly experience the results from his introduction of changes to the model. Thus the ISM can reflect a change in the real-world situation or new knowledge gained by the modeller. We have described in chapter 6 the roles of ISMs in four main fields most relevant to BPR: for HCI, the ISMs can be used for interface construction; for the development of processes from the ISMs, an observation-oriented analysis and an associated simulation of behaviour can be done along the construction of the model; for requirements engineering, the ISMs serve as a prototype which help to understanding the current problems and visualise the reality of the future system; for decision making the ISMs can be used to explore a set of alternatives for the problems.
- **Computer-Based Artefacts** In EM the computer is used as an interactive and open-ended artefact for facilitating knowledge construction by situated modelling. This is different to the conventional use of the computer as just an application tool for knowledge representation. In EM the computer is used to generate the metaphorical representation of particular states. It helps not only knowledge representation, but also knowledge construction, to enrich the modeller's knowledge. With the aid of networked communication, all the participants in the modelling process can interact with each other in a visible and communicable manner through the synchronisation between the evolution of computer models and individual participant's insights.
- **The Unified Procedure of Development** One advantage of EM is that it makes it possible to take account of the context not only in the construction stage, but also in the operational stage of the system development (Sun, 1999). As illustrated in Figures 4.2, 5.3 and the SPORE framework in Figure 6.1, the knowledge captured by the modeller during the modelling process can be coupled with his existing knowledge of both the external subjects and embedded in the computer-based model, which in turn can be used as the basis for further interaction.

Participative Process Modelling and Participative BPR

We have emphasised in chapter 3 and chapter 6 that BPR will not be successful without the support and active participation of its people, and concluded that BPR should be people-centred. In chapter 6 we proposed two concepts to emphasise the importance of people participation in the modelling process: *participative process modelling* and *participative BPR*. The SPORE framework, and the EM approach proposed in section 6.4 to system development and BPR, were all centred around this 'tenet'. Through the user participation in the modelling process, we can take the holistic view of the cross-functional interactions and processes in the real-world context rather than the piecemeal engineering of isolated parts of a system (either computer system or business system). The main potential advantage of EM is that it provides an ideal environment for participants to interact with each other in a flexible and open-ended manner. This character and flexibility of ISMs encourage a different kind of relationship between human modellers and the automated business activity. And the potential benefits of introducing the SPORE framework here are its flexibility, openness and the richness of interaction possible between many participants in the modelling environment. This interpersonal interaction within the distributed EM environment can model the agencies in two levels: (1) the modeller as an *external* observer can shape the agency in the context of his role in the task, such as developers or users (for this we referred to 'participative modelling of processes' in chapter 6); or (2) the modeller as an *internal* observer can act as an agent to carry out the interaction between agents through pretend play ('modelling of participative process'). This being-participant-observer approach (as named by Sun, 1999) under the EM framework enables the participants to shape the agency within the system in their customary context rather than the modeller's context. And participants' interaction with, as well as the construction of, ISMs enable a mode of operation which is loosely tied to a routine process, and encourages a creative, opportunistic, situated (re-)thinking and problem-solving activity.

8.2 Summary of Contributions

The principal contribution and focus of this work is the introduction of Empirical Modelling for integrating the development of software systems and business process reengineering, and promoting human participation in both processes. The following subsections outline the primary contributions of this thesis and the limitations of this research.

Primary Contributions

The primary contributions of this thesis can be summarised as follows:

- The framework of applying EM for software system development has been proposed by previous researchers in this group (that is, Ness, 1997 and Sun, 1999). This thesis extends their work by investigating the potential of EM as an approach for linking the development of software systems with BPR in which such systems are playing the role of an enabler for BPR. For example the SPORE framework, which was proposed for the situated modelling of cultivating system requirements, is extended to be a framework for the modelling of business situations. As described in chapter 6, many existing BPR analyses do not address the implementation issues of both BPR and its supporting systems. The EM approach, which combines both the implementation and other activities in system development, can address such issues by integrating the analysis and implementation of the supporting system for BPR.
- This thesis introduces the concepts 'participative process modelling' and 'participative BPR', and proposes two levels of modelling for integrating both contextual modelling and system development (cf. section 6.4). Through the distributed EM tools, different participants, either in the business area (such as managers, personnel or BPR analysts), or in the technical area (such as system designers or programmers), should be involved in the modelling process and communicate with each other by interacting with ISMs.
- The investigation of the comparison between EM, object-orientation and use-case approach for system development. This comparison clarifies some concepts which may seem similar in each

approach but their ontological status may be fundamentally different. Clarifying such differences will give a direction for further work on investigating the potential of EM in both system development and business modelling which will be pointed out in the next section.

Limitations of the Research

During the research, this author has met some difficulties and several limitations. Such limitations may partly result from the scope of this research, and partly from limited time and resources available.

- The aim of this research has been to introduce EM as the approach for different groups of people (who are directly or even indirectly relevant) to be involved in the process of system development or business modelling. However there are some social issues, such as user satisfaction or worker resistance, which cannot be fully addressed in our approach. This is what Checkland (1993) emphasises that social phenomena are too complex to model especially when we are a part of them. Such social issues can only be highlighted through other (non-technical) methodologies which – as beyond the scope of this thesis – will be another research subject in other disciplines such as psychology or business.
- The human interaction is so complex that no methods can totally model or describe such behaviours (cf. SSM in (Checkland and Scholes, 1990) and 'ethnomethodology' in (Goguen, 1996)). The existing distributed EM tools (**`dtkeden`**) provides four modes which are obviously not sufficient for modelling of complex business situations. Further a more friendly user-interface design for existing EM tools, such as icon or window-based interface, could be helpful for users without technical background to be involved and thus participate in the process of modelling.
- Some concepts about object-orientation described in chapter 2, as well as the comparison between EM and OO made in chapter 5, result from the best knowledge of the author at the time of his research. However, as new concepts and new applications have been continuously added to the object-oriented methodologies[2], some problems and disadvantages mentioned may have been

2. For example the association 'uses' in the use case model described in (Jacobson et al., 1992) has been replaced by 'includes' in the new version of UML (UML 1.3 or newer).

modified and improved. Similarly in BPR, new concepts and new approaches may have been developed after this research. Further investigation and comparison between EM and other methods may be needed and will be helpful for clarifying the situations to which EM can be applied.

8.3 Further Work

There is clearly future work to be done on exploring the scalability of the EM approach and the derivation of other applications from the models. The research represented in this thesis has addressed some of the fundamental problems with system development and BPR, and these give the direction for the further work in these areas. This section provides an overview of some areas of future interest.

The first is the possible applications of EM in software system development. The work in this research was to investigate the development of software systems for the support of BPR. However for software system development itself (or software engineering), EM is potentially applicable to this research area and the relevant research connecting EM with software development is ongoing in this group. For example, the support by EM for situated problem-solving activity in requirements engineering, and reengineering the user interfaces through exploring statecharts (Beynon et al., 1999). Further work may include the detailed investigation of the difference between EM and other development methods (such as OO or Harel's statecharts), especially in their different modelling philosophy. The evaluation of computer-mediated interaction among participants through the distributed nature of EM models, instead of traditional face-to-face interaction, is another direction for investigation of EM in software development, especially to identify how participative process modelling can contribute in this area.

The second is the investigation of the linkage of different spheres relevant to BPR in which EM may make a contribution. For example, as pointed out in chapter 6, EM's emphasis on the process of model construction can address many of the significant issues raised by Warboys et al. (1999) such as the modelling of software and organisational processes or evolutionary design of software systems. Applied to human-computer interaction (HCI), EM has the potential for applications in scenario-based design which include system development, object-orientation and process modelling. EM may also have poten-

tial in decision support systems which can provide an environment for the learning of modellers through the modelling process, rather than merely providing several alternatives. EM can contribute in this area because decision-making in an organisation is always influenced by the context, which involves people with different perceptions and interests. One clue for investigating how the EM approach will be applied to these areas is through the concept of 'participative' modelling which was the core of this thesis.

In addition, the systems thinking and the evolutionary paradigm for system development are also reference points for further work of EM. Apart from system development or BPR, the further development of EM should be emphasised in taking the holistic view to include the environment of the applications developed. In this perspective, the open-ended and situated characteristics of EM show its potential for application in many different disciplines.

A major issue for the use of EM in serious system development is the problem of scalability for large, realistic applications. This has been identified as a key issue in previous theses (e.g. Ness, 1997 and Sun, 1999). This thesis has made more detailed proposals for a unified development approach (cf. subsection 5.3.1) and we have outlined in the final paragraph of subsection 6.4.3 one aspect of how a 'frozen' EM model may be optimised for local efficiency. It may be that there will be a trade-off between the extent of such local optimisations and the adaptability of the original EM model. Current models (e.g. the case studies of the digital watch and the warehouse in chapter 7) are of the order of several hundreds definitions. There is clear need for further work to explore the scalability and management issues with models using several thousand definitions.

APPENDIX A

LSD Account for the Warehouse

```
AGENT foreman(w) {
    STATE
    ORACLE
        rf_on_transport[4],                    /* Information in RF4, filled by Forklift Operator */
        rf_redistribution_confirm[2]
    HANDLE
        rf_warehouse[1...5],
        tf_foreman_name[1...5],
        rf_job_number[1...5],
        rf_item[1...5],
        rf_quantity[1...5],
        rf_from_place[1...5],
        rf_to_warehouse[1...5],
        rf_date[1...5]
    DERIVATE
    PROTOCOL
        *** Deciding the distribution ***
            --> write rf_warehouse[1...5]; write tf_foreman_name[1...5];
                write rf_job_number[1...5]; write rf_item[1...5];
                write rf_quantity[1...5]; write rf_from_place[1...5];
                write rf_to_warehouse[1...5]; write rf_date[1...5];
            --> *** Keep RF5 and pass RF1...4 to warehouse worker; ***
        (rf_on_transport[4] == 'Yes') || (rf_redistribution_confirm[2] == 'Done')
            --> *** Update quantity ***      /* After finishing loading/unloading, update quantity */
        (rf_redistribution_confirm[2] != 'Done')
            --> *** Pass RF2 back to office *** /* Redistribution error! */
}

AGENT warehouseWorker(w) {
    STATE
    ORACLE
        rf_item[1...4],
        rf_quantity[1...4],
        rf_from_place[1...4],
        rf_date[1...4],
        rf_item[2],
        rf_quantity[2],
        rf_date[2]
    HANDLE
        rf_moving_pending[1...4],
        rf_loading_time[1...4],
        rf_loading_platform[1...4],
        rf_redistribution_confirm[1...4],
        rf_to_place[2],
        rf_redistribution_confirm[2]
    DERIVATE
    PROTOCOL
        *** Receiving RF1...4 from foreman ***
```

```
        --> read rf_item[1...4]; read rf_quantity[1...4];
            read rf_from_place[1...4]; read rf_date[1...4];
            write rf_moving_pending[1...4];
        --> write rf_loading_time[1...4];
            write rf_loading_platform[1...4];
        --> *** Pass RF1,2,3 to office and RF4 to forklift operator; ***
                                        /* For loading */
    *** Item not enough ***
        --> write rf_redistribution_confirm[1...4] == ('Error' && 'Item not enough');
        --> *** Pass RF1...4 back to foreman; ***
    *** Receiving RF2 from office ***
            --> read rf_item[2]; read rf_quantity[2]; read rf_date[2];
            --> write rf_to_place[2];
            --> *** Pass RF2 to forklift operator; *** /* For unloading */
        *** All places full ***
            --> write rf_redistribution_confirm[2] == ('Error' && 'All places full');
            --> *** Pass RF2 back to foreman; ***
}

AGENT forkliftOperator(w) {
    STATE
    ORACLE
        rf_item[4],
        rf_quantity[4],
        rf_from_place[4],
        rf_date[4],
        rf_loading_time[4],
        rf_loading_platform[4],
        rf_item[2],
        rf_quantity[2],
        rf_date[2],
        rf_arrival_time[2],
        rf_unloading_platform[2]
    HANDLE
        rf_on_transport[4],
        rf_redistribution_confirm[4],
        rf_redistribution_confirm[2]
    DERIVATE
    PROTOCOL
        *** Receiving RF4 from warehouse worker ***
            --> read rf_item[4]; read rf_quantity[4];
                read rf_from_place[4]; read rf_date[4];
                read rf_loading_time[4]; read rf_loading_platform[4];
            --> *** Have the items ready when/where the truck is expected; ***
            --> write rf_on_transport[4] == 'Yes';/* After loading */
            --> *** Pass RF4 to foreman; *** /* Finsahing loading */
        *** Truck didn't come ***
            --> write rf_redistribution_confirm[4] == ('Error' && 'Unable to load items');
            --> *** Pass RF4 back to foreman; ***
    *** Receiving RF2 from warehouse worker ***
        --> read rf_item[2]; read rf_quantity[2];
            read rf_date[2]; read rf_arrival_time[2];
            read rf_unloading_platform[2];
        --> *** Move the items to new place; ***
        --> write rf_redistribution_confirm[2] == 'Done';
                                        /* After unloading */
        --> *** Pass RF2 to foreman; *** /* Finishing unloading */
```

```
        *** Truck didn't arrive ***
            --> write rf_redistribution_confirm[2] == ('Error' && 'Unable to unload items');
            --> *** Pass RF2 back to foreman; ***
}

AGENT office {
    STATE
    ORACLE
        rf_warehouse[1,2,3],
        rf_job_number[1,2,3],
        rf_to_warehouse[1,2,3],
        rf_date[1,2,3],
        rf_redistribution_confirm[2],
        tf_driver,
        tf_on_going,
        tf_from_warehouse,
        tf_loading_date,
        tf_to_warehouse,
        ttp_driver[1],
        ttp_date[1],
        ttp_from_warehouse[1],
        ttp_to_warehouse[1]
    HANDLE
        rf_driver[1,2,3],
        rf_arrival_time[1,2,3],
        rf_unloading_platform[1,2,3],
        rf_redistribution_confirm[1,2,3],
        ttp_on_going[1],
        tf_driver,
        tf_on_going,
        tf_from_warehouse,
        tf_loading_date,
        tf_to_warehouse
    DERIVATE
    PROTOCOL
        *** Receiving RF1,2,3 form warehouse worker ***
            --> read rf_warehouse[1,2,3]; read rf_job_number[1,2,3];
                read rf_to_warehouse[1,2,3]; read rf_date[1,2,3];
            --> *** According to TF decide which driver; ***
            --> write rf_driver[1,2,3]; write rf_arrival_time[1,2,3];
                write rf_unloading_platform[1,2,3];
            --> *** Keep RF1,2 and pass RF3 to the driver; ***
        *** Receiving TTP1 from driver ***
            --> read ttp_driver[1]; read ttp_date[1];
                read ttp_from_warehouse[1]; read ttp_to_warehouse[1];
                write ttp_on_going[1]; write tf_driver;
                write tf_on_going; write tf_from_warehouse;
                write tf_loading_date; write tf_to_warehouse;
            --> *** Pass RF2 to destination warehouse; ***
    *** If no truck available for that date ***
        --> write rf_redistribution_confirm[1,2,3] == ('Error' && 'No truck available');
        --> *** Pass RF1,2,3 back to foreman; ***
                                        /* Redistribution error! */
    *** (receive RF2 from destination foreman)
    rf_redistribution_confirm[2] != 'Done'
        --> *** Pass RF2 back to the original foreman; ***
                                        /* Redistribution error! */
```

```
}

AGENT driver(d) {
    STATE
    ORACLE
        rf_warehouse[3],
        rf_job_number[3],
        rf_to_warehouse[3],
        rf_date[3],
        rf_loading_time[3],
        rf_loading_platform[3],
        rf_driver[3],
        rf_arrival_time[3],
        rf_unloading_platform[3]
    HANDLE
        ttp_driver[1,2],
        ttp_date[1,2],
        ttp_job_number[1,2],
        ttp_from_warehouse[1,2],
        ttp_loading_platform[1,2],
        ttp_loading_time[1,2],
        ttp_to_warehouse[1,2],
        ttp_unloading_platform[1,2],
        ttp_arrival_time[1,2]
    DERIVATE
    PROTOCOL
        *** Receiving RF3 from office
            --> read rf_warehouse[3]; read rf_job_number[3];
                read rf_to_warehouse[3]; read rf_date[3];
                read rf_loading_time[3]; read rf_loading_platform[3];
                read rf_driver[3]; read rf_arrival_time[3];
                read rf_unloading_platform[3];
                write ttp_driver[1,2]; write ttp_date[1,2];
                write ttp_job_number[1,2]; write ttp_from_warehouse[1,2];
                write ttp_loading_platform[1,2];
                write ttp_loading_time[1,2]; write ttp_to_warehouse[1,2];
                write ttp_unloading_platform[1,2];
                write ttp_arrival_time[1,2];
            --> Keep TTP2 and pass TTP1 back to office; ***
        *** Time is 'up' --> Drive truck to the warehouse; ***
}

AGENT rf(n=1...5) {                          /* Redistribution Form */
    STATE
        rf_warehouse[n]=@,                   /* Information in RF1,2,3,4,5 */
        tf_foreman_name[n]=@,                /* Information in RF1,2,3,4,5 */
        rf_job_number[n]=@,                  /* Information in RF1,2,3,4,5 */
        rf_item[n]=@,                        /* Information in RF1,2,3,4,5 */
        rf_quantity[n]=@,                    /* Information in RF1,2,3,4,5 */
        rf_from_place[n]=@,                  /* Information in RF1,2,3,4,5 */
        rf_to_warehouse[n]=@,                /* Information in RF1,2,3,4,5 */
        rf_date[n]=@,                        /* Information in RF1,2,3,4,5 */
        rf_moving_pending[n]=@,              /* Information in RF1,2,3,4 */
        rf_loading_time[n]=@,                /* Information in RF1,2,3,4 */
        rf_loading_platform[n]=@,            /* Information in RF1,2,3,4 */
        rf_driver[n]=@,                      /* Information in RF1,2,3 */
        rf_arrival_time[n]=@,                /* Information in RF1,2,3 */
```

```
        rf_unloading_platform[n]=@,          /* Information in RF1,2,3 */
        rf_on_transport[n]=@,                /* Information in RF4 */
        rf_to_place[n]=@,                    /* Information in RF2 */
        rf_redistribution_confirm[n]=@       /* May appear in RF1,2,3,4 */
    ORACLE
    HANDLE
    DERIVATE
    PROTOCOL
}

AGENT ttp(t=1,2) {                           /* Truck Transportation Plan */
    STATE
        ttp_driver[t]=@,
        ttp_date[t]=@,
        ttp_job_number[t]=@,
        ttp_from_warehouse[t]=@,
        ttp_loading_platform[t]=@,
        ttp_loading_time[t]=@,
        ttp_to_warehouse[t]=@,
        ttp_unloading_platform[t]=@,
        ttp_arrival_time[t]=@,
        ttp_on_going[t]=@
    ORACLE
    HANDLE
    DERIVATE
    PROTOCOL
}

AGENT tf {                                   /* Transportation Form */
    STATE
        tf_driver=@,
        tf_on_going=@,
        tf_from_warehouse=@,
        tf_loading_date=@,
        tf_to_warehouse=@
    ORACLE
    HANDLE
    DERIVATE
    PROTOCOL
}
```

APPENDIX B

Details of the Agency in the Warehouse Process

Agent: Foreman

1. (Redistribution between warehouses) foremen decide the redistribution jobs for the specific week, with reference of Table G (items/places board), fill into Table A and then send to the office.
   - **Oracle:** place(G), item(G), quantity(G).
   - **Handle:** item(A), quantity(A), fromPlace(A), toWarehouse(A), finishByDate(A).
2. (Local redistribution) foremen decide the items to be moved within the warehouse, with reference of Table G (items/places board), fill into Table B and then send to the warehouse workers. [Note: in this stage the moving time and forklift in Table B is empty.]
   - **Oracle:** place(G), item(G), quantity(G).
   - **Handle:** item(B), quantity(B), fromPlace(B), toPlace(B), Date(B).

Agent: Warehouse Worker

1. (Redistribution between warehouses) after receiving Table C & D from office (via foreman), the warehouse workers decide the platform and forklift operator for each job, and which place to keep the items received, with reference of Table F (platform details), Table G (place details) and Table E (forklift operator daily schedule).
   - **Oracle:** item(C), quantity(C), loadingTime(C), item(D), quantity(D), unloadingTime(D), platform(F), time(F), forklift(F), time(E), place(G), item(G), quantity(G).
   - **Handle:** platform(C), platform(D), toPlace(D), forklift(B), forklift(C), forklift(D).
2. (Local redistribution) after receiving Table B from foreman, warehouse workers decide the forklift operator and moving time for each job with reference of Table E (forklift operator daily schedule).
   - **Oracle:** time(E).
   - **Handle:** movingTime(B).
3. Inform truck drivers the loading and unloading platforms.
4. Warehouse workers update Table E (forklift schedule) and Table F (platform details) according to Table B, C and D.
   - **Oracle:** platform(C), platform(D), forklift(F).

- **Dependency:** item(E) **is** item(B/C/D), quantity(E) **is** quantity(B/C/D), from(E) **is** fromPlace(B/C)/platform(D), to(E) **is** toPlace(B)/platform(C/D), time(E) **is** movingTime(B)/loading-Time(C)/unloadingTime(D), driver(E) **is** driver(C/D), item(F) **is** item(C/D), quantity(F) **is** quantity(C/D), fromToPlace(F) **is** fromPlace(C)/toPlace(D), time(F) **is** loadingTime(C)/unloadingTime(D), forklift(F) **is** forklift(B/C/D), driver(F) **is** driver(C/D).

5. Pass Table E to each forklift operator.
6. After receiving Table E from forklift operators, update Table G (place details).
   - **Oracle:** item(E), quantity(E), from(E), to(E), jobDone(E).
   - **Handle:** item(G), quantity(G).

Agent: Forklift Operator

1. Forklift operators do the redistribution jobs according to his daily schedule (Table E).
2. After finishing the redistribution jobs, forklift operators fill into jobDone in Table E and send it back to warehouse workers.
   - **Oracle:** jobKind(E), item(E), quantity(E), from(E), to(E), time(E).
   - **Handle:** jobDone(E).

Agent: Office

1. When received Table A from foremen, office will check Table H (truck schedule) and Table I (each driver daily schedule), then decide which driver and loading time for each job. Fill these information in Table I. [Note: in this stage the platform and forklift in Table C, the loading and unloading platforms in Table I are empty.]
   - **Oracle:** item(A), quantity(A), toWarehouse(A), finishByDate(A), driver(H), warehouses(H), time(H), item(I), quantity(I).
   - **Handle:** loadingTime(I).
   - **Dependency:** foreman(C, #) **is** foreman(A, #), item(C, #) **is** item(A, #), quantity(C, #) **is** quantity(A, #), fromPlace(C, #) **is** fromPlace(A, #), toWarehouse(C, #) **is** toWarehouse(A, #), fromWarehouse(I, #) **is** fromWarehouse(C, #), toWarehouse(I, #) **is** toWarehouse(C, #), item(I, #) **is** item(C, #), quantity(I, #) **is** quantity(C, #).
2. According to Table C, office produce Table D which shows all the redistribution jobs to the warehouse (for example Coventry) at the specified date.
   - **Oracle:** toWarehouses(C), date(C).
   - **Dependency:** item(D, #) **is** item(C, #), quantity(D, #) **is** quantity(C, #), fromWarehouse(D, #) **is** fromWarehouse(C, #), toWarehouse(I, #) **is** toWarehouse(C, #).
3. With reference of Table H and I, office decide the driver and unloading time for each job. [Note: at this stage the platform, toPlace and forklift in Table D are empty.]
   - **Oracle:** item(D), quantity(D), fromWarehouse(D), driver(H), warehouses(H), time(H), item(I), quantity(I).

- **Handle**: unloadingTime(I).
- **Dependency**: fromWarehouse(I, #) **is** fromWarehouse(D, #), toWarehouse(I, #) **is** toWarehouse(D, #), item(I, #) **is** item(D, #), quantity(I, #) **is** quantity(D, #).

4. Office fill/update Table C, D and H (according to Table I).
   - **Oracle**: jobNo(I).
   - **Dependency**: loadingtime(C, #) **is** loadingTime(I, #), unloadingtime(D, #) **is** unloadingTime(I, #), driver(C, #)/driver(D, #) **is** driver(I, #), warehouse(H, #) **is** fromWarehouse(I, #)/toWarehouse(I, #), time(H, #) **is** loadingTime(I, #)/unloadingTime(I, #).
5. After deciding the trucks, office will send Table C (without platform details) as well as Table D (unloading information) back to the warehouse. [Note: (1) the platforms in both Table C and Table D and toPlace in Table D are empty; (2) not all jobs in Table A will be allocated on 10 June 2000.]
6. Office send each truck driver Table I.

Agent: Truck Driver

Truck driver drives lorry to warehouses according to his daily schedule (Table I).

APPENDIX C

Paper-Based Tables/Forms Used During the Warehouse Process

Table A-1: One-Week Redistribution Jobs from Warehouse Coventry (sent by foreman)

Job #	Foreman	Items	Qty	From (P)	To Warehouse	Finished by
100001	*Bill*	*Screws*	*550*	*4*	*London*	*30 Nov 2001*
100002	*Susan*	*Bananas*	*1500*	*2*	*London*	*10 Nov 2001*
100003	*Bill*	*Oil Drums*	*200*	*1*	*Manchester*	*20 Nov 2001*
100004	*Susan*	*Computers*	*120*	*5*	*London*	*20 Nov 2001*
100005	*Susan*	*Bananas*	*3000*	*3*	*London*	*30 Nov 2001*

Table A-2: One-Week Redistribution Jobs from Warehouse London (sent by foreman)

Job #	Foreman	Items	Qty	From (P)	To Warehouse	Finished by
200001	*Nick*	*Bananas*	*3000*	*3*	*Coventry*	*10 Nov 2001*
200002	*Nick*	*Bananas*	*1500*	*2*	*Manchester*	*20 Nov 2001*
200003	*Buck*	*Oil Drums*	*700*	*5*	*Manchester*	*20 Nov 2001*
200004	*Buck*	*Screws*	*1000*	*4*	*Coventry*	*30 Nov 2001*

Table A-3: One-Week Redistribution Jobs from Warehouse Manchester

Job #	Foreman	Items	Qty	From (P)	To Warehouse	Finished by
300001	*Chris*	*Computers*	*30*	*1*	*Coventry*	*30 Nov 2001*
300002	*Alex*	*Oil Drums*	*950*	*3*	*London*	*20 Nov 2001*
300003	*Alex*	*Oil Drums*	*40*	*4*	*London*	*30 Nov 2001*
300004	*Alex*	*Screws*	*1000*	*5*	*Coventry*	*10 Nov 2001*

Table B: Local Redistribution Jobs within Warehouse Coventry

Job #	Foreman	Items	Qty	From (P)	To (P)	Moving Time	Forklift
000001	*Susan*	*Computers*	*60*	*4*	*5*	*09:30-10:30*	*Lucy*
000002	*Bill*	*Oil Drums*	*125*	*3*	*1*	*11:00-12:00*	*Tim*
000003	*Susan*	*Screws*	*1930*	*1*	*4*	*15:30-16:30*	*Tim*
000004	*Bill*	*Bananas*	*900*	*3*	*2*	*14:30-15:30*	*Lucy*
Within Warehouse: Coventry		*Date: 10 November 2001*					

Table C: Redistribution Jobs from Warehouse Coventry (10 November 2001)

Job #	Fore-man	Items	Qty	From (P)	To Ware-house	Loading Time	Plat-form	Driver	Fork-lift
100002	*Susan*	*Bananas*	*1500*	*2*	*London*	*09:00-10:00*	*B*	*Gale*	*Tim*
100003	*Bill*	*Oil Drums*	*200*	*1*	*Manchester*	*11:00-12:00*	*A*	*Adam*	*Lucy*
100005	*Susan*	*Bananas*	*3000*	*3*	*London*	*13:00-14:00*	*C*	*David*	*Tim*
From Warehouse: Coventry			*Date: 10 November 2001*						

Table D: Redistribution Jobs to Warehouse Coventry (10 November 2001)

Job #	Items	Qty	From Ware-house	Plat-form	Unloading Time	Driver	To (P)	Forklift
200001	*Bananas*	*3000*	*London*	*A*	*17:00-18:00*	*David*	*2*	*Tim*
300001	*Computers*	*30*	*Manchester*	*C*	*17:00-18:00*	*Gale*	*4*	*Lucy*
To Warehouse: Coventry		*Date: 10 November 2001*						

Table E: Forklift Operator (Lucy) Daily Schedule (10 November 2001)

Job #	Job Kind	Items	Qty	From	To	Time	Driver	Job Done
000001	*Moving*	*Computers*	*60*	*4*	*5*	*09:00-10:00*	***	
100003	*Loading*	*Oil Drums*	*200*	*1*	*A*	*11:00-12:00*	*Adam*	
000004	*Moving*	*Bananas*	*900*	*3*	*2*	*14:30-15:30*	***	
300001	*Unloading*	*Computers*	*30*	*C*	*4*	*17:00-18:00*	*Gale*	

Table F: Platform Schedule Details

Plat-form	Job #	L/UL	Items	Qty	From/To Place	Time	Forklift	Driver
A	*100003*	*Loading*	*Oil Drums*	*200*	*1*	*11:00-12:00*	*Lucy*	*Adam*
A	*200001*	*Unloading*	*Bananas*	*3000*	*2*	*17:00-18:00*	*Tim*	*David*
B	*100002*	*Loading*	*Bananas*	*1500*	*2*	*09:00-10:00*	*Tim*	*Gale*
C	*100005*	*Loading*	*Bananas*	*3000*	*3*	*13:00-14:00*	*Tim*	*David*
C	*300001*	*Unloading*	*Computers*	*30*	*4*	*17:00-18:00*	*Lucy*	*Gale*
Date: 10 November 2001								

Table G: Items/Place Details

Place	Item	Qty	Item	Qty	Item	Qty	Item	Qty
1	*Bananas*	*550*	*Computers*	*100*	*Oil Drums*	*1230*	*Screws*	*300*
2	*Bananas*	*2100*	*Computers*	*45*	*Oil Drums*	*0*	*Screws*	*3000*
3	*Bananas*	*450*	*Computers*	*92*	*Oil Drums*	*240*	*Screws*	*0*

Table H: Truck Schedule (10 November 2001)

Driver	Job #	Warehouses	Time	Job #	Warehouses	Time
Adam	*100003*	*Coventry–Manchester*	*11:00-14:00*	*300002*	*Manchester–London*	*15:00-18:00*
David	*100005*	*Coventry–London*	*13:00-15:00*	*200001*	*London–Coventry*	*16:00-18:00*
Gale	*100002*	*Coventry–London*	*09:00-12:00*	*200003*	*London–Manchester*	*12:30-15:00*
	300001	*Manchester–Coventry*	*15:30-18:00*			

Table I: Driver (Gale) Daily Schedule (10 November 2001)

Job #	From Ware-house	Loading Time	Load' P'form	To Ware-house	Unload-ing Time	UnLoad' P'form	Items	Qty
100002	*Coventry*	*09:00-10:00*	*B*	*London*	*11:00-12:00*	*A*	*Bananas*	*1500*
200003	*London*	*12:30-13:00*	*A*	*Manchester*	*14:00-15:00*	*B*		
300001	*Manchester*	*15:30-16:00*	*C*	*Coventry*	*17:00-18:00*	*C*	*Computers*	*30*

APPENDIX D

Scripts for the Warehouse System

Script A: The Paper-Based Model for the Warehouse System

A.1 The DoNaLD and Scout Script for Figure 7.5 (in the Server Model)

```
%eden
OpenDisplay("door_screen", 600, 400);
proc display_screen : door_screen { DisplayScreen(&door_screen, "door_screen"); }

%scout
window warehousePict = {
   type: DONALD,
   pict: "WarehouseProcess",
   box: [{0, 0}, {600, 400}],
   xmin: 0, ymin: 400, xmax: 600, ymax: 0,
   bgcolor: "blue",
   fgcolor: "yellow",
   border: 1,
   sensitive: ON
};
display dispWarehouse;
dispWarehouse = < warehousePict >;
door_screen = dispWarehouse;

%donald
viewport WarehouseProcess
label wh1, fm1, ww1, fo1, off, dri
label wh2, fm2, ww2, fo2
line wh111, wh112, wh121, wh122
line 111, 112, 113, 121, 122, 123, 131, 132, 133, 141, 142, 143, 151, 152, 153
line 161, 162, 163, 171, 172, 173, 181, 182, 183, 191, 192, 193
label 114, 124, 134, 144, 154, 164, 174, 184, 194
real ptX, ptY

wh1 = label("Warehouse A",    {500,  50})
fm1 = label("(Foreman A)",    {500, 150})
ww1 = label("(W. Worker A)",  {500, 250})
fo1 = label("(Forklifter A)", {500, 350})
off = label("(Office)",       {300,  50})
dri = label("(Truck Driver)", {300, 250})
```

```
wh2 = label("Warehouse B",    {100,  50})
fm2 = label("(Foreman B)",    {100, 150})
ww2 = label("(W. Worker B)",  {100, 250})
fo2 = label("(Forklifter B)", {100, 350})

wh111 = [{450, 60}, {550, 60}]
wh112 = [{450, 62}, {550, 62}]
wh121 = [{ 50, 60}, {150, 60}]
wh122 = [{ 50, 62}, {150, 62}]

111 = [{550, 150}, {580, 150}]
112 = [{580, 150}, {580, 245}]
113 = [{580, 245}, {550, 245}]
114 = label("RF1,2,3,4", {560, 200})

121 = [{550, 255}, {580, 255}]
122 = [{580, 255}, {580, 350}]
123 = [{580, 350}, {550, 350}]
124 = label("RF4", {580, 300})

131 = [{450, 350}, {420, 350}]
132 = [{420, 350}, {420, 150}]
133 = [{420, 150}, {450, 150}]
134 = label("RF4", {420, 300})

141 = [{450, 250}, {400, 250}]
142 = [{400, 250}, {400,  45}]
143 = [{400,  45}, {340,  45}]
144 = label("RF1,2,3", {420, 120})

%eden
func changeWidth { para lineNo;
   execute("
   A_1"// lineNo //"1 = 'linewidth=5';\n
   A_1"// lineNo //"2 = 'linewidth=5';\n
   A_1"// lineNo //"3 = 'linewidth=5';\n
   ");
};

A_wh1 = "color=red";   A_fm1 = "color=green"; A_ww1 = "color=green";
A_fo1 = "color=green"; A_off = "color=white"; A_dri = "color=white";
A_wh2 = "color=red";   A_fm2 = "color=green"; A_ww2 = "color=green";
A_fo2 = "color=green";
A_wh111 = "color=white"; A_wh112 = "color=white";
A_wh121 = "color=white"; A_wh122 = "color=white";
A_114 = "color=gray"; A_124 = "color=gray"; A_134 = "color=gray";
A_144 = "color=gray"; A_154 = "color=gray"; A_164 = "color=gray";
A_174 = "color=gray"; A_184 = "color=gray"; A_194 = "color=gray";
```

A.2 Script for Redistribution Form (RF) shown in Figure 7.10

```
%scout
window f = {
   type: TEXT
   frame: ([{0, 0}, {500, 700}])
};
window RF1 = {
   type : TEXT
   frame: ([{20, 25}, {55, 35}])
   string: "RF 1"
   border: 1
   alignment: CENTRE
   sensitive: ON
   fgcolor: "blue"
   bgcolor: "yellow"
};
window RF2 = {
   type : TEXT
   frame: ([{65, 25}, {100, 35}])
   string: "RF 2"
   border: 1
   alignment: CENTRE
   sensitive: ON
   fgcolor: "blue"
   bgcolor: "yellow"
};
window RFfill = {
   type : TEXT
   frame: ([{20, 410}, {210, 422}])
   string: "Redistribution Forms Fill"
   border: 1
   alignment: CENTRE
   sensitive: ON
   fgcolor: "blue"
   bgcolor: "yellow"
};
window RFtitle = {
   type : TEXT
   frame: ([{40, 60}, {200, 70}])
   string: "REDISTRIBUTION FORM"
   alignment: CENTRE
   fgcolor: "yellow"
};
window RFcopy = {
   type : TEXT
   frame: ([{220, 60}, {255, 72}])
   string: "Copy:"
```

```
   alignment: LEFT
   fgcolor: "yellow"
};
window copyText = {
   type : TEXTBOX
   frame: ([{260, 60}, {5, 1}])
   fgcolor: "yellow"
};
window RFwarehouse = {
   type : TEXT
   frame: ([{25, 90}, {97, 102}])
   string: "Warwhouse:"
   alignment: LEFT
};
window warehouseText = {
   type : TEXTBOX
   frame: ([{100, 90}, {20, 1}])
   border: 1
   sensitive: ON
};

display screenRF = <RFtitle/RFcopy/copyText/RFwarehouse/warehouseText/RFforeman/
   foremanText/RFjobNo/jobNoText/RFitem/itemText/RFquantity/quantityText/RFfromPlace/
   fromPlaceText/RFtoWarwhouse/toWarehouseText/RFdate/dateText/RFforemanSignature/
   foremanSignatureText/RFsignatureDate/signatureDateText/RFredistribution/
   redistributionText/RFsubtitle1/RFsubtitle2/RFmovePending/movePendingText/RFloadTime/
   loadTimeText/RFloadPlatform/loadPlatformText/RFdriver/driverText/RFarrivalTime/
   arrivalTimeText/RFunloadPlatform/unloadPlatformText/RFsubtitle3/RFsubtitle4/
   RFonTransport/onTransportText/RFtoPlace/toPlaceText/RF1frame/RF2frame/RF3frame/
   RF4frame/RF/RFfill>;

%eden
m = RFv;
proc RF1_click { renewObs("mainRForm"); showRF(1); m = RFv1; };
proc RF2_click { renewObs("mainRForm"); showRF(2); m = RFv2; };
proc RF3_click { renewObs("mainRForm"); showRF(3); m = RFv3; };
proc RF4_click { renewObs("mainRForm"); showRF(4); m = RFv4; };
proc RF5_click { renewObs("mainRForm"); showRF(5); m = RFv5; };

proc RFfill_click {
   auto i;
   for (i = 1; i <= m#; i++) {
      sendServer("","
      mainRForm[" //str(m[i])// "][1] =\"" //str(warehouseText_getText())// "\";\n
      mainRForm[" //str(m[i])// "][2] =\"" //str(foremanText_getText())// "\";\n
      mainRForm[" //str(m[i])// "][3] =\"" //str(jobNoText_getText())// "\";\n
      mainRForm[" //str(m[i])// "][4] =\"" //str(itemText_getText())// "\";\n
      mainRForm[" //str(m[i])// "][5] =\"" //str(quantityText_getText())// "\";\n
```

```
mainRForm[" //str(m[i])// "][6] =\"" //str(fromPlaceText_getText())// "\";\n
mainRForm[" //str(m[i])// "][7] =\"" //str(toWarehouseText_getText())// "\";\n
mainRForm[" //str(m[i])// "][8] =\"" //str(dateText_getText())// "\";\n
mainRForm[" //str(m[i])// "][9] =\"" //str(foremanSignatureText_getText())//"\";\n
mainRForm[" //str(m[i])// "][10]=\"" //str(signatureDateText_getText())// "\";\n
mainRForm[" //str(m[i])// "][11]=\"" //str(redistributionText_getText())// "\";\n
mainRForm[" //str(m[i])// "][12]=\"" //str(movePendingText_getText())// "\";\n
mainRForm[" //str(m[i])// "][13]=\"" //str(loadTimeText_getText())// "\";\n
mainRForm[" //str(m[i])// "][14]=\"" //str(loadPlatformText_getText())// "\";\n
   mainRForm[" //str(m[i])// "][15]=\"" //str(driverText_getText())// "\";\n
   mainRForm[" //str(m[i])// "][16]=\"" //str(arrivalTimeText_getText())// "\";\n
   mainRForm[" //str(m[i])// "][17]=\"" //str(unloadPlatformText_getText())// "\";\n
   mainRForm[" //str(m[i])// "][18]=\"" //str(onTransportText_getText())// "\";\n
   mainRForm[" //str(m[i])// "][19]=\"" //str(toPlaceText_getText())// "\";\n
   ");
};
sendServer("","
for (k = 1; k <= 5; k++) {\n
   for (l = 1; l <= 19; l++) {\n
      if (mainRForm[k][l] != '@') {} else {mainRForm[k][l]='';};
};\n };\n");
};

func showRF { para copyNo;
RForm is mainRForm;
copyText_setText("* "//str(copyNo)//" *");
warehouseText_setText(RForm[copyNo][1]);
foremanText_setText(RForm[copyNo][2]);
jobNoText_setText(RForm[copyNo][3]);
itemText_setText(RForm[copyNo][4]);
quantityText_setText(RForm[copyNo][5]);
fromPlaceText_setText(RForm[copyNo][6]);
toWarehouseText_setText(RForm[copyNo][7]);
dateText_setText(RForm[copyNo][8]);
foremanSignatureText_setText(RForm[copyNo][9]);
signatureDateText_setText(RForm[copyNo][10]);
redistributionText_setText(RForm[copyNo][11]);
movePendingText_setText(RForm[copyNo][12]);
loadTimeText_setText(RForm[copyNo][13]);
loadPlatformText_setText(RForm[copyNo][14]);
   driverText_setText(RForm[copyNo][15]);
   arrivalTimeText_setText(RForm[copyNo][16]);
   unloadPlatformText_setText(RForm[copyNo][17]);
   onTransportText_setText(RForm[copyNo][18]);
   toPlaceText_setText(RForm[copyNo][19]);
};
```

A.3 Script for the Foreman Window (in the Client Model)

```
%eden
OpenDisplay("foremanA_screen", 1050, 500);
proc display_screen : foremanA_screen { DisplayScreen(&foremanA_screen,
   "foremanA_screen"); }
RFv = [1, 2, 3, 4, 5]; RFv1 = [1, 2, 3, 4, 5];
RFv2 = []; RFv3 = []; RFv4 = []; RFv5 = [];
renewObs("mainRForm");
renewObs("mainITableA");

%eden
include("/dcs/emp/cheny/PUBLIC/D-warehouse/RFscreen.e");
include("/dcs/emp/cheny/PUBLIC/D-warehouse/ITscreen.e");
include("/dcs/emp/cheny/PUBLIC/D-warehouse/ITscreenA.e");
include("/dcs/emp/cheny/PUBLIC/D-warehouse/whPicture.e");

%scout
window PassRF1234 = {
   type : TEXT
   frame: ([{20, 5}, {240, 15}])
   string: "Pass RF1...4 -> W. Worker A"
   border: 1
   alignment: CENTRE
   sensitive: ON
   fgcolor: "red"
   bgcolor: "white"
};

foremanA_screen = <RF1/RF2/RF3/RF4/RF5/PassRF1234/ITbotton> & screenRF & screenIT & <f> & dispWH;

%eden
proc PassRF1234_click : PassRF1234_mouseClick {
   include("/dcs/emp/cheny/PUBLIC/D-warehouse/foremanAPassClick.e");
};

         /*** The following is the script of 'foremanAPassClick.e' ***/
%eden
sendClient("foremanA"," %scout\n
   foremanA_screen = <RF5/ITbotton> & screenRF & screenIT & <f> & dispWH;\n
");

sendClient("workerA"," %scout\n
   workerA_screen = <RF1/RF2/RF3/RF4/PassRF123/PassRF4/ITbotton/PTbotton> &
   screenRF & screenIT & screenPT & screemWorkerA & dispWH;\n
");

sendServer("","changeWidth('1');");
```

Script B: The EDDI Table Defined in the Server Model

```
%eddi
AllTable is (Coventry + London + Manchester);
```

Script C: The Foreman (in Warehouse Coventry) Script

C.1 The EDDI Table

```
%eddi
Coventry (FromHouse char, JobNo int, Foreman char, Items char, Qty int, FromPlace int, ToWarehouse char,
    FinishedBy char, R_Date char, L_Pform char, L_Time_From char, L_Time_To char, L_Forklift char,
    Driver char, UL_Pform char, To_Place char, UL_Time_From char, UL_Time_To char, UL_Forklift char,
    JobDone char);
Coventry << ["Coventry",100001,"Bill","Screws",550,4,"London","30/11/01","","","","",
   "","","","","","","","",""];
Coventry << ["Coventry",100002,"Susan","Bananas",1500,2,"London","10/11/01","","","",
   "","","","","","","","","",""];
Coventry << ["Coventry",100003,"Bill","Oil Drums",200,1,"Manchester","20/11/01","","",
   "","","","","","","","","","",""];
Coventry << ["Coventry",100004,"Susan","Computers",120,5,"London","20/11/01","","","",
   "","","","","","","","","","",""];
```

C.2 The Window Interface and the Functions of Buttons Defined by EDEN

```
%eden
OpenDisplay("coventry_screen", 500, 500);
proc display_screen : coventry_screen { DisplayScreen(&coventry_screen, "coventry_screen"); }
include("/dcs/emp/cheny/PUBLIC/Warehouse-System/foremanWindow.e");
jobNumberC = 100005;

proc addButton_click : addButton_mouse_1 {
   if(addButton_mouse_1[2]==4) {
      changeData();
      execute(%eddi\n
         Coventry << [\"Coventry\","
         // str(jobNumberC) // ",\"" // c2 // "\",\"" // c3 // "\"," // c4 // ","
         // c5 // ",\"" // c6 // "\,\"" // c7
         // "\,\"\"",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\",\"\"];";
      jobNumberC++;
   };
};

proc viewData_click : viewData_mouse_1 {
   if(viewData_mouse_1[2]==4) {
      c8 = jobNoModifyText_getText();
```

```
for (i = 2; i <= (Coventry#); i++) {
   if (Coventry[i][2] == int(c8)) {
      j = i;
      jobNoText_setText(str(Coventry[j][2]));
      foremanText_setText(str(Coventry[j][3]));
      itemText_setText(str(Coventry[j][4]));
      quantityText_setText(str(Coventry[j][5]));
      fromPlaceText_setText(str(Coventry[j][6]));
      toWarehouseText_setText(str(Coventry[j][7]));
         dateText_setText(str(Coventry[j][8]));
      };
   };
};

proc modify_click : modify_mouse_1 {
   if(modify_mouse_1[2]==4) {
      c8 = jobNoModifyText_getText();
      for (i = 2; i <= (Coventry#); i++) {
         if (Coventry[i][2] == int(c8)) {
            j = str(i);
            changeData();
            Coventry[j][3] = c2;   Coventry[j][4] = c3;
            Coventry[j][5] = c4;   Coventry[j][6] = c5;
            Coventry[j][7] = c6;   Coventry[j][8] = c7;
            break;
         };
      };
   };
};

func changeData {
      c2 = foremanText_getText();     c3 = itemText_getText();
      c4 = quantityText_getText();    c5 = fromPlaceText_getText();
      c6 = toWarehouseText_getText(); c7 = dateText_getText();
};

proc cancel_click : cancel_mouse_1 {
   if(cancel_mouse_1[2] == 4) {
      jobNoText_setText(str(jobNumberC));
      clearTable();
   };
};

func clearTable {
   foremanText_setText("");      itemText_setText("");
   quantityText_setText("");     fromPlaceText_setText("");
   toWarehouseText_setText(""); dateText_setText("");
```

```
   jobNoModifyText_setText("");
};

proc viewTable1_click : viewTable1_mouse_1 {
   if(viewTable1_mouse_1[2]==4) {
      execute("%eddi\n
         ?Coventry %JobNo,Foreman,Items,Qty,FromPlace,ToWarehouse,FinishedBy;");
   };
};
```

```
proc viewTable2_click : viewTable2_mouse_1 {
   if(viewTable2_mouse_1[2]==4) {
      execute("%eddi\n  ?PlaceDetailsCoventry;");
   };
};
```

Script D: The Office Personnel Script

D.1 The EDDI Tables

```
%eddi
FromCoventry is AllTable: FromHouse=='Coventry' %JobNo,FromHouse,ToWarehouse,
   FinishedBy,R_Date,Driver,L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;
FromLondon is AllTable: FromHouse=='London' %JobNo,FromHouse,ToWarehouse,
   FinishedBy,R_Date,Driver,L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;
ToCoventry is AllTable: ToWarehouse=='Coventry' %JobNo,FromHouse,ToWarehouse,
   FinishedBy,R_Date,Driver,L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;
ToLondon is AllTable: ToWarehouse=='London' %JobNo,FromHouse,ToWarehouse,
   FinishedBy,R_Date,Driver,L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;
Adam is AllTable: Driver=="Adam" %JobNo,FromHouse,ToWarehouse,R_Date,Driver,
   L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;
David is AllTable: Driver=="David" %JobNo,FromHouse,ToWarehouse,R_Date,Driver,
   L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;
Gale is AllTable: Driver=="Gale" %JobNo,FromHouse,ToWarehouse,R_Date,Driver,
   L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;
```

D.2 The Window Interface and the Functions of Buttons Defined by EDEN

```
%eden
OpenDisplay("office_screen", 500, 500);
proc display_screen : office_screen { DisplayScreen(&office_screen, "office_screen"); }
include("/dcs/emp/cheny/PUBLIC/Warehouse-System/officeWindow.e");

proc viewData_click : viewData_mouse_1 {
   if(viewData_mouse_1[2]==4) {
      renewObs("AllTable");
```

```
c8 = jobNoModifyText_getText();
for (i = 2; i <= (AllTable#); i++) {
   if (AllTable[i][2] == int(c8)) {
      j = i;
      jobNoText_setText(str(AllTable[j][2]));
      foremanText_setText(str(AllTable[j][3]));
      fromPlaceText_setText(str(AllTableAllTable[j][1]));
      itemText_setText(str(AllTable[j][4]));
      quantityText_setText(str(AllTable[j][5]));
         toWarehouseText_setText(str(AllTable[j][7]));
         dateText_setText(str(AllTable[j][8]));
         OfficeRDateText_setText(str(AllTable[j][9]));
         OfficeLTFromText_setText(str(AllTable[j][11]));
         OfficeLTToText_setText(str(AllTable[j][12]));
         unloadTFromText_setText(str(AllTable[j][17]));
         unloadTToText_setText(str(AllTable[j][18]));
         driverText_setText(str(AllTable[j][14]));
      };
   };
};

proc modify_click : modify_mouse_1 {
   if(modify_mouse_1[2]==4) {
c8 = jobNoModifyText_getText();
for (i = 2; i <= (AllTable#); i++) {
   if (AllTable[i][2] == int(c8)) {
      j = str(i);
      changeData();
      sendServer("","
         AllTable[" // j // "][9]  = \"" // str(c2) // "\";\n
         AllTable[" // j // "][11] = \"" // str(c3) // "\";\n
         AllTable[" // j // "][12] = \"" // str(c4) // "\";\n
         AllTable[" // j // "][17] = \"" // str(c5) // "\";\n
         AllTable[" // j // "][18] = \"" // str(c6) // "\";\n
         AllTable[" // j // "][14] = \"" // str(c7) // "\";
      ");
      break;
   };
   };
};

func changeData {
      c2 = OfficeRDateText_getText();  c3 = OfficeLTFromText_getText();
      c4 = OfficeLTToText_getText();   c5 = unloadTFromText_getText();
      c6 = unloadTToText_getText();    c7 = driverText_getText();
};
```

```
proc clear_click : clear_mouse_1 {
   if (clear_mouse_1[2]==4) {
      OfficeRDateText_setText("");    OfficeLTFromText_setText("");
      OfficeLTToText_setText("");     unloadTFromText_setText("");
      unloadTToText_setText("");      driverText_setText("");
   };
};
```

```
proc viewCoventry1_click : viewCoventry1_mouse_1 {
   if(viewCoventry1_mouse_1[2]==4) {
      execute("%eddi\n  ?FromCoventry");
   };
};

proc viewLondon1_click : viewLondon1_mouse_1 {
   if(viewLondon1_mouse_1[2]==4) {
      execute("%eddi\n  ?FromLondon");
   };
};

proc viewCoventry2_click : viewCoventry2_mouse_1 {
   if(viewCoventry2_mouse_1[2]==4) {
      execute("%eddi\n  ?ToCoventry");
   };
};

proc viewLondon2_click : viewLondon2_mouse_1 {
   if(viewLondon2_mouse_1[2]==4) {
      execute("%eddi\n  ?ToLondon");
   };
};

proc go1_click : go1_mouse_1 {
   if(go1_mouse_1[2]==4) {
      renewObs("AllTable");
      c8 = finishByInViewText_getText();
      execute("%eddi\n  ?AllTable:FinishedBy==\"" // str(c8) // "\"%JobNo,FromHouse,
         ToWarehouse,FinishedBy,R_Date,Driver,L_Time_From,L_Time_To, UL_Time_From,
         UL_Time_To;");
   };
};

proc viewAdam1_click : viewAdam1_mouse_1 {
   if(viewAdam1_mouse_1[2]==4) {
      execute("%eddi\n  ?Adam");
   };
};
```

```
proc go2_click : go2_mouse_1 {
   if(go2_mouse_1[2]==4) {
      renewObs('AllTable');
      c8 = viewDriverInViewText_getText();
      execute('%eddi\n  ?AllTable:R_Date==\"' // str(c8) // '\"%JobNo,R_Date,Driver,
         FromHouse,ToWarehouse,L_Time_From,L_Time_To, UL_Time_From, UL_Time_To;');
   };
};
```

Script E: The Warehouse Worker (in Warehouse Coventry) Script

E.1 The EDDI Tables

```
%eddi
WorkerInCoventry is ((AllTable: FromHouse == "Coventry")
   + (AllTable: ToWarehouse == "Coventry")) - (AllTable: R_Date == "");
PlatformCoventry is ((AllTable: FromHouse == "Coventry")
   + (AllTable: ToWarehouse == "Coventry"))
   - (AllTable: R_Date == "").((AllTable: L_Pform != "")+(AllTable: UL_Pform != ""));
Lucy is (AllTable: L_Forklift == "Lucy") + (AllTable: UL_Forklift == "Lucy");
Tim is (AllTable: L_Forklift == "Tim") +(AllTable: UL_Forklift == "Tim");
```

E.2 The Window Interface and Functions of Buttons Defined by EDEN

```
%eden
OpenDisplay("coventryWorker_screen", 500, 500);
proc display_screen : coventryWorker_screen { DisplayScreen(&coventryWorker_screen,
    "coventryWorker_screen"); }
include("/dcs/emp/cheny/PUBLIC/Warehouse-System/workerWindow.e");

proc viewData_click : viewData_mouse_1 {
   if (viewData_mouse_1[2]==4) {
      c8 = jobNoModifyText_getText();
      for (i = 2; i <= (WorkerInCoventry#); i++) {
         if (WorkerInCoventry[i][2] == int(c8)) {
            j = i;
            jobNoText_setText(str(WorkerInCoventry[j][2]));
            foremanText_setText(str(WorkerInCoventry[j][3]));
            itemText_setText(str(WorkerInCoventry[j][4]));
            quantityText_setText(str(WorkerInCoventry[j][5]));
            fromPlaceText_setText(str(WorkerInCoventry[j][1]));
            rDateText_setText(str(WorkerInCoventry[j][9]));
            LTFromText_setText(str(WorkerInCoventry[j][11]));
            LTToText_setText(str(WorkerInCoventry[j][12]));
            ULTFromText_setText(str(WorkerInCoventry[j][17]));
            ULTToText_setText(str(WorkerInCoventry[j][18]));
```

```
            loadPlatformText_setText(str(WorkerInCoventry[j][10]));
            loadForkliftText_setText(str(WorkerInCoventry[j][13]));
            unloadPlatformText_setText(str(WorkerInCoventry[j][15]));
            unloadForkliftText_setText(str(WorkerInCoventry[j][19]));
            toPlaceText_setText(str(WorkerInCoventry[j][16]));
         };
      };
   };
};
```

```
proc modify_click : modify_mouse_1 {
   if (modify_mouse_1[2]==4) {
      c8 = jobNoModifyText_getText();
      for (i = 2; i <= (AllTable#); i++) {
         if (AllTable[i][2] == int(c8)) {
            j = str(i);
            changeData();
            sendServer("","
               AllTable[" // j // "][10] = \"" // str(c2) // "\";\n
               AllTable[" // j // "][13] = \"" // str(c3) // "\";\n
               AllTable[" // j // "][15] = \"" // str(c4) // "\";\n
               AllTable[" // j // "][19] = \"" // str(c5) // "\";\n
               AllTable[" // j // "][16] = \"" // str(c6) // "\";
            ");
            break;
         };
      };
   };
};

func changeData {
      c2 = loadPlatformText_getText();     c3 = loadForkliftText_getText();
      c4 = unloadPlatformText_getText();   c5 = unloadForkliftText_getText();
      c6 = toPlaceText_getText();
};

proc cancel_click : cancel_mouse_1 {
   if(cancel_mouse_1[2] == 4) {
      loadPlatformText_setText("");     loadForkliftText_setText("");
      unloadPlatformText_setText("");   unloadForkliftText_setText("");
      toPlaceText_setText("");
   };
};

proc viewAll_click : viewAll_mouse_1 {
   if(viewAll_mouse_1[2]==4) {
      execute("%eddi\n  ?WorkerInCoventry:FromHouse == 'Coventry' %JobNo,R_Date,
         FromHouse,L_Time_From,L_Time_To;");
```

```
      execute("%eddi\n  ?WorkerInCoventry:ToWarehouse == 'Coventry'
         %JobNo,R_Date,ToWarehouse,UL_Time_From,UL_Time_To;");
   };
};

proc viewLucy1_click : viewLucy1_mouse_1 {
   if(viewLucy1_mouse_1[2]==4) {
      execute("%eddi\n  ?Lucy: L_Forklift == 'Lucy'
         %JobNo,R_Date,L_Forklift,L_Time_From,L_Time_To;");
      execute("%eddi\n  ?Lucy: UL_Forklift == 'Lucy'
         %JobNo,R_Date,UL_Forklift,UL_Time_From,UL_Time_To;");
   };
};

proc viewTim1_click : viewTim1_mouse_1 {
   if(viewTim1_mouse_1[2]==4) {
      execute("%eddi\n  ?Tim:L_Forklift == 'Tim'
         %JobNo,R_Date,L_Forklift,L_Time_From,L_Time_To;");
      execute("%eddi\n  ?Tim:UL_Forklift == 'Tim'
         %JobNo,R_Date,UL_Forklift,UL_Time_From,UL_Time_To;");
   };
};

proc go1_click : go1_mouse_1 {
   if(go1_mouse_1[2]==4) {
      c8 = byDateInViewText_getText();
      execute("%eddi\n  ?PlatformCoventry:R_Date==\"" // str(c8) // "\"%JobNo,R_Date,
         Driver,L_Time_From,L_Time_To,L_Pform,UL_Time_From,UL_Time_To,UL_Pform;");
   };
};

proc viewItems_click : viewItems_mouse_1 {
   if(viewItems_mouse_1[2]==4) {
      execute("%eddi\n  ?PlaceDetailsCoventry;");
   };
};
```

Bibliography

Ackoff, R. L. (1959). "Games, Decisions, and Organisations", in von Bertalanffy, L., Rapoport, A. (eds.) *General Systems*, Vol. 4, pp. 145-150

Adzhiev, V. D., Beynon, W. M., Cartwright, A. J., Yung, Y. P. (1994). "A Computational Model for Multiagent Interaction in Concurrent Engineering", in *Proceedings of the Second International Conference on Concurrent Engineering and Electronic Design Automation (CEEDA'94)*, Bournemouth University, United Kingdom, 7-8 April 1994, pp. 227-232

Arnott, D. R., O'Donnell, P. A. (1994). "Business Process Re-engineering and Decision Support Systems: A Cautionary Tale", in (Glasson et al., 1994), pp. 127-135

Baets, W. R. J. (1998). *Organisational Learning and Knowledge Technologies in a Dynamic Environment*. Boston: Kluwer Academic Publishers

Beynon, W. M. (1994). "Agent-oriented Modelling and the Explanation of Behaviour", Invited Paper in *Proceedings of the International Workshop -- Shape Modelling Parallelism, Interactivity and Applications*, Technical Report TR-94-1-040, Department of Computer Software, University of Aizu, Japan, September 1994, pp. 54-63

Beynon, W. M. (1997). "Empirical Modelling for Educational Technology", in *IEEE Proceedings of the Second International Conference on Cognitive Technology (CT'97)*, University of Aizu, Japan, 25-28 August 1997, pp. 54-68

Beynon, W. M. (1998). "Empirical Modelling and the Foundation of Artificial Intelligence", in Nehaniv, C. L. (ed.) *Proceedings of the International Workshop on Computation for Metaphors, Analogy and Agents*, University of Aizu, Japan, April 1998 (also in *Lecture Notes in Artificial Intelligence*, 1562, pp. 322-364)

Beynon, W. M., Cartwright, R. I. (1995). "Empirical Modelling Principles for Cognitive Artefacts", in *Proceedings of the IEE Colloquium: Design Systems with Users in Mind: The Role of Cognitive Artefacts*, December 1995, Digest No. 95/231, pp. 8/1-8/8

Beynon, W. M., Cartwright, R. I., Cartwright, A., Yung, Y. P. (1996). "Abstract Geometry for Design in an Empirical Modelling Context", Research Report CS-RR-319, Department of Computer Science, University of Warwick

Beynon, W. M., Cartwright, R. I., Rungrattanaubol, J., Sun, P. H. (1998). "Interactive Situation Models for Systems Development", Research Report CS-RR-353, Department of Computer Science, University of Warwick

Beynon, W. M., Cartwright, R. I., Sun, P. H., Ward., A. (1999). "Interactive Situation Models for Information Systems Development", in *Proceedings of the 3rd World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the 5th International Conference on Information Systems, Analysis and Synthesis (ISAS'99)*, Orlando, USA, 31 July – 4 August 1999, Volume 2, pp. 9-16

Beynon, W. M., Chen, Y. C., Hseu, H. W., Maad, S., Rasmequan, S., Roe, C., Rungrattanaubol, J., Russ, S. B., Ward, A., Wong, A. (2001). "The Computer as Instrument", in Beynon, W. M., Nehaniv, C. L., Dautenhahn, K. (eds.) *Cognitive Technology: Instruments of Mind – Proceedings of the Fourth International Conference on Cognitive Technology (CT-2001)*, LNAI 2117, University of Warwick, United Kingdom, 6-9 August 2001. Berlin: Springer-Verlag, pp. 476-489

Beynon, W. M., Joy, M. S. (1994). "Computer Programming for Noughts-and-Crosses: New Fountiers", in the *6th Annual Workshop Programme of Psychology of Programming Interest Group (PPIG'94)*, Open University, United Kingdom, 6-8 January 1994

Beynon, W. M., Rasmequan, S., Russ, S. B. (2000a). "The Use of Interactive Situation Models for the Development of Business Solutions", in *Conference Proceedings of the Workshop on Perspectives in Business Informatics Research (BIR-2000)*, Rostock, Germany, 31 March – 1 April 2000

Beynon, W. M., Rasmequan, S., Russ, S. B. (2000b). "An Experience-Based Approach to Decision Support Systems", Position Paper for *IFIP TC8 / Working Group 8.3, International Conference on Decision Support Through Knowledge Management*, Stockholm, Sweden, 9-11 July 2000

Beynon, W. M., Rasmequan, S., Russ, S. B. (2002). "A New Paradigm for Computer-Based Decision Support", to appear in a special issue of *Decision Support Systems: the International Journal*, Vol. 33, Issue 2, pp. 127-142

Beynon, W. M., Russ, S. B. (1994). "Empirical Modelling of Requirements", Research Report CS-RR-227, Department of Computer Science, University of Warwick

Beynon, W. M., Slade, M., Yung, Y. P. (1990). "Protocol Specification in Concurrent Systems Software Development", Research Report CS-RR-163, Department of Computer Science, University of Warwick

Beynon, W. M., Sun, P. H. (1998). "Interactive Situation Models for Program Comprehension", Research Report CS-RR-352, Department of Computer Science, University of Warwick

Beynon, W. M., Ward, A., Maad, S., Wong, A., Rasmequan, S., Russ, S. B. (2000c). "The Temposcope: a Computer Instrument for the Idealist Timetabler", in *Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling*, Constance, Germany, 16-18 August 2000

Boehm, B. W. (1976). "Software Engineering", in *IEEE Transactions on Computers*, Vol. C-25, No. 12, pp. 1226-1241

Boehm, B. W. (1981). *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall

Boulding, K. E. (1956). "General Systems Theory – The Skeleton of Science", in *Management Science*, Vol. 2, No. 3, pp. 197-208

Brödner, P. (1995). "The Two Cultures in Engineering", in Göranzon, B. (ed.) *Skill, Technology and Enlightenment: On Practical Philosophy*. London: Springer-Verlag, pp. 249-260

Brooks, F. P., Jr. (1987). "No Silver Bullet: Essence and Accidents of Software Engineering", in *IEEE Computer*, Vol. 20, No. 4, pp. 10-19; also in (Brooks, 1995), pp. 177-203

Brooks, F. P., Jr. (1995). *The Mythical Man-Month: Essays on Software Engineering*. Reading, Massachusetts: Addison-Wesley

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester, England: John Wiley &Sons

Butler, C. (1994). "The Role of I/T in Facilitating BPR: Observations from the Literature", in (Glasson et al., 1994), pp. 147-160

Campbell-Kelly, M., Aspray, W. (1996). *Computer: A History of the Information Machine*. New York: BasicBooks

Capra, F. (1996). *The Web of Life: A New Synthesis of Mind and Matter*. London: HarperCollins Publishers

Carroll, J. M. (ed.) (1995). *Scenario-Based Design: Envisioning Work and Technology in System Development*. New York: John Wiley & Sons

Cartwright, R. I. (1998). *Geometric Aspects of Empirical Modelling: Issues in Design and Implementation*, PhD Thesis, Department of Computer Science, University of Warwick

Checkland, P. (1993). *System Thinking, System Practice*. Chichester, England: John Wiley & Sons Ltd.

Checkland, P. (1999). *Soft Systems Methodology: A 30-Year Retrospective*. Chichester, England: John Wiley & Sons Ltd.

Checkland, P., Scholes, J. (1990). *Soft Systems Methodology in Action*. Chichester, England: John Wiley & Sons Ltd.

Chen, Y. C. (1996). *The Use, Abuse and Security of Internet Payment Systems*, MSc Dissertation, Department of Information Systems, The London School of Economics and Political Science, University of London

Chen, Y. C., Russ, S. B., Beynon, W. M. (2000a). "Empirical Modelling for Business Process Reengineering: An Experience-Based Approach", in *Conference Proceedings of the Workshop on Perspectives in Business Informatics Research (BIR-2000)*, Rostock, Germany, 31 March – 1 April 2000

Chen, Y. C., Russ, S. B., Beynon, W. M. (2000b). "Participative Process Modelling", in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC-2000)*, Sheraton Music City Hotel, Nashville, Tennessee, USA, 8-11 October 2000, WP-1.6-5

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks", in *Communication of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387

Codd, E. F. (1979). "Extending the Database Relational Model to Capture More Meaning", in *ACM Transactions on Database Systems*, Vol. 4, No. 4, December 1979, pp. 397-434

Codd, E. F. (1982). "Relational Database: A Practical Foundation for Productivity", in *Communications of the ACM*, Vol. 25, No. 2, pp. 109-117

Cockburn, A., Fowler, M. (1998). "Question Time! About Use Cases", in *Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '98)*, 18-22 October 1998, Vancouver, Canada, pp. 226-229

Cook, S. (1994). "Analysis, Design, Programming: What's the Difference?", in O'Callaghan, A. J., Leigh, M. (eds.) *Object Technology Transfer: Meeting the Training Needs of Software Development.* Henley on Thames. England: Alfred Waller Limited, pp. 55-63

Cook, S., Daniels, J. (1994). *Designing Object Systems: Object-Oriented Modelling with Syntropy.* Englewood Cliffs, NJ: Prentice Hall

Corrigan, S. (1996). "Human and Organisational Aspects of Business Process Reengineering", Research Report, Institute of Work Psychology, University of Sheffield

Coulson-Thomas, C. (ed.) (1994). *Business Process Re-engineering: Myth & Reality.* London: Kogan Page Limited

Crowe, M., Beeby, R., Gammack, J. (1996). *Constructing Systems and Information: A Process View.* London: The McGraw-Hill Companies

Dano, B., Briand, H., Barbier, F. (1996). "Progressing Towards Object-Oriented Requirements Specifications by Using the Use Case Concept", in *Proceedings of the IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, 11-15 March 1996, Friedrichshafen, Germany, pp. 450-456

Davenport, T. H. (1993). *Process Innovation: Re-engineering Work through Information Technology.* Boston: Harvard Business School Press

Davenport, T. H. (1996). "Why Re-engineering Failed: The Fad that Forgot People", in *Fast Company*, Premier Issue, pp. 70-74

Davenport, T. H., Short, J. E. (1990). "The New Industrial Engineering: Information Technology and Business Process Redesign", *Sloan Management Review*, Vol. 31, No. 4, Summer 1990, pp. 11-27

Davis, A. M., Hsia, P. (1994). "Giving Voice to Requirements Engineering", in *IEEE Software*, Vol. 9, No. 5, pp. 70-79

Evans, M., Beynon, W. M., Fischer, C. M. (2001). "Empirical Modelling for the Logistics of Rework in the Manufacturing Process", in *Proceeding of the 16th Brazilian Congress of Mechanical Engineering (COBEM)*, Uberlandia, Brazil, 26-30 November 2001, pp. 226-234

Farkas, M., Beynon, W. M., Yung, Y. P. (1993). "Agent-oriented Modelling for a Billiards Simulation", Research Report CS-RR-260, Department of Computer Science, University of Warwick

Fernandes, K., Raja, V., Keast, J., Beynon, W. M., Chan, P. S., Joy, M. (2000). "Business and IT Perspectives on AMORE: a Methodology using Object-Orientation in Re-engineering Enterprises", in Henderson, P. (ed.) *Systems Engineering for Business Process Change: New Directions*, Springer-Verlag, December 2001

Ferrie, J. (1995). "Business Processes – A Natural Approach", in *Proceedings of Forum 1 in ESRC Business Processes Resource Centre*, University of Warwick, 19 September 1995

Fichman, R. G., Kemerer, C. F. (1997). "Object Technology and Reuse: Lessons from Early Adapters", in *IEEE Computer*, Vol. 30, No. 10, October 1997, pp. 47-59

Financial Times (1994). "Giant with Feet of Clay: Tom Lloyd Offers a Contrasting View of Business Process Reengineering", in *The Financial Times*, 5 December 1994, page 8

Firesmith, D. G. (1998). "Use Cases: the Pros and Cons", Research Report, Knowledge Systems Corporation, Cary, North Carolina, USA

Fischer, C. N., Beynon, W. M. (2001). "Empirical Modelling of Products", in *Proceedings of the International Conference on Simulation and Multimedia in Engineering Education*, Phoenix, Arizona, USA, January 2001, pp. 20-26

Galliers, R. D. (1991). "A Scenario-Based Approach to Strategic Information Systems Planning", in Blackham, R. B., Flood, R. L., Jackson, M. C., Mansell, G. J., Probert, S. V. E. (eds.) *Systems Thinking in Europe*. New York: Kluwer Academic/Plenum Publishers, pp. 73-87

Galliers, R. D. (1992). "Soft Systems, Scenarios and the Planning and Development of Information Systems", in *Systemist*, Vol. 14, No. 3, August 1992, pp. 146-159; also in *Journal of Information Systems*, Vol. 3, No. 3, July 1993, pp. 199-213

Galliers, R. D. (1993). "Information Systems and Business Re-engineering", Warwick Business School Research Paper No. 89, May 1993, University of Warwick

Galliers, R. D. (1995). "Re-Orienting Information Systems Strategy: Integrating Information Systems into the Business", in (Stowell, 1995a), pp. 51-74

Galliers, R. D. (1998). "Reflections on BPR, IT and Organisational Change", in (Galliers and Baets, 1998), pp. 225-243

Galliers, R. D., Baets, W. R. J. (eds.) (1998). *Information Technology and Organisational Transformation*. Chichester, England: John Wiley and Sons Ltd.

Gammack, J. (1995). "Modelling Subjective Requirements Objectively", in (Stowell, 1995a), pp. 159-185

Gattorna, J. (ed.) (1990). *The Gower Handbook of Logistics and Distribution Management*, Fourth Edition. Aldershot, England: Gower Publishing

Gerrits, H. (1994). "Business Modelling Based on Logistics to Support Business Process Re-engineering", in (Glasson et al., 1994), pp. 279-288

Glasson, B. C., Hawryszkiewycz, I. T., Underwood, B. A., Weber, R. A. (eds.) (1994). *Business Process Re-engineering: Information Systems Opportunities and Challenges* (*Proceedings of the IFIP TC8 Open Conference on Business Process Re-Engineering: Information Systems Opportunities and Challenges*, Queensland Gold Coast, Australia, 8-11 May 1994). Amsterdam: Elsevier Science

Goguen, J. A. (1994). "Requirements Engineering as the Reconciliation of Technical and Social Issues", in Jirotka, M., Goguen, J. A. (eds.) *Requirements Engineering: Social and Technical Issues*, London: Academic Press, pp. 165-199

Goguen, J. A. (1996). "Formality and Informality in Requirements Engineering", in *Proceedings of the Second International Conference on Requirements Engineering (ICRE '96)*, Colorado Springs, Colorado, USA, 15-18 April 1996, pp. 102-108

Graham, I. (1994). *Object Oriented Methods*. Harlow, England: Addison-Wesley

Graham, I. (1996a). "Task Scripts, Use Cases and Scenarios in Object Oriented Analysis", in *Object Oriented Systems*, Vol. 3, No. 3, pp. 123-142

Graham, I. (1996b). "Some Problems with Use Cases... and How to Avoid Them", in Patel, D., Sun, Y., Patel, S. (eds.) *Proceedings of OOIS'96 – Object-Oriented Information Systems, Third International Conference*, December 1996, London, pp. 18-27

Hammer, M (1990). "Re-engineering Work: Don't Automate, Obliterate", *Harvard Business Review*, July-August 1990, pp.104-111

Hammer, M., Champy, J. (1993). *Reengineering the Corporation: A Manifesto for Business Revolution*. New York: Harper Business

Harel, D. (1992). "Biting the Silver Bullet: Toward a Brighter Future for System Development", in *IEEE Computer*, Vol. 25, No. 1, January 1992, pp. 8-20

Harrington, J. (1991). *Organisational Structure and Information Technology*. New York: Prentice-Hall

Holtham, C. (1994). "Business Process Re-engineering – Contrasting What It Is with What It Is Not", in (Coulson-Thomas, 1994), pp. 60-81

Humphries, M. W., Dy, M. C. (1996). "Bridging the Reengineering and IT Gap: Integrating BPR and System Development Methodologies", White Paper, Intranet Business Systems, Inc., Pasig City, The Philippines

Hutchins, E. (1995). *Cognition in the Wild*. Cambridge, Massachusetts: The MIT Press

Hutchison, A. (1994). "CSCW as Opportunity for Business Process Re-engineering", in (Glasson et al., 1994), pp. 309-318

Jackson, M. (1995). *Software Requirements & Specifications – a Lexicon of Practice, Principles and Prejudices*. Wokingham, England: Addison-Wesley Publishing Company

Jacobson, I. (1994). "Basic Use-Case Modelling (Continued)", in *Report on Object-Oriented Analysis and Design (ROAD)*, Vol. 1, No. 3, September/October 1994, p. 7

Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Harlow, England: Addison-Wesley

Jacobson, I., Ericsson, M., Jacobson, A. (1995). *The Object Advantage: Business Process Reengineering with Object Technology*. Wokingham, England: Addison-Wesley

Jacobson, I., Griss, M., Jonsson, P. (1997). *Software Reuse: Architecture Process and Organisation for Business Success*. Harlow, England: Addison-Wesley

Johnson, P., Johnson, H., Wilson, S. (1995). "Rapid Prototyping of User Interfaces Driven by Task Models", in (Carroll, 1995), pp. 209-246

Johnson, J. C., Wood, D. F. (1996). *Contemporary Logistics*. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Jordan, N. (1968). "Some Thinking about 'System'", Chapter 5 in *Themes in Speculative Psychology*. London: Tavistock Publishing

Kahen, G., Lehman, M. M. (2000). "A Brief Review of Feedback Dimensions in the Global Software Process", Position Paper in the *International Workshop on Feedback and Evolution in Software and Business Processes (FEAST 2000)*, 10-12 July 2000; also Technical Report 2000/15, Department of Computing, Imperial College, University of London

Kaindl, H. (1999). "Difficulties in the Transition from OO Analysis to Design", in *IEEE Software*, Vol. 16, No. 5, September/October 1999, pp. 94-102

Kawalek, P., Leonard, J. (1996). "Evolutionary Software Development to Support Organisational and Business Process Change: A Case Study Account", in *Journal of Information Technology*, Vol. 11, No. 3, pp. 185-198

Kutschker, M. (1995). "Re-engineering of Business Processes in Multinational Corporations", Working Paper No. 95-4, Katholische Universitat Eichstatt, Germany

Lehman, M. M. (1989). "Uncertainty in Computer Application and its Minimisation through the Engineering of Software", in *Software Maintenance Research and Practice*, Vol. 1, No. 1, pp. 3-27

Lehman, M. M. (1991). "Software Engineering, the Software Process and their Support", in *Software Engineering Journal*, Vol. 6, No. 5, pp. 243-258

Lehman, M. M. (1992). "Evolution of Processes and Process Models", in *IOPener, the Newsletter of the IOPT Club*. Bath: Praxis Systems Ltd., Vol. 1, No. 4, p. 4

Lehman, M. M. (1997). "Feedback in the Software Process", Position Paper in the *UK Software Engineering Association (SEA) Workshop: Research Directions in Software Engineering*, Imperial College, University of London, 14-15 April 1997

Lehman, M. M. (1998). "Feedback, Evolution and Software Technology – The Human Dimension", in *ICSE 20 Workshop on Human Dimension in Successful Software Development*, Kyoto, Japan, 20 April 1998

Lehman, M. M. (2000). "Rules and Tools for Software Evolution Planning and Management", Position Paper in the *International Workshop on Feedback and Evolution in Software and Business Processes (FEAST 2000)*, Imperial College, University of London, 10-12 July 2000

Lewis, P. J. (1993). "Towards an Interpretative Form of Data Analysis for the Soft Systems Methodology", in Stowell, F. A., West, D., Howell, J. G. (eds.) *Systems Science: Addressing Global Issue*. New York: Plenum, pp. 391-396

Lewis, P. J. (1995). "New Challenges and Directions for Data Analysis and Modelling", in (Stowell, 1995a), pp. 186-205

Loomes, M.J., Jones, S. (1998). "Requirements Engineering: A Perspective Through Theory-Building", in *Third IEEE International Conference on Requirements Engineering (ICRE'98)*, Colorado Springs, Colorado, USA, 6-10 April 1998, pp. 100-107

Loomes, M. J., Nehaniv, C. L. (2001). "Fact and Artifact: Reification and Drift in the History and Growth of Interactive Software Systems", in Beynon, W. M., Nehaniv, C. L., Dautenhahn, K. (eds.) *Cognitive Technology: Instruments of Mind – Proceedings of the Fourth International Conference on Cognitive Technology (CT-2001)*, LNAI 2117, University of Warwick, United Kingdom, 6-9 August 2001. Berlin: Springer-Verlag, pp. 25-39

Lundeberg, M. (1994). "Widening Our Views of Information Systems Development", in (Glasson et al., 1994), pp. 7-15

MacIntosh, R., Francis, A. (1997). "The Market, Technological and Industry Contexts of Business Process Re-Engineering in UK Businesses", Research Report, Department of Management Studies, Glasgow Business School

Magad, E. L., Amos, J. M. (1995). *Total Materials Management: Achieving Maximum Profits Through Materials/Logistics Operations*, Second Edition. New York: Chapman & Hall

McBreen, P. (1998). "Using Use Cases for Requirements Capture", Research Report, McBreen Consulting, New York

McGinnes, S. (1992). "How Objective is Object-Oriented Analysis?", Paper presented at *CAISE '92 – The Fourth Conference on Advanced Information Systems*, Manchester, United Kingdom, May 1992

Meester, P. C., Post, J. (1998). "LUCIA Accelerates Service Delivery: A Case Study of Business Process Re-engineering", in (Galliers and Baets, 1998), pp. 261-268

Meyer, B. (1990). "The New Culture of Software Development", in *Journal of Object-Oriented Programming (JOOP)*, Vol. 3, Issue 4, pp. 76-81

Mingers, J. (1995). "Using Soft Systems Methodology in the Design of Information Systems", in (Stowell, 1995a), pp. 18-50

Morris, D., Evans, G., Green, P., Theaker, C. (1996). *Object Oriented Computer Systems Engineering*. London: Springer-Verlag

Neo, B. S. (1994). "IT-enabled Business Re-engineering: A Study of Singapore's Trade Clearance Process", in (Glasson et al., 1994), pp. 433-442

Ness, P. E. (1997). *Creative Software Development: An Empirical Modelling Framework*, PhD Thesis, Department of Computer Science, University of Warwick

Norman, D. A. (1991). "Cognitive Artefacts", in Carroll, J. M. (ed.) *Designing Interaction: Psychology at the Human-Computer Interface*. Cambridge University Press

Objective Systems SF AB (1994). "Objectory", in Hutt, A. (ed.) *Object Analysis and Design: Description of Methods*. New York: Wiley-QED, pp. 107-117

O'Callaghan, A. J. (1994). "Object Technology Skills: Why, What and How", in O'Callaghan, A. J., Leigh, M. (eds.) *Object Technology Transfer: Meeting the Training Needs of Software Development*. Henley on Thames, England: Alfred Waller Limited, pp. 3-30

Odell, J. J. (1992). "Dynamic and Multiple Classification", in *Journal of Object Oriented Programming (JOOP)*, Vol. 4, Issue 8, pp. 45-48

Parets, J., Torres, J. C. (1996). "Software Maintenance versus Software Evolution: An Approach to Software Systems Evolution", in *Proceedings of the IEEE Symposium and Workshop on Engineering of Software-Based Systems*, 11-15 March 1996, Friedrichshafen, Germany, pp. 134-141

Parnas, D. L. (1996). "On the Criteria to Be Used in Decomposing Systems into Modules", in Laplante, P. (ed.) *Great Papers in Computer Science*. St. Paul, MN: West Publishing Company, pp. 433-441

Paul, R. J. (1993). "Dead Paradigms for Living Systems", in Whitley, E. A. (ed.) *First European Conference on Information Systems*, 29-30 March 1993. Henley, England: Operational Research Society, pp. 250-255

Peltu, M., Clegg, C., Sell, R. (1996). "Business Process Re-engineering: The Human Issues", in *Proceedings of Forum 4 in ESRC Business Processes Resource Centre*, University of Warwick, 30 April 1996

Pettigrew, A. M. (1985). *The Awakening Giant: Continuity and Change in ICI*. Oxford: Basil Blackwell

Pettigrew, A. M. (1998). "Success and Failure in Corporate Transformation Initiatives", in (Galliers and Baets, 1998), pp. 271-289

Pressman, R. S. (2000). *Software Engineering: a Practitioner's Approach*, Fifth Edition. London: McGraw-Hill

Quaddus, M. A. (1994). "An MCDM Based Interactive Support System with Application to Business Process Redesign", in (Glasson et al., 1994), pp. 493-502

Rasmequan, S., Roe, C., Russ, S. B. (2000). "Strategic Decision Support Systems: An Experience-Based Approach", in *Proceedings of the 18th International Association of Science and Technology*

for Development (IASTED) Conference on Applied Informatics, Innsbruck, Austria, 14-17 February 2000

Rasmequan, S., Russ, S. B. (2000). "Cognitive Artefacts for Decision Support", in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC-2000)*, Sheraton Music City Hotel, Nashville, Tennessee, USA, 8-11 October 2000

Rawles, S. (1997). "A Comparative Analysis of Object-Oriented and Empirically-Modelled Development", Third-Year Project Report, Department of Computer Science, University of Warwick

Regnell, B. (1999). *Requirements Engineering with Use Cases – A Basis for Software Development*, PhD Thesis, Department of Communication Systems, Lund Institute of Technology, Lund University, Sweden

Riemer, K. (1998). "A Process-Driven, Event-Based Business Object Model", in *Proceedings of Second International Workshop in Enterprise Distributed Object Computing (EDOC'98)*, 3-5 November 1998, La Jolla, California, USA, pp. 68-74

Robinson, S. (1994). "Implications of Business Process Re-engineering for the Management of Telework", in (Coulson-Thomas, 1994), pp. 127-141

Roe, C., Beynon, W. M., Fischer, C. N. (2001). "Empirical Modelling for the Conceptual Design and Use of Engineering Products", in *Proceedings of the International Conference on Simulation and Multimedia in Engineering Education*, Phoenix, Arizona, USA, January 2001, pp. 27-32

Rushton, A., Oxley, J., Croucher, P. (1989). *The Handbook of Logistics and Distribution Management*, Second Edition. London: Kogan Page Limited

Russ, S. B. (1997). "Empirical Modelling: The Computer as a Modelling Medium", in *BCS Computer Bulletin*, April 1997, pp. 20-22

Scarbrough, H., (1996). "The Relevance of Socio-Technical Theory to Business Process Re-design", Report of the Focus Group Meeting, Warwick Business School, 11 December 1996

Schader, M., Korthaus, A. (1998). "Modelling Business Processes as Part of the BOOSTER Approach to Business Object-Oriented System Development Based on UML", in *Proceedings of the Second International Enterprise Distributed Object Computing (EDOC'98) Workshop*, 3-5 November 1998, La Jolla, California, USA, pp. 56-67

Sherlock, J. (1994). *Principles of International Physical Distribution*. Oxford: Blackwell Publishers Ltd.

Sherwood-Smith, M. (1994). "People Centred Process Re-engineering: An Evaluation Perspective to Office System Re-design", in (Glasson et al., 1994), pp. 535-544

Simon, H. A. (1996). *The Sciences of the Artificial*, Third Edition. London: MIT Press

Simons, A. J. H. (1999). "Use Cases Considered Harmful", in Mitchell, R., Wills, A. C., Bosch, J., Meyer, B. (eds.) *Proceedings of the 29th Conference in Technology of Object-Oriented Languages and Systems (TOOLS/Europe-29 '99)*, Nancy, France, 7-10 June, 1999, pp. 194-203

Simsion, G. C. (1994). "A Methodology for Business Process Re-engineering", in (Glasson et al., 1994), pp. 61-69

Stewart, T. A. (1993). "Reengineering – The Hot New Managing Tool", in *Fortune*, 23 August 1993, pp. 33-37

Stowell, F. A. (ed.) (1995a). *Information Systems Provision – The Contribution of Soft Systems Methodology.* London: McFraw-Hill Book Company

Stowell, F. A. (1995b). "Empowering the Client: The Relevance of SSM and Interpretivism to Client-Led Design", in (Stowell, 1995a), pp. 118-139

Suchman, L. A. (1987). *Plans and Situated Actions: The Problem of Human-Machine Communication.* Cambridge: Cambridge University Press

Sully, P. (1993). *Modelling the World with Objects.* Englewood Cliffs, NJ: Prentice Hall

Sun, P. H. (1999). *Distributed Empirical Modelling and its Application to Software System Development*, PhD Thesis, Department of Computer Science, University of Warwick

Sun, P. H., Russ, S. B., Chen, Y. C., Beynon, W. M. (1999). "Cultivating Requirements in a Situated Requirements Engineering Process", Research Report CS-RR-357, Department of Computer Science, University of Warwick

Todd, S. J. P. (1976). "The Peterlee Relational Test Vehicle", in *IBM System Journal*, Vol. 15, No. 4, pp. 285-308

Turner, J. A. (1998). "The Role of Information Technology in Organisational Transformation", in (Galliers and Baets, 1998), pp. 245-260

Ulrich, W. M. (1995). "Business Process Redesign and the Legacy Systems Challenge", in *CrossTalk (The Journal of Defense Software Engineering)*, Vol. 8, No. 1, January 1995

van Meel, J. W., Bots, P. W. G., Sol, H. G. (1994). "Towards a Research Framework for Business Engineering", in (Glasson et al., 1994), pp. 581-592

Varela, F. J. (1979). *Principles of Biological Autonomy.* New York: North-Holland

Varela, F. J., Thompson, E., Rosch, E. (1991). *The Embodied Mind: Cognitive Science and Human Experience.* Cambridge, MA: MIT Press

Vidgen, R., Rose, J., Wood, B., Wood-Harper, T. (1994). "Business Process Reengineering: The Need for a Methodology to Revision the Organisation", in (Glasson et al., 1994), pp. 603-612

Vogel, D. (1994). "Re-engineering Towards the Meeting of the Future", in (Glasson et al., 1994), pp. 35-45

von Bertalanffy, L. (1968). *General System Theory: Foundations Development Applications.* New York: George Braziller

Warboys, B. (1994). "Reflections on the Relationship between BPR and Software Process Modelling", in *Lecture Notes in Computer Science*, Vol. 881, pp. 1-9

Warboys, B., Kawalek, P., Robertson, I., Greenwood, M. (1999). *Business Information Systems: A Process Approach.* Berkshire, England: McGraw-Hill Publishing

Winograd, T. (1995). "From Programming Environments to Environments for Designing", in *Communications of the ACM*, Vol. 38, No. 6, pp. 65-74

Winograd, T., Flores, F. (1987). *Understanding Computers and Cognition.* Reading, Massachusetts: Addison-Wesley Publishing Company

Yu, E. S. K., Mylopoulos, J (1994). "From E-R to 'A-R' – Modelling Strategic Actor Relationships for Business Process Reengineering", in Loucopoulos, P. (ed.) *Proceedings of 13th International Conference on the Entity-Relationship Approach (ER'94) – Business Modelling and Re-Engineering*, LNCS 881, Manchester, United Kingdom, December 1994. Berlin: Springer-Verlag, pp. 548-565