Towards efficacious groupware development: an Empirical Modelling approach

by

Zhan En Chan

A thesis submitted to

THE UNIVERSITY OF WARWICK

in partial fulfilment for the admission to the degree of

Doctor of Philosophy in Computer Science

Department of Computer Science

University of Warwick

July 2009

Table of Contents

List of Figures and Tables

Acknowledgement

I would like to thank my supervisor, Dr Meurig Beynon, who has been guiding me and sharing my emotions throughout my research journey at University of Warwick. I am indebted to him for substantial assistance in correcting language mistakes during the writing of this thesis. I am also indebted to Dr Meurig Beynon for his enthusiasm, patience, and invaluable help in all aspects during the writing of this thesis.

Special thanks goes to Dr Steve Russ and Dr Mike Joy, who have given me invaluable insight on early ideas in forming the structure of this thesis. I am also grateful to Prof Erkki Sutinen, who has given insightful comments about the overall orientation of this thesis and possible future research directions.

I would like to thank my parents who have been taking care of my poor health, and sharing my stress, depression, and joy since I was born. They are patient and encouraging. Without their support, I would never been able to realise this thesis.

I would also like to thank Dr Roger Packwood and the IT Services for their help in setting up equipment for the case studies. Lastly but not least, thank you to the present and the past members of the Empirical Modelling group who have been participating in the group seminars and provoked valuable ideas in relation to my research: Dr Charles Care, Dr Ashley Ward, Dr Antony Harfield, Dr Chris Roe, Russell Boyatt, Karl King, George Efstathiou, Nick Pope, Timothy Heron, Sunny Chang, Amanda Wright, Chris Brown.

Declaration

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 discussion of the process of co-evolution of the developer's understanding and the artifact in section 3.1 draws on the ideas presented in a paper (Beynon et al., 2008) published in the "The 20th Annual Psychology of Programming Interest Group Conference (PPIG 2008)". The discussion of the role-shifting phenomena discussed in section 3.3 draws on a presentation (Chan, 2006) in the "Warwick Postgraduate Colloquium in Computer Science (WPCCS '06)", and a workshop paper (Beynon and Chan, 2006) presented in the Distributed Participatory Design (DPD) workshop that was held in conjunction with NordiCHI 2006. The preliminary results from the case study on the cricket project in section 6.4 were presented in the "Warwick Postgraduate Colloquium in Computer Science (WPCCS '08)" (Chan, 2008). Part of the data used in the case study on the distributed jugs construction in section 6.2 was jointly collected by Antony Harfield and myself. The role-shifting phenomenon observed in section 6.2 and section 6.3 was briefly presented in the DPD 2006 workshop and in WPCCS '06. The discussion of the case study on the cricket project draws on the technical research report CS-RR-444 (Beynon and Chan, 2009).

The ideas behind practising an EM approach to collaborative modelling presented in this thesis were also inspired by the reworking of the Clayton Tunnel railway accident model, the related presentation in the "Warwick Postgraduate Colloquium in Computer Science (WPCCS '05)" (Chan, 2005) and the poster presented in the EU TEL Kaleidoscope Network of Excellence Showcase 2005 in Oberhausen, Germany (Harfield et al., 2005).

Abstract

Groupware development can be conceived as one particular branch of software systems development. Research into groupware development faces both methodological challenges as in classical software development, and socio-technical issues as identified in the CSCW literature. On the one hand, the paradigm needs to accommodate the changing context for use, facilitate the effective communication between developers and users, and maintain the conceptual integrity of the system. On the other hand, it has to deal with the dynamic nature of groups and the emerging work practices during the development process. As Grudin (1988) pointed out, the social and organisational aspects within the groupware development (and use) often lead groupware to fail. Individual differences make it unlikely that two groups are in reality identical. Consequently, groupware development should be seen and treated as an organic process, in which the groupware is grown by the owners with the aid of professional developers rather than constructed by professional developers alone. In this thesis, I propose a new conception of efficacious groupware development which draws on the ancient Chinese philosophical notion of shi (as interpreted by Jullien (1995)).

In this thesis, I argue that Empirical Modelling (EM) potentially offers a conceptual framework well-suited for efficacious groupware development. In the process, I propose a new conceptual framework for practising an EM approach in a groupware development context. EM is a body of principles and tools which embraces an experimental, interactive, and open approach towards systems development through the exploration of observation, dependency and agency. The proposed conceptual framework, known as GroupPIE, is based on the principles of EM. This is built upon previous research into EM, particularly Sun's Distributed Empirical Modelling (DEM). In contrast to DEM, this thesis focuses on the micro-level of collaborative modelling. In particular, it considers how EM might facilitate the collaboration amongst the modellers and interaction between the modellers and the evolving artifact which takes place in groupware development.

The thesis draws on various case studies from undergraduate projects and research projects which have practised an EM approach to collaborative modelling. The case studies suggest that the participants' knowledge of the situation co-evolves with the artifact under construction and that there is role-shifting behaviour through the collaborative modelling. Drawing on the case studies, this thesis argues that an EM approach to collaborative modelling potentially facilitates genuine participation. This challenges the accepted ideas about the role of participants (or actors) and the relationship between them in the groupware development process. It also suggests that EM potentially facilitates a notion of participatory development which is “more” human-centred. On this basis, I argue that EM is potentially better-suited for realizing the vision of efficacious groupware development.

Abbreviations

ADM	Abstract Data Machine (in the EM context)
AT	Activity Theory
CM	Collaborative modelling (cf. chapter 5)
CSCL	Computer Supported Collaborative Learning
CSCW	Computer Supported Cooperative Work
DCog	Distributed Cognition
DEM	Distributed Empirical Modelling
DPD	Distributed Participatory Design
EGD	Efficacious Groupware Development (cf. chapter 7)
EM	Empirical Modelling (cf. chapter 4)
FOSS	Free and Open Source Software
GD	Groupware development
HCD	Human-centred design
HCI	Human-Computer Interaction
IS	Information System
IT	Information technology
ICT	Interaction and Communication Technologies
LSD	LSD Notation (in the EM context)
ODA	Observables, dependencies, agents, and agencies (in the EM context)
PD	Participatory Design
UCD	User-centred design

Chapter 1 Introduction

Prior to postgraduate studies, I worked in a few organisations as a systems developer and a systems administrator[1]. As an on-site systems developer, like many other systems developers, my main duty was to write software for my organisation. But, interestingly, I occasionally found myself involved in other organisational operations which a ‘normal’ systems developer would not perform (e.g. promoting products in the public space). Equally, the users were sometimes more than just users – they could develop ‘simple programs’ for themselves. This is particularly true in small and medium sized enterprises (SMEs), where there tends to be a more flexible organisational structure.

Reflection on these personal experiences suggests some interesting questions. On the one hand, it could be argued that fulfilling non-technical roles is part of the systems developer’s duty. On the other hand, I may ask “what is the boundary of the systems developer’s duties?”, “can systems developers in reality only work as ‘developers’?” and “can users in reality only work as ‘users’?”.

Another interesting reflection from my industrial experience is that, despite the fact that there were off-the-shelf (and open source) alternatives, organisations that employed on-site developers tend to build and re-build software of similar kinds over and over again. The interesting question to ask is thus “why would the organisations have to reinvent the wheel over and over again?” One reason is that the organisation process is always situated and evolving at all times. It is unclear how an off-the-shelf component (or system) can be

[1] The difference between a systems developer and a systems administrator is that the former requires knowledge in software construction and the latter requires knowledge in computer hardware. As a systems developer, I mainly wrote software using generic programming languages such as Java and C, and as a systems administrator, I mainly monitored and maintained computer equipment (e.g. servers, routers) using little system-level programming with programming languages such as C and Perl.

customised to suit the organisational needs until the on-site developers and the *real* users get their hands on it. In SMEs, with limited resources to spend, evaluating off-the-shelf components could be a time-consuming activity carrying a degree of risk. This is partially due to the fact that seemingly well-suited off-the-shelf components might prove to be unsuitable after the evaluation, so that the evaluation effort would be wasted, and partially due to the fact that there is no guarantee that off-the-shelf components can be customised to fit new requirements[2]. An economic factor, namely, the cost-to-develop vs. the cost-to-buy-and-customise, could be relevant in this situation. In labour-intensive countries, e.g. China, the labour cost to develop software from scratch is most likely cheaper than the cost of buying an off-the-shelf component and customising it. These considerations all suggest that, to some extent at least, systems development is not merely a technical, but also a socio-technical process.

This thesis studies the development of one particular species of software – groupware – and is motivated by my own experience in groupware development. Groupware development is often a part of a larger systems development project and the developers are working very closely with the other stakeholders (in some cases sharing the same open office space). In contrast to other species of software, groupware is more sensitive to the socio-technical challenge (cf. Grudin, 1994a). Consequently, I argue that groupware development should be seen and treated as an organic process, in which the groupware is grown by the owners with the aid of professional developers rather than constructed by professional developers alone. To realise this perspective, this not only requires the development and proper use of technology, but also harmonious design between the human and the technological. As de Vreede and Guerrero (2006) observed:

> *"… It appears that technology alone seldom is the answer. What is needed is the conscious and harmonious design of collaboration processes and technologies." (ibid, p.571)*

[2] The work practices are more frequently changing in SMEs, especially young SMEs.

1.1 Thesis aim and scope

Empirical Modelling (EM) research was initiated by Meurig Beynon at University of Warwick more than 20 years ago. It has been developed as an approach to computer-based modelling which can be linked to William James's Radical Empiricism – a philosophical foundation that is radically different from that upon which mainstream computer science is based (Beynon, 2005). Over the past two decades, more than 100 peer-reviewed papers and more than a dozen doctoral research theses related to EM have been published. For more details, see chapter 4 and the EM website (http://www.dcs.warwick.ac.uk/modelling).

The aim of this thesis is to show that EM is an alternative and potentially a better conceptual framework for groupware development that accommodates the dynamic nature of group work that the groupware aims to support. More precisely, it tackles three research questions:

i) What are the main challenges in human-centred groupware development?

ii) How can EM be practised in a groupware development context?

iii) Why does EM potentially offer a better conceptual framework?

To answer these questions, I draw on literature on groupware development, human-computer interaction (HCI), computer supported cooperative work (CSCW), participatory design (PD), and Empirical Modelling (EM) in the broader context. In respect of EM, I draw on two strands of previous research:

i) research that concerns EM tool support and the interaction between the modeller and the model, and

ii) research concerned with the potential of EM as a systems development approach (cf. figure 1.1).

Y. W. Yung (1990), Y. P. Yung (1993), P-H. Sun (1999) A. Ward (2004) were the main contributors to the principal tool support (tkeden and dtkeden) for EM. The theses by the brothers Y. W. and Y. P. Yung are both concerned with the low-level differences between EM and other programming paradigms. In particular, Yung (1993) gives a detailed account of the differences in various aspects: *state*, *dependency*, agent, and *definitive notations*.

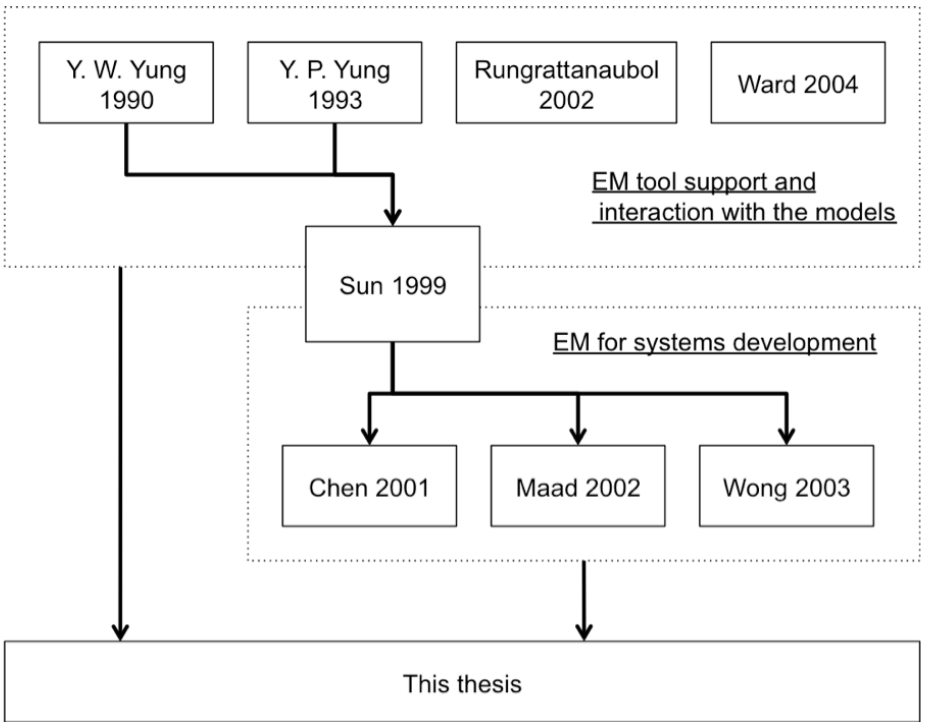


Figure 1.1 – Relationships between this thesis and other doctoral theses on EM

In her doctoral thesis "*A treatise on Modelling with definitive scripts*", Rungrattanaubol (2002) improved our understanding of the core activity of EM: Modelling with Definitive Scripts (MwDS). In particular, she considered in depth how a modeller interacts with states through the (re)definitions of observables, dependencies, agents, and agencies in EM. Ward (2004) explained the difficulty in supporting *dependency* – a key concept in EM (cf. §4.1.1) – on the sequential von Neumann machine hardware.

Sun (1999) introduced a conceptual framework for practising EM in a distributed environment, namely Distributed Empirical Modelling (DEM), in his doctoral thesis: "*Distributed Empirical Modelling and its Applications to Software System Development*". His work improved our understanding the potential of EM in distributed modelling contexts. Since DEM is based on distributed cognition theory (Hutchins, 1995), DEM is evidently more concerned with structured organisations rather than the dynamic structured organisations that might be found in the collaborative modelling context. Sun (1999) also presented a Situated Process of Requirements Engineering (SPORE) to show the potential of the DEM framework. Based on the DEM and the SPORE frameworks, some researchers had shown the potential of practising an EM approach in various domains. For example, Chen (2001) showed the potential of EM for participative development in business process re-engineering. Maad (2002) showed how the SPORE framework could be used in a financial domain. More recently, Wong (2003) consolidated the EM principles in a Definitive Modelling Framework (DMF). He showed how the developers and the users might interact through the construction of EM models within a SICOD[3] framework in ubiquitous computing.

1.2 Related work

In this section, I briefly discuss a few research works that have a similar outlook with this thesis in the broader context.

The Evolving Artifact Approach

One of the key aspects in groupware development is to co-construct a shared knowledge of the future work practices that will be facilitated by the developing artifact through the development process. Ostwald (1996) argues that the knowledge of future work practices can neither be elicited by observing nor by analysing how the users get their work done within the current work practices. For this reason, Ostwald (1996) proposed the Evolving Artifact Approach (EVA), an evolutionary and participatory approach that “operationalizes the idea of knowledge construction for system development” (ibid). This thesis shares a common interest in the entire lifecycle of systems development (discussed in section 2.5). However, this thesis argues that the users should have equal opportunities to get their hands on co-constructing the artifact within the same modelling space as the developers through EM (discussed in chapter 3, 5, and 7). The philosophical foundation behind EM is rather different from that behind traditional programming. In particular, as will be discussed in chapter 4, EM is based on observation, experimentation, and experience[4].

Meta-design

The idea of viewing systems development as a continual process can also be found in the meta-design conceptual framework (Fischer and Scharff, 2000). Fischer and Scharff (2000)

[3] SICOD stands for “Soft Interfaces for the Control of Devices”

[4] The term ‘experience’ is used in the sense of William James’s (1912) Radical Empiricism.

observe that it is difficult to design a closed system, and assert that, however well such a system is designed, the user will nevertheless discover “mismatches between their problems and the support a system provides”. They suggest that the *meta-designer* should *underdesign* (Brand, 1995) the system and let the end-users extend and evolve the system in an open-ended manner, which aligns with the “Seeding, Evolutionary Growth, Reseeding Model” (Fischer et al., 1994). The outlook of the GroupPIE framework (cf. chapter 7) and the efficacious groupware development perspective (cf. chapter 2) are quite similar to the meta-design framework: both evolutionary and participatory. However, “design” in meta-design refers to “[the] activities in which a person wishes to act as an active participant and contributor in personally meaningful activities”, which does not necessarily involve the lower level development activities, e.g. system implementation. Therefore, despite the fact that developers and users can be thought as *designers* in the meta-design framework, the imbalance of influence to the artifact construction potentially remain there (cf. chapter 3).

Participatory programming

Research into human-oriented systems development is not new; it was part of the early movement in Participatory Design (PD) (Gasson, 2003; also cf. §2.5.1). While PD, the notion of participation, and the issues surrounding participation will be described in detail in chapter 2 and 3, one particular work in PD is worth mentioning here.

Letondal and Mackay (2004) reported a participatory design project with a group of bioinformatics practitioners who had received professional training in both biology and computer science. The primary aim of their project was to explicitly support end-user programming through PD. The project became interesting as they found two types of bioinformaticians: i) those who maintain a strong interest in biology but are also interested in creating tools for other biologists and themselves to use; ii) those who are “seduced” by the computing tools and are forced to create tools for their problems. This motivated Letondal and Mackay to define the term *participatory programming* as “a logical extension of participation design, in which users participate in the creation of software tools that they can ultimately tailor and program themselves.”

Whereas no assumption is made about the technical competence of the participants in EVA or meta-design, the bioinformaticians in Letondal and Mackay's study were trained to be professional software developers (through their interdisciplinary education). In other words, the bioinformaticians were the developers as well as the users. As the popularity of interdisciplinary programmes, such as the Computer and Business Studies[5] undergraduate course at University of Warwick, increases, we shall see more cases like Letondal and Mackay's. Though there have been some similar findings in other fields, e.g. in free and open source software (FOSS) (Nakakoji et al., 2002) development, only a few researchers (such as (Letondal and Mackay, 2004)) have considered such highly conflated roles in systems development to be possible.

Although in fact EM is not the same as *programming* (cf. chapter 4), *participatory programming* is certainly related to the broader context of this thesis, particularly to the GroupPIE framework that I am going to discuss in chapter 7.

1.3 Research Contribution

The main contribution of this thesis, firstly, is that it identifies a promising configuration for practising an EM approach to groupware development that may lead to *efficacious groupware development*. Secondly, this thesis provides an account of the interplay among the conceptions and issues surrounding participation that leads to a better understanding of the problems in supporting human-centred groupware development. Thirdly, it improves our understanding of the potential of EM in a collaborative modelling context. This is supported with four case studies that cover a range of situations that may frequently occur in a collaborative modelling context. The work provides a promising foundation for further development of EM principles and tool support for collaborative modelling. Lastly, the *efficacious groupware development* perspective is itself a contribution, which provides a unified organic process view of groupware development based on Jullien's interpretation of ancient Chinese philosophy in his book "*The Propensity of Things*".

[5] These students are trained with programming knowledge and business knowledge. They can develop their career in both directions: software development or business management. In the latter case, they may become *hidden professional programmers*.

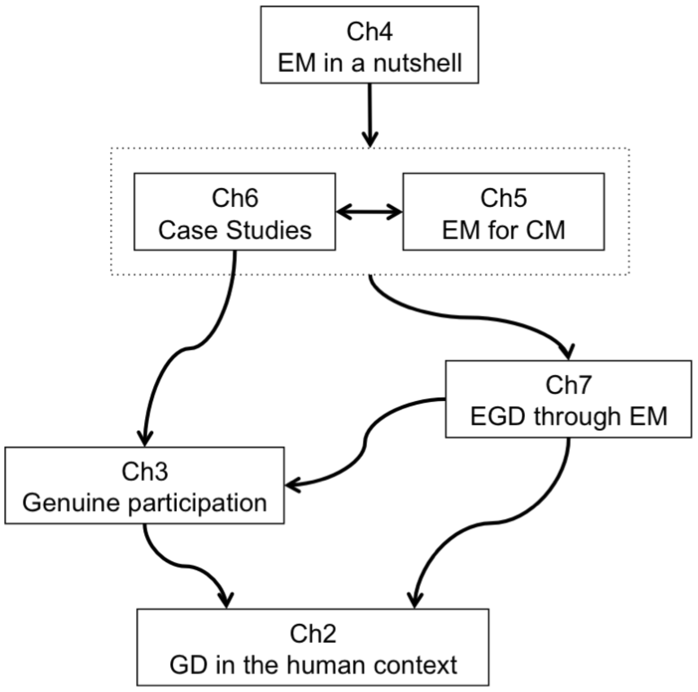


Figure 1.2 – Relationships between chapters in this thesis

1.4 Overview of the thesis

In relation to the aforementioned research questions, this thesis can be divided into three sections (cf. figure 1.2).

The first section (chapter 2 and 3) reviews research into groupware development and argues that several conceptions behind its development are problematic. It answers the question "What are the main challenges in human-centred groupware development?". In chapter 2, I argue that support for co-evolution between the groupware and the group activities is the main issue in integrating groupware into its human context harmoniously. This challenges accepted ideas about the scope of development and the nature of participation. Towards the end of chapter 2, I propose a unified organic process view – the *efficacious groupware development* perspective – that views the development activities as finding an efficacious disposition, which does not distinguish contexts and roles of participants throughout the development and evolution of groupware. In chapter 3, I argue that the contexts of development and use, the roles of participants, and the nature of participation are interdependent concepts. Instead of re-conceptualising them, I argue for the support of *genuine participation* which conflates contexts and roles, and see participants as human-beings. This creates a conceptual orientation for the rest of the thesis.

The second section (chapter 4, 5, and 6) discusses EM principles and tools, and explores EM potential in collaborative modelling. It answers the questions “What is EM?” and “How can EM be practised in a collaborative modelling context?”. EM is a modelling approach which is based on experience, interaction, observation, and experimentation. In chapter 4, I explain the principles, the philosophical outlook, and the practical EM tool support. Drawing on various previous research into EM (cf. figure 1.1), I argue that the EM process is open and flexible, and that EM artifacts (or models) are provisional, situated, tacit, interactive, and ostensive (in the sense of Gooding’s construal). In chapter 5, I define what I mean by *collaborative modelling*. I argue that collaborative modelling involves diverse configurations that can neither be prescribed nor framed easily. A better approach to support collaborative modelling is to take this heterogeneous nature into account and allow flexible reconfiguration. I also examine the potential of EM in a collaborative modelling context, and explain why DEM is not good enough for collaborative modelling. In chapter 6, I demonstrate the potential for practising an EM approach in the context of collaborative modelling with four case studies. These case studies cover situations that may occur frequently in the context of collaborative modelling.

The last section (chapter 7) considers EM in the context of groupware development. It answers the questions “Why does EM potentially offer a better conceptual framework?” and “How can EM be practised in a groupware development context?. Based on the principles of EM and the case studies, I argue that EM is a human-centred development approach when it is practised in a collaborative modelling context. Therefore, I argue that the potential of EM in collaborative modelling can be exploited in groupware development. Such an orientation potentially yields a better conceptual framework that gives a holistic account of groupware development from its first conception to its eventual disposal. This is a step towards realising efficacious groupware development.

Chapter 2 Groupware development in the human context

In the past few decades, since the intervention of personal computers, computational artifacts have been proliferating dramatically. Computer technology has entered our daily lives in many aspects, from organized to unstructured ad-hoc activities, from office to home, from formal work to entertainment, and it is playing an essential role in human communication and collaboration. Yet, the diverse, pervasive and ubiquitous use of computer technology poses new challenges to research into the development of computer technology when seamless integration to human activities and context are the primary concerns. Research communities such as Human Computer Interaction (HCI), Computer Supported Cooperative Work (CSCW), Participatory Design (PD), and Information Communication Technology (ICT) focus on different facets in the development of computer technology and yet share a broader common concern: how can computer technology support human activities efficaciously. In CSCW, the challenge is referred to as the *social-technical gap*: the gap between what we can support technically and what we must support socially (Ackerman, 2000). It is this socio-technical challenge that frames the broad context for this chapter. Instead of studying the broader context, this chapter focuses on the development of groupware, a kind of computer system which is more sensitive to the socio-technical challenge than other species.

This chapter develops an argument that there is no silver bullet for groupware development. Successful groupware development requires insightful understanding of how work is carried out in practice, and contemporary research does not offer a unified coherent theoretical framework for this purpose. Beginning with a taxonomy of group work, section 2.1 discusses the five important aspects of group work. That is, group awareness and coordination of work, articulation work, creativity, experimentation, knowledge sharing and knowledge construction. These are the basic aspects that successful groupware has to support.

Groupware is a socio-technological system and its development is unusually challenging due to the dynamic nature of groups and the evolving environment for group work. Section 2.2 discusses the socio-technical problem and the issues that are at the centre of concerns in groupware development. Contemporary paradigms for groupware development are reviewed in section 2.3. Due to the differences among groups and the dynamic nature of group work, a generic approach to groupware development is unlikely to suit all types of group work or all groups. Section 2.4 discusses the evolving nature of human activity, current theoretical influences on groupware development, and paradigms to evolve groupware. Indeed, in supporting human activities, the groupware must be able to *evolve*, allowing re-configuration throughout its lifecycle. Section 2.5 discusses a novel perspective on groupware development that is *efficacious* in the sense discussed by Jullien (1995) with reference to the notion of disposition.

2.1 Supporting group work

In order to provide effective computer support for group work, it is vital that groupware developers and groupware methods developers recognize the different types of group work and their nature. It is also essential to know that all successful groupware has some common features. In this section, a taxonomy of group work will be derived from the classification of groupware and five common features of successful groupware will be discussed.

2.1.1 Taxonomy of group work

Classifications of groupware such as (Johansen, 1988), (Teufel et al., 1995) and (Ellis et al., 1991) provide useful information to help users of off-the-shelf software to understand the kind of support that a specific groupware is offering and therefore help them to select the appropriate tool according to their needs. These classifications also provide a shared reference point, which eases the communication among groupware researchers, developers and other participants who are involved in the development process. Indeed, each classification scheme reflects a facet of the whole spectrum of group work.

Johansen (1988) proposed a two dimensional taxonomy for groupware. He suggests that groupware can be conceived to support collocated or distributed groups in the space dimension, and asynchronous or synchronous work in the time dimension. Most of the groupware classifications are derived from Johansen's space/time classification. For example, Grudin (1994b) has extended Johansen's taxonomy to take account of the predictability of space and time constraints, which results in a 3x3 matrix, and the impact this has on the way in which group work is organised. Andriessen (2003 p.12) combines Johansen's classification with five functions of ICT in the group process. Recently, Penichet et al. (2007) produced a more complicated 7-dimensional matrix, which combine CSCW characteristics (i.e. information sharing, communication, and coordination) with a modified space/time dimension matrix. Johansen's and Grudin's classification were popular in 1990s and remain highly cited amongst CSCW researchers.

There are other classifications of groupware which do not follow Johansen's dimensional perspective. For example, the 3C model (Teufel et al., 1995) reflects a triangular relationship among communication, coordination and cooperation (cf. Borghoff and Schlichter, 2000), but such reflection may contradict the four-level of communication perspective (Borghoff and Schlichter, 2000, p.110). Ellis et al. (1991) differentiates groupware according to which application features they support. In Ellis et al's classification, different groupware may share common features. Because of this, compared to Johansen's classification and its derivatives, these classifications reflect less useful perspectives for understanding the characteristics of group work.

Apart from space, time, predictability, and CSCW characteristics, group work can also be classified as structured or unstructured, planned or ad-hoc, data centric or process centric, and according to other dimensions such as group size (cf. Johansen, 1988; van der Aalst, 2007). Moreover, CSCW characteristics can be extended into a full spectrum of degrees of engagement (which will be discussed in chapter 5). There is endless possibility in classifying group work. Certainly, we can enrich the taxonomy in an open-ended way with additional dimensions through further analysis of group work, but more significantly, we shall recognise that group work in practice is usually dynamic. It rarely settles into one particular mode of interaction. Actual human collaborations are fluid and do not usually follow a single pattern of work.

2.1.2 Awareness and Coordination

Awareness of other people's work is crucial in a group work setting. As Dourish and Bellotti (1992) describe:

> *"awareness is an understanding of the activities of others, which provides a context for your own activity. This context is used to ensure that individual contributions are relevant to the group's activity as a whole, and to evaluate individual actions with respect to group goals and progress. The information, then, allows groups to manage the process of collaborative working." (ibid)*

This understanding is an ongoing interpretation of both one's own and others' activities and artifacts involved in the process of group work (Chalmers, 2002), and it enables an individual to make conscious decisions (Borghoff and Schlichter, 2000) that promote a harmonious interaction, which eventually improves the productivity of the group (Kirsch-Pinheiro et al., 2003).

Dourish and Bellotti's (1992) notion of awareness is generally referred to as "group awareness" (Gutwin, 2004). In sociology, awareness is referred as shared cognition or team knowledge (Andriessen, 2003; Cannon-Bowers and Salas, 2001). In terms of computer support, awareness can be further categorized according to the nature of information it discloses and supports, e.g., presence awareness, emotion awareness, workspace awareness (Gutwin, 1997), change awareness (Tam and Greenberg, 2004), activity awareness (Hayashi et al., 1999; Carroll et al., 2006), gaze awareness (Ishii and Kobayashi, 1992).

According to Dourish and Bellotti (1992), awareness has dual roles in group work: a high-level of awareness of other people's work helps to avoid redundancy of work, while a low-level awareness allows "fine-grained shared working and synergistic group behaviour". (Dourish and Bellotti, 1992) The support to awareness is important for coordinating group work (Dourish and Bellotti, 1992) and this inevitably involves division-of-labour. Lack of support to awareness in some situations may have unpleasant consequences, e.g. presence disparity in mixed-presence groupware (Tang et al., 2004).

Although the impact of awareness on group work is clear, implementing awareness support in groupware remains challenging. As Chalmers (2002) pointed out "representation and interpretation affect the degree and character of awareness afforded by computer systems: awareness of people and of information artifacts". This highlights the complexity and the socio-technical nature (cf. §2.2.1) of the issue. Further, awareness information can overload the recipient if not carefully designed (Kirsch-Pinheiro et al., 2003). In some situations, it may become a disturbance and significantly degrade the performance of an individual and the system (Dourish and Bellotti, 1992). It is difficult to maintain a balance between the amount of information to be disclosed and the degree of awareness of other people's work. This is because different activities and different group, with few exceptions, require different kind of awareness support (Kirsch-Pinheiro et al., 2003). This makes it hard to generalize approaches to providing support for awareness and groupware developers often have to build this support from scratch (Kirsch-Pinheiro et al., 2003).

2.1.3 Articulation work

Research into articulation work has gained much attention in CSCW since Schmidt and Bannon (1992) observed that the lack of focus of the field "may hinder its further development and lead to its dissipation", and suggested that "CSCW should be conceived of as *an endeavor to understand the nature and requirements of cooperative work with the objective of designing computer-based technologies for cooperative work arrangements*" (ibid, p.5, italics in original).

The notion of articulation work was first introduced by a sociologist, Anselm Strauss, in 1985 (Strauss, 1985; Fjuk et al., 1997). It is used as a framework to study the interwoven nature of the division-of-labour and the interdependence among individuals in group work (Fjuk et al., 1997). Articulation work refers to the interaction between individuals that is necessary to, e.g. coordinate, align, schedule, and integrate interdependent activities which occur during the course of cooperative work (Schmidt and Simone, 1996; Lee, 2005). Strauss (1993) defines:

> *"Articulation stands for the coordination of lines of work. This is accomplished by means of the interactional process of working out and carrying through of work-related arrangements. Articulation varies in degree and duration depending upon the degree to which arrangements are in place and operative." (Strauss, 1993, p.87)*

Articulation work is different from cooperative work. While the former is concerned with coordinating individual tasks taking into account the distributed nature of group work, the latter merely concerns the distribution of tasks to individuals who shared a common workspace, as Schmidt and Simone (1996) pointed out:

> *"Cooperative work is constituted by the interdependence of multiple actors who interact through changing the state of a common field of work, whereas articulation work is constituted by the need to restrain the distributed nature of complexly interdependent activities." (Schmidt and Simone, 1996, p.158)*

The relationship between cooperative work and articulation work is recursive since the management of an articulation work can invoke another articulation work or cooperative work, which again may yield another articulation work (Star and Strauss, 1999, Schmidt and Simone, 1996). This recursion, in theory, can be infinite, but, as Schmidt and Simone (1996) have pointed out, real world work practices usually terminate such recursions before they degenerate into chaos.

Supporting the ongoing articulation work is one of the key issues in CSCW (Schmidt and Bannon, 1992). Central to articulation work is the concern for informal interactions, that are not explicit in the formal description of work, to bring the work back on to track when there are unexpected contingencies and breakdowns (Fjuk et al., 1997; Star and Strauss, 1999; Evans et al., 2001). Schmidt and Simone (1996) observe that articulation work may become intuitive and remain invisible in collocated group work where actors can interpret others' activities through their bodily actions. Unfortunately, articulation work becomes "extremely complex and demanding" in a real world complex group work setting (Schmidt and Bannon, 1992; Schmidt and Simone, 1996). Providing computer support for articulation work remains difficult due to the situated characteristics of the work activity (cf. Suchman, 1987; Grinter, 2000). Indeed, it may entail a thorough analysis of the culture and politics of the work, if the work is to be made visible (Star and Strauss, 1999; Schmidt and Bannon, 1992).

2.1.4 Creativity

Creativity has been a topic for research in sociology and psychology, but only recently has it caught the attention in HCI and CSCW, e.g. collaboratories (Sonnenwald, 2003; Farooq et al., 2005). Despite the fact that human creativity has the characteristics of being "inventive, wily, sly, cunning, and crafty" (Spinuzzi, 2003, quoted by Kaptelinin and Nardi, 2006), individual creative contributions are essential in group work because they involve "an instant restructuring of the whole representation of a problem" (Kaptelinin and Nardi, 2006; also cf. Wertheimer, 1966). Although this process is entirely internal to an individual, Kaptelinin and Nardi (2006, p.210) also acknowledge that "a subject can reframe a representation to successfully solve a problem", and the importance of this framing of a problem in problem solving has already been stressed by Jackson (2001).

Although creativity is still by-and-large viewed as a quality of an individual and is sometimes viewed as a gift endowed at birth, there is no doubt that many creative ideas are the result of collaborative work (Nijstad et al., 2006). Fischer (2005) echoes this view by saying:

> *"Much human creativity is social, arising from activities that take place in a social context in which interaction with other people and the artifacts that embody collective knowledge are essential contributors." (ibid)*

A similar perspective can also be found in (Kaptelinin and Nardi, 2006), in which the authors argue that "[t]here is no lone genius" (ibid, p.214).

Group creativity is not merely a sum of individuals' creative work as if they are working alone, but it is an emerging property of the group (Stahl, 2006; Kaptelinin and Nardi, 2006). However, there is much evidence in sociology suggesting that groups are far less productive and creative than the sum of their individual talents might indicate, e.g. (McGrath, 1984; Mullen et al., 1991; Nemeth and Nemeth-Brown, 2003; Paulus et al., 2006). It is not unusual to find that groups are making poorer decision than individuals (Nemeth and Nemeth-Brown, 2003). Nijstad et al. (2006) observed that this reduced productivity is due to 3 reasons: i) group members must express their ideas in turn; ii) group members are unwilling to share their ideas[6], iii) high-performing individuals tend to match their effort to that of their low-performing colleagues. While there is seemingly little computer technology can do in case (ii) and (iii) (Nijstad et al., 2006), there is certainly a role that computer technology can play in supporting creativity in group work.

In tackling (i), Nemeth and Nemeth-Brown (2003) stress the benefits of allowing different perspectives in the group. This proposal is strengthened by Nijstad et al's (2006) observation that if individuals could express their ideas simultaneously through e.g. typing and writing, though not verbal communication, there would be less productivity loss (2006). Similar thinking along these lines can be found in (Farooq et al., 2005), in which they propose to give support for i) divergent and convergent thinking, ii) the development of shared objectives, and iii) reflexivity.

It is worth noting that the development of computer support for creativity is still at an early stage. As Kaptelinin and Nardi (2006) point out, there is still no framework for analyzing how effects at an individual level can cause effects at the group level.

2.1.5 Experimentation

Following the creativity thread that has been discussed in the previous section, there are situations in which one may wish to explore an innovative idea further, e.g. to explore a configuration to see if it fits the situation at hand. This can be due to uncertainties about the problem at hand, the desire to gain further understanding about the current situation, or imperfect knowledge that necessitates learning by doing. I refer to this kind of problem solving related activity as *everyday experiment*.

Everyday experiment is different from a controlled experiment that is usually carried out in a

[6] In some situations, individuals may be reluctant to share their perspectives because it may conflict with the group's perspective. In order to maintain a consensus, they usually drop their perspectives silently (Baron et al. 1992; Paulus et al. 2006). Janis (1982) refers to this phenomenon as 'Groupthink'. This usually results in reduced creativity for the group (Nemeth and Nemeth-Brown 2003).

laboratory[7]. In contrast to laboratory experimentation, everyday experiment is informal and casual in nature, e.g. it is subject to fewer safety regulations. In the broad sense, experiments can be classified as *exploratory* experiments or *post-theory* experiments, depending on whether "there is a reliable basis for prediction, or criterion for successful outcome, drawn from theory or previous experience" (Beynon and Russ, 2008).

Experimentation has been seen as significant in innovative work[8], e.g. product design (Thomke, 1998), because "all experiments yield information that comes from understanding what does work and what does not work" (Thomke, 2003, p.20). Although Thomke (2003) is more concerned with the systematic experimentation involved in the use of computer simulations, models, and prototypes in controlled environments, I believe that the less-rigorous everyday experiment can also stimulate new ideas in group work.

2.1.6 Knowledge sharing and knowledge construction

As Divitini et al. (1993) observe, knowledge is crucial for group learning and its continuous development:

> *"The knowledge generated and used by the members of an organization in their cooperative work is a fundamental context for the effective accomplishment of their on-going activities as well as a basic background for the future ones." (ibid)*

Knowledge sharing and knowledge construction are two closely related concerns that are sometimes regarded as one concern under the umbrella term of knowledge management. Knowledge sharing can be conceived as part of the collaborative knowledge construction process (cf. Stahl, 2006). It concerns the exchange of information, i.e. knowledge, among individuals in groups, and it is one of the basic group processes (Andriessen, 2003). Sharing knowledge, like sharing awareness information (cf. §2.1.2), also suffers from similar social

[7] In the narrow sense, an experiment can only be carried out in a laboratory environment, in a controlled fashion.

[8] There is a subtle distinction between creativity and innovation. Amabile et al. (1996) define "… creativity as the production of novel and useful ideas in any domain [and] innovation as the successful implementation of creative ideas within an organization. In this view, creativity by individuals and teams is a starting point for innovation; the first is a necessary but not sufficient condition for the second" (ibid).

concerns – there may be no direct benefit for sharing knowledge with the group, and no guarantee that the other group members may benefit from the shared knowledge.

Knowledge construction concerns the emergence of knowledge and is usually part of the collaborative learning process in a group. Because of that, it is more often studied in the context of, e.g. computer supported collaborative learning (CSCL), knowledge management, and organisational studies.

Andriessen (2003) described two strategies, originally proposed in (Hansen et al., 1999), to provide technological support for knowledge management in groups. The first approach, the *codification strategy* entails codifying knowledge. This can be successful in relation to storing a large amount of information, but is less effective in relation to practical useful *knowledge*[9] (Andriessen, 2003). Andriessen (2003) argues that the limitations of codification stem from the loss of the personal experience and the contextual information that is originally embedded in the *knowledge*. The second approach, the *personalisation strategy*, focuses on inter-personal communication and knowledge sharing, mentoring, and *communities of practice* (CoP) (Andriessen, 2003). In this approach, the technological support is limited to providing pointers to the expertise (e.g. yellow pages) and promoting the communication between people (Andriessen, 2003). Andriessen (2003) stresses that each strategy has its own merits – the former is better for routine work, and the latter is more suitable for non-routine work.

2.2 Issues in groupware development

In this section, I first define what I mean by groupware, then I move on to the discussion of the socio-technical problem in groupware development. I then revisit Brooks's (1987) "No Silver Bullet" argument. Finally I argue that there is no standard way to approach groupware development and, due to its socio-technical nature, groupware development is a situated activity.

[9] Andriessen (2003) pointed out that, in the strict sense, once knowledge is stored in an 'impersonal system', e.g. a computer system, it is "no longer knowledge, it has become information".

2.2.1 Groupware and CSCW

Many people believe that the study of groupware is equivalent to the study of CSCW. However, this is not quite the case (cf. Schmidt and Bannon, 1992). The term 'groupware' was used by Johnson-Lenz and Johnson-Lenz (1982) before the term CSCW was coined, and it was originally used to refer to the software that supports group processes (Penichet et al., 2007). CSCW is a multidisciplinary research field with a much broader scope, which includes, e.g., computer science, psychology, and sociology.

Groupware, literally, is a combination of the word 'group' and the word 'software' which suggests a meaning of *groups' software* or *software for groups*. However, the term does not have a standard definition in the literature. For example, some people consider workflow systems as groupware, but others do not (cf. Bannon, 1992a). Johansen (1988) defines

> *"Groupware is a generic term for specialized computer aids that are designed for the use of collaborative work groups. Typically, these groups are small project-oriented teams that have important tasks and tight deadlines. Groupware can involve software, hardware, services and/or group process support." (Johansen, 1988; cited by Borghoff and Schlichter, 2000, p.92)*

Ellis et. al. (1991) define groupware as:

> *"computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment." (ibid)*

More recently, Orlikowski and Hofman (1997) describe groupware technology in the following term:

> *"Groupware technologies provide electronic networks that support communication, co-ordination and collaboration through facilities such as information exchange, shared repositories, discussion forums and messaging." (Orlikowski and Hofman, 1997)*

Johansen's (1988) definition includes all kinds of computer technologies for supporting collaborative work. It indicates that groupware development should include the development of hardware, software, and other kinds of system support. In Ellis et. al. (1991), the focus is on providing a shared workspace for a common goal. Orlikowski and Hofman (1997) believe that groupware should focus on supporting group interaction through technologies, with no particular emphasis on supporting surrounding human activities carried out without directly interacting with computers. This divergence in perspectives is highlighted by Grudin (1991a), who points out that researchers even disagree on the classification of email. Whereas Kraut believes that email is the only successful groupware, Bannon and Schmidt argue that email is merely an enabling technology (Grudin, 1991a).

The term "computer-supported cooperative work" (CSCW) was coined by Paul Cashman and Irene Greif in a workshop held in 1984 (Grudin, 1994b). Greif (1988) described CSCW as "an identifiable research field focused on the role of the computer in group work" (quoted in Andriessen, 2003, p.10). However, the scope of CSCW was broadened to not only finding a role for the computer but to understanding how people actually work when Wilson (1991) said:

> *"CSCW is a generic term which combines the understanding of the way people work in groups with the enabling technologies of computer networking, and associated hardware, software, services and techniques" (quoted by Borghoff and Schlichter, 2000, p.92).*

This broadening of scope becomes yet clearer when Bannon and Schmidt (1989) write:

> *"CSCW should be conceived as an endeavor to understand the nature and characteristics of cooperative work with the objective of designing adequate computer-based technologies". (ibid, p.360)*

In the context of this thesis, the term CSCW is used to refer to the research field and its theoretical foundations for building computer systems, and the term groupware is used to refer to software in the computer system (in the narrow sense).

2.2.2 The socio-technical challenge

When compared with single user software development, groupware development is more challenging. On the one hand, groupware developers are facing all the problems that are inherited from typical software development (cf. Sommerville, 2004). On the other hand, they are facing both technical and social challenges that are unique to groupware development (cf. Borghoff and Schlichter, 2000). These challenges, however, are intertwined in nature. A technically sound solution can fail due to the social issues of the group (Grudin, 1994a). As Grudin (1994a) writes:

> *"Groupware may be resisted if it interferes with the subtle and complex social dynamics that are common to groups" (Grudin, 1994a).*

Similarly, a socially sound solution may not be realisable due to the technical difficulties. Success and failure of groupware are equally hard to predict (Grudin, 1994a). A groupware that works for one group may not work for another. This suggests that the process of developing a groupware is a socio-technical challenge that has to be addressed *in situ*.

Five aspects of the socio-technical challenge

The CSCW research community had already recognized the socio-technical nature of groupware development in the 1990s (cf. Grudin, 1988; 1991a; 1994a; 1994b). Many experiences and lessons had been reported in journals[10], conferences[11], and workshops[12]. I shall not repeat their findings here or attempt to give a comprehensive survey of the literature. Instead, to put this thesis into context, I derive five aspects of the socio-technical challenge, mainly drawing on the work by Grudin (1988; 1991a; 1994a; 1994b), Ellis et al. (1991), and Ackerman (2000). These aspects are discussed below.

The organisational aspect

Grudin (1994a) observed that the benefit that one might receive from using a groupware depends on a number of factors. These include the user's preferences, backgrounds, roles,

[10] E.g. Computer-Supported Cooperative Work, Human-Computer Interaction, Interacting with Computers.

[11] E.g. International Conference on Computer-Supported Cooperative Work (CSCW), European Conference on Computer-Supported Cooperative Work (ECSCW), International Conference on Supporting Group Work (GROUP), International Conference on Computer-Human Interaction (CHI), Nordic Conference on Computer–Human Interaction (NordiCHI).

[12] E.g. Collaboration Researchers International Workshop on Groupware (CRIWG).

and tasks (Grudin, 1994a). There is not necessarily a direct connection between the work, e.g. using the groupware to share information, and the benefit of the work (Grudin, 1994a). For example, in a groupware that provides a shared calendar feature for project management purposes, the benefit from using such feature is greater for a manager of the team, rather than her team members (Grudin, 1994a).

A related issue is the conflict of interest between individuals and groups or among individuals. Conflict of interest is common in organizations (cf. Brooks, 1999). Conflict of interest can become a barrier to achieving critical mass in groupware adoption (as will be topical in discussing the Adoption aspect below) or reduce the efficacy of the groupware. Markus and Connolly (1990) discuss the conflict of interest issue thoroughly, where they call it the *interdependence problem*. Grudin (1994a) refers to it as the *Prisoner's dilemma*. According to Markus and Connolly (1990), there are two types of interdependence in the context of groupware. The first one is *usage interdependence*, which is defined as "a person's ability to use an application in one way depends on another individual's different use of the same application" (Markus and Connolly, 1990). The second one is *payoff interdependence*, which occurs when "one person's use of an application creates costs or benefits (negative or positive externalities) for others who use the application in essentially similar ways" (Markus and Connolly, 1990). While usage interdependence always results in different or asymmetric payoffs to the users, payoff interdependence may result in asymmetrical or symmetrical payoffs (Markus and Connolly, 1990).

The third issue which falls into the organizational aspect is the decision making associated with groupware development. For instance, Grudin (1994a) observed that managers who have the power to make decisions can be falsely informed by their intuition. Grudin (1994a) argues that this can be due to underestimating the down side of the system, or to failure to foresee the difficulties that may be entailed in developing, evaluating, and using the groupware. In addition, "[i]ntuition fails when the intricate dynamics of such situations are not appreciated" (Grudin, 1994a, p.101).

For the first issue in the organisational aspect, incentives to compensate for individual effort (i.e. extra work) in using the groupware system are crucial (Grudin 1988; 1994a; Ackerman, 2000). These incentives can be provided either in the groupware or within the organizational context (Ackerman, 2000). The key idea is to ensure that individuals perceived a potential benefit in using the groupware. For the second issue, negotiation may be needed to resolve such conflicts of interest (Ackerman, 2000). For the third issue, Grudin (1994a) emphasizes the importance of user involvement and democratic decision-making in the development of groupware. For example, paradigms such as Participatory Design may help to fulfill this need.

The social aspect

Ackerman (2000) observes that human social activities are “fluid and nuanced”. Ackerman (2000), citing Goffman’s work (1971), argues:

> *“people have very nuanced behaviour concerning how and with whom they wish to share information. People are concerned about whether to release this piece of information to that person at this time, and they have very complex understandings of people’s views of themselves, the current situation, and the effects of disclosure” (Ackerman, 2000)*

For example, people may resist using a groupware for project management that automatically sends notification message to supervisors and directors when the deadline of a piece of work is reached (Borghoff and Schlichter, 2000). As Grudin (1994a) pointed out, groupware might fail if it disturbs the social processes of the group. It is often hard to develop and to support the level of flexibility and detail of human social activities in computer systems (Ackerman, 2000), because “[the] computer is happiest in a world of explicit, concrete information” (Grudin, 1994a, p.97). Due to this difficulty, it may be easier to allow the social mechanisms to regulate the work which the groupware is supporting (Ackerman, 2000).

Work practices

Although there is usually a standard procedure for work, Grudin (1994a) observed that people quite often do not follow the standard procedure for work. Grudin (1994a) argues that “working to rule” may bring the work to halt, and “A wide range of error handling, exception handling, and improvisation are characteristic of human activity" (Grudin, 1994a). Sachs (Sachs, 1995) distinguishes two types of work. The first type of work is organizational work, which follows formal organizational procedures and is characterized as explicit and structured (Kaptelinin and Nardi, 2006). The second type is informal (i.e. ad-hoc) and unstructured work, which follows work practices and focuses on conceptual understanding (Kaptelinin and Nardi, 2006). According to Kaptelinin and Nardi (2006), citing Sachs's work (Sachs, 1995), groupware developers need to take both types of work into account if the system is to be used to support work. However, most designs are merely informed by the formal work view (Kaptelinin and Nardi, 2006). Grudin (1994a, p.98) argues that if we follow this path of design, we may require "AI techniques beyond the state of the art … to make the system useful". Grudin (1994a) suggests tailoring as a way to address this problem.

A number of ethnomethodology analyses studied the failure of workflow systems in the 1990s (Kaptelinin and Nardi, 2006). Kaptelinin and Nardi (2006) argue that these failures were due to placing the design focus on optimising the effectiveness and efficiency of the organizational tasks rather than on how people actually work. (Kaptelinin and Nardi, 2006). To design better groupware, researchers have been employing field studies, such as ethnography, to inform their design decisions (cf. Ackerman, 2000). However, even after two decades, CSCW researchers still know little about how people actually work[13].

Evaluation

According to Grudin (Grudin, 1994a), groupware is difficult to evaluate for several reasons:

1. It is not easy to replicate all factors, e.g. social, political, economical, and organisational, that affect the use of groupware in laboratory environment.
2. It is hard to schedule an evaluation due to the number of people involved.
3. It is hard to expose interaction issues which can usually only be discovered over a long period of time.

[13] This observation was made by Christian Heath who gave the concluding remarks in the Colloquium on "Challenging Groupware: Emerging configurations for distributed interaction" on 12 February 2008. The author attended this colloquium (cf. also the blog by Sigfridsson at http://socgsd.blogspot.com/2008/02/colloquium-on-challenging-groupware.html).

4. It is hard to identify the factors that lead to success or failure. For instance, a talented group may be able to adapt a seriously flawed system (Grudin, 1994a). This also makes generalizing from previous experience difficult, if not impossible.

Adoption

For a single-user application, persuading one out of five users to use the product is a big success; but for groupware, only one out of five members of the group using the groupware is a huge disaster (Grudin, 1994a). Unfortunately, the adoption process of groupware varies from group to group. This is because of individual differences (e.g. skills, knowledge, beliefs) and the context in which the group is situated (e.g. organization, social practice, and politics). Strictly speaking, there are no two identical groups.

The efficacy, i.e. usefulness, of the groupware requires a critical mass of members of the group to adopt it (Grudin, 1994a). This critical mass is usually high, i.e. the groupware may fail even when only a few individuals refuse to use it (Grudin, 1994a). A groupware is less likely to gain a critical mass of adoption if it presents social or organizational issues as I discussed above. Other issues such as usability and their physical location of the hardware may also affect the possibility of achieving a critical mass (cf. Grudin, 1994a).

The socio-technical gap

Ackerman (2000) argues that the socio-technical challenge in groupware development is due to a mismatch between the social requirements and the technical feasibility. Ackerman (2000) refers to this mismatch as the *socio-technical gap*, as he writes:

> *"The socio-technical gap is the divide between what we know we must support socially and what we can support technically. Exploring, understanding, and hopefully ameliorating this social-technical gap is the central challenge for CSCW as a field and one of the central problems for HCI. Other areas of computer science dealing with users also face the socio-technical gap, but CSCW, with its emphasis on augmenting social activity, cannot avoid it." (ibid)*

In arguing for the significance of the socio-technical gap, Ackerman (2000) has considered three arguments:

The socio-technical gap was due to groupware developers' habit of ignoring social factors. However, Ackerman (2000) argues that researchers have gradually accepted the existence of the gap and recognized its nature. As Ackerman (2000) observed, "[s]ocial nuance and flexibility were slowly added to all CSCW systems, as the undesirability of being explicit became an assumption with CSCW" (Ackerman, 2000). For example, according to Ackerman (2000), lessons were learnt from The Coordinator (Flores et al., 1988) and Answer Garden (Ackerman and Malone, 1990), and the inflexible access control was improved when Answer Garden 2 was developed (Ackerman and McDonald, 1996).

It is tempting to think that the gap will disappear when new technology becomes available, but Ackerman (2000) argues that this gap is unlikely to go away because there has been no success in the past twenty years or so despite the tremendous effort in attempting to bridge the gap in other areas of research, such as artificial intelligence, information technology, and information science (Ackerman, 2000). Furthermore, Ackerman argues the efficacy of other approaches, such as neural networks, has not been proved. He also argues that the gap may be due to a fundamental problem in the von Neumann architecture on which modern computing is based:

> *"As Hutchins and others have noted (Hutchins, 1995, Clark, 1997) the standard model of the computer over the last thirty years was disembodied, separated from the physical world by ill-defined (if defined) input and output devices. In this view, the von Neumann machine came to be socially inflexible, decontextualized, and explicit. … the existing von Neumann architecture led to programming practices that in turn led to explicit and inflexible systems using data that were far too simplified over the natural world" (Ackerman, 2000).*

The gap could be mended by a co-evolutionary perspective. The rationale behinds the co-evolutionary perspective is that people are more flexible than machines and therefore should able to adapt themselves to the machine efficiently and effectively. Ackerman (2000) explains that the co-evolutionary perspective can hardly be dismissed because it is hard to see that human culture would not adapt to any technology. However, there is evidence that poorly designed technology can result disasters and inevitably have to be *re-designed* (e.g. the Clayton Tunnel railway disaster in 1861 (cf. Harfield et al., 2005). As the subtitle on the CPSR[14] website: "Technology is driving the future… It is up to us to do the steering".

Nevertheless, there is an implicit assumption in Ackerman's (2000) claim concerning the provenance of the socio-technical gap – that we can know what the social requirements are. Unfortunately, after twenty years of empirical studies, the community has not come up with a well-defined set[15] of social requirements that a groupware must support in order to ensure its success. And, despite Ackerman's (2000) hopes for technological solutions to meet social requirements, he does acknowledge that there will always be a compromise between the ideal "technical working" system and an "organizationally workable" system.

2.2.3 Brooks's "No Silver Bullet" argument

In his seminal paper "No Silver Bullet – Essence and Accidents of Software Engineering", Fred P. Brooks Jr. argues that the difficulties for software development can be divided into the essential and accidental[16] (Brooks, 1987; 1995; Fraser et al., 2007). Brooks (1995) argues that, unless the incidental difficulties are 90% of all the difficulties, spending time in tackling incidental difficulties will not result in significant improvement for the software development process as a whole. According to Brooks (1987), the essential difficulties are *born* with software development, while the accidental difficulties stem from its production process. Brooks (1987; 1995) argues that the hardest part in software development is the *mental crafting* of the conceptual construct, not its implementation process. For Brooks (1987), developing software is always challenging due to its essential complexity, conformity, changeability, and invisibility. Brooks (1987) explains:

[14] Computer Professionals for Social Responsibility, http://cpsr.org/

[15] There are new findings about how people work together continuously, but it is also the case that generalisation is difficult, if not impossible.

[16] Brooks explained that the word "accidental" is somehow misleading. He explains: "By *accidental*, I did not mean *occurring by chance*, nor *misfortunate*, but more nearly *incidental*, or *appurtenant*" (italics in original, (Brooks 1995, p.209)). Brooks used the word "incidental" instead of "accidental" in a recent workshop in OOPSLA '07.

Complexity – there are no two parts that are exactly the same beyond the statement level, and complexity scales up nonlinearly in size (Brooks, 1987). However, Brooks (1995) explains that most of the complexity in software is due to the representation involved in its implementation, i.e. what is accidental in Brooks's terms, such as algorithms, connectivity, and data structures.

Conformity – this concern relates to the artificial nature of software and the human agency imposed on it. Whereas physical laws can guide the modelling of natural phenomena, there are no fundamental principles that one can follow when making complex objects in software development (Brooks, 1987). Software developers have to conform to different arbitrary standards from time to time, from systems to systems, not due to physical necessity but due to the fact that these standards are designed by different people (Brooks, 1987). In the other words, *software does not conform to any principles that can be found or can be established in nature*. This makes software engineering different from other engineering disciplines as well.

Changeability – software inevitably changes from time to time because "the software product is a cultural matrix of applications, users, laws, and machine vehicles. These all change continually, and their changes inexorably force change upon the software product" (Brooks, 1987, p.12).

Invisibility – Unlike a floor plan, which can be represented in a geometric reality, software is hard to represent in two or more dimensions (Brooks, 1987; 1995). The reality of software is not in a physical space; one cannot interact with software abstractions as when interacting with other physical objects. Software is inherently invisible and unvisualisable (Brooks, 1987).

Brooks (1987) has considered four strategies for attacking the essential difficulty of software development, i) buying instead of building in-house software, ii) rapid prototyping in supporting requirements elicitation and refinement, iii) growing instead of building software, iv) investing in great designers. Much has changed in the software industry since the publication of Brooks's (1987) seminal paper, e.g. the propagation of object-oriented methods and tools, the advocacy of agile development paradigms. Even so, despite the fact that more than 20 years has passed and the responses of Brooks's critics[17], e.g. (Harel, 1992) and (Cox, 1990), the author argues that Brooks's (1987) argument concerning the nature of software development still holds. Indeed, these essential difficulties constitute part[18] of, in Brooks' terms, the *essential* difficulty in designing a groupware for supporting a group because groupware is one of the species of software. Of particular relevance amongst Brooks's (1987) recommendations for tackling the essential difficulties of software development is the notion of growing software. I shall argue that groupware should be *grown organically* within its production environment. That is, it should be developed by those who use the groupware in the actual environment in which the actual group work is carried out. I will elaborate how this can be done through an Empirical Modelling approach in chapter 7.

2.3 Contemporary groupware development paradigm

This section describes the contemporary approach to groupware development. This includes Tang's (1991) iterative process for studying and supporting cooperative work (cf. §2.3.1), the ethnography-oriented design tradition since 1990s (cf. §2.3.2), and the recent call for increasing the level of user participation, e.g. through Participatory Design (cf. §2.3.3).

2.3.1 The iterative process

Tang (1991) described an iterative approach for studying cooperative work (see figure 2.1)[19]. This approach consists of three stages, i) observe, ii) understand, iii) support. Based on the description in Tang (1991), Borghoff and Schlichter (2000), and other reports in the literature,

[17] In his response to Harel's (1992) questioning of the need to make a distinction between the essential and the accidental, Brooks (1995) argues that it is necessary and "[it] is absolutely central to understanding why software [construction] is hard" (Brooks 1995, p.214).

[18] The other part of the essential difficulty is the socio-technical challenge, which was discussed in section 2.2.1

[19] Initially, it was a research approach for studying collaborative work. John Tang uses his iterative approach to study and develop collaborative drawing tool support for a small group of designers in connection with his doctoral research (cf. Greenburg 2008). While this approach may look over-simplified in the light of other more sophisticated software engineering methodologies, it became popular in the 1990s and most of the groupware development projects nowadays more or less follow Tang's approach. i.e. observe, understand, and support. Greenburg (2008) credits Tang's contribution retrospectively, and refers to Tang's approach as "group-centered" design.

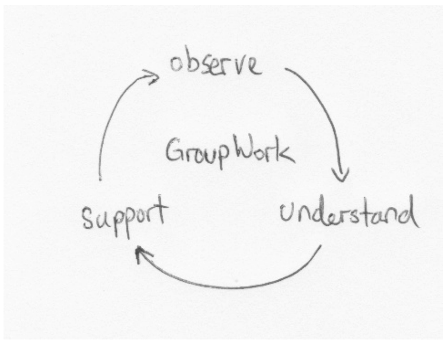


Figure 2.1 – Tang's iterative process for groupware development[20]

these three stages can be characterised as:

i. Observing how people carry out the work – this may involve field studies.

ii. Understanding how people carry out the work – if groupware has been deployed, analyze and evaluate how the group adopts the groupware in its context.

iii. Developing computer tool support – i.e. groupware, for the group, based on the analysis and evaluation.

However, there is no predefined entry point for this approach (Borghoff and Schlichter, 2000). For example, one development may begin with an observation on how the group carries out the work, while another may start from building computer tool support for the group and observe its adoption later. This development process may go through various cycles before it can achieve a satisfactory groupware to support the group (Dourish, 1995). Similar iterative approaches have also been discussed elsewhere, e.g. (Boehm, 1986), in the context of systems development.

2.3.2 Ethnomethodology and Ethnography

Despite the merits of an iterative approach, such as is depicted in figure 2.1, more informative techniques for studying the workspace for groupware are required. Ethnomethodology (Garfinkel, 1967) and ethnography have been gradually adapted in groupware development and other systems designs since 1990s (cf. Anderson, 1994;

[20] This figure is adapted from (Tang, 1991)

Dourish and Button, 1998; Dourish 2001, 2006; Harper 2000). As Dourish and Button (1998) have observed, the use of ethnography became popular for studying workspaces in HCI and CSCW after Lucy Suchman's (1987) work – *Plans and Situated Actions*. It is worth noting that ethnomethodology is not the same as ethnography. While ethnography is a technique that anthropologists use to inquire into human societies (Crabtree 2003; Dourish 2001), ethnomethodology is "a particular analytic orientation to the practical issue of the problem of social order" (Dourish and Button, 1998). Ethnomethodology is often perceived as the same as ethnography because ethnomethodologists often use ethnographical data in their studies, and their differences can only be detected in their "analytic mentality" (Dourish and Button, 1998). Indeed, other social scientists, e.g. activity theorists, may use ethnography as a data collection technique (Crabtree, 2003).

The data generated by an ethnographical investigation can be characterized as situated, qualitative, and open-ended (Dourish, 2006). This is because ethnographical investigation pays particular attention to people as the subject of study, and produces an account – from the investigator's perspective and arguably on a subjective basis – of how people work. The virtue of ethnography is that it not only guides the observation, but it also provides a framework for evaluating and understanding the group work in its context. In relation to, the iterative approach described in section 2.3.1 (also cf. figure 2.1), ethnography naturally fits into the observation stage. For instance, according to Hughes et al. (Hughes et al., 1995), ethnography can play the role of:

i. A 'quick and dirty' technique to get groupware developers to achieve preliminary understanding of the workspace in a relatively short time frame.

ii. A concurrent investigation which is carried out in parallel with the technical development and informs design decisions periodically.

iii. An evaluation technique to study the groupware prototype.

In reality, ethnographical investigations are usually carried out in parallel with the development process (Borghoff and Schlichter, 2000) and it becomes a division of labour (Dourish and Button, 1998). Since ethnographers are typically social scientists and are usually not involved in the groupware implementation at all, the only point of contact between ethnographers and groupware developers is in team meetings at which ethnographers report their findings to the groupware developers, or when ethnographers formulate their findings into some form of requirements. What is more, the ethnographers are usually seen, not as people directly involved in the work process, but as the *spokesman* for the workers (Hughes et al., 1994). In this case, ethnography is arguably not part of the groupware development process at all (Dourish and Button, 1998). The conflict between the ethnographers and the systems developers reflects the divide between two disciplines, social science and computer science, and leads to what Grudin and Grinter (1995) have called "the ethnographer's dilemma"[21]. Dourish (Dourish, 2006) argues that this conflict is due to the lack of a fundamental connection between them[22].

Apart from the practical difficulties, the use of ethnography in groupware development has invited other criticism. Harper (2000) makes the criticism that some researchers are using the term as a buzzword to denote close observation research projects and that how these inquiries were structured was usually unspecified. The descriptive nature of ethnography has also invited criticism. For instance, Kaptelinin and Nardi (2006) criticize the heavy use of text as the main medium for reporting ethnographic studies. They argue that this makes it hard to generalize, and therefore diminishes its value when compared to other theoretical approaches to interactive systems design[23]. There are complaints that ethnography is not generating the "right materials for design" (Harper, 2000). Moreover, ethnography cannot capture the users' crafting skills and their reflection on the work. In discussing an ethnographic study of air-traffic control, Shapiro (1994) reports that the ethnographers failed to reveal the organisational arrangements for air-traffic controllers and the personal attributes that might have affected their work. In the end, there is simply no firm definition of

[21] Dourish and Button (1998) further break down the *ethnographer's dilemma* into the *paradox of system design* and the *paradox of technomethodology*.

[22] Dourish and Button (1998) have a proposal – technomethodology – of how ethnography and systems design may be connected together at the analytic level, instead of at the empirical level (i.e. 'implications for design' (Dourish 2006). Discussion of Dourish and Button's proposal is beyond the scope of this thesis. For the interested reader, the author recommends reading (Dourish and Button 1998), (Dourish 2001, ch. 3), and (Dourish 2006).

[23] Kaptelinin and Nardi (2006) also criticized Garfinkel's position on ethnomethodology as "radical antitheory", which "avoids generalization and abstraction" (Kaptelinin and Nardi 2006, p.17).

what ethnography *is*, particularly in CSCW (Harper, 2000).

2.3.3 Participatory design

Participatory Design (PD)[24] is a collection of principles and practices that is primarily aimed at making computer technologies and social institutions more responsive to human needs (PDC, 2008). In fact, PD has been applied outside computer systems development, e.g. in engineering projects (Grønbaek et al., 1993). Traditional PD practices only cover the design and planning activities before procurement (e.g. Bødker et al., 2004), but modern PD practices have broadened to cover most activities in the entire development process, from early project planning to systems evaluation (cf. Muller, 2001)[25]. PD has become popular in systems development since 1990s. This is due to its promise of active and direct user participation (cf. Bødker et al., 2004). That is, the workers who will eventually use the computer system should be involved in its design process and the decision-making that may affect its workspace (Greenbaum, 1993). This strong commitment is sometimes deemed to be bringing democracy and respect to the workspace and to systems development (cf. Kensing and Blomberg, 1998). The strong commitment in PD has a root in 1970s, where the workers and their unions feared that the introduction of computer technologies would eventually lead to job losses (Kensing and Blomberg, 1998). It was conceived as a political strategy of systems design – a strategy to rebalance the power between the workers and the management (Kensing and Blomberg, 1998).

In the context of groupware development, user participation is seen as a remedy to overcome part of the socio-technical challenge (cf. Mambrey and Pipek, 1999). Karasti (2001) reports a propitious experience of integrating ethnography into PD through, what he called, *appreciative intervention* in a workshop[26] in relation to the development of a teleradiology system. In the workshop, ethnographers, developers, and the workers carried

[24] The detailed history of Participatory Design, particularly the politics involved in introducing computer technologies into the workspace, is beyond the scope of this thesis. See (Kensing and Blomberg 1998), and (Schuler and Namioka 1993).

[25] In fact, there is no standard answer to the question "What is PD?" (cf. Törpel2005).

[26] A Participatory Design technique that is used to co-construct the future work practices with the workers.

out sustained analysis of the workers' everyday practice through pre-recorded video clips. According to Karasti (2001), the main benefits of such approach are:

i. The overall systems development is oriented towards the actual work practices.

ii. The user participation has informed and is intertwined with the systems development, evaluation, and envisionment.

iii. The tension between the developers and the workers is lessened.

iv. The developers were made aware of what is important and must be preserved in the workspace.

Recently, Pekkola et al. (2006) proposed a participatory iterative approach for information systems development[27] based on their experience of groupware development projects. Their participatory iterative approach begins with project definition and has eight stages. After the initial requirements elicitation, they follow an iterative process for prototyping (i.e. analysis and design, implementation, deployment), gather user feedback, and re-analyse the requirements. Although Pekkola et al. (2006) have emphasized user participation in their development process, their approach does not make use of some of the important features of PD, e.g. joint decision-making in the design, future workshops. It also omits the ethnographical investigation that would be included in most groupware development. In this approach, the authors hope to make a blend between PD, an iterative approach, and an ethnographical approach. However, I argue that this blend is not smooth. If PD is intended to be the potential key to tackling the socio-technical challenge, it requires integration of a kind closer to what Karasti (2001) has used in the teleradiology project. Nevertheless, I am going to study this *participation problem* in groupware development in depth in chapter 3, as it is central to the theme of this thesis.

2.4 Accommodating evolving human activity

In this section, I discuss the evolving nature of human activity in groupware development

[27] Compared with CSCW and PD, research into Information Systems Development is more organisation-oriented and its development process is not flexible enough to cope with changing context (cf. Pekkola et al. 2006).

and review approaches and strategies that have been used to make the groupware “softer” to cope with this evolution.

2.4.1 Evolving nature of human activity

There is no doubt that humans are living in an ever-changing world. To adapt ourselves to this ever-changing world, our activities evolve from time to time. In fact, we often change the *activity* to adapt to the situation in which the activity is carried out (Suchman, 1987). As Béguin and Clot (2004) pointed out “activity cannot be reduced to execution procedures applied more or less passively … even the most repetitive movement of a production line worker is always unique” (ibid, p.57). Think of the activity of enjoying a cup of coffee in a café. There may be a limited repertoire of actions that can be carried out in this simple activity, e.g. buying a cup of coffee from the till, sweetening the coffee, finding a seat, reading a book. However, the occurrence of and the sequence of the actions and operations that constitute the activity can be quite different[28]. For example, one may find a seat before buying a cup of coffee from the till, add more milk because there is no more sugar left, read a the same book but start a different page, and so forth. In this example, although the overall organisation of activity is more or less the same, the *detail* in the activity is always different and its actions are always *unique*. This is because actions are situated in the context of the activity (cf. Suchman, 1987). Béguin and Clot (2004) also remind us that, in other circumstances, the organisation of an activity *can* be changed:

> *“Obstacles, disagreements [and] conflicts encountered in activity generate tension ... and invite the subject to mobilize and develop [new forms of organization]” (quoted in Kaptelinin and Nardi, 2006, p.220)*

Situated actions (Suchman, 1987), in fact, are not only limited to everyday life activities – they can also be found in the use of computer technologies. For example, there is some flexibility about the sequence of actions in the activity of sending an email to a co-worker, as long as the send button is clicked in a graphical interface of an email application. In fact, we

[28] From an Activity Theory perspective, an activity is composed of actions, and actions are composed of operations (cf. Kaptelinin and Nardi 2006).

use computer technologies in whatever way is appropriate to our needs, rather than necessarily respecting their *designed* usage. From the systems developer perspective, it is an unanticipated use. Indeed, the level of unanticipated use of computer systems has surprised many systems developers and this has resulted in an emerging research topic in HCI[29].

2.4.2 Evolution of groupware

Groupware cannot be developed and remain *as is* if its aim is to support the evolving nature of human activity. Many concepts and design principles have been developed in the literature to promote software flexibility. Certainly, groupware developers may apply these concepts and design principles to make groupware cope with the evolving nature of our activities. Although the aims of these concepts are similar, I argue that there are subtle differences among these concepts and techniques in the literature in relation to their scope of concern and intervention. I discuss these concepts briefly.

Extension

The idea of designing extensible software is not new. This idea can be traced back to the 1970s when the software industry faced the challenge of changing requirements after the software was deployed. Parnas (1978) reports four common issues[30] in the software systems at that time which led to difficulties in extending or tailoring the software for different needs. Parnas (1978) asked us to consider software as a set of components ("software as a family of programs" in Parnas's terms) and to identify a minimal set of components ("minimal subset" in Parnas's terms) for basic usage, such that additional functionality can be derived from this minimal set.

In view of subsequent development, the idea of making software extensible is closely related to the idea of making software reusable. That is, the software extension which is built on the base system, which in turn can be seen as being "reused". The notion of extension is the

[29] There is a tension between 'curriculum for use' and 'unanticipated use' (Bertelsen, 2006).

[30] Three out of four were related to modularity in software architecture. Unsurprisingly, a lot of novel ideas in this paper have been taken up in structured programming, and in object-oriented programming.

simplest concept to cope with changing requirements. Nowadays, quite a number of applications, e.g. Firefox and Adobe Photoshop, support additional features through plug-ins, i.e. an extension component that can be plugged into the application for additional features. In the context of groupware, this concept of extensibility was exemplified by available architectures, e.g. (Gibbs, 1989; Hummes, 2000). However, this concept can only be used to add new features, but not to rectify major faults in the existing software[31].

Tailoring and Customization

Tailoring can be viewed as the "further development of an application during use to adapt it to needs that were not accounted for in the original design" (Mørch et al., 1998). As Stiemerling et al. (1997) put it, "[t]ailorability is a property of software which allows to change certain aspects of the software in order to meet different user requirements". Another similar concept to tailoring is *customization* (Mackay, 1990). Some researchers view customization as the same as tailoring, e.g. (Wulf, 1999). Others distinguish customization from tailoring. For instance, Bentley and Dourish (1995) see the customization process as consisting of tailoring activities. The concept of tailoring goes beyond the concept of extension. Tailorable software must have an extensible architecture underneath[32], and the flexibility to fulfill the dynamically evolving requirements by adding new features or changing the behaviour of the software is provided to the end-users without rewriting or recompiling the software (Stiemerling et al., 1997). That is, the *tweaking* of the software is shifted to the end-user from the developer. According to Stiemerling et al. (1997), this shift is necessary because "the continuous involvement of systems developers may retard or even impede a necessary adaptation of the software."

In the context of groupware development, tailorability is often seen as a mandatory feature of groupware due to the rapidly changing context in organisational work (Mørch et al., 1998).

[31] This is because the base system is assumed to remain unchanged. While the extension components can be added anytime in the use context, the extensible architecture is designed and developed in the main development process. In a highly modularized operating system, e.g. Linux, it is possible to replace a faulty component without redeveloping the entire system. By definition, this form of component replacement is *patching*, not *extending*, the existing software.

[32] Technically, this is often component-based, e.g. (Stiemerling 1997, Stiemerling and Cremers 1998)

In a workshop on tailorable groupware, Mørch et al. (1998) identify four levels of tailoring activities:

i. Changing parameters of the application.

ii. Changing the composition of the application.

iii. Extending the application with high-level computational mechanisms, e.g. APIs.

iv. Modifying the source code of the application.

Tailoring at the third and the fourth level in the above classification is sometimes viewed as end-user programming (Nardi and Miller, 1990; Nardi, 1993). Arguably, the fourth level of tailoring activities, i.e. modifying the source code of the application, has gone beyond the boundary of "tailoring". I argue that such activity is more appropriately referred to as re-development or maintenance of the application, depending on the nature of such modification. A similar perspective is adopted in (Teege et al., 1999).

Tailoring may also be a recipe for achieving a critical mass of adoption of the groupware, as it allows different members of a group to customize the groupware to fulfill their different personal needs (Bentley and Dourish, 1995). However, as Teege et al. (1999) pointed out, tailoring may redistribute the work and the benefit in a group – an issue that I have discussed in the organisational aspect of the socio-technical challenge (section 2.3.2). In a study of tailorability in a visualization system, Tim Mansfield reported that users might not fancy using advanced tailoring features of an application due to psychological barriers (Mørch et al., 1998).

Appropriation, Adoption, and Adaptation

From a theoretical perspective, the concept of *appropriation* in interactive systems design was borrowed from Adaptive Structuration Theory (Andriessen et al., 2003; Dourish, 2003). From the AST perspective, appropriation concerns the mode or fashion in which the group uses, adapts, and adopts technology in relation to its *structural features* and the *spirit* of those features[33] (Andriessen et al., 2003; Poole and DeSanctis, 1990). However, Andriessen et al. (2003) argue that framing the concept of appropriation from a social-scientific perspective "does not cover the design of customizable or tailorable artifacts which may let use evolve into co-construction."

From a more practical perspective, the concept of appropriation is similar to the concept of customization and tailoring, but it has a broader scope which concerns how people adopt and adapt technology and integrate the customised technology into their context through continue (re-)configuration of the technology. As Dourish (2003) put it:

> *"Appropriation is the way in which technologies are adopted, adapted and incorporated into working practice. This might involve customisation in the traditional sense (that is, the explicit reconfiguration of the technology in order to suit local needs), but it might also simply involve making use of the technology for purposes beyond those for which it was originally designed, or to serve new ends." [ibid, p.463]*

Within this broadened scope, appropriation becomes an inevitable process for groupware adoption and adaptation. As Dourish (2003) pointed out, the success of appropriation not only depends on the flexibility of the technology but also the social practice in which the appropriation takes place. Indeed, adoption and adaptation are determinant factors for the success of failure of a groupware (Grudin, 1994b; Palen, 1997). Dourish (2003) asks us to view the appropriation of technology in relation to communities of practice, a context in which technology plays important roles. To paraphrase:

i. The features of the social-technological system become meaningful to people as they develop understandings about how the representations the social-technological system offer may affect their work and how the representations

[33] Andriseen et al. (2003) write "The use of advanced information technology depends on the features and spirit of the tool, but also on structures provided by the task and the environment. Moreover, when a technology is used, its output – e.g. proposals, new ideas, but also emerging ways of using the technology in interaction – becomes in turn a new source of structures. This is the process of 'structuration', i.e. the production (and reproduction) of emergent social structures. When emergent rules or behaviours are accepted over time they become institutionalised and will determine subsequent behaviour. This process of institutionalising meanings and rules in a group is called 'appropriation'".

relate to other people or entities.

ii. The social-technological system provides a means for people to can interpret and understand others' actions, both implicit and explicit.

Evolution

Evolution is not another buzzword to describe the process of adoption or adaptation of groupware. Instead, it is a perspective at the holistic level that emphasizes a focus on the evolutionary process of groupware rather than giving exclusive attention to the technological imperative perspective or the organisational imperative perspective (cf. Andriessen et al., 2003).

Evolution can be characterized by its emphasis on the gradual and mutual adjustment, both in social and technical aspects, instead of sudden changes which may deliberately be imposed by other factors, e.g. politics in the organization, rather than the actual need derived from day to day work practice (cf. Andriessen et al., 2003). This perspective is compatible with many existing concepts, e.g. adoption, tailoring, customization, incorporation, and appropriation (Andriessen et al., 2003). However, it contradicts Orlikowski and Hofman's (1997) improvisational model for change management for groupware which suggests anticipated changes interleaved with emergent and opportunity changes[34].

Since the evolution of the groupware occurs in a context in which the social, the organization, and individual aspects are all evolving and affecting each other, it has therefore sometimes been viewed as a co-evolution process (Rogers, 1994). I will discuss the co-evolution in more detail in relation to collaborative modelling in Chapter 5.

[34] Orlikowski and Hofman's (1997, p.13) describes anticipated changes as "changes that are planned ahead of time and occur as intended", emergent changes as "changes that arise spontaneously from local innovation and that are not originally anticipated or intended", and opportunity-based changes as "changes that are not anticipated ahead of time but are introduced purposefully and intentionally during the change process in response to an unexpected opportunity, event, or breakdown".

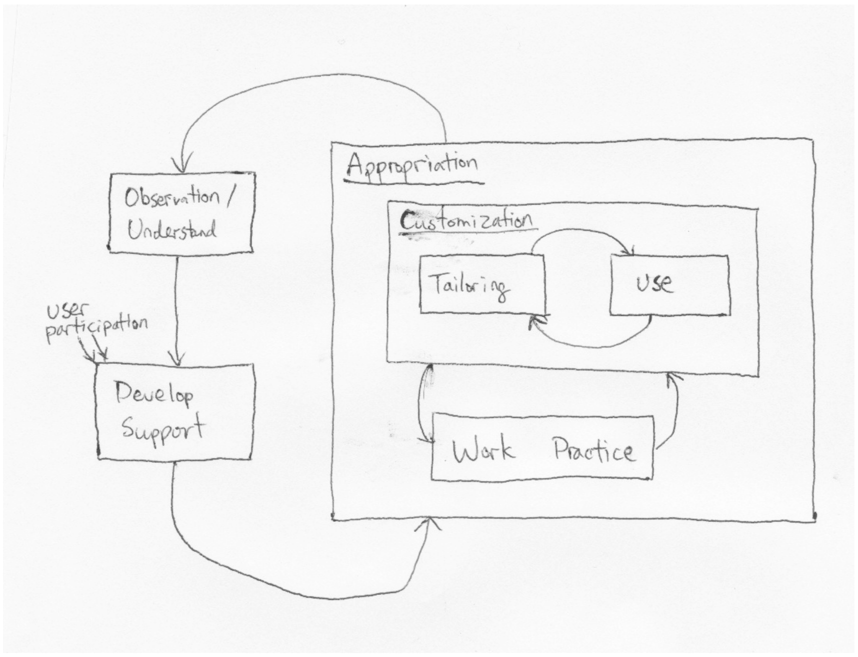


Figure 2.2 – A model for groupware evolution

A model for groupware evolution

To help the reader to understand what I meant by groupware evolution in relation to the existing concepts in the literature and in the context of this thesis, I have developed a simple model which integrates all the aforementioned concepts, i.e. tailoring, customization, appropriation, adoption and adaptation, into a coherent model. This model takes inspiration from the iterative process to groupware development proposed by Tang (1991) that I discussed in section 2.3.1, the evolutionary process described in (Wulf and Rohde, 1995), and the STEPS process model (Floyd et al., 1989).

Figure 2.2 illustrates how these concepts can be put together and shows the scope of their concerns. Individual members of the group can tailor the groupware to her local context of use. However, the local context of use may change from time to time, therefore may require further tailoring to suit the evolving needs. All these tailoring activities are circumscribed within the customization process[35]. Individuals can customize the groupware according to the communities of practice to which they belong. As in tailoring, there is a continuous

[35] MacKay (1990, p.23) asserts that "Customizations are preserved in a form accessible to users, which may be shared". This suggests that MacKay considers customization to be an individual process rather than a collective process.

interplay between customization and the working practices. According to MacKay (1990), individuals' customizations can be shared and therefore can be viewed as a collaborative activity within the organization. All the customizations take place within the context of appropriation, and the appropriation is carried out at the group level.

By integrating all these concepts into a coherent framework, I do not intend to disregard or be disrespectful of the theoretical grounds or traditions behind accepted concepts, such as Adaptive Structuration Theory behind the concept of appropriation.

2.5 Towards efficacious groupware development

So far in this chapter, I have studied the challenges in groupware development – i.e. why groupware is so difficult to develop compared to other species of software. I have also discussed a number of concepts and techniques that attempt to make groupware easier to adapt to the human context and enable it to cope with the evolving nature of group work – i.e. the changing context of group work in different aspects, which includes the social, the organizational, and the work practices.

In the final section of this chapter, I argue that an approach to groupware development that is *efficacious* must adopt a human-centred approach[36]. Section 2.5.1 discusses respects in which current approaches to groupware development fail to be human-centred. In section 2.5.2, I then critique the notion of participation in the current participatory approaches to systems development. I use the term "efficacious" here in the sense in which it is used by the French philosopher François Jullien. Jullien (1995) adopts the term in relation to his lengthy discussion of the interpretations of the Chinese word *shi* (勢) as *efficacious disposition* in his inquiry into ancient Chinese philosophy. I will briefly discuss this concept in section 2.5.3.

[36] By emphasising a human-centred approach, I do not mean to discard the group-centred approach to groupware development that we discussed in section 2.3. Instead, I regard human-centred thinking as transcending group-centred thinking.

2.5.1 From group-centred to human-centred for groups

As I discussed in section 2.3, there is a strong connection between the development of user-centred design and the development of the group-centred approach to groupware development. Research into user-centred design in HCI shares many common themes with research into groupware development, e.g. social aspect and the appropriation process. To some extent, the group-centred approach addressed many concerns regarding the limited scope of the notion of *user interface* that the first generation HCI did not take into account (cf. Grudin, 1990; Bannon, 1992b). It was in this context that the research field CSCW received much attention.

The major difference between user-centred design and the group-centred approach is their scope – the group-centred approach pays particular attention to the interaction between group members. In this sense, groupware development may follow an expanded-scope user-centred paradigm. More importantly, current group-centred approaches may share a common perspective on *user* as the term is interpreted in user-centred design. This perspective puts it emphasis on *a human as a technology user in a particular domain* rather than *a human who has a felt life* (cf. Gasson, 2003; Bødker, 2006; McCarthy and Wright, 2004). This narrower perspective precludes developers examining the issues that are central to human-centred design (Gasson, 2003). According to Gasson (2003), human-centred design promotes the following features:

1. Flexible computer technologies that cope with people and allow them to manage and to shape their work.
2. Priority given to human communication and collaboration over computer-based information processing, which is the opposite of the traditional technology-oriented approach.
3. Computer technologies that allow the use of implicit knowledge, i.e. tacit and skill-based, together with the explicit, e.g. rule-based, knowledge. If the computer technologies merely support explicit knowledge and not its implicit counterparts, the implicit knowledge becomes marginalised because the explicit knowledge is no longer contextualised (Gasson, 2003).

4. The design of computer technologies that address the normative expectation of the technology (cf. Kuhn, 1996). Computer technologies are intertwined with the social expectations that circumscribe the design of the technologies.

5. Designs that gives joint consideration to the questions of "What is technically feasible?" and "What is socially desirable?"[37] (cf. Kuhn, 1996).

6. Socio-technical computer technologies that support meaningful and enriched work (cf. Gill, 1991). This precludes the separation of the planning tasks from the doing tasks, since such separation may deskill those workers who are ill-equipped for meaningful decision making or exception handling (cf. Cooley, 1987).

Approach	Intended Focus	Actual Focus
Traditional IS design approaches	The structuring of ill-structured problems: goal-driven decomposition.	Explicit (management) focus is goal-driven and decompositional. Implicit strategies are opportunistic, to deal with goal-emergence.
Prototyping and participatory design	Negotiation and exploration of ill-structured problems.	Iterative and cyclical process of stakeholder involvement, limited by political selection of user-representatives and technology-centered requirements focus.
Interaction design	Exploration of IT-supported user work-processes.	Technology-centered, individual user focus. Assumes consensus among system users, with well-understood IS goals.
UML and Use-Cases	Modeling of business processes and user-interactions with intended IT system.	Models formal information-processing (business processing rules). Technology-centered and decompositional (so no opportunity to redefine goals as these emerge through design process).
Agile Software Development	Adaptation of an evolving system design, based on user interaction and scenario generation.	Technology centered prototyping, accomplished by the development of individual user-scenarios.

Table 2.3 – Summary of the intended focus and actual focus of contemporary systems development approaches[38]

[37] The socio-technical gap, as described by Ackerman (and as discussed in §2.3.2), points to a similar but larger divide. Instead of “what is socially desirable?”, Ackerman asks “what must be supported socially?”.

[38] This table is adopted from (Gasson 2003)

Based on the above principles, Gasson (2003) examined five contemporary systems development approaches[39] from an Information Systems Development (ISD) perspective, and concluded that contemporary systems development approaches put too much emphasis on the technology-focused perspective and do not satisfy human-centred design principles fully (cf. table 2.3). Gasson (2003) argues that:

> *"… each of these approaches focuses on user-centeredness at the expense of human-centeredness, because of their technology-centered focus … system stakeholders -- the intended "victims and beneficiaries" of the proposed information system (Checkland, 1981) – should be enabled to negotiate the role and purposes of the system with other stakeholders, non-technical as well as technical." (Gasson, 2003, p.39)*

As Gasson (2003) pointed out, this idea is not new – it was part of the early movement of PD (cf. Mumford, 1983). However, it has been difficult to implement due to the goal-driven and technological-focused traditions in ISD (Gasson, 2003). Gasson (2003) further argues that:

> *"most user-centered approaches are concerned with closing down a technology-centered and goal-directed IS problem-definition, not about exposing (or opening up) the social and organizational context (the design "problem") to examination and debate … The most human-centered of the methods discussed, participatory design approaches, do not change the fundamental nature of the "circular" system development life-cycle. They merely "rotate" the life-cycle through 90°, so that the cycle is driven by user-evaluation of system design requirements, rather than by technical evaluation of system design requirements. This rotation does not question many of the essential contradictions of the traditional perspective, because it inherits the "problem closure" life-cycle emphasis." (ibid, p.39)*

[39] These include Participatory Design, two varieties of agile development (Adaptive Software Development and Extreme Programming), use cases in UML, and the "Interaction Design" paradigm in HCI. However, Gasson (2003) did not mention the group-centred approach that we discussed in section 2.3.

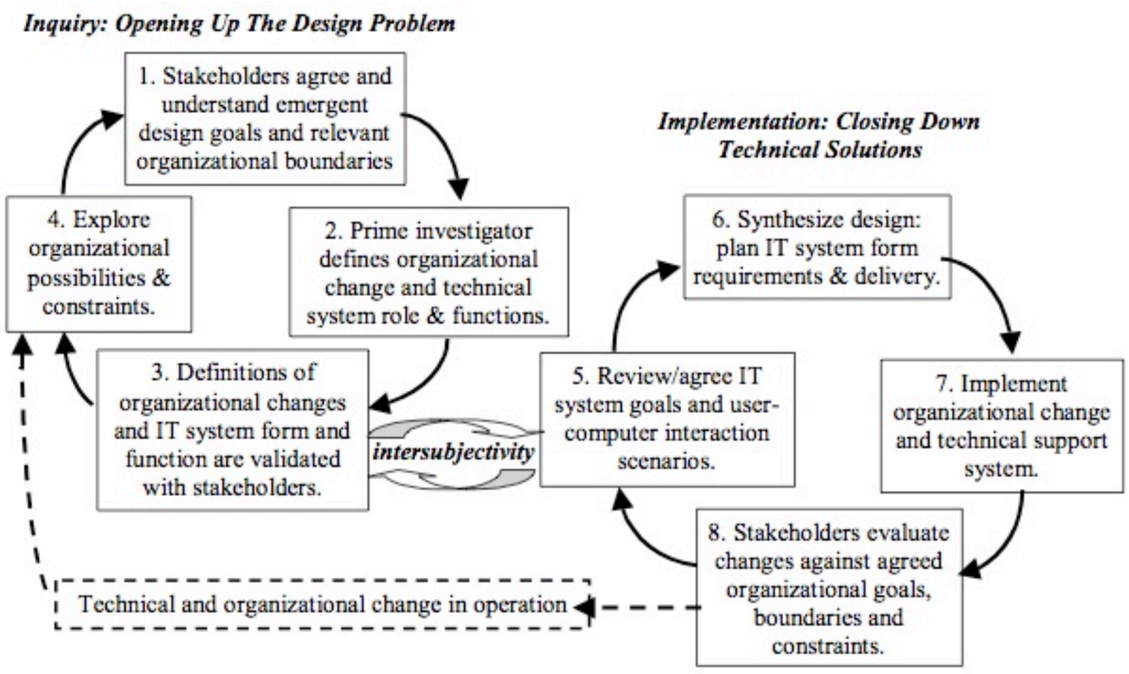

Inquiry: Opening Up The Design Problem
1. Stakeholders agree and understand emergent design goals and relevant organizational boundaries
2. Prime investigator defines organizational change and technical system role & functions.
3. Definitions of organizational changes and IT system form and function are validated with stakeholders.
4. Explore organizational possibilities & constraints.
intersubjectivity
Implementation: Closing Down Technical Solutions
5. Review/agree IT system goals and user-computer interaction scenarios.
6. Synthesize design: plan IT system form requirements & delivery.
7. Implement organizational change and technical support system.
8. Stakeholders evaluate changes against agreed organizational goals, boundaries and constraints.
Technical and organizational change in operation

Figure 2.4 – Gasson's dual-process model[40]

To satisfy the human-centred concerns, Gasson (2003) proposed a dual-process model (cf. figure 2.4), which is based on a case study of the systems development in an engineering firm. Gasson (2003) claims that the model "represents an optimal way of managing the dialectic between subjective, organizational problem inquiry and goal-directed, process and technical solution design." The two cycles, "opening up design problem" and "narrowing down potential solutions" can be carried out inter-changeably to cater for design inquiry and technical implementation. In contrast to the traditional waterfall model for ISD, Gasson's (2003) dual-process model fills in the missing pieces of the "systems development puzzle" with the "opening up" cycle (on the left) that is aimed at guiding the "pre-requirements" activities. However, I argue that the dual-process model merely represented the system development process of the engineering company where the study was carried out, and, at best describes a process that caters for, addresses, and balances the concerns raised from both the user's and the developer's worlds. It hardly convinces one to believe that this model can address the human-centred concerns or be applicable to other forms of systems development, such as groupware development. From my perspective, the main concerns

[40] This figure is adopted from (Gasson1998)

with Gasson's proposal are:

i. In stage 2 of the "opening up" cycle, the prototype of the organisational and technological change is produced by the principal investigator. Although other stakeholders may have been consulted in the first instance (as depicted in figure 2.4), the orientation of this "opening up design problem" process suggests that the inter-subjective design perspective is vulnerable to being overridden from a managerial perspective. This is somehow contrary to the principles of PD and Gasson's initial argument for human-centredness.

ii. The non-technical stakeholders, e.g. workers and their managers, can only discuss or review the systems implementation at the transition stage (stage 3 to stage 5 and *vice versa*). It is also the only point of contact between the technical and non-technical teams. Further, the discussion and review around the design goals and boundary between the technical and non-technical teams is likely to be carried out at managerial level in practice. The dual-process does not pre-empt such "representative meetings".

iii. The technical development process seems to be hidden out from the scene. It more or less follows a mini-waterfall model, i.e. the development process looks like a black box process. The dual-process model seems to promote minimal contact between the developers and the non-technical stakeholders. It does not promote an ongoing interaction between these two parties, which a human-centred approach should promote.

To some extent, the CSCW and groupware development research community has gradually taken the aforementioned human-centred concerns into consideration over the last twenty years. However, as Gasson (2003) observed in the IS and HCI literature, the CSCW literature rarely discusses how these concerns are considered, especially in a holistic view at the methodology level, during the development of groupware. Experience from field studies usually reports the scattered results but does not report how they actually made it work.

The problem with the group-centred approach, or user-centred approach in general, is not with its promises, its philosophy, or its intended outcome. The *real* problem is the over-emphasis on the organisational and technical goals rather than the emphasis on each stakeholder's *felt-life* (cf. McCarthy and Wright, 2004). Because of this misaligned emphasis, many so-called user-centred, group-centred, and human-centred approaches fail to live up to their billing. If these approaches cannot fulfil their promise, they may become no more than labels for research funding.

Although Gasson's dual-process model potentially violates some basic tenets of human-centred design, there is little doubt that the model has opened a debate about whether the "pre-requirements" activities (in the traditional ISD sense) should be considered as part of the systems development. Indeed, the human-centredness considerations (especially the bridging between the technical and the non-technical worlds) behind Gasson's dual-process model are worth reiterating, and group-centred approaches should learn from these principles. In the next section, I discuss the notion of participation in relation to the human-centredness concern.

2.5.2 The notion of participation

User participation is thought to be a remedy for relieving the tensions between stakeholders (e.g. developers, workers, and managers) of the computing technologies, and for guiding the development towards a successful direction. It is suggested that user participation can increase the democracy of decision-making during the development process (Wulf and Rohde, 1995) and it can improve the relationship and the mutual understandings between stakeholders (Karasti, 2001; Béguin, 2003; Cluts, 2003; DePaula, 2004). From the developer's perspective, this increases the likelihood of getting the user to accept unanticipated changes, when that happens, during the systems development. From the user's perspective, this results in a 'better system' because of their participation (Pekkola et al., 2006). Although early studies (e.g. Cavaye, 1995; Olson and Ives, 1981) argued that there was insufficient evidence to establish a positive correlation between user participation and system success, recent studies do show a positive correlation between user participation and the likelihood of system acceptance and adoption, both of the design and the actual system (Woods, 2002; He and King, 2008), which is a critical factor in determining the success and failure of groupware development (Grudin, 1994a). Indeed, user participation is a key component in the principles of human-centred design (cf. Gasson, 2003). However, user participation alone may not be enough to ensure the success of a systems development project (He and King, 2008). Some researchers even suggest that user participation can increase the complexity of managing the systems development project, and may result in its abandonment if user participation is not properly managed (Gasson, 1999; Howcroft and Wilson, 2003). Howcroft and Wilson (2003) argue that, in the worst case, the systems development process becomes an arena for negotiating private interests. For instance, it is possible that users abuse the participatory opportunity to realise crazy ideas or want to explore ideas so ambitious as to be outside the scope of the systems development. For this reason, some researchers have suggested that the intervention of the users is problematic and should be avoided in systems development (cf. Howcroft and Wilson, 2003). Despite the concern that user involvement may invite political conflicts among stakeholders, it is generally believed that user involvement may increase the overall satisfaction of systems development.

The degree of *user participation* varies in different styles of systems development, and in different projects: from ‘participation’ merely in requirement analysis, to making joint design decisions, to joint appropriation of the system, to end-user development. It is surprising that research into user participation has proliferated so rapidly without agreement on a precise definition of the concept. Perhaps the most satisfactory *definition* of user participation is given by Barki and Hartwick (1989), in which the authors distinguish *user participation* from *user involvement* in systems development. Barki and Hartwick (1989) suggest that *user participation* should be used to refer "the behaviors and activities that the target users or their representatives perform in the systems development process" (ibid, p.59) and *user involvement* should be used to refer to "a subjective psychological state of the individual and defined as the importance and personal relevance that users attach either to a particular system or to IS in general, depending on the users’ focus" (ibid, p.59-60). With this distinction between two concepts, it is possible that a user is participating but does not necessarily feel involved in the systems development process, no matter whether the participation is voluntary or forced (Barki and Hartwick, 1989).

In order to get the users to enrol in the development process, systems developers often emphasize its human-centred aspect (Howcroft and Wilson, 2003). Howcroft and Wilson (2003) write:

> *"The promise of empowerment is to give employees more power to use their judgement and discretion in their work, thereby encouraging them to utilise their skills and experience for the benefit of the organisation." (ibid, p.9)*

Genuine participation implies a degree of democratisation by way of equal opportunities and responsibility. In the context of systems development, this implies high degree of joint ownership and joint responsibility for the system and its development process among all stakeholders. However, this is easier said than done. As Howcroft and Wilson (2003) write:

> *"User participation, because it brings together a broad range of actors with varying interests, potentially carries with it more latent conflicts than other forms of systems development." (ibid, p.15)*

Research into user participation in systems development (e.g. Howcroft and Wilson, 2003) suggests that, apart from users' attitudes and methodologies, *structural constraints* in the organisation play substantial roles in the business of participation. Howcroft and Wilson (2003) observed that user participation is often being used as a managerial token gesture for various non-technical purposes, e.g. to promote their humanitarian image or to enrol users' commitment to the organisational changes early in the systems development project. When this tokenism exists, participating users are often carefully selected according to compliance to the managerial perspective (Gasson, 2003; Howcroft and Wilson, 2003). This is because the managers, not the users or the developers, often have the power to initiate user participation, to determine its extent, and to select which approach adopted (Howcroft and Wilson, 2003).

Apart from the imbalance of power distribution in systems development, the mismatch of skills and expertise between developers and users also contributes to the difficulty of achieving genuine participation (cf. Howcroft and Wilson, 2003). Despite the fact that a number of approaches have been developed to promote a higher degree of user participation with this mismatch in mind, however, none of them comes close to fulfilling all the human-centred principles that I discussed in section 2.5.1. In her NordCHI 2006 keynote speech, Susanne Bødker (2006) remarked “the human actor needs to step out of the role as worker in a particular practice, and participate in design as a person who brings her entire life to the design.” When considering how far the current implementations of PD fall short of Bødker’s ideal notion of participation, we are led to question whether it is the current approaches that are ineffective or whether the conceptual constructs that underlie current approaches (or more generally, modern systems development) are appropriate for promoting genuine participation.

2.5.3 The efficacious disposition perspective

Earlier in this chapter (in section 2.4.2), I synthesized an evolutionary process model to describe groupware evolution. This model is, in fact, compatible with an emerging perspective in interactive product design, and in HCI and CSCW in general, which argues that the design process does not stop when the piece of computer technology is handed over to its consumer (cf. e.g. Henderson and Kyng, 1992). In this sense, the boundary between design and use become blurred as the consumer will continue to *design-in-use* (Henderson and Kyng, 1992). That is, the role of the computer technology is continuously shaped through continued reconfiguration during its use. Consequently, designs of interactive products have aimed at providing flexibility for their consumers to *appropriate* its use rather than prescribing use (cf. Suchman, 2007).

My perspective on groupware development here is similar to, though different from, the *design-in-use* interactive product design perspective. My perspective is neither that associated with professional software development (in the sense of traditional software development) nor that associated with end-user software development. It transcends the notion of end-user software development but not to the extent that it encompasses professional software development with semi-formal (e.g. UML) or formal methods (e.g. B-method) (this will be explained in more detail in chapter 7). While I agree that the scope of the *design* process of groupware has now been extended into the use context, I argue that the entire artifact cycle should be thought as a process in which some or all of the stakeholders are continuously searching for *efficacious dispositions* for the artifact and its stakeholders throughout the artifact's life.

In his book "The propensity of things", the philosopher François Jullien describes *efficacious dispositions* as one of the meanings of the Chinese concept *shi* in the context of ancient Chinese philosophy. He argues that the Chinese interpretation of reality "appears to proceed through the understanding of the disposition of things" (Jullien, 1995). Jullien then writes[41]:

> *"One starts by identifying a particular configuration (disposition, arrangement), which is then seen as a system according to which things function: instead of the explanation of causes, we have the implication of tendencies. In the former, one must always find an external element as an antecedent, and reasoning can be described as regressive and hypothetical. In the latter, the sequence of changes taking place stems entirely from the power relations inherent in the initial situation, thereby constituting a closed system: in this case we are dealing not with the hypothetical but with the ineluctable. In the context of natural phenomena and in first philosophy, this ineluctability of tendency can be expressed by the term shi, translated as either "tendency" or "propensity" depending on the word chosen by the first Western interpreters of Chinese thought as they tried to convey its originality." (Jullien, 1995, p.221)*

Therefore, obstacles (including social, organisational, and technical) and the changing nature of these obstacles in groupware development become "natural things" and "natural occurrence" when viewed through the lens of the *efficacious disposition*. The focus of the groupware development is then shifted to finding a suitable configuration for the situation by configuring and arranging existing ingredients in the situation from time to time, taking the

[41] Jullien's (1995) work was originally published in French. The quote used here is taken from the English translation version by Janet Lloyd. In her translator's note (Jullien 1995, p.9), Lloyd expresses the difficulty in translating the French word *dispositif* into English, where in turn Jullien had also mentioned his difficulty in translating the Chinese word *shi* (勢) (ibid, p.16). According to Lloyd, "*Dispositif* refers to the efficacy of a *disposition* (whether strategic, aesthetic, etc.), its capacity to function spontaneously and inexhaustibly. Every configuration or disposition possesses an inherent potential or propensity that is fulfilled by the *dispositif*. In other words, a *dispositif* is a "setup", a colloquial expression to which I have often resorted in preference to longer, heavier phrases such as those above." (ibid, p.9)

evolving nature of all these ingredients into account. All activities in the groupware development can then be viewed as being concerned with *efficacious dispositions*, including both technical and non-technical activities.

The efficacious disposition perspective does not attempt to draw a hard boundary between the context of development and use, and does not divide the roles between stakeholders. Instead, this perspective blends the contexts and roles of the stakeholders in relation to the *development* of the artifact (i.e. groupware in the context of groupware development). The treatment of stakeholders in this perspective is not the same as in traditional software development, but closer to that in Participatory Design, which views *user* as "categories describing persons differently positioned, at different moments, and/or with different histories and future investments in the project of technology development." (Suchman, 2007, p.278-279) Furthermore, the efficacious disposition perspective is human-centred. It puts its emphasis on the support for meaningful and enriched human work by disposition, that is by (re-)configuration, of objects within the context of work. This shifts the focus from development as an exercise in prescription to development as exploration of the most *efficacious* configuration for the work.

In the next chapter, I discuss the role of developers and users, their relationships, and the conflation of development and use contexts due to their role-shifting behaviour. I also reformulate the notion of participation and the development process upon which the rest of this thesis will be based.

Chapter 3 The participation problem: roles, participation, and the conflation of contexts in groupware development

In the previous chapter, I have discussed the issues and challenges surrounding groupware development. I have explained that Brooks's (1987) "No Silver Bullet" argument has particular relevance for groupware development as a special branch of software development. Based on Brooks's argument, the most efficacious groupware should be developed by the group of people[42] who will eventually use it, where the groupware *grows* until its final disposal. This implies that efficacious groupware development requires incremental development and continuous *fine-tuning* similar to that in crafting (cf. Brooks, 1987) in order to provide adequate support to the evolving contexts[43]. This *crafting* process may be continued in the use context. In this sense, neither the activity of coding nor the activity of use can be used to characterize the *stage* of the evolving groupware. In other words, the context of development and the context of use are conflated.

The conflation of the development context and the use context challenges the traditional concept of participation in groupware development through pre-conceived roles. In this conception, the roles of the participants are usually assumed unchanged throughout the

[42] Although this usually refers to end-users, the group may consist of other stakeholders such as managers and system developers.

[43] The evolving contexts take account of environment, human activity, the understanding of the developers, users and other stakeholders.

development and participation usually refers to end-user participation in the developer's space. In a conflated context, however, the concepts of role and participation are blurred. The roles of participants are usually dynamic throughout the development and participation may be conceived as multi-directional. It is the case that the end-user participates in the developer's space, but it is also the case that the developer participates in the end-user's space. In fact, the conflation of contexts intertwines with the phenomenon of dynamic roles. On the one hand, the conflated context seems to promote dynamic roles by allowing participants to interact with the evolving groupware in various ways, such as to construct, to (re-)configure, to observe, or to use[44]. On the other hand, dynamic roles will also cause conflation of the contexts. This can happen when the participants are developing and using the evolving groupware in the same shared context.

In this chapter, I discuss the issues surrounding the conflation of the context of development and the context of use, the concept of role and participation, role-shifting phenomenon in systems development, and the impacts of these issues and conceptual variations on groupware development. In section 3.1, I examine the way in which the developer's understanding and the artifact co-evolve. This co-evolution phenomenon has been noted by Naur (1985a) and has been discussed among activity theorists in interaction design, e.g. Kaptelinin and Nardi (2006). It also exemplifies reality construction that Floyd (1992) has observed in software development. The co-evolution phenomenon suggests that the development of the artifact is intertwined with the developer's understanding of the problem situation. Therefore, granting equal opportunity to all participants to *craft* the groupware may enhance its fitness for the evolving context and enhance the understanding of the participants.

Developer and user are two crucial roles in the evolution of groupware. However, the concepts of developer and user remain controversial. In the traditional conception, the role of an individual in groupware development is more often determined by their organisational

[44] There are many different ways to interact with the evolving groupware, e.g. observing the changes being made by other participants at real time, learning about the groupware by exploring without an explicit goal, etc. From a software engineering perspective, the "construction" task can further be broken down into coding, testing, debugging, evaluating, etc.

roles rather than by their capabilities. This leads to the situation in which individuals usually play a single, static and separate role throughout the groupware development in the context of a high division of labour. In practice, it is possible for a *user* to do the programming task if she is equipped with the programming knowledge. In section 3.2, I look critically at the concepts of developer and user. I argue that the tension between developers and users is caused by the assumption that an individual can only play a single and static role. I then turn to discussing the possibilities and the issues of developer-as-user and user-as-developer.

Theoretically, an individual rarely plays multiple roles in groupware development because this requires quite different sets of skills and knowledge. In practice, however, it is not uncommon for individuals to have to shift their roles in different situations and at different times. In section 3.3, I report two examples of systems development where this role-shifting behaviour was observed, one by Friis (1998) and the other by Danielsson (2004). I argue that this role-shifting phenomenon should not be viewed either as unstable, temporary, or exceptional behaviour of individual participants. In these examples, a number of participants shifted their roles from passive users to active users, and from active users to pseudo-developers.

In section 3.4, I revisit the notion of participation (see section 2.5.2). I argue that the implicit assumption behind the concept of participation – as the term in used in PD – that the *participants* are non-developers – is problematic when individuals can shift their roles throughout the development process. In fact, the development process should be viewed as a co-construction process where individuals' understanding may co-evolve with the groupware throughout its development and evolution.

In section 3.5, I summarise this chapter, highlighting the implications for developing an *environment* for groupware development, drawing on the discussions from section 3.1 to section 3.4. I argue that the traditional concept of role and participation may be an obstacle to realising efficacious groupware development. To remedy this situation, we may want to consider an alternative conception of role and participant in a conflated context of development and use. This chapter concludes with a discussion on the implications of these changes for the existing conception of groupware development.

3.1 The co-evolution between the developer's understanding and the artifact[45]

It has been suggested that there is a close connection between the developer's understanding that is associated with the software development process and the construction of the actual artifact, i.e. the software under construction. Peter Naur (1985a) was one of the first researchers to acknowledge the importance of recognising such a relationship:

> *"a vital issue is the proper view of the development process considered as an interplay of intuitive knowledge had by the programming person and the real world accessible to that person, which includes the texts used or produced by the person." (Naur 1985a, p.73)*

As figure 3.1 depicts, the developer's intuitive knowledge will gradually evolve in relation to the development of artifacts throughout the development. In a software development project, the developer begins with her initial knowledge (K1) of the real world (W). As the development progresses, the developer's knowledge of the problem domain and the artifact will gradually be improved (K2) as the artifact is being developed (A1). At the same time, the artifact (i.e. the software and any documentation describing the software including any specifications and manuals) is actually developed within the real world. Further development will bring the artifact into a more complete state (A2) and the developer will achieve a better understanding of the artifact (K3). Although the development of the artifact may be restricted by specifications[46], there is no limit to how far the developer's understanding may be developed. Indeed, this co-evolution occurs no matter which methodology for software development is adopted (Naur, 1985a).

[45] Part of the argument in this section has been presented in a paper published in The 20th Annual Psychology of Programming Interest Group Conference 2008 (Beynon et al. 2008).

[46] The scope of the artifact might be limited in the context of traditional software development, but is subject to no limits if the development adopts an evolutionary perspective, as we discuss in section 2.4.

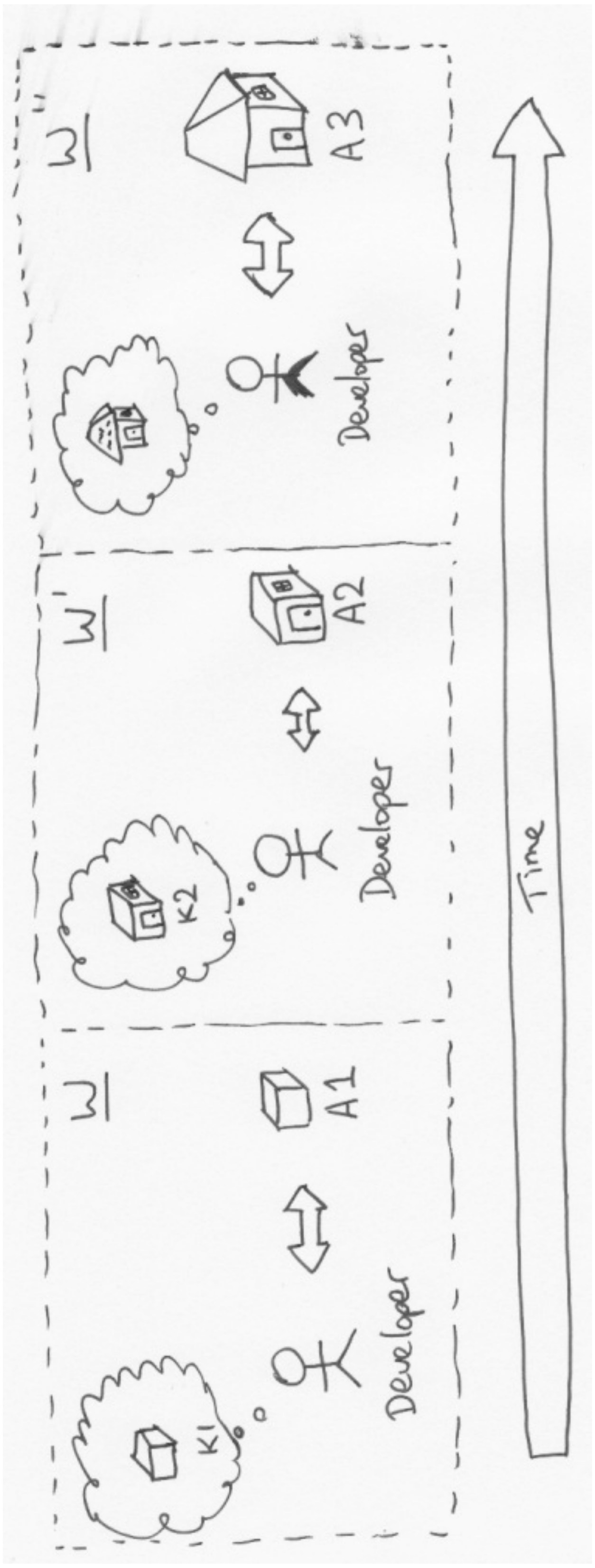


Figure 3.1 – The co-evolution between the developer's understanding and the development of an artifact

For Naur (1985a), the interplay between the developer's understanding and the artifact produced by the developer is driven by human intuitive knowledge. Following the above discussion; in order to grasp K2, the developer has to relate her knowledge of the artifact A1 to her knowledge of the world, i.e. part of K1. This interplay only makes sense if it is "understood as elaborate, purposeful actions undertaken by the programmer, depending on intuitive insight at every turn" (Naur, 1985a).

However, there is a subtle difference between what can and cannot be regarded as intuition. According to the Oxford English Dictionary (2008), *intuition* refers to "The immediate apprehension of an object by the mind without the intervention of any reasoning process"; "Immediate apprehension by the intellect alone"; "Immediate apprehension by sense"; or an instance of "Direct or immediate insight". What Naur (1985a) refers to as intuition here is more closely matched to immediate apprehension without the intervention of any reasoning process, by the intellect alone, or direct or immediate insight, rather than merely an "immediate apprehension by sense". Indeed, what we can apprehend immediately is highly affected by experience and training, i.e. what we already know (Beynon et al., 2008). Therefore, the extreme forms of guesswork with minimal understanding that novice developers may resort to might not involve intuition in this sense – such activity might not involve significant "apprehension of the program by the mind" (Beynon et al., 2008). In contrast, a purposeful trial-and-error activity of an experienced developer might be guided by "apprehension by the intellect alone".

Naur (1985a) considers the potential hazards of human intuition. On the one hand, Naur is concerned with ways of avoiding the flaws of human intuition through the use of methodologies and techniques. On the other hand, he argues that formal methods do not guarantee the absences of flaws and acknowledges the importance of human intuition:

> *"The claim is often made that certain forms, or formalizations, will guarantee the absence of flaws of arguments. What seems to lie behind such claim is the fact that by the use of certain kinds of formalizations it is possible to formulate the connections between statements corresponding to a proof in terms of rules for manipulating the statements. While this property is of great*

> *interest as a matter of principle, and also is the necessary basis for mechanical proof construction and verification, and occasionally is used in the reasoning carried out by people, it provides no guarantee for the absence of flaws in the arguments used in software development making use of formalizations … Avoiding flaws in that modelling undoubtedly depends to some extent on the form or language used for the description. However, what is the most suitable form or language for any situation will depend not only on the degree of formalization of the description, but equally on the character of the aspect of the world to be modelled and the background experience of the programmer who has to establish the description. " (Naur 1985a, p.77)*

From Naur's perspective, the involvement of human intuition is indispensable in software development. Naur (1985a; 1985b) observes that the developer's intuitive knowledge, i.e. the developer's understanding, is so pervasively involved in software development that it has only attracted researchers' attention when it is exposed in exceptional circumstances, e.g. failure as a result of acting on intuitions. Naur (1985a) also observes a difficulty in discriminating developer's intuitive knowledge while acknowledging that the developer's knowledge "at any time is an indivisible whole". This view is closely connected to the outlook of William James (1912; cf. Beynon et al., 2008). Furthermore, such a kind of knowledge can hardly be formulated into rules, as such rules further require other sets of rules to explain how to apply them and so forth, and this becomes "an infinite regress" (Naur 1985a; 1985b).

In another work, Naur (1985b) proposes a "programming as theory building" perspective on software development[47]. Unlike Turski and Maibaum (1987), who adopted a formal conceptualisation for their theory building perspective, Naur adopts a 'softer' notion of theory, as developed by Gilbert Ryle (1949). Naur (1985b) explains briefly that, in Ryle's sense, "a person who has or possesses a theory … knows how to do certain things and in

[47] Naur (1985b) views programming embracing "the whole activity of design and implementation of programmed solutions", i.e. the entire lifecycle of software development.

addition can support the actual doing with explanations, justifications, and answers to queries, about the activity of concern." This means that the developer who builds the software possesses a *knowledge that enables her to perform activities* that transcend what is captured in other document-based artifacts (Naur, 1985b). According to Naur (1985b), these activities may include:

i) relating the software to the affairs of the world which will be supported by the software;

ii) explaining and justifying each part of the software; and

iii) responding to any demand to change the software *constructively*.

As stressed by Naur (1985b), this notion of theory is not the same as what is often called *tacit knowledge*, nor is it the same as intuitive knowledge such as I discussed earlier in this section. While the former is closely related to the implicit knowledge of its design rationale and the structure of software, the latter involves a more general insight into the program.

The most interesting thing about Naur's theory building perspective is that it emphasizes the development of the theory, which has "primacy" over other document-based artifacts such as specifications and user manuals (Naur, 1985b). This also contributes to the growth of the developer's intuitive knowledge of the software. As Naur argues, "an essential part of any [software], the theory of it, is something that could not conceivably be expressed, but is inextricably bound to human beings", therefore, "the proper, primary aim or programming is, not to produce programs, but to have the programmers build theories of the manner in which the problems at hand are solved by program execution." (Naur, 1985b). This puts Naur's theory building perspective in a position that is highly contrasted with the product development perspective, where the primary focus is on documentation and the software itself, and where the development is thought of as guided by a prescriptive specification. To some extent, Naur's theory building perspective conveys the need to attend to the social issues and the situated nature of software development – which are not popular concerns in software engineering, but are topical in research areas such as PD, CSCW, and HCI. These concerns are echoed in Brooks's (1987) *growing* software metaphor and Floyd's (1992) notion of *reality construction*. Floyd (1992, p. 95; cited in Rönkkö, 2007, p.689) argues that:

- *We do not analyze requirements; we construct them from our own perspective. This perspective is affected by our personal priorities and values, by the methods we use as orientation aids, and by our interaction with others constructing requirements from their perspective. Requirements are governed by perspective. In most cases they reflect divergences in perspective and are subject to temporal changes.*
- *We do not apply predefined methods, but construct them to suit the situation at hand. There are no such things as methods per se – what we are invariably concerned with are processes of situative method development and application. We select methods and adopt them. What we are ultimately doing in the course of design is developing our own methods.*
- *We do not refer to fixed means of implementation that only take effect later on when working out the details of implementation decisions. Instead, we construct the meaningful use of means of implementation by testing, selecting, or complementing what is already available*

Brooks's (1987) *growing* software metaphor, to a certain degree, highlighted the need to understand how support for the developer's intuition and communication can be embodied in the artifact itself (cf. Beynon et al., 2008). The idea that understanding can be implicit in an artifact and interpretations of interactions with it is vividly expressed in Gooding's notion of *construal* (Beynon et al., 2008). The idea that the developer's understanding of the artifact will evolve throughout the construction of the artifact is a central tenet of constructivist computing (cf. Harfield, 2008), and the idea that interaction with the artifact may evolve from time to time due to its changing environment is one of the main concerns in Activity Theory (cf. Kaptelinin and Nardi, 2006).

3.2 Reconceptualising the developer and the user

In this section, I argue that the traditional narrow conceptions of developer and user maintain a divide between two parties (cf. §3.2.1). The divide in the conception has raised two issues:

i) The disparity between the influence that the developer and the user can exercise over the construction of the artifact (cf. §3.1)

ii) Difficulty in conceptualise the role-shifting phenomena when user involvement and workspace democracy is increased. (cf. §3.3)

To understand the root causes of these issues, I examine the broadening conception of developer and user in section 3.2.2 and section 3.2.3 respectively. Despite a trend towards broadening the conceptions of developer and user, the confusions that arise within the narrow conceptions may not be completely resolved. This is because (cf. Kuutti, 2001; Bannon, 1992b):

i) The narrow conceptions are not completely eradicated.

ii) Discussion is still set within the traditional conception of systems development, i.e. the divide still exists and when the user can perform supposedly developer's activities it breaks out of the traditional conception.

Consequently, I suggested a social role perspective for the conception of developer and user (cf. §3.2.4). That is, they should be thought as roles that can be taken by persons instead of entities of systems development in the traditional narrow conception. In addition, each person may play the role of developer and user at the same time or different times, as well as other social roles.

The discussion in this section focuses on the relationship between developer and user. The discussion of other issues, viz. the role-shifting phenomena in systems development and the need to revise the notion of participation due to the conflation of contexts will be postponed to later sections (cf. §3.3 and §3.4) of this chapter.

3.2.1 The tension between the developers and the users

In systems development, different stakeholders usually possess different or even conflicting perspectives about what artifact is to be built and how it should be built. This potentially creates a tension among all stakeholders, particularly between the developers and the users due to their different interests. In the worst case, the systems development process becomes an arena for negotiating their private interests (Howcroft and Wilson, 2003). From the organisational perspective, the existence of the tension is unrelated to the development approaches practised, and it is hard to dismiss due to its social nature – it exists anyway irrespective of the role of the actors. Since the difference of interests is mainly between the developers and the users, I argue that the tension is caused by the preconceived roles of developer and user (i.e. what they can do, are supposed to do, and are allowed to do) and the implicit assumption that a person may play only one role in most of the conceptual frameworks for systems development. Furthermore, the co-evolution between the construction of the artifact and developer's understanding (cf. §3.1) hints at an imbalance between the developers' and the users' access to the knowledge acquisition through the construction of the artifact – as far as the traditional conception is concerned, the developers have direct access, while the users can only passively influence the construction.

In the traditional conception of systems development, there is a clear separation between the developers and the users. An individual who has a *developer* role in systems development is expected and is allowed to build the system, while an individual who has a *user* role is expected and is allowed to exploit the system for her work. This separation is due to several implicit assumptions in the traditional conception:

i) The developers and the users possess different sets of skills, knowledge, and abilities, and they are working in different workspaces, on different sites, at different time, etc.

ii) The developers and the users are working in separated contexts: the developer (role) is working within the development context, and the user (role) is working within the use context [48].

Fischer (1999) even goes further to suggest that there is a *symmetry of ignorance* between the developers and the users, i.e. the developer is ignorant of the user's space, and the user is ignorant of the developer's space, and these spaces are disconnected. From the role

[48] The separation of contexts becomes apparent in product development, when the systems development is considered as a software factory. In the author's own working experience in the e-commerce industry, the development environment (i.e. the software and hardware configuration for developer to construct, test, and evaluate the system) is usually separated from, and often different from, the production environment due to resource limitations. Even in some in-house development projects, customers are usually unwilling to invest in equipment for the developers and unwilling to share their brand new production hardware for the development. These differences make it hard for the developers to discuss and relate innovative ideas to the users.

theory perspective, the traditional conception of the developer and the user are reciprocal (Biddle and Thomas, 1966). That is, there is a producer-consumer relationship between the developer role and the user role. This may promote a conflict of interest between the developer and the user.

The divide between the conceptions of developer and user can also be found in the fundamental concepts of Information Technology (IT). Alter (2000) observed that the developer-oriented perspective (with reference to the narrow conceptions of developer and user; in Alter's terms, the IT perspective) tends to emphasize the interests of developers and hence view the users as more distant from the developer compared to the business perspective[49]. In this sense, the conception seems to suggest a necessary divide between the developer and the user in order to achieve the integrity of the traditional conception of systems development – i.e. the relationship between the developer and the user has to be reciprocal and is necessarily facilitated by document-based communication such as requirements specification. Another reason for this divide may be due to the fact that the roles of individuals in relation to the development process are often restricted by their organisational roles and the culture of participation in the organisation (cf. §2.5.2).

As I discussed in section 3.1, there is an intimate co-evolutionary relationship between the developer's understanding and the software (Naur, 1985a), and to that extent the developer who constructs the software may be considered as holding the *theory* of the software (Naur, 1985b). Since the users cannot interact with the artifact as the developers do (lacking the intimate co-evolutionary relationship), the developer can be viewed as having a privileged influence over the artifact in the systems development process, and the user can only passively influence the development of the artifact even though the *ownership of the artifact* is later transferred to the user. In other words, the intimate interplay between developer's understanding and the artifact results in a disparity of influence between the developers and the users. This disparity, together with the divide between the developers and the users in

[49] According to Alter (2000), the business perspective is the perspective that is taken by non-IT professionals such as staff in sales, marketing, engineering, general management, etc. These people are interested in how the system achieves their goals effectively and how it might provide a better work life (Alter 2000). They may like, dislike, or tolerate doing work in relation to systems development (Alter 2000).

the traditional conception, contributes to a tension between the developers and the users in systems development.

3.2.2 Broadening the conception of user

Common sense suggests that the term *user* refers to the individual who will ultimately use the artifact as constructed by the developer. However, careful examination in the literature suggests that the concept of user covers a wide range of possibilities. Drawing on von Hippel (1986), Bannon (1992b), Wallace (1999), Alter (2000), Kuutti (2001), Lamb and Kling (2003), Iivari (2006), the user can be thought as:

- a component in an organisational system (influenced by organisational studies)
- a source of errors (influenced by cognitive psychology and human factors)
- a social actor / a human actor (influenced by anthropology and micro-sociology)
- a consumer (influenced by marketing and design)
- a learner (influenced by learning theory and pedagogy)

The conceptions of the user has also been characterised in narrower and broader terms in different contexts. In the narrow conception, such as was topical during the first wave HCI, the users are conceived as "… at worst, idiots who must be shielded from the machine, or at best, simply sets of elementary processes of "factors" that can be studied in isolation in the laboratory" (Bannon, 1992b). The adoption of this limited scope user conception can be found in many disciplines such as Software Engineering (SE), Information Systems (IS), and Human Computer Interaction (HCI). For instance, the user is treated as the source of information within the traditional waterfall systems development model. In interactive systems and products design within the research community of HCI, this notion of user is exemplified by the popularity of user-centred design (UCD), which is "a philosophy based on the needs and interests of the user" (Norman, 2002, p.188). Although UCD claims that it will create a more usable and understandable system or product for the user (Norman and Draper, 1986), it is based on some implicit assumptions:

i. The user can be studied in laboratory settings and can be modelled accurately with conceptual tools such as user and task modelling[50] (Bannon, 1992b).

ii. The interaction between the user and the machine can be isolated from the social context in which the user is living (Lamb and Kling, 2003).

These implicit assumptions were broadly made in HCI but were not questioned until the “second-wave” of HCI (Kaptelinin et al., 2003; cf. Bannon, 1992b). The main issue with this the narrow-scoped user conception is that it does not take the complex *interface* that the user is facing into account (Bannon, 1992b). Grudin (1990) argues that the user not only interacts with the computer, but also other people, artifacts and processes in the context in which the user is situated, and all of these are significant in shaping the user’s interaction with the computer. That is, the computer is only one of the many entities in the *user’s interface*. Indeed, it is through these seemingly non-essential everyday social interactions that we articulate our work, and many groupware applications have failed due to their lack of concern for the social aspects of the user (as I discussed in §2.2.2).

Consequently, broader conceptions of user have emerged, e.g. *user as human actor* (Bannon, 1992b), *user as social actor* (Lamb and Kling, 2003). In these conceptions, the social aspects of the user, such as the organisational structure and the social environment that circumscribe the user, are taken into account. Moreover, user are conceived as active agents who are competent in their work practices and “wish to accomplish tasks, to understand what is going on, and are willing to jump ahead and explore the computer system on their own if, for example, the tutorial material is unclear or too pedantic” (Bannon, 1992b). Bannon (1992b) argues “By using the term "human actors" emphasis is placed on the person as an autonomous agent that has the capacity to regulate and coordinate his or her behaviour, rather than simply being a passive element in a human-machine system.” Furthermore, the users are thought of as “partners in an ongoing and continuous social interaction with various communities … their thinking is shaped by their personal histories and by their membership in those various cultural communities” (Kuutti, 2001). Besides

[50] User modelling and task modelling are the conceptual tools that a usability engineer may use to analyse the user’s behaviour and the user’s task during interactive systems design. These conceptual tools are based on the first generation cognitive psychology theories. From my observation, these conceptual tools are still being used, at the moment, in practice.

these social factors, the broadening conception of user has recently been influenced by the consumer perspective, which originated from consumer product design. In this consumer perspective, the user is conceived to have “emotions and needs for pleasure and self-expression” apart from rationality and reason (Kuutti, 2001; also cf. McCarthy and Wright, 2004).

User-as-learner

Apart from the social and consumer experience aspects, the broader conception should include the learning aspect of the user. Quite often, the user has to going through a learning process before she can fully adapt to the system and properly configure the system for her work. However, this aspect is usually overlooked. As Kuutti (2001) pointed out, “When designers think about users, they rarely think of learning and dynamics, neither in the small – users as learning to master the technology device itself – nor in the large – users as learning something through the technology.” In fact, the learning of the user not only occurs in the use context, but also occurs in the development context. For instance, if the user is actively involved during the development process, she may also need to learn about the future configuration of the computer system and changes to her work and workspace in order to give appropriate comments regarding the design of the system. All these suggest that the learning of a user is pervasively intertwined with the entire development/use process.

User-as-developer

Apart from the above aspects, there is still (at least one) aspect missing from the scene: the *user as developer*. If users are considered to be social actors who are able to perform operations informed by their knowledge (i.e. capabilities, skills, trainings, experiences, etc) and relate the artifact to their work *in situ* (cf. Suchman, 2007), there is no reason why the users should be treated as “technically-incompetent” (i.e. as only able to carry out activities other than development activities, even though they may be technically-competent). Furthermore, users may take the “first steps” to programming if they believe the profit of the program is greater than the investment of their attention (Blackwell 2002; 2004). For instance, if the user believes learning to use the macro feature in a word processor and writing a macro may save time in the future for a repetitive task, she may choose to learn and write the program rather than perform it manually. Indeed, the (main) difference between professional developers and end-user developers is whether they regard programming as their primary job:

> *"A "professional" programmer might be defined as someone whose primary job function is to write or maintain software. A "novice" programmer might be defined as someone who is learning how to program ... "end-user programmers" (EUP) are people who write programs, but not as their primary job function ... but some EUPs, such as chemists or other scientists, may need to learn to use "regular" programming languages such as C or Java to achieve their programming goals" (Myers, 2006)*

This argument is supported by recent research, which suggests that, in the US, a large number of end-users may do programming in their job (Scaffidi, 2005; Myers et al., 2007) In fact, members of the younger generation are often more experienced in using computer technologies than their teachers. It is not uncommon for high school and college students to be required to perform information search from the web, to write reports using word processors, and to do a few programming exercises (Xiao et al., 2005). More recently, Berlinger et al. (2008) argue that the next challenge for the HCI community is to find ways to enable non-professional software developers (i.e. initially conceived to be users) to create, modify, and extend software artifacts. All of these arguments suggest that the users may play a developer role in the development process.

3.2.3 Broadening the conception of developer

To some extent, developer is a collective term which is used to refer to the people who carry out technical activities that are directly related to the software construction process, e.g. requirements analysis, designing the artifact, programming, testing, debugging, and deployment. However, these activities are often carried out by different people in large software teams due to high division of labour. In these teams, the role of a developer may be further broken down into several roles. For instance, system analysts capture requirements from the users and design the system, programmers implement and debug the system, and testers evaluate the system. In contrast, a developer in a small development team may be responsible for all these activities at the different stages, or even concurrently, during the systems development process. This suggests that the conception of developer can vary from several specialised roles to a general role. This can be partly due to the chosen paradigm for development, and partly due to the division of labour in the organisation structure.

The term “developer” can be viewed as an umbrella term useful to people outside the software development community. It can refer to system analysts, programmers, testers, etc. However, the diverse activities in the systems development process raise questions such “What characterises a developer?” and “What is the role of the developer in the development process?”

In the traditional conception, the developer is conceived as a constructor whose main responsibility is to carry out technical activities that are at the core of the software construction process, e.g. requirements analysis, designing the artifact, programming, testing, debugging, and deployment. Such a conception is narrow – at worst, the developer is thought as a person who merely converts the given specification of the system into the real system – and it presumes minimal contact, communication and interaction between the developers and the users. Furthermore, this conception emphasizes a reciprocal role relationship between the developer and the user, which may provoke the tension that I discussed earlier (cf. §3.2.1).

To some extent, the narrow conception of developer is derived from the traditional conception of software development. The main issue in this conception is that it does not take other aspects (faces/facets) of the developer into account other than the formal systems development aspect. In practice, developers are not merely responsible for software construction; they may also need to carry out non-technical activities that are related to the construction, which may include learning the user’s domain/workspace (as a learner), playing the role of the user to envisage user’s experience in the future configuration (as a user), teaching and training the users to use and to configure the system (as a teacher and/or consultant). Indeed, these non-technical activities broadened the conception of developer. As with the broader conception of user, the broader conception of “developer” views the developer as a *participant* who brings in all of his knowledge and his life into the systems development process, rather than merely as a technical guru. To some extent, this broader conception is motivated by the growing trend towards increasing user involvement in systems development and it is highly influenced by the approach that the development team has adopted.

Developer-as-learner

It is very difficult, if not impossible, to design an artifact with prior knowledge. This is because the design problem is a *wicked problem* (Rittel and Webber, 1973). It is not always possible to be a *reflective practitioner* (Schön, 1983). Developers often have to face new domains and unfamiliar situations in this era of rapid technological change. Indeed, systems development is always situated – there are no two identical problems and no two identical groups of stakeholders. This makes reusing or relating to previous experience difficult.

Quite often, developers will acquire knowledge (or learn) about the users’ domain and their workspace throughout the design and development of the artifact when practicing an incremental development approach to systems development. When practising the traditional systems development process, the developers may have to gain substantial knowledge about the user’s domain before an appropriate design is developed. No matter which development approach is practised, the systems development process can be viewed as a mutual learning process between the developers and the users which has the learning activity at the centre of its focus (Béguin 2003). In fact, learning occurs among all stakeholders. In the light of the symmetry *of ignorance* perspective (Fischer 1999), this is because stakeholders (developers, users, managers, etc) have different perspectives and expertise such that they learn from each other through the development process (a common space which can be thought of a *boundary object* (Star and Griesemer, 1989).

Besides learning about the users’ domain and their workspace, the developer also learns through the construction of the artifacts (cf. §3.1). Furthermore, it is not uncommon that novice developers have to learn how to use the development tools[51] on-the-fly. This suggests that the learning process (both individual and collective) is intertwined with the development process, and the developer sometimes has to *act* as a learner in the systems development process.

Developer-as-user

During the development process, developers may sometimes, arguably, play the role of a user of the developing artifact. This often happens in exploratory systems development (cf. e.g. Yung, 1993), in which the developers may play the role of users and *use* the artifact to envisage the users' experience in order to gain insight for further development. However, unless the developers are developing software for themselves (e.g. in the developers of the IBM Jazz project[52]), it is questionable whether such development is effective, as the role-playing users lack the skills that *real* users possess. Consequently, the *use* scenarios that the developers engage in are closely connected with the objective of learning how the users interact with the system. Such interactions would otherwise be regarded as no more than testing or debugging activities.

3.2.4 Reconsidering the conception of developer and the user: a role or a person?

The purpose of the following discussion is to establish a coherent view and interpretation of human-centred design, a topic to be discussed throughout the rest of this thesis.

A few researchers have already argued that the separation of roles between developers and users in systems development should not be binary (Nardi, 1993; Fischer et al., 2005). The spectrum between users and developers should range continuously, from passive consumer to active consumer, to end-user, to normal user, to power user, to domain designer, to meta-

[51] This includes seeking references for API libraries and components off the shelf (COTS).

[52] This is an example of a context in which the developers can play the role of users actively. (cf. the presentation at http://www.eclipsecon.org/2007/index.php?page=sub/&id=4248 on "Jazz in Action - Building Jazz with Jazz" in EclipseCON 2007)

designer[53] (Nardi, 1993; Fischer et al., 2005). The discussion above (i.e. §3.2.2 and §3.2.3) suggests a trend towards expanding the scope of design (cf. Kaptelinin and Nardi, 2006), and a trend towards broadening the conceptions of developer and user. The broadening scope of developer and user has motivated us to reconsider the conception of developers and users in systems development. On the one hand, the terms “developer” and “user” can be conceived as labels for entities in the development process, so that a developer is a person whose main responsibility is to construct the artifact, but (depending on her knowledge, ability, skills and training) is also capable of performing any other activities as a human being. Similarly, a user can be a person whose main responsibility is to use the artifact in her work context, but who is capable of performing any activities as a human being. On the other hand, these concepts can be conceived as roles, so that a person can be a *user* as well as a *developer*, at different times or at the same time. This can be explained by *social role* theory, which suggests that a person can have multiple roles (cf. e.g. Biddle and Thomas, 1966).

Taking the above discussion into consideration, the narrow conceptions that I discussed in section 3.2.2 and 3.2.3 seem to suggest *a single static role perspective*, while the broader conceptions seem to suggest *a person perspective with multiple roles and continuously being broadened with newly discovered roles*. The person perspective is more aligned with social role theory, which suggests that each person may have multiple roles at different times. However, the broader conceptions seem to maintain the divide between developers and users that can be found in the traditional narrow conceptions. The broader conceptions still pay little or no attention to the fact that developers and users are capable of performing other activities, e.g. user-as-developer or developer-as-user, which may be thought of as irrational or irrelevant within the thinking derived from the traditional narrow conceptions (cf. §3.2.1).

[53] Fischer (1999) uses the term ‘designer’, while I use the term ‘developer’ to refer to the similar conceptions but to place particular emphasis on the fact that systems development is not merely *design* – it includes both design and implementation, and these two activities are intertwined.

Although many researchers have criticised the narrow conception of user (e.g. Bannon, 1992b; Kuutti, 2001), they have also stated that there is nothing wrong with taking such narrow conceptions. This reflects the fact that research communities still have reservations about adopting a broader conception that is independent from the traditional conception, or an alternative conception that is better suited to human-centred design. This may be because there are implicit interdependencies between the conceptions of developer and user and the development paradigms. To some extent, the conceptions of developer and user are bound up with the development paradigms – development paradigms set the scope of the design process. The conception of the development paradigm influences how the developers and users are involved (this is a collaboration issue) and how the development process is carried out (this is structural issue). If adopting broader conceptions of developer and user entails dispensing with the traditional (i.e. narrower) conceptions entirely, this may violate the underlying assumptions and integrity of the development paradigm. Since the broader conceptions are derived from the narrower conceptions within the traditional conception of systems development, it is hard to conceptualise the dynamic role-shifting phenomena in systems development (cf. §3.3). While the narrow conceptions are restricted to a single static role, the broader conceptions do not account for how and why the roles may be shifted. Indeed, the imbalanced relationship between "developer" and "user" becomes apparent when human-centred aspects are taken into consideration.

Because of all of the issues discussed above, I argue that, at the very least, a new methodology should not be based on the narrow conceptions of developer and user, as this limits our understanding of what users are *really* capable of and blinds us to the possibility that the users may progress towards developer roles as they gain sufficient technical knowledge regarding the artifact in construction. However, the diverse conceptions associated with the same terms may lead to confusion. Kuutti (2001) criticises the fact that the term "user" is adopted for different perspectives at different times, and different focuses in different research areas. This can lead to confusion, as the same terms are being used for different meanings and these meanings are often implicitly associated with different perspectives taken by practitioners and researchers (Alter, 2000). The narrow conceptions of user have not and will not completely disappear, and we do not seem to have a better term other than *user* even though the term has been confusing (Kuutti, 2001). For this reason, adopting the role perspective (as mentioned above), in which developer and user are roles and a person can hold both roles at different times, seems less problematic.

3.3 The role-shifting phenomena[54]

If the developer and the user are roles for a person, it is necessary to conceptualise the possible role-shifting in systems development. In this section, I argue that the role-shifting phenomenon is not uncommon in general group work and in systems development, where human actions (including group work) are highly situated (Suchman, 1987). In these contexts, the shift in roles is enabled by the configuration of the situation (e.g. members of the group having overlapping skill-sets) and is motivated by local needs (e.g. to cope with changing context of work).

From a social role theory perspective, a *role* is "any set of behaviors that has a socially agreed upon function and an accepted code of norms" (Newman and Newman, 2007; also cf. Biddle and Thomas, 1966), and it is filled with expectations of self and others (Newman and Newman, 2007). Put simply, a role can be used to predict the behaviour of an individual who performs the role, and at the same time the behaviour of an individual may reflect the role that she is playing. For instance, an individual who has a *developer* role in systems development is expected and is allowed to build the system. Similarly, an individual who has a *user* role is expected and is allowed to exploit the system for her work. However, it is inappropriate to assume that a person can only play a single static role. Since a person may have multiple roles[55] (Merton, 1968), the notion of role-shifting as I discuss below can be considered as the shift of an individual's primary role.

---

[54] Part of the argument in this section has been presented in the Warwick Postgraduate Colloquium in Computer Science (WPCCS '06), University of Warwick (Chan 2006), and later in a paper (Beynon and Chan 2006, available at http://extra.shu.ac.uk/paperchaste/dpd/participants.html, last accessed on 17 February 2009) for the Distributed Participatory Design workshop that was held in conjunction with NordiCHI 2006 in Norway.

[55] In role theory, there is a subtle difference between what is known as *multiple roles* and *role-set*. According to Merton (1968), *multiple roles* refers to the set of roles that are associated with multiple social statuses that a person is holding, while a *role-set* refers to the "complement of role relationships which persons have by virtue of occupying a particular social status" (p.423). The notion of multiple roles I use here is closer to Merton's notion of role-set.

3.3.1 Role-shifting in group work

In order to achieve a common goal effectively and efficiently, it is not unusual that a group activity is broken down into a number of tasks and carried out by different people, i.e. division of labour. It is also not unusual that members of a group shift their roles due to local needs throughout the course of a group activity. For example, consider the event of organising a house party of a small group of friends, e.g. a dozen of people. There is usually a significant amount of work between preparing to hold the party and cleaning up after the party. Although the party may be "managed" by party organisers (i.e. a few individuals of the group), the work is often broken down into many tasks and shared by most, if not all, of the participants. Initially, individuals may have "well-specified" duties, e.g. preparing food, setting up a tent, serving drinks, etc. However, the duties and the roles of individuals may be shifted either temporarily or permanently during the party. For instance, Anthony was socialising and serving drinks to other party participants in another corner of the house initially, but then went to help Mark in the kitchen to make the barbecue skewers because there is shortage of labour in the food preparation "team". Anthony initially knows nothing about how to make skewers and is not aware of hygiene issues in preparing food. However, through observing how Mark is making the skewers and learning-by-doing, Anthony becomes proficient in making the skewers.

In this example, Anthony's role has shifted, from drink server to a food preparer, due to the need of extra labour in food preparation. This role-shifting was enabled by the possibility that he can learn the new skills for food preparation and takes up the new role. Upon reflection, Anthony may be reluctant to shift his role if he does not see the need (i.e. shortage of labour), or if he does not want to move into the new role because he is a bad learner or the new role requires professional skills of which he is not capable. Consequently, one may argue that such role-shifting behaviour (i.e. flexible roles) only occurs in informal activities such as voluntary work, which can be characterised by its softer and decentralised power structure. Furthermore, it is suggested that people conform to the expectation of a given role if the punishment for shifting away from the given role is substantial enough (Milgram 1974). However, one counter-argument is that decentralised power structures (i.e. decision by consensus) can also be implemented in work. For instance, Constantine (1991; 1993) introduced the concept of structured open team, which can be viewed as the combination of the *closed paradigm* (i.e. traditional hierarchical organisation paradigm) and *open paradigm* (i.e. adaptive collaboration process). The merit of this 'hybrid' model is that it promotes flexible communication and efficient collaboration for group work yet it is within a formal and static hierarchical organisational structure (Constantine, 1993). It is of particular interest that it promotes "rotation of roles to promote flexibility and skill acquisition" (Constantine, 1993). In this sense, it is well-aligned with the kind of role-shifting that I am concerned with in the example of organising a house party. Indeed, the concept of structured open team has been successfully applied in software development projects (e.g. Rettig, 1990; Thomsett, 1990).

3.3.2 Role-shifting in systems development

In the context of systems development, it is not uncommon for individuals to have to shift their roles (in relation to the development activities) in different situations and at different times, especially in PD projects when participants are actively involved in the systems development process.

In Friis (1998) paper[56], Friis reports a field study of a systems development project for an organisation across two sites. In Friis's study, all participants were considered as actors – this included researchers, future users of the computer system, and developers (i.e. system analysts and programmers) – and about half of the future users had participated in the development. The development process followed an exploratory prototyping approach, in which the prototype served as complementary to the other requirement specification documentation (cf. Friis, 1988). Further, the development was highly influenced by its future users (to some extent, it was user-driven) as the users sat together with the researchers and built a "logical data base model" that was based on their structured descriptions of the problem, and later implemented by the in-house developers (Friis, 1988).

The virtue of Friis's study lies in the harmonious collaboration between various actors, and

[56] Friis (1988) was published as "Action research on systems development: case study of changing actor roles" in "Computer and Society".

the shifts in their roles, in relation to the systems development activities, throughout the development process. Friis (1998) observed a number of participants who had shifted their roles from time to time. The role-shifting behaviour was evidently reflected from the participants' interaction styles with the system and their behaviour towards the other personnel. To increase the validity of the observation, Friis discussed the role-shifting findings with the participants (Friis, 1988). Interestingly, most of them (except two programmers who showed no interest in this discussion) agreed with Friis's findings regarding their changing roles in the development process (Friis, 1988). Throughout the development, Friis (1998) had detected five user roles: the "traditional user", the "interested user", the "analyzing user", the "designing user", and the "evaluating user". As the development progressed, the users became bolder in their interaction with the computer system (Friis, 1988). Some users became curious about the feasibility of their proposed solutions ("interested users") and some even wanted to participate and to influence the development on the technical side ("analyzing user") (Friis, 1988). Some users wanted to make decisions about what would and would not be computerised ("designing user") (Friis, 1988). Friis (1998) observed that some "designing users" were interested in what is behind the user interface, and argued that these users might be able to build components of the system if appropriate tools were given. Friis (1998) further observed that some users felt that they were the "system owner", and therefore were confident in evaluating, testing, and making modifications to the system in a later stage of development.

Friis noticed that this role-shifting behaviour was not limited to the users, but also occurred among the developers. On the one hand, three system analyst roles were detected: the "traditional analyst", the "collaborating expert", and the "teaching-consultative expert". On the other hand, two programmer roles were detected: the "traditional programmer", and the "teaching programmer". Throughout the development, and through increased interaction with the users, the developers gradually shifted towards the role of teacher and/or consultant (Friis, 1988). Initially, the system analysts had no contact with the users or treated the users as source of information ("traditional analyst"). In the role of "collaborating expert", a system analyst invites users to actively participate in the analysis work (Friis, 1988). As the development progressed, the system analysts had shifted into the role of "teaching-consultative expert", providing guidance and help to the users during the analysis and, in some cases, teaching the users to use high-level compilers to build the prototype (Friis, 1988). Although the programmers are the ones who actually implement the system with lower-level tools (and are not supposed to interact with the users in the first instance), a similar teacher-student relationship was also observed between the programmers and the users (Friis, 1988).

Friis (1998) considered the role-shifting behaviour as "preliminary" findings because they were unexpected results: the original aim of the study was "to develop a mutual code for better communication between the DP experts and the users for the work with requirement specifications" (ibis, p.30). However, despite its "preliminary" nature, it is clear that participants had transcended their preconceived roles in Friis's study. To some extent, the users in Friis's study had gradually shifted into a role close to what is known as end-user development or end-user programming (Nardi, 1993).

Apart from Friis's study, similar role-shifting behaviour was also detected in other PD projects. For instance, Danielsson (2004) observed a role-shifting behaviour among the participants in the development of a mobile learning environment for students in higher education over a two-year period. According to Danielsson (2004), the project initially followed a user-centred design (UCD) approach (Norman and Draper, 1986). In UCD, the focus is to develop an understanding of the possible interaction between the user and the artifact in the context of use. Hence, the participating students (who were the future users of the system) were initially treated as the source of information. However, researchers observed that the students were unable to envisage the future configurations (Danielsson, 2004). In order to enhance the students' ability to imagine the future environment, the development process was driven into a learner-centred design (LCD) approach (Danielsson, 2004; Soloway et al., 1994), where the development focus shifted to supporting diversity, motivation and growth (Quintana et al., 2001). As the development progressed, the students "could *interact with the designers* and directly see future design possibilities" (Danielsson, 2004, p.51). In a later stage, the project team found themselves using PD techniques, e.g. future workshops.

In Danielsson's study, the shift from user-centred, to learner-centred, to participatory design was strongly in evidence: the development focus was shifted, the scope of concerns was broadened, the conceptual roles of the students were shifted, and the techniques that were used were diversified throughout the project. The shift in the roles of the students were not merely a coincidence, but inevitable (Danielsson, 2004). The role-shifting was in part due to the fact that the students were not familiar with the future configuration – there was no counterpart in their experience to which the students could relate. In order to promote user participation, the focus of the project was shifted and, at the same time, the role of the participants was shifted from a user role to a learner role. In addition, role-shifting may also have been promoted by the situated nature of the systems development – it is unlikely that two systems and their development process would be identical. This makes it hard for the participants to maintain preconceived roles. As Danielsson (2004) noted, the approach that was used in their project cannot be properly reconciled with the description of the perceived approaches, i.e. UCD, LCD, or PD.

The role-shifting phenomenon among the participants is apparent in both Friis's and Danielsson's studies: a number of participants shifted their roles from passive users to active users, from active users to pseudo-developers, and from pseudo-developers to co-developers. From a traditional perspective on roles in relation to systems development, it is tempting to regard such role-shifting behaviour is exceptional, if not abnormal and irrational. However, from a social perspective, an individual requires some form of motivation (i.e. a need for changing role) in order to change her role, in addition to possessing appropriate skills and/or appropriate training (i.e. having the potential for changing role). Where motivation is concerned, it is clear that the users were motivated by their personal interest to the system and the needs for appropriate support to their existing and future work (in Friis's case), and learning (in Danielsson's case). Where potential for role-shifting is concerned, the users were able to obtain help and guidance to learn about the technical possibilities (and even the programming skill in Friis's case).

Another argument might be used to dismiss the role-shifting behaviour. It could be attributed to *powerful intervention* by the researchers as they entered the systems development project so that the situation might revert to normal when the researchers had left the scene (Friis, 1988). In Friis's study, the situation did revert to normal when the researchers had left the project. As some of the users complained, "You come here and promise a lot, and you even show us the promised land, and how to get there – then you leave and all is back to 'normal'!" (Friis, 1988). Although researchers using participative observation might have made their best effort to avoid interference, it is as hard to guarantee the absence of such interference as to guarantee the absence of interference from the management of the organisation (cf. the political and organisational issues in user participation I discussed in §2.5.2). Hence, it is hard to see how such arguments may be dismissed entirely.

Despite these issues, there is a growing interest in recent HCI research (cf. e.g. Beringer et al., 2008; Fischer and Ostwald, 2002; ODS, 2009) in democratizing the design space for the users to create, modify, or to extend software systems. The rationale behind this trend is that the users are the owners of the problems and computer systems "may be better and more efficiently used if we … let the users solve their own problems, we should not do that for them" (Friis, 1984). In fact, this matter is closely linked to the espousal of user-oriented development, in which the development is driven by the users. Indeed, the high degree of workspace democracy (such as is advocated in PD) might be viewed as a catalyst for the role-shifting behaviour in systems development – as it encourages the users to contribute their diverse perspectives; but the users cannot do so without being deeply involved and engaged in learning about the developer's initial vision and the technical possibilities. At the same time, as the users seek help and guidance from the developers, the developers move to a teacher and consultant position. Eventually, the users shift into a role of pseudo-developer or co-developer. However, there is concern that users may not be able to produce such high quality systems as professional developers as they are not trained for rigorous quality assurance techniques nor able to identify an accurate and complete set of requirements (Friis, 1988; also cf. Davis, 1984). Even worse, the user may not be aware of the consequences of that their design. This could result in what Friis (1998) called *automatization in absurdum* – the worker redesign away his own job through a high degree of automation of the present work process that the worker is supposed to handle.

The role-shifting phenomenon becomes more apparent in the context of free and open-source systems development (FOSS). Nakakoji et al. (2002), based on their case studies in four FOSS projects, observed that some initially passive users may become active developers after a period of engagement and active contribution to the project They argue that the "evolution" of the users is facilitated by two factors:

i) the existence of motivated members who aspire to play roles with larger influence,

ii) the social mechanism of the community that encourages and enables such individual role changes.

Nakakoji et al's observation is consistent with Lave and Wenger's (1991) theory of Legitimate Peripheral Participation (LPP), which suggests that new members of a community of practice will learn the skills from the more senior members of the group (cf. Nakakoji et al., 2002).

To some extent, the interplay between the users' understanding of the developing artifact and the developers' role-shifting is similar to the co-evolution between the developer's understanding and the artifact that I discussed in section 3.1. Despite the concerns raised above, the role-shifting phenomenon is seemingly genuine. Neither should it be viewed as unstable, temporary, or exceptional behaviour of individual participants. Consequently, the question to ask is not "Will role-shifting occur or not", but "How far may we allow role-shifting to go?" and "How do we support it?"

3.4 Participatory development

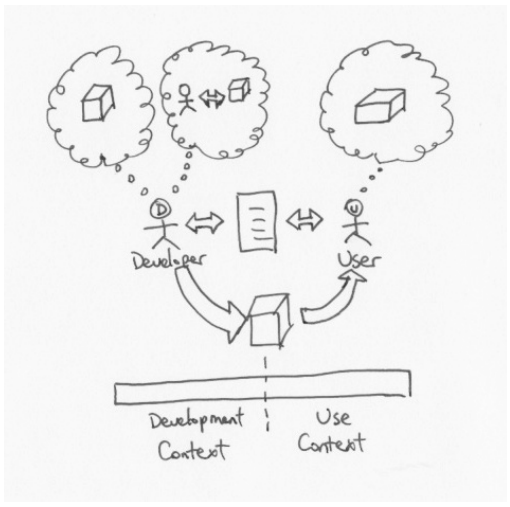
In the previous sections, I argued that the disparity between the influence of the developer and the user over the construction of the artifact in systems development is a consequence of the traditional conceptions of developer and users (cf. §3.1 and §3.2). This disparity may be mitigated by promoting a higher degree of user involvement and workspace democracy in systems development. On the one hand, a higher degree of user involvement enables the users to envisage and experience the systems as early as possible and, therefore, to communicate and negotiate their perspectives with the developers if the system does not fulfil their needs. On the other hand, a greater degree of workspace democracy allows users to actively influence the design decisions and, therefore, increases the level of satisfaction and the chance of acceptance.

Apart from the practical issues discussed in section 2.5.2, the promotion of a higher degree of user involvement and workspace democracy probes issues in the traditional conception of systems development – the boundaries between contexts of development and use, and the boundaries between the roles of developer and user become blurred. Moreover, as the user involvement and workspace democracy progress towards *genuine participation* (cf. §2.5.2), shifting in roles between developer and user may result. This suggests that the existing conception of user participation may not reflect this conflation well. In fact, the concepts of developer, user, and participation are all interdependent and co-related; when revising the conceptions of developer and user, the conception of participation may have to be reconsidered as well.

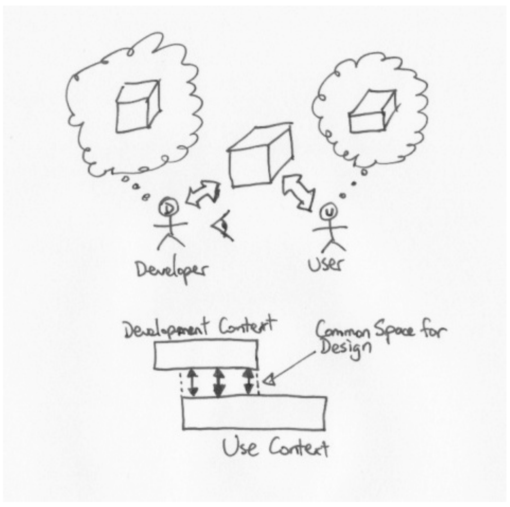
In this section, I examine the conception of participation, its relationship with the conflating contexts and roles, and propose a conceptual framework which views the systems development as a co-construction and co-evolution process that incorporates the role-shifting phenomena I discussed in section 3.3.

3.4.1 Who is the participant?

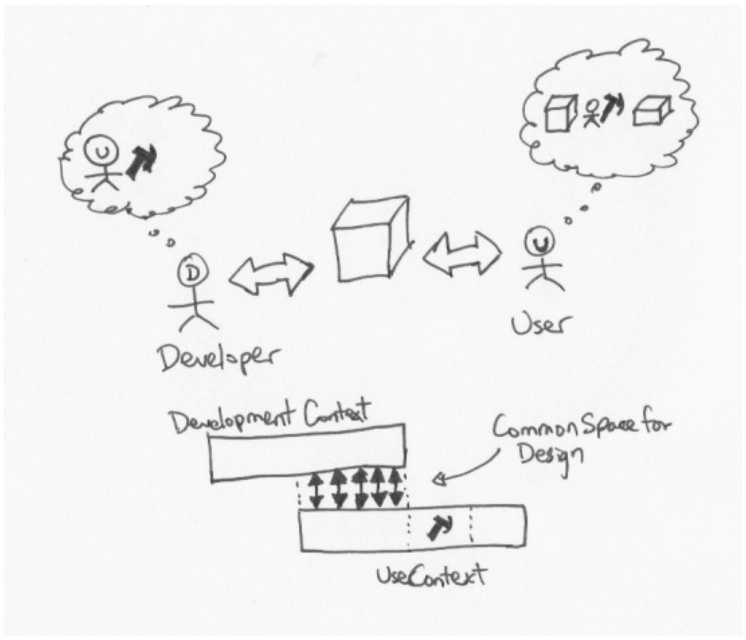
Central to the conception of participation is the question "Who is the participant?" The answer to this question is dependent on which perspective is taken. From a *developer-oriented perspective*, it is the case that the user participates in the development process. In this perspective, the developer has the overall control of the process and, therefore, the user may be conceived as entering the developer's space (i.e. user participation). From a *user-oriented perspective*, the user has the overall control of the process and, therefore, it is the developer who participates in the user's world (i.e. developer participation). From a *holistic perspective* (e.g. project management), all stakeholders may be thought as participants and, therefore, all of them are participating in a shared workspace. From a practical perspective, it seems that there is nothing wrong in adopting any of these perspectives. Indeed, a number of techniques and paradigms have been developed for different situations, ranging from a user-oriented to a developer-oriented perspective and from early to late stages in systems development (cf. Muller, 2001). However, from a conceptual perspective, I argue that both developer-oriented and user-oriented perspectives are biased and they may increase the tension between the stakeholders. Besides, the holistic perspective (i.e. common workspace) is well-aligned with the *symmetry of ignorance* perspective:



(a)



(b)



(c

Figure 3.2 – The spectrum of user participation in systems development

“The 'symmetry of ignorance' requires creating spaces and places that serve as boundary objects where different cultures can meet. Boundary objects serve as externalizations that capture distinct domains of human knowledge, and they have the potential to lead to an increase in socially shared cognition and practice." (Fischer, 1999)

Indeed, the holistic perspective not only offers better status for all participants, it also offers a more human-centred perspective:

i) It changes the relationship between the developer and the user, i.e. neither the developer nor the user dominates the development process.

It puts the emphasis on the collaboration instead of the separation between participants such as the developer and the user.

3.4.2 Participation and the conflation of contexts

The promotion of a higher degree of user involvement and workspace democracy blurs the boundaries between contexts of development and use. Figure 3.2 illustrates the spectrum of user participation in systems development that can be roughly divided into three different regions:

a) The level of user involvement in the development process is minimal. The developers and the users, therefore, mainly communicate through documents. The users may not have the opportunity to experience the artifact until late in the development process. The traditional “waterfall” approach to systems development falls into this category.

b) There is an increased level of user involvement but limited interaction between the developers and the users in the development process. At this level, users are invited to design meetings and are encouraged to express their diverse perspectives. Meanwhile the developers actively observe how the users interact with the artifact. Systems development approaches such as evolutionary prototyping (with or without ethnographical studies) fall into this category.

c) The developers and the users co-design and interact through the artifact, to the extent that face-to-face (or mediated) interaction becomes indispensable. In some cases, the users may construct or modify the artifact, with or without the guidance of the developers (e.g. tailoring and end-user development). This is an ideal scenario for systems development, yet none of the existing approaches fall into this category. However, Participatory Design comes closest – due to its advocacy of equal opportunity for participation and multiple communication channels (cf. Mambrey and Pipek, 1999).

As the degree of user participation increases, the context of development and the context of use are being conflated and create a common space for 'interaction' and 'participation'. In fact, the nature of interaction and participation are quite different in these cases, and they also establish different kinds of tensions between the developers and the users. In scenario (a), interaction and participation is often used to 'define' the boundary of development and the scope of use; thus the tension is stemming from negotiating the boundary. In scenario (b), interaction and participation is for verifying the observation and improving the fitness of the artifact from the developer's perspective; thus, the tension is stemming from interpreting the situation. In scenario (c), interaction and participation is an indispensable part of the whole process, in which the developers and users learn about each other's concerns, develop joint-ownership of the development process, and co-construct the artifact; thus the tension is stemming from the gap between what they can do and what they are allowed to do. It is worth noting that the above partitioning of the user participation in systems development should not be considered as representative of all possibilities. For instance, it is possible to collaborate as in scenario (b) but for the developers to also develop tools or mechanisms for the users to customerise the artifact as in scenario (c). However, it is in scenario (c) that the conflation of the development and the use contexts becomes apparent. On the one hand, the development process (i.e. both design and implementation activities) may continue in the user's workspace, i.e. in the use context in the traditional conception. On the other hand, the artifact may be trialled in the user's workspace, i.e. *using* the artifact in its state-of-development. In this sense, the context of development and the context of user are overlapped and the boundary between them is blurred.

3.4.3 Participation and the conflation of roles

As mentioned earlier in section 3.2, within the traditional narrow conception of systems development, developers are conceived as constructors and users are conceived as non-IT professionals who use the artifact constructed by the developers. As the paradigm of participation shifts towards the scenario depicted in figure 3.2c, it is not only the contexts of development and use that become blurred – the roles of developer and user also begin to conflate. This raises a question regarding the conceptions of developer and user – both conceptions break out of the scope of traditional conception, and have violated the implicit assumptions behind the traditional conception.

One way to restore the integrity of these conceptions is to take a social role perspective (cf. §3.2.4). This allows the participants to play multiple roles at different times or at the same time, and to shift their roles throughout the systems development[57]. That is, a participant can have a developer role and a user role. In fact, a person can be a learner, an observer, a consultant, etc. In this sense, the "roles" of a participant at different moments in time become conflated (i.e. *role conflation*). Moreover, as the "role" of the participant is changing from time to time, it becomes unclear which "role" the person is holding at any given moment (i.e. within a short time of observation). However, this conflation has its merits: it does not merely allow participants to envisage the developing artifact from different perspectives, but also allows them to contribute from different perspectives and with different skill sets. Furthermore, this role-shifting phenomenon is not uncommon in systems development (cf. §3.3), and it becomes apparent when the user involvement and workspace democracy progresses towards *genuine participation* (cf. §2.5.2). All these considerations suggest that the conception of participation, developer, and user are all inter-related. By reconsidering the conceptions of developer and user, the conception of participation (as well as the framework for participatory development) may have to be reconsidered.

3.4.4 Participation, co-construction, and co-evolution

There is a close relationship between participation, co-construction and co-evolution in

[57] The assumption is that the participant has or will possess the appropriate skill sets.

systems development. One the one hand, a high degree of user involvement and workspace democracy results in a co-construction relationship between participants instead of a reciprocal developer-user relationship. In this sense, participants are co-designing and co-constructing the artifact through interaction with other participants within and outside the artifact itself. This means the users can also construct or modify the artifact, with or without the guidance of the developers. On the other hand, *genuine participation* gives a better experience of learning about the artifact through the co-evolution process during artifact construction (cf. §3.1). In the case of *genuine participation*, this process is no longer proprietary to the developers; it is accessible by all participants, which enables them to gain insight about the artifact and to contribute freely with different perspectives and with different knowledge. This may lead to a more human-centred systems development.

Figure 3.4 illustrates a conceptual model for participatory development which is based on the co-construction scenario (cf. figure 3.2c), and the co-evolution relationship that I discussed in section 3.1. This model takes a social role perspective, which considers “developer” and “user” as roles rather than entities in systems development. It potentially removes the conceptual boundary between “developer” and “user” by incorporating the idea

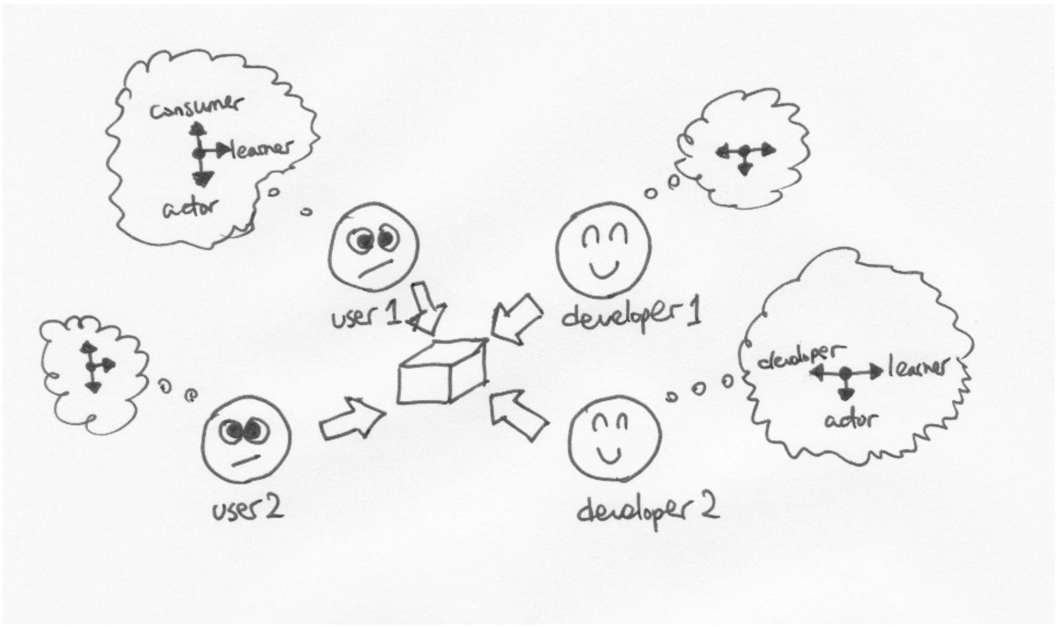


Figure 3.3 – A model of systems development with broader conceptions of developer and user

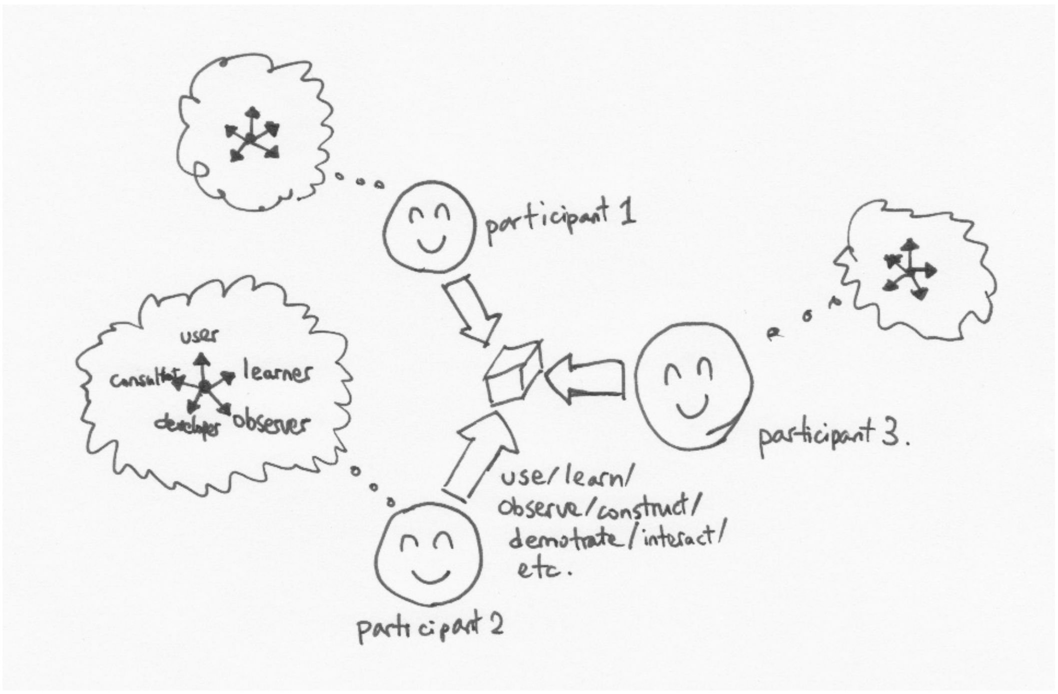


Figure 3.4 – A conceptual model for participatory development

of conflated roles that I discussed in the section 3.4.3 – so that a participant may have multiple roles, e.g. user, learner, developer, observer, consultant, etc, at any period of time. The conceptual differences between the participatory development model (cf. figure 3.4) and the systems development model with the broadened traditional conceptions of developers and users (cf. figure 3.3) are:

i) The participatory development model does not maintain a divide between the developer and the user as in the traditional conception. This divide limits our perception of what actions a "developer" and a "user" can perform.

ii) The systems development model with the broadened traditional conceptions is based on a single static role perspective, while the participatory development model is based on a social role perspective (that allows a person to play multiple roles). In the participatory development model, the "role" of a participant depends on the activity and the situation in which she is engaged, and her role may shift from time to time.

iii) There is virtually no limitation on the number of "roles" that a participant can take in the participatory development model, as they are treated as human beings who can perform any action they are capable of. This is, perhaps, the most significant conflict with the traditional view of systems development.

3.5 Implications for groupware development

"Groupware developers need to be conscious of the potential effects of technology on people, their work and interactions." (Ellis et al., 1991)

In this chapter, I have examined the conception of developer and user in relation to the degree of user participation and workspace democracy. I argue that the boundary between contexts of development and use and the boundary between the "roles" (in the sense of traditional conceptions) of developer and user is blurred and becomes conflated when the degree of user participation and workspace democracy moves towards *genuine participation*. I have also discussed the co-evolution between the developer's understanding and the artifact, and the role-shifting phenomenon in the context of systems development in detail. Throughout this chapter, I gradually developed the argument that the conceptions of developer, user, and participation in systems development are intertwined. That is, reconceptualising one of these concepts will affect the relationship between them and our conceptual understanding of systems development. While these concepts are topical in the systems development research, our particular interest is in how these concepts affect research into groupware development paradigms. Drawing on the conceptual analysis and the discussion in the earlier sections, I conclude this chapter by identifying seven implications for groupware development:

i) Allow users as active and direct an influence over the artifact construction as the developer has

ii) Support seamless shifting of roles

iii) Support the acquisition and refinement of development related skills

iv) Support diverse perspectives and integration

v) Support genuine participation

vi) Do not limit the user engagement in development to a sample of users

vii) Consider an alternative conceptual framework for participatory development

These implications, as discussed below, can be grouped under four headings, namely, co-evolution (i), role-shifting (ii, iii), participation (iv, v, vi), and conception (vii). I argue that a conceptual framework and *environment* that takes these implications into account is potentially better suited to addressing the human-centred aspects (as discussed in §2.5.1) and leads to efficacious groupware development (as discussed in § 2.5.3).

3.5.1 Co-evolution

The way in which the developer's understanding co-evolves with the artifact suggests that there is an essential ingredient of knowledge which is embedded in the artifact itself. The developer's understanding is growing through continuous interaction with and construction of the artifact. The intimate relation between the developer's understanding and the artifact also suggests that only the developer who produces the artifact may have access to this knowledge. This knowledge can be viewed as 'a theory' in the sense of Ryle (1949). I shall refer to this knowledge, which is derived through interaction with the evolving artifact, as *co-evolved understanding*.

The extent to which the developer is privileged over the user through access to co-evolved understanding becomes apparent when the development process affords less user participation and workspace democracy. This is the exact opposite of the high degree of user participation and workspace democracy (compare figure 3.2a with figure 3.2c). This makes the developer the main and the most influential contributor to the development of the artifact, which is thus more developer-oriented.

Implication I: Allow users as active and direct an influence over the artifact construction as the developer has

In the context of groupware development, this may raise the issues I discussed in section 2.2.2 because the users are not involved in the process. It may resulting systems that are based heavily on what the developers (including the ethnographers) can observe and interpret, which may not *fit* into group requirements (e.g. it may violate the work practice, disturb the social process, etc). It may also make the users reluctant to adopt and adapt to the system. Therefore, as I argued in section 3.1, it is important for the users to have as active and direct influence over the artifact construction as the developer. Moreover, this allows the users to understand the system and context of the development process better.

3.5.2 Role-shifting

In the context of groupware development, role-shifting is actually a two-fold issue. On the one hand, the participants may shift their roles in relation to the development (e.g. developer, user). On the other hand, practitioners (i.e. future users) who are involved in the development process may also shift their roles in relation to their work (i.e. "user" roles) [58]. In section 3.3, I argue that the role-shifting phenomenon is not uncommon in systems development. In fact, the degree of role-shifting is influenced by the extent to which user participation and workspace democracy is implemented. In other words, a higher degree of user participation and workspace democracy invites shifting in development-related roles.

Implication II: Support seamless shifting of roles

As many have argued (e.g. Grudin, 1991b; Ellis et al., 1991; Karasti, 2001), a high degree of user participation and workspace democracy increases the likelihood of successful groupware adoption and adaption, and may result in more human-centred groupware. For these reasons, a high degree of user participation and workspace democracy is seemingly indispensable for efficacious groupware development. When the users are highly engaged in the development process (as in Friis's (Friis, 1988) case study), curious users may wish to shift into a more active development role as they become more engaged. Indeed, in this context, participants of systems development shifted their roles seamlessly – they were only aware of the shift in their roles when they were interviewed by the researchers (cf. §3.3.2). Therefore, I argue that groupware development should provide support for participants to shift development-related roles seamlessly whenever a high degree of user participation and

[58] The former issue has been discussed in detail in section 3.3.2, while the latter has been briefly discussed in section 3.3.1. Since the focus of this thesis is on the development process of the groupware, the latter issue is beyond the scope of this thesis.

workspace democracy is preferable.

Implication III: Support the acquisition and refinement of development related skills

In order to enable the user to *shift* their roles towards the developer seamlessly, it is necessary for groupware development frameworks to provide adequate support for the user to acquire and refine their development related skills. Furthermore, social role theory suggests that a person may have to learn about the behaviour of the new roles when the person takes the new role and before the person become proficient in the new role (cf. e.g. Biddle and Thomas, 1966; Merton, 1968). In this sense, section 3.3 confirmed that participants of group work in general and of systems development specifically might have to learn the new skills through on-the-job training, as in the party example (cf. §3.3.1) and both Friis's (Friis, 1988) and Danielsson's (Danielsson, 2004) case studies (cf. §3.3.2) respectively.

3.5.3 Participation

As a high degree of user participation and workspace democracy is crucial in groupware development, it is more likely the participants will generate more ideas on how the system should be built. This, in turn, may require support for more diversified perspectives and a development process more oriented towards *genuine participation*.

Implication IV: Support diverse perspectives and integration

When the imbalance between the degree of co-evolved understanding of the participants is removed (cf. §3.1) and the users are allowed to shift their roles towards "developer", the users will potentially be able to make contributions from different perspectives and different roles. In this sense, all participants may directly influence, construct, or modify the system (as in the case of end-user development). Consequently, the development process will gradually become user-oriented. From a conceptual perspective, this may result in conflated contexts and conflated roles (as I discussed in section 3.4). From a pragmatic perspective, this potentially brings in more diverse and potentially conflicting perspectives. Since the user-developers are now having active and direct influence over the artifact construction, the degree of diversity in perspectives (as well as in interaction and communication between participants) is potentially much wider than the situation in developer-oriented development. Therefore, proper support for integrating diverse perspectives of participants becomes important.

Implication V: Support genuine participation

As argued in section 3.2 and 3.4, increased user participation and workspace democracy may lead to *genuine participation*. From a holistic view, the development process is leaning more towards the notion of *genuine participation* when all participants act like developers who can directly influence, to construct, or to modify the system. However, *genuine participation* is not a “silver bullet” for groupware development – not only may *genuine participation* be hard to realise due to subtle social issues and organisational constraints, it can be problematic when the development is *too much* user-oriented, again, due to the very same factors. While these issues may beyond the scope of this thesis, they are unfortunately critical factors for success or failure in groupware development (cf. §2.2.1). In this sense, allowing and limiting the potential for *genuine participation* becomes a dilemma. Therefore, it is necessary to strike a balance between a high degree of user participation and workspace democracy and *genuine participation*. As far as groupware development frameworks are concerned, the important consideration is to allow room for such potential to be realised. This is because it is difficult to judge the degree of user participation and workspace democracy for the development of the groupware for a particular group – it depends on the social, cultural, and organisational characteristics of the group of participants (i.e. users, managers, developers, etc) rather than solely upon technical issues.

Implication VI: Do not limit the user engagement in development to a sample of users

Despite the fact that user participation has gradually been incorporated into systems development methodologies in response to criticisms, real *user-centred* design is hardly achieved in reality because of the difficulties in getting active user involvement (cf. Bannon, 1992b). Consequently, a reduced degree of user involvement, e.g. representative users, is often employed in practice. This is particularly true when the systems development is leading to mass selling products (Bannon, 1992b). The problem here is that a “sampling technique” of this nature cannot be used in groupware development. The individuals act differently when no account is being taken of the group dynamics, so such sampling methods are liable to fail if the group is not involved and studied as a single unit. For this reason, what is needed in groupware development is *group participation*, i.e. the participation of the entire future user-group, rather than *user participation*.

3.5.4 Conception

Although the traditional narrow conception adopted by user-centred design is, to some extent, based on individualistic cognitive science theories, there is nothing wrong in taking the traditional narrow conception of users and developers (cf. Lamb and Kling, 2003, Kuutti, 2001). However, it becomes problematic when it is applied to a context in which a high degree of user participation and workspace democracy is indispensable, such as groupware development.

Implication VII: Consider an alternative conceptual framework for participatory development

As mentioned earlier, the conceptions of developer, user, and participation in systems development are intertwined. Reconceptualising one of these concepts will affect the relationship between them and our conceptual understanding of systems development. Earlier in this chapter, I argue that there is a conceptual divide between the developer and the user in the traditional conceptions. This divide not only affects how we may conceive participation in systems development, but is also an obstacle to conceiving participatory development projects, where roles and contexts can be highly conflated when user engagement and workspace democracy are in place.

In contrast to other branches of systems development, groupware development can be characterised by its particular emphasis on tackling the socio-technical aspects that arise during its development process. Central to this concern is exploring the best possible ways to integrate the human activity and the technological system harmoniously. To transcend the traditional conception and move towards efficacious groupware development (in the sense that I discussed in section 2.5), user engagement and workspace democracy seem indispensable. Therefore, the concept of developer and user, their contexts, and the “roles” of participants in the development process must of their *nature* be conflated. Moreover, if the participants’ roles are not static, the conceptual framework has to support the notion of role-shifting. To this end, it may be worth considering an alternative conception that takes all these considerations into account, such as the participatory development model I described in figure 3.4 (cf. §3.4.4). This is not only to help the practitioners and the researchers to conceptualise the kind of participation and collaboration that is required in the context of groupware development, but also to serve as a guide to the development of socio-technical approaches for efficacious groupware development.

In the next chapter, I will describe the merits of Empirical Modelling (EM) as an alternative conceptual framework for systems development. In chapter 5 and 6, I will argue that EM has the potential for collaborative modelling. In chapter 7, I consider groupware development as an instance of collaborative modelling, and argue that EM may be a potential framework for efficacious groupware development.

Chapter 4
Empirical Modelling in a nutshell

Empirical Modelling (EM) research was initiated by Meurig Beynon at University of Warwick more than 20 years ago. The name *Empirical Modelling* was adopted in 1994. EM can be conceived as an exploratory and human-centered approach to software systems development (Yung, 1993, Wong, 2003, Sun, 1999). The word *empirical* reflects two characteristics of the EM approach, namely, the experimental and the experiential (cf. Cartwright, 1999). In contrast to mainstream computing, Empirical Modelling is founded on William James's *Radical Empiricism* – a philosophical foundation that is radically different from that upon which mainstream computer science is based (Beynon, 2005). The potential of EM is not restricted to software systems development. Various researchers have explored the potential of EM in numerous application domains. These include computer-aided design (Cartwright, 1994), computer graphics (Cartwright, 1999), business decision-support systems (Rasmequan, 2001), financial modelling (Maad, 2002), participatory business process reengineering (Chen, 2001), computer-supported learning (Roe, 2003; Harfield, 2008).

This thesis develops a conceptual framework for efficacious groupware development that is based on the principles of EM (cf. chapter 7). As a preparation for the reader to fully apprehend the conceptual framework, this chapter discusses the fundamental principles of EM, its philosophical foundations, and the differences to other programming paradigms. As I mentioned in chapter 2, articulation work is one of the central concerns in collaborative work. For this reason, the discussion in this chapter is primarily concerned with the single modeller context, i.e. with how EM is carried out in a single modeller context. EM in a multiple modeller context and its fitness for collaborative modelling will be discussed in chapter 5.

In section 4.1, I briefly describe the basic features of EM. These can be divided into three categories: the ODA framework, the modelling process, and the situated and cognitive aspects of the EM model. The discussion here aims to provide the reader with enough background to appreciate the discussion of the philosophical foundation of EM in section 4.2.

In section 4.2, the focus is moved to the philosophical issues that EM has taken seriously during the development of its principles over the past two decades. These include William James's (1912) Radical Empiricism, David Gooding's (Gooding, 1990) notion of construal, and some of the writings of Brian Cantwell Smith (Smith, 1987). This section reviews a contribution to the philosophical foundation of EM made by the author, based on Russell Norwood Hanson's (1958) philosophical inquiry into the notion of observation and theory building in empirical science.

Due to its distinctive theoretical foundation, EM has numerous distinctive features in practice and cannot be regarded as programming in the sense of traditional programming. Section 4.3 explains the ontological and practical differences between EM and other programming paradigms, and how the principal tool – tkeden – supports the experimentation principle of EM in practice.

4.1 Concepts of Empirical Modelling

In this section, I briefly describe the basic concepts of EM: the ODA framework, the modelling process, and the situated and cognitive aspects of the EM model. The discussion here is not attempting to formalize these concepts into a unified framework as other authors have done (e.g. Wong, 2003), but aims to provide the reader with enough background to understand the discussion of the philosophical foundation of EM in the next section.

4.1.1 The ODA framework

EM researchers use the acronym ODA to refer to the key elements in an EM model, namely, *observable*, *dependency*, *agent*, and *agency*. The ODA framework refers to the meaningful use of these key elements during the construction of an EM model.

Observable and observation

In EM, *observables* are entities that can be directly apprehended by a human observer. This can be virtually *anything* whose identity can be established through the experience of a human observer (Beynon, 2008). For instance, an observable can be a value (e.g. 8), a description (e.g. “the ball has flown in the air”), a feeling (e.g. happiness), an event (e.g. an apple has fallen from the apple tree), a volume (e.g. 500ml), a relative perception (e.g. loud music), a status (e.g. the train engine is switched on), etc.

In the construction of an EM artifact (also an EM model), the identification of observables is through making careful and meaningful *observation* of the referent, whether it is a thing in the real world or it is an idea in the mind of the human observer who made the observation. When the observation refers to a thing in the real world, the notion of observation in EM is similar to that in scientific experiment, where the EM artifact can be viewed as construal in Gooding’s sense. However, observation admits different interpretations according to the context and status of the observer (see §4.2.1 below). What is regarded as relevant (and irrelevant) is subjective and reflects the current interest and *mode of observation* of the human observer (Beynon, 2008). It also depends on the level of detail of the observation and the current configurations of the environment within which the observation is made. This makes the identification of an observable, its presence and existence, a situated and subjective matter.

In contrast to the systematic identification process (e.g. Noun-Verb analysis in object oriented analysis and design (cf. Meyer, 1997) for entities (e.g. objects, and variables in objects) in traditional programming, the identification of observables in EM is much more fluid. Observables may be defined, redefined and removed throughout observation and interaction with the EM artifact during the modelling process. Taking all these characteristics into consideration, the intended character of *observable* in EM is much broader than what is known as a *variable* in the sense of traditional programming.

Dependency

In an EM model, a *dependency* can be represented by a *definition*, which defines the relationship between observables. Typically, such a definition has the following form:

$$d \text{ is } f(x_1, x_2, x_3 \ldots x_n)$$

where d, x_1, x_2, $x_3 \ldots x_n$ are observables, *is* is the keyword for separating the dependent and the observables upon which it depends. With this definition, observable d changes when any of the observables x_1, x_2, $x_3 \ldots x_n$ changes. Conceptually, the state-changing activity behind a definition can be interpreted as a cause-effect link between observables. Technically, it can be viewed as a transparent and atomic transaction for change propagation among observables. However, it is inappropriate to think that the stage-changes of observables are separate change-update steps as in traditional programming (e.g. change listeners in Java). The *state-changing* activities among observables occur simultaneously and atomically as in everyday experience of concurrency. For instance, as in everyday experience: expectations are framed by previous experience of the reliability of observations; observables and dependencies may be revised in response to new observations; dependencies reflect how state-changes to observables are perceived to be indivisibly linked (Sun, 1999). In these respects, the actual operation of a dependency in an EM model is closer to the dependency mechanism found in spreadsheets rather than component and service dependency as in Enterprise Java Beans (EJB) or *dependency injection* in design patterns.

Agent and Agency

In EM, an *agent* is typically conceptualised as a set of definitions and observables that causes state-changes within an EM model, and *agency* is associated with the "attributed responsibility (or privilege) for a state change to an agent" (Sun, 1999). In contrast to Artificial Intelligence (AI), EM has taken a much broader notion of agent. In classical AI, agents are thought of as entities that perform preconceived actions based on a stimulus-response model. In EM, by contrast the notion of agent is closer to what we may experience in attempting to understand a phenomenon in our everyday life. In this sense, agents can be "anything that has the capacity to change state" in an object (Harfield, 2008; also cf. Wong, 2003). For instance, I wake up when the alarm goes off (i.e. the alarm acts as an agent which changes my state to 'awake').

Like observables and dependencies, agents may be defined, refined and removed in response to the current *state-as-experienced* by the modeller (see §4.1.2 below). In addition, the attribution of agency is "shaped by the explanatory prejudices and requirements of the external observer, and by the past experience of the system" (Beynon, 1997[59] cited in Sun, 1999, p.26). Considering the previous example, it may not be the alarm that wakes me up, but an irritating beeping sound (which it may be that the alarm has generated) that wakes me up. Whether the agency involved in waking me up is attributed to the 'alarm' or to the 'irritating beeping sound' depends on the perception of the external observer.

In relation to the identification of agents and agency during the modelling process, Beynon (2008) describes three views of agents:

View 1: every observable or object is an agent, as is the external observer

In this view, an agent is an association of observables. Since every observable can potentially act as an agent that has potential effects on other agents (i.e. changing their states), an agent can be a collection of one or more observables that are co-existent and co-absent (Beynon, 2008; Wong, 2003). An analogy in object-oriented modelling may be a class with properties only (Wong, 2003).

View 2: agents are objects responsible for particular state changes

In this view, agents are deemed responsible for state changes to other observables or agents (Beynon, 2008). For example, the church bell sounds when somebody pulls the bell-rope. It is, of course, impossible to say that pulling the rope is the only way to cause the bell to sound. However, careful observations (e.g. there is a priest pulling the rope when the bell has been sounding) may influence the external observer to believe that such agency (i.e. connecting the sounding of the bell with the pulling of the rope) exists. In EM models, this kind of agency can be express

[59] The lecture notes for a MSc module (Beynon 1997) are now part of the "Introduction to Empirical Modelling (CS405)" course (cf. Beynon 2008)

using definitions (as discussed above), where state changes to x_1, x_2, x_3 … x_n trigger state-change of d.

View 3: virtual agency in the closed-world

In this view, the context of observation for the agent is so circumscribed that there is little doubt about its existence and presence (Beynon, 2008). Moreover, the behaviour of the agent is so reliable that its operations can be conceived in terms of stimulus-response actions (Beynon, 2008). Such a kind of agent can be considered as *closed* in the sense that further exploration is unlikely to add new insight or to change its behaviour (Wong, 2003).

Figure 4.1 illustrates the differences between the three views of agents. From a systems development perspective, the interest of the EM process is in the progression of agents from view 1 to view 3 through continuous observation of, interaction with, and experimentation with the EM artifact (i.e. the EM model). This process for circumscribing agency, where the development of the EM model progresses from obscure knowledge of agents to many circumscribed agents, is elsewhere known as the agentification process (e.g. Wong, 2003).

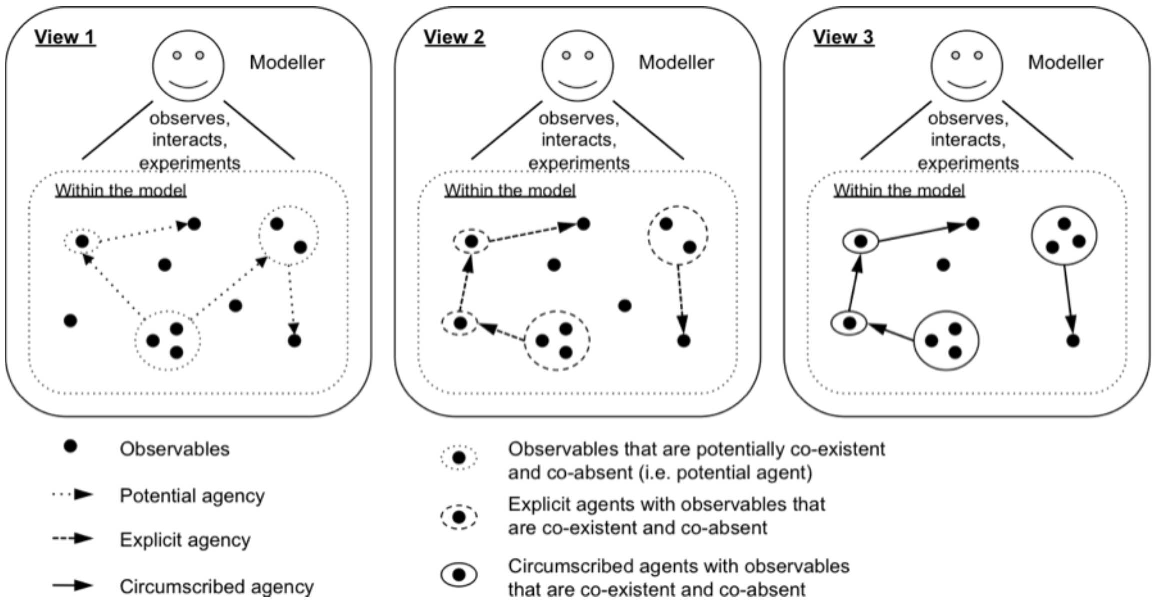

View 1
Modeller
observes, interacts, experiments
Within the model
View 2
Modeller
observes, interacts, experiments
Within the model
View 3
Modeller
observes, interacts, experiments
Within the model
Observables
Potential agency
Explicit agency
Circumscribed agency
Observables that are potentially co-existent and co-absent (i.e. potential agent)
Explicit agents with observables that are co-existent and co-absent
Circumscribed agents with observables that are co-existent and co-absent

Figure 4.1 – Three views of agent and agency in EM

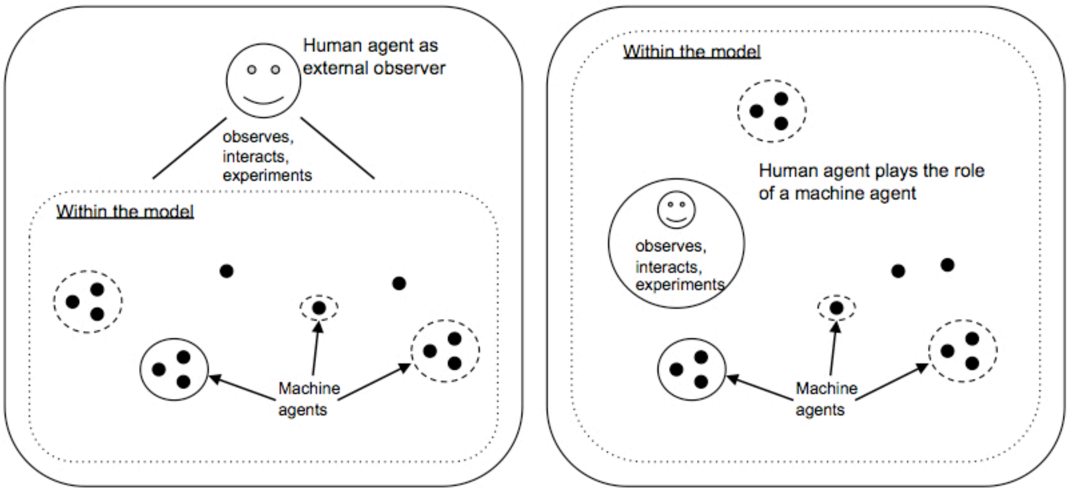

Human agent as external observer
observes, interacts, experiments
Within the model
Machine agents
Within the model
Human agent plays the role of a machine agent
observes, interacts, experiments
Machine agents

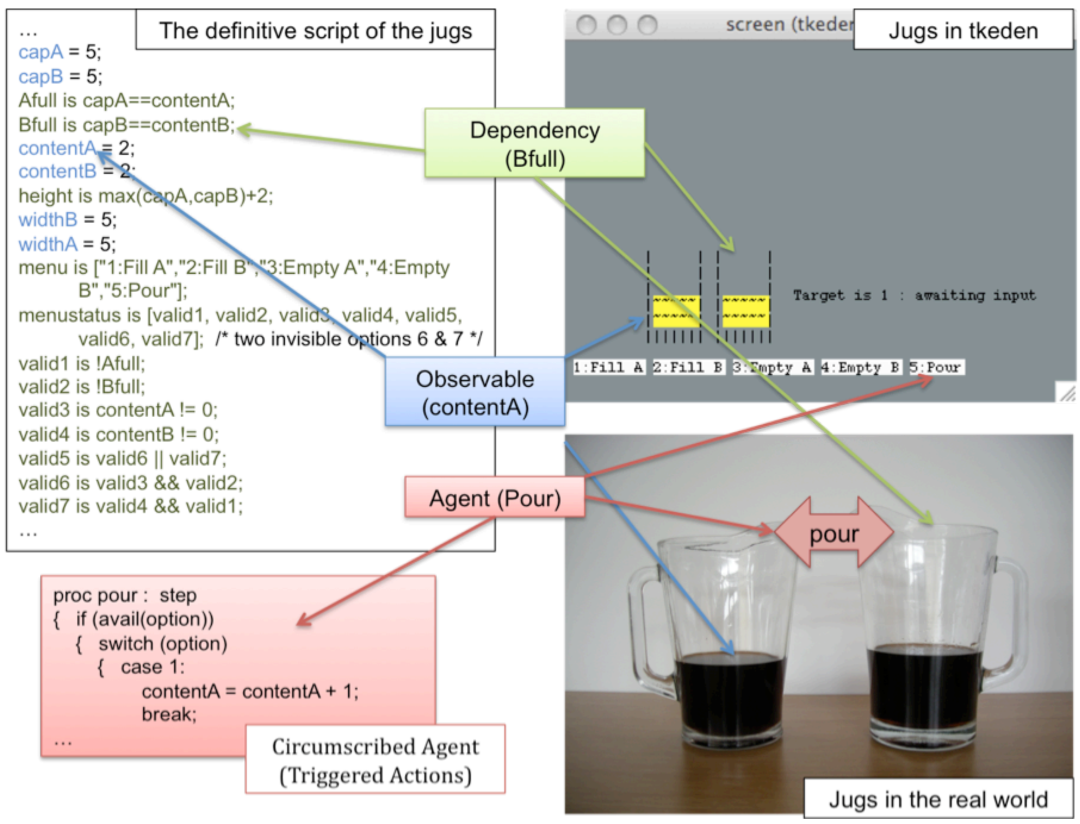
Figure 4.2 – External observer vs. internal observer in EM

It is worth noting that at any time during the EM process, the modeller (i.e. the human observer) may play the role of an agent as part of the experimentation and may also interact with other agents in the model through definitions and re-definitions. For this reason, it is important to distinguish two types of agent (and agency) within the EM process, namely, human agents and machine agents. The archetype for a *human agent* is the modeller who either, as an external observer, plays the role of super-agent that has the overall power to intervene the animation of the EM model, or, as an internal observer, plays the role of a machine agent within the model. Figure 4.2 illustrates the difference between a human agent as an external observer and a human agent as an internal observer.

When unspecified, the term 'agent' is used to refer to the agents in an EM model, i.e. the machine agents. In EM, the actions of a human agent can be circumscribed into a machine agent after prolonged observations (i.e. through the agentification process (Wong, 2003)). For example, the train drivers in the Clayton Tunnel model were initially human agents at the beginning of the modelling process, but some became machine agents in the model as the modellers gradually circumscribed them.

Illustrating ODA in the Jugs model

As I discussed earlier, *observables*, *dependencies*, *agents*, and *agencies* (ODA) in EM are open to change in response to continuous observation. However, for the sake of illustration, I shall temporarily ‘freeze the temporal factor’ of the modelling process. As depicted in figure 4.3, the ODA in the EM jugs model (EMArchive: jugsBeynon2008), as represented both in the definitive script and on the tkeden screen, intimately link to its referent in the real world. For instance, the liquid with Jug A is captured by the observable *‘contentA’* in the definitive script and is shown as yellow blocks on the tkeden screen in the jugs model. Similarly, the dependency “*Bfull is capB==contentB”* captures the definition that “Whether Jug B is full is depending on whether the liquid in Jug B has reached the capacity of Jug B”. In EM, dependency continues to hold even when, e.g., the size of Jug B has changed. This is in contrast to ‘dependency’ in conventional computer programming (as I mentioned in §4.1.1). The ‘pour’ agent closely resembles the pour action that can be performed on the pair of jugs by a human agent.



```
...
capA = 5;
capB = 5;
Afull is capA==contentA;
Bfull is capB==contentB;
contentA = 2;
contentB = 2;
height is max(capA,capB)+2;
widthB = 5;
widthA = 5;
menu is ["1:Fill A","2:Fill B","3:Empty A","4:Empty
        B","5:Pour"];
menustatus is [valid1, valid2, valid3, valid4, valid5,
        valid6, valid7];  /* two invisible options 6 & 7 */
valid1 is !Afull;
valid2 is !Bfull;
valid3 is contentA != 0;
valid4 is contentB != 0;
valid5 is valid6 || valid7;
valid6 is valid3 && valid2;
valid7 is valid4 && valid1;
...
```

```
proc pour :  step
{  if (avail(option))
   {  switch (option)
      {  case 1:
            contentA = contentA + 1;
            break;
...
```

Figure 4.3 – The relationship between the definitive script, the screen, and its referent in the jugs model (EMPA: jugsBeynon1988)

4.1.2 The EM process

Central to the EM process (i.e. the modelling process) is the notion of *modelling state-as-experienced* (Beynon, 2008) and *modelling with definitive scripts* (MwDS) (cf. Rungrattanaubol, 2002).

Modelling state-as-experienced

In traditional computer science, state is typically characterised in relation to the *internal abstract behaviour* of the *program*, which assumes the behaviour of a program can be fully apprehended within a fixed context for use. This conception can be reflected in the adoption of state machines, e.g. Turing Machines, state charts in mainstream computing, e.g. software engineering. In contrast, EM has a broader conception of state – a sense of state such as we apprehend in an everyday phenomenon. Such an everyday sense of state is open to diverse interpretations and can be refined through continuous observations and interactions that are based on personal experience (Harfield, 2008). It is this everyday sense of state, which Beynon (2008) calls *state-as-experienced*[60], that the EM process is most concerned with.

Figure 4.3 shows the 'final' jugs model (EMPA: jugsBeynon2008) as it was submitted to the EM repository. It consists of two jugs and a pouring agent. The model reflects the aspect of the situation as taken into account by Beynon at the time of submission, but this neither means that the situation had been modelled fully nor that it would be the same had another modeller modelled the same situation. For instance, the scripts of the model reflect the fact that modeller, when he made his observation of the jugs, was not interested in (or did not experience issues concerned with) the viscosity of the liquid in jugs, the speed of pouring action, the fact that the jugs liquid may be poured to somewhere other than into the jug next to it, or that the glass jug may be broken if it is filled with boiling liquid, and so on and so forth.

[60] Note that, state-as-experienced is not to be equated with 'what the modeller actually experiences at a particular time'. Instead, the term relates rather to the conception of state, and in particular to the potential to take account of more than has been explicitly observed and recorded.

Modelling with definitive scripts

“State-as-experienced encompasses the holistic experience of interacting with the computer-based artefact” (King, 2007, p.30). To support modelling state-as-experienced, EM artifacts (i.e. EM models) are represented in definitive scripts. This allows the modeller to construe[61] the state in many semantically different ways by adding new definitions and making redefinitions to the definitive script (Beynon, 2008). For example, a redefinition may represent a state-change of an observable in the model, a refinement of a dependency based on new observations, an interaction with the model, a re-attribution of agency, etc.

A *definitive script* typically consists of a number of definitions and these can be in various *definitive notations*. At the conceptual level, definitions define observables, dependencies, and agents (i.e. the ODA framework as discussed in §4.1.1). For effective modelling purposes, a number of definitive notations have been developed in the past 20 years for modelling in various domains in addition to the general purpose EDEN notation (Yung, 1993). These include, for instance, DoNaLD for line drawings (Yung, 1993), SCOUT for screen layouts and organising of information on screen (Yung, 1993), Sasami for 3D modelling (Carter, 2000), ARCA for Cayley diagramming (Beynon, 1983; Ward, 2004), EDDI for database modelling, etc. In the EM process, the modeller may observe, interact or experiment with the EM model. Hence, the major activity in the EM process can be viewed as *modelling with definitive scripts* (Rungrattanaubol, 2002).

In systems development, two typical kinds of activities can be identified (Beynon et al., 2006): exploratory and prescribed. While the former focuses on sense-making (i.e. process focus), the latter focuses on meeting predetermined goals (i.e. product focus). In contrast to a traditional perspective on systems development, EM has less interest in the predetermined goals, and is more concerned with sense-making activities through interaction with the computer-based artifacts – that is, through modelling with definitive scripts. As shown in figure 4.4, some of these exploratory actions can be later transmuted into actions that are no longer ‘sense-making’, as when a *ritualised* pattern of interaction and interpretation is

[61] The term ‘construe’ is used to refer to the activity of *making a construal* in the sense introduced in Gooding (1990). The connection between EM and Gooding’s idea on construal will be elaborated in §4.2.

discovered during the modelling process. Consequently, the EM process is often fluid and dynamic.

As shown in figure 4.5, the process of sense-making and ritualisation in EM involves continuous observation and interaction with the definitive script that reflects the modelling situation and the referent. In principle, the EM process will progress until the set of definitions reflect (or closely approximate to) some state of the referent as it is experienced by the modeller. However, state-as-experienced is an evolving object of study – observing the referent and the model and interacting with the model may provoke further insights, and therefore, create a *new* state-as-experienced. For this reason, this process may never be considered as 'completed'.

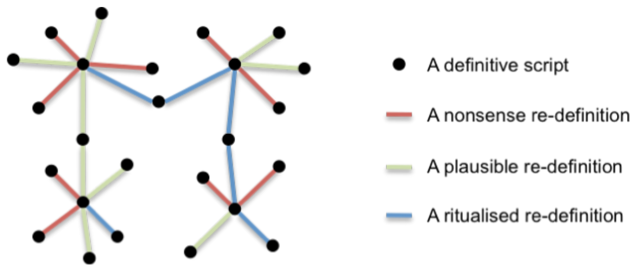

● A definitive script
🟥 A nonsense re-definition
🟩 A plausible re-definition
🟦 A ritualised re-definition

Figure 4.4 – Exploring ritualised definitions in EM[62]

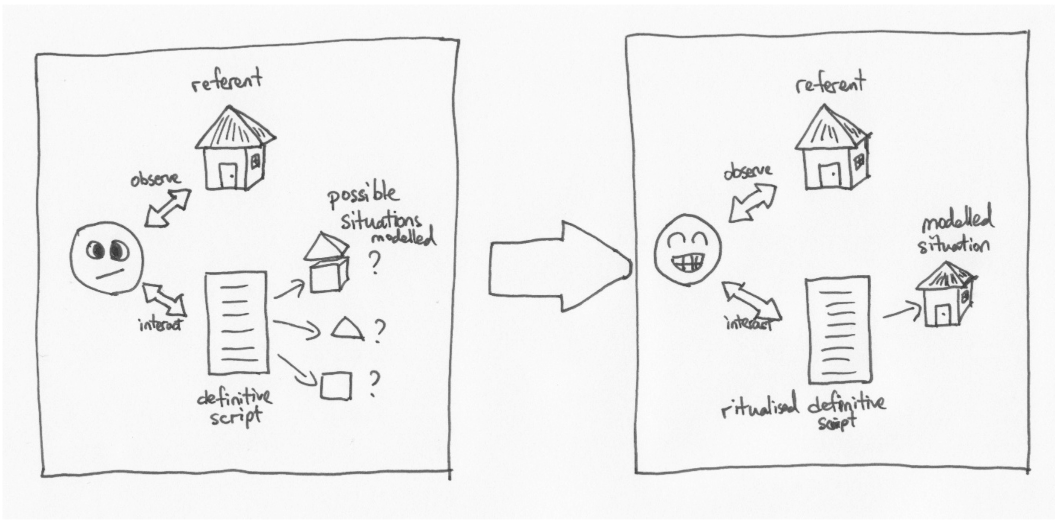


Figure 4.5 – Sense-making and ritualisation in EM[63]

[62] This figure is modified from (Beynon 2008).

4.1.3 The EM artifact

As I discussed in section 4.1.2, EM artifacts (or EM models) are typically represented by definitive scripts which make use of various definitive notations. However, it is a mistake to conceive EM artifacts as merely ‘scripts’. The script as-is can merely reflect a snapshot of the current status of the artifacts and the historical definitions and re-definitions. They do not reflect the modeller’s experience with the EM artifact and with the referent. Indeed, the interaction between the modeller and the EM artifact and between the modeller and the referent provokes ‘meaning’ for the interaction that ‘scripts’ cannot describe.

In traditional programming, the ‘meaning’ of a program is typically prescribed by its abstract behaviour in a particular context. In contrast, EM artifacts are *cognitive artefacts*, i.e. a tool that aids individual thoughts (Beynon and Cartwright, 1995). The meaning of EM artifacts is typically shaped by the modeller’s personal experience through continuous observation of and interaction with the artifact and its referent, though it is possible to prescribe an EM artifact in the same way as in traditional programming. In the same vein, Beynon argues:

> *“EM construals do not require an intended functional specification or a specific context for observation and interpretation. This is a matter of ontology, and is significant despite the fact that an EM construal can exhibit system-like and program-like behaviour subject to restricted forms of interaction. Indeed, imposing a strict functional specification and specific modes of observation and interpretation upon an EM artefact subverts its character as a construal.” (Beynon, 2009)*

To highlight the cognitive aspect of EM artifacts, Beynon (Beynon, 2008) adopts the term *construal* from Gooding (Gooding, 1990), where the term is used to refer to the pre-theory conceptual constructs that experimental scientists use to record and communicate their understanding of an observed phenomenon:

> *“Construals are a means of interpreting unfamiliar experience and communicating one’s trial interpretations. Construals are practical,*

[63] This figure is inspired by (Beynon 2008) and (Harfield 2008).

situational and often concrete. They belong to the pre-verbal context of ostensive practices." (Gooding, 1990, p22)

"... a construal cannot be grasped independently of the exploratory behaviour that produces it or the ostensive practices whereby an observer tries to convey it." (Gooding, 1990, p88)

In Gooding's (1990) sense, construals are provisional, situated and often associated with tacit knowledge that is in a pre-verbal context that may only be conveyed through direct demonstrations (i.e. ostensive practices). To this end, King (King, 2007) summarised the essence of construal elegantly:

"… a construal is a particular understanding of a situation, "a communicable representation" developed empirically as the result of situated observation, interpretation and experiment. It is always subject to future embellishment and re-interpretation." (King, 2007, p.32)

As mentioned earlier, an EM artifact is always open to changes in response to the modeller's new insight into the referent through continuous observations. These observations are based on the modeller's subjective and personal experience of the situation being observed. More importantly, this allows the modeller to explore through and within the model to find *an efficacious configuration* that reflects the state-as-experienced (by way of adding new definitions or changing the existing definitions in the model). Like Gooding's construal, EM artifacts are provisional, situated, tacit, interactive, and ostensive (cf. e.g. Beynon, 2008, Harfield, 2008). To exploit these characteristics, EM models are often viewed as *interactive situation models* (ISM), a term coined by Pi-Hwa Sun (Sun, 1999; Wong, 2003, Beynon et al., 1999). ISMs complement text-based documents and are to be used to construe *situations* in systems development (e.g. the situations in user-artifact interaction) and as "a communicable representation" to cultivate requirements (Sun, 1999). Figure 4.3 shows a snapshot of a jugs model (EMPA: jugsBeynon1988). Such a model can be thought of an ISM because it facilitates the analysis of the *use* situations for a pair of jugs. Besides being used to analyse interaction between users and artifacts, EM models have also been used in other domains. These include geometric modelling (Cartwright, 1999), creative product design

(Ness, 1997), education (Roe 2002), systems development (Wong, 2003), and constructivist computing (Harfield, 2008).

From a systems development perspective, the status of an EM artifact may be considered to be dynamic[64]:

> **EM artifact as *artifact*** – when viewed merely as an artifact, an EM artifact is simply a source of experience for the modeller. That is to say, there are observables, dependencies, agents and agencies associated with the artifact, but there is not necessarily an external significance for the ODA patterns that the modeller encounters in the artifact.
>
> **EM artifact as *construal*** – when viewed as an construal, an EM artifact serves an exploratory and explanatory function. The observables, dependencies, agents and agencies typically have counterparts in some external phenomenon, but the correlation between experience of the artifact and experience of the phenomenon is provisional in character. The ODA patterns in the construal are subject to change as the modeller's understanding of the external phenomenon develops, and as the construal potentially acquires a clearly defined referent through further experimentation.
>
> **EM artifact as *model*** – an EM artifact can be viewed as an model when it can be placed in a context in which the ODA patterns it embodies are closely matched to an external referent. In this case, subject to discretionary interaction on the part of the modeller, there will be a precise correspondence between observables, dependencies, agents and agencies in the EM artifact and in its referent. For instance, interacting with the model may be a reliable guide to what happens when interacting with its referent, and vice versa.

[64] Notice that the stage of the EM model is not the same as the process stage as in systems development such as analysis, design, implementation, and maintenance. The ideas for this classification is drawing on two unpublished manuscripts written by Meurig Beynon: i) a working document Beynon prepared for an AHRC ICT Methods Network Expert Workshop held at Loughborough University in May 2008; and ii) a presentation manuscript Beynon prepared for the introductory talk for the Thinking Through Computing Workshop held at University of Warwick on 2-3 November 2007.

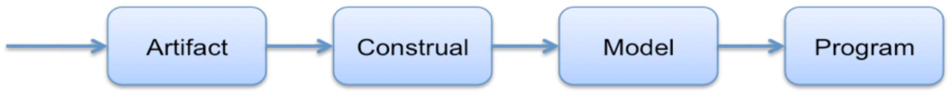


Figure 4.6 – The EM perspective on artifact, construal, model, program

> **EM artifact as *program*** – an EM artifact can be viewed as an program when it can be placed in a context in which the ODA patterns it embodies are closely matched to an external referent that has the characteristics of a computing device. That is to say, within a suitable circumscribed context, there are patterns of interaction and interpretation with the artifact that can be reliably reproduced and that realise purposeful changes that can serve a 'computational' function.

Figure 4.6 depicts a possible systems development progression view of an EM artifact, from artifact to program. Note that the dynamic status of the EM artifact is not an indication that the EM development process has well-defined stages similar to those in traditional systems development, such as analysis, design, implementation and maintenance. It is inappropriate to conceive EM model-building as proceeding through transformation of an initial artifact to a construal, to a model, then to a program. The actual status of an EM model can be much more fluid and subtle. As argued previously, EM artifacts are always open for changes and new interpretations based on continuous observations and at the discretion of the modeller. The status of the artifact reflects possible views and interpretations that can be made by the modeller at any given time during the development, depending on the maturity of their understanding of the model and its referent and the context. At any one time, the status of a given artifact may sustain more than one interpretation, depending on how the modeller chooses to interact with it. In the richness and diversity of interrelated resources that inform the modeller's understanding, the scope of the EM process and model-building activities is analogous to lifelong learning (cf. Beynon and Harfield, 2007).

4.2 The philosophical foundation of Empirical Modelling

Much has been written in the EM literature about the philosophical foundations for EM. Relevant works (e.g. Beynon, 2005; Beynon and Russ, 2008; Beynon et al., 2008; King, 2007) relate to connections with William James's (1912) *Radical Empiricism*, David

Gooding's (1990) *Experiment and the Making of Meaning*, and various works by Brian Cantwell Smith, particularly his *Two Lessons of Logic* (Smith, 1987). Familiarity with these philosophical connections is most helpful in understanding this thesis, but a resume of the relevant ideas is beyond the present scope. The author has however made his own original contribution to scholarship on the philosophical foundation of EM with reference to Russell Norwood Hanson's (1958) *Patterns of Discovery* (cf. §4.2.1) as will be briefly outlined in the following.

4.2.1 The notion of observation and theory building

Among all the EM concepts, the notion of observation is arguably the most important – because all the activities in EM are either initiated or triggered by observation (cf. §4.1.1). From a philosophical perspective, observation is more than just visionary experience or *seeing with eyeballs*. Observation is a personal experience. In his influential work *Patterns of Discovery*, Norwood Hanson (1958) argues that observation is a complex and unified process that involves visionary sense-datum experience (i.e. "*seeing as"* in Hanson's terms) and conceptual organisation (i.e. "*seeing that"* in Hanson's terms). Based on his analysis of how physicists observe, he argues that *seeing as* and *seeing that* are indispensable and inextricably linked:

> *"Seeing is not only the having of a visual experience; it is also the way in which the visual experience is had." (Hanson, 1958, p.15) "'Seeing as' and 'seeing that' are not components of seeing, as rods and bearings are parts of motors: seeing is not composite. … 'Seeing as' and 'seeing that', then, are not psychological components of seeing. They are logically distinguishable elements in seeing-talk, in our concept of seeing." (Hanson, 1958, p.21)*

Hanson's argument is that in observation seeing and thinking are intertwined. This is in contrast to the traditional analytical philosophy which treats the mind (i.e. thinking) and the body (i.e. the sensory experience) as separable. Hanson further argues that observation is actually shaped by the observer's prior knowledge of the phenomena (i.e. is *theory-laden* in Hanson's terms). In the other words, the observer knows what to observe prior to the observation being made, and what can be observed is influenced by the 'theory' (e.g.

knowledge, trainings, experience, etc.) that the observer already has. Following Hanson's argument, observation is not an equivalent process to all people: the *interpretations, connexions*, and conceptual organisation involved in the observation are different for all people.

EM is concerned with modelling through a notion of observation that is analogous to what is described by Hanson. In EM, observation is intricately intertwined and inseparable within the EM process. The EM process is advanced through the modeller's insights that are provoked within the context of observation. Through observing, experimenting and interacting with the EM model and its referent in the real world, the knowledge of the referent in the real world and the knowledge of the EM model can inform each other in the modeller's head.

The notion of observation as described here is relevant to abductive reasoning. Hanson (1958) argues that abduction (also known as retroduction) (cf. Peirce, 1931-1958) plays a crucial role in the scientific theory building process and that the influence of abduction is often overlooked in most hypothetical-deductive (H-D) accounts of physical theory[65]. He examines the experimental process that leads to the discovery of facts. He argues that, contrary to H-D accounts, which presume that a hypothesis is the primary ingredient in all experiment, many scientific theories are actually 'developed backwards'. That is to say, the scientist begins with the results, not from a hypothesis, and usually seeks an explanation of his/her observation of the results. Hanson studies the process that eventually led Galileo to formulate the conception of constant acceleration between 1604 to 1632. He further argues that Johannes Kepler was in fact working bottom up from Tycho Brahe's data, from "explicanda to explicans", when he discovered that Mars' orbit was actually elliptical (Hanson, 1958, p.85). These examples of scientific theory development lead Hanson to assert that scientific reasoning and the process of theory building do not follow a hypothetical-deductive system:

[65] Peirce writes "[induction] sets out with a theory and it measures the degree of concordance of that theory with fact. It never can originate any idea whatever. No more can deduction. All the ideas of science come to it by the way of Abduction. Abduction consists in studying facts and devising a theory to explain them. Its only justification is that if we are ever to understand things at all, it must be in that way. Abductive and inductive reasoning are utterly irreducible either to the other or to Deduction, or Deduction to either of them …" (quoted in Hanson 1958, p.85)

"By the time a law has been fixed info an H-D system, really original physical thinking is over. The pedestrian process of deducing observation statements from hypotheses comes only after the physicist sees that the hypothesis will at least explain the initial data requiring explanation. This H-D account is helpful only when discussing the argument of a finished research report, or for understanding how the experimentalist or the engineer develops the theoretical physicist's hypotheses; the analysis leaves undiscussed the reasoning which often points to the first tentative proposals of laws" (Hanson 1958, p.70-71)

"H-D accounts all agree that physical laws explain data, but they obscure the initial connexion between data and laws; indeed, they suggest that the fundamental inference is from higher-order hypotheses to observation statements. This may be a way of setting out one's reasons for accepting an hypothesis after it is got, or for making a prediction, but it is not a way of setting out reasons for proposing or for trying an hypothesis in the first place." (Hanson, 1958, p.71)

For Hanson, the intellectuality of empirical science actually involves both discovering the phenomena and working out a H-D account to convince others to accept that account as an objective explanation. The intentional character of EM, as an approach to computer-based modelling, is precisely concerned with discovery and identification of the agents in the context of observation that is analogous to the discovery process in empirical science. As Beynon and Russ (Beynon and Russ, 2008) pointed out, EM has been developed as an approach to fit the purpose of *pre-theory* experimentation with computing. Such interest in pre-theory experiments indicates that EM pays attention to the importance of abductive influences in the creation of innovative artifacts that is subsumed in the systems development process. As mentioned in section 4.1, the EM process does not predefine any path for exploration. It cannot (and should not) be seen as an approach that exclusively follows a deductive or an inductive approach to computer-based modelling.

4.3 Characteristics of practical EM

Despite the fact that the central activity of EM is modelling with program-like definitive notations, EM is often disregarded as an approach for programming. This section concerns the practical (and technical) differences between EM and common programming paradigms in various aspects. These include conceptual constructs, instant feedback, dialogical interaction, openness and flexibility, and experimentation. This is not meant to establish a dichotomy between EM and other programming paradigms, but rather, to help the reader to understand EM. This is done by comparing, contrasting and relating EM concepts to concepts in other programming paradigms.

4.3.1 The tkeden tool

Tkeden is the primary modelling tool for EM. It is a modelling environment which supports the core EM activity – *modelling with definitive scripts* (cf. §4.1.2). Tkeden is developed using the C language and the Tcl/Tk toolkit, and it runs on various platforms[66] (Ward, 2004). Perhaps the most noticeable and most discussed[67] feature of the tool is its dependency maintainer. The built-in dependency maintainer keeps the conceptual integrity of the model by maintaining the atomicity of the state-changes among observables within an EM model (cf. Wong, 2003), as they are experienced by the modeller when interacting with the referent in the sense of everyday concurrency (cf. §4.1.1).

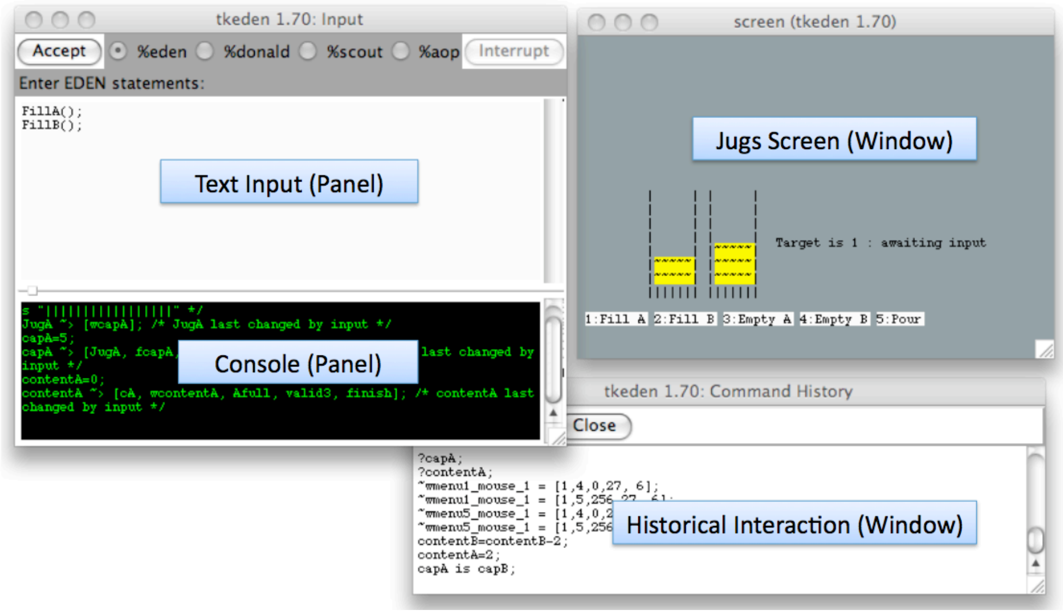
The development of tkeden can be dated back to 1987, when Y. W. Yung developed EDEN (the Evaluator for DEfinitive Notations) in his final-year undergraduate project. Yung's (Yung, 1990) EDEN later became ttyeden, which is a terminal-based EDEN interpreter and the precursor of tkeden (Ward, 2004). The first version of tkeden was developed by Y. P. Yung in connection with his post-doctoral research (Yung, 1996; Ward, 2004), and there have been a number of subsequent contributors including Pi-Hwa Sun (1999), Ashley Ward (2004),

[66] These include Solaris SunOS Linux, MacOS, MacOS X, and Microsoft Windows

[67] Much of the literature in EM has discussed, and more or less emphasized, this feature in comparison with other programming languages and environments, e.g. (Wong 2003), (Heron 2002), (Sun 1999), (Yung 1993).

Antony Harfield (2008), Charles Care, Russell Boyatt, and myself[68].

Figure 4.7 shows the current tkeden modelling environment (version 1.71) loaded with the jugsBeynon2008 model, a standard example of an EM model. The environment includes an input panel for the modeller to interact with the model in a command-line fashion, a console panel for text-based responses (e.g. the definition of an observable when queried), a window for displaying the graphical components of the model[69], and a window for displaying the interaction history. It is also possible to invoke a window for displaying the current set of definitions that constitute the model. As shown on the toolbar of the input panel in figure 4.7, the current tkeden modelling environment, by default, pre-loads four core definitive notations when it is invoked, namely, EDEN, DoNaLD, SCOUT, and AOP:



```
FillA();
FillB();
```

```
s "||||||||||||||||" */
JugA ~> [wcapA]; /* JugA last changed by input */
capA=5;
capA ~> [JugA, fcapA,                last changed by
input */
contentA=0;
contentA ~> [cA, wcontentA, Afull, valid3, finish]; /* contentA last
changed by input */
```

```
Target is 1 : awaiting input
```

```
?capA;
?contentA;
~wmenu1_mouse_1 = [1,4,0,27, 6];
~wmenu1_mouse_1 = [1,5,256
~wmenu5_mouse_1 = [1,4,0,2
~wmenu5_mouse_1 = [1,5,256
contentB=contentB-2;
contentA=2;
capA is capB;
```

Figure 4.7 – The tkeden modelling environment loaded with the Jugs model (EMPA: jugsBeynon2008)

[68] My interest has been in improving and extending dtkeden (the distributed version of tkeden which is developed by Pi-Hwa Sun for distributed modelling) for collaborative modelling purposes.

[69] In theory, the number of screens of a model can vary from zero (i.e. no graphical components) to the max number that is allowed by Tcl/Tk and the host operating system.

The ***EDEN*** notation is a general purpose language which supports the principles of EM. It was initially developed in Y. W. Yung's (1987) final-year undergraduate project. Technically, the EDEN language has a C-language-like syntax and it allows the modeller to construct models with a mix of definitive constructs (i.e. observables, dependencies, and agents) and procedural constructs (i.e. sequential functions and procedures). However, procedural constructs can be re-defined in Eden (the existing definition of an identifier is *replaced by a new definition* rather than *overloaded* in an object-oriented language such as C++) (cf. Yung and Yung, 1988).

DoNaLD (a Definitive Notation for Line Drawing) notation was developed by Steve Hunt and Tim Bissell on the basis of the ideas introduced by David Angier in his final-year undergraduate project[70] (cf. Wong, 2003; Beynon et al., 1986). Unlike Eden, DoNaLD is a pure definitive notation, and allows observables to be arranged in a hierarchical structure (cf. Beynon et al., 1986). Line drawing agents may consist of primitive agents (e.g. *line*, *shape*, *arc*, *circle*, *ellipse*, *rectangle*) and other line drawing agents, and they are grouped into *viewports* (i.e. a drawing canvas).

SCOUT (the definitive notation for SCreen layOUT) was originally developed for describing screen layout, i.e. prototyping text-based user-interfaces, in Y. P. Yung's (Yung, 1988) final-year undergraduate project. However, the usage of SCOUT soon exceeded its intended scope. It has been used in various ways not initially envisaged by the original author of the notation. For example, it has been extensively used to specify panels to arrange information on the screen (EMPA: projecttimetableKeen2000, racingGardner1999), and as a presentation tool (EMPA: drawSlideKing2004). This exemplifies the way in which the users may 'redesign' the use an EM artifact in ways which it is hard, if not impossible, for the 'designer' to limit.

The ***AOP*** notation[71] is a definitive notation which allows the development of new

[70] The DoNaLD interpreter was first implemented by Y. W. Yung in 1987, and subsequently enhanced by Chan (1989) and Parsons (1991).

[71] The Agent Oriented Parser (AOP) was originally developed by Chris Brown (2001) in his final-year

notations without re-developing tkeden source code, or even on the fly! (cf. Harfield, 2002; 2008) For instance, in an undergraduate project, Graeme Cole has developed a notation for the player to interact with the wumpus game (EMPA: wumpusCole2005).

An EM model often uses various definitive notations. For example, King exploited both EDEN and SCOUT in the Sudoku model (EMPA: sudokuKing2006). Beynon exploited EDEN, DoNaLD, and SCOUT in the Lift model (EMPA: liftBeynon2003). It is worth emphasizing that additional notations (e.g. EDDI, a definitive notation for database modelling) can be loaded on demand.

Further discussion of the history and the technical implementation of the tool are beyond the scope of this thesis. However, a more detailed discussion of its history and its implementation is available in Ashley Ward's (Ward, 2004) doctoral thesis. Ward was the primary developer of the tool between 1996 and 2005.

4.3.2 Conceptual constructs

Petre and Winder (1990) described a spectrum of programming languages between two extreme paradigms, imperative and declarative. While object-oriented and procedural languages fall into the imperative paradigm, functional and logic programming languages are declarative. Definitive notations (in EM), however, are neither imperative nor declarative, but somewhere between these two extremes (Yung, 1993). In section 4.1, I have discussed the basic concepts in EM. Unlike previous comparisons (e.g. Yung, 1996; Heron, 2002; Wong, 2003; Harfield, 2008), the following discussion does not contrast *modelling with definitive notation*s (MwDN) with individual programming paradigms in detail (as they cannot be 'compared' effectively because they are based on a different conceptual foundation), nor compare EM with programming in general. Instead, I relate, compare, and contrast conceptual constructs in EM's definitive notations with the closest conceptual constructs in other programming paradigms (cf. table 4.8).

undergraduate project, and enhanced by Antony Harfield (2002). It was turned into the aop notation by Charles Care in 2005, and was incorporated into tkeden (with minor modifications) by Antony Harfield in 2005 (cf. http://www2.warwick.ac.uk/fac/sci/dcs/research/em/notations/aop/).

Conceptual constructs in definitive script (EM)	**The closest conceptual constructs in other programming paradigm**
Observable	Variable in imperative programming paradigm. A cell in a spreadsheet.
Definition (or dependency)	Formula in spreadsheet.
Agent	Classless object in a prototype-based programming paradigm.
Virtual agent	Classless object in multiple namespaces in a prototype-based programming paradigm.
Triggered action (agent)	Event handler in an event-driven programming paradigm.

Table 4.8 – Conceptual constructs in EM versus the closest conceptual constructs in other programming paradigms

Observable

Technically, an observable in a definitive notation resembles a variable in an imperative programming paradigm; they are both a kind of identifier for some objects in a program (in the former) and in a model (in the latter). Furthermore, values can be assigned to both variables and observables. However, the identification of a variable is in relation to an abstract context, whereas in definitive notation (or EM) it is in relation to the experience of a human observer with reference to some referent in the real world (cf. §4.1.1).

Another point of technical difference between a variable and an observable is that the latter can be assigned with a definition (or dependency). In this sense, an observable is similar to a cell in a spreadsheet, whose value can be assigned using a formula involving other data values. However, an observable is type-less (e.g. in EDEN), which is different from a cell in a spreadsheet.

As mentioned earlier in section 4.1.1, the intended character of observable in EM is much broader than variables in traditional programming. Definitive notations address this concern by allowing the assignment of a definition to an observable. Since the definition of an observable gives information about how the state-change of the observable is produced and how it relates to other observables in the model, this can be thought of as giving a 'meaning' to the observable (Yung, 1993). Traditional program variables do not necessarily have a 'meaning' in this sense.

```
a is b + c;
b is d;
```

(a)

```
a is f(b);
b is g(c);
c is h(a);
```

(b)

Figure 4.9 – Cascading definitions and cyclic definitions in EDEN

Definition (or dependency)

As mentioned above, a definition (or dependency) in definitive notation is technically similar to a formula in a spreadsheet application. State-changes between the observables on the left hand side and the right hand side of the definition (cf. §4.1.1) are maintained as indivisible actions, in a similar fashion as transactions in data manipulation languages in databases. As in many spreadsheet applications, definitions can be cascaded but not cyclic. For instance, it is possible to have the EDEN definitions as in figure 4.9a, but not as in figure 4.9b.

Agent

If agents are seen as a natural extension of objects (cf. the discussion in (Wong, 2003)), perhaps the closest conceptual construct to an agent in EM is the classless object in a prototype-based programming language (cf. Noble et al., 2001). One example of such programming language is Javascript. In contrast to the manipulation (i.e. creation, inheritance, reuse) of class-based objects in object-oriented programming, manipulation of classless objects is done through cloning of existing objects in the program. However, the cloning process in definitive notations in EM is not the same as the cloning as in, e.g. Javascript. While the latter is supported by built-in functions in the language, cloning of agents in definitive notations needs to be done manually by the developer. The potential advantage of the classless structure is that the developer (or modeller) does not have to plan the object structure in the modelling process in advance, and the manual cloning makes every detail of a cloned agent explicit – which is necessary in EM, where the semantics of all features in the model is to be derived from their counterparts in the referent. The disadvantage is that such explicit cloning can make maintaining large models difficult (King, 2007).

```
proc yield : contentA
{
        writeln("The content of Jug A has changed !");
}
```

Figure 4.10 – Triggered action in EDEN

Virtual agent

The *virtual agent* is a special kind of agent in EDEN that was introduced by Pi-Hwa Sun in relation to his work in dtkeden (a distributed version of tkeden). A virtual agent in EDEN has the form:

virtualAgentName_identifier

where *virtualAgentName* is the name of the virtual agent, the underscore character (viz. '_') is the delimiter for virtual agent in EDEN, and *identifier* is the observable, dependency, or name of an agent within the context of the virtual agent. For Sun (1999), virtual agents are used as a mechanism for the human agent at the dtkeden server to "introduce definitions in a context *as if* they were being generated by an agent in a similar fashion" (Sun, 1999, p.171, italics in original). Therefore, virtual agents can be thought of as classless objects in multiple namespaces, as in prototype-based programming paradigms.

Triggered action

The EDEN notation, unlike the auxiliary pure definitive notations in the tkeden environment, supports triggered actions. Technically, a triggered action is similar to an event handler (also known as listener in some language) in event-driven programming paradigms – the event handler is invoked when a given event is raised in another thread within a program. Such event handling mechanisms are typically used to process input/output data (e.g. file access) or graphical user interface events (e.g. mouse clicks).

In EM, a triggered action is used to represent an agent that responds to a particular state-change of an observable. For instance, the EDEN fragment in figure 4.10 depicts a triggered action *yield* that will print a message to the console when the content of Jug A changed[72]. A triggered action can be thought of as an event handler because it responds to a particular state-change and handles multiple state-changes of various observables in a similar fashion as event handler. The qualities of triggered actions in EDEN are much enhanced by their use in close conjunction with definitive scripts. The appropriate use of triggered actions in conjunction with definitive scripts typically rationalises the way in which updates of observables are synchronised and greatly simplifies the maintenance of triggered actions.

The above sub-sections have reviewed the features of the tkeden tool and associated conceptual constructs. As is depicted in Figure 4.14 (on page 130), the qualities of EM as a modelling approach stem from key technical characteristics. These include – the fact that EM scripts are interpreted rather than compiled, that they exploit definitive notations rather than conventional procedural commands, and that sets of definitions entered into the model serve as a record of the interaction history. To help the reader to appreciate the distinctive character of EM, I shall now discuss specific qualities of the modelling approach in such a way as to expose the contribution that each of these technical characteristics makes.

4.3.3 Instant feedback

The tkeden modelling environment potentially offers instant feedback to the modeller as she develops her construal. The term *instant feedback* here refers to the fact that some state-changes of the model (or some actions) are visible to the modeller promptly or within a short period of time after a redefinition is input to tkeden.

This instant feedback feature can be attributed to the fact that definitive notations are interpreted, i.e. that tkeden animates the EM models by interpreting the definitive scripts, instead of compiling. Figure 4.11 illustrates the difference in feedback cycles between interpretation and compilation. In the case of interpretation, the program (or script) is analysed, parsed, and compiled in one step. In contrast, the program is stored in a machine executable form. The programmer (or modeller) will have to invoke a compiled program (or model) after compilation. Despite the better code security and performance that compilation

[72] Assuming the Jugs model (EMPA: jugsBeynon2008) is loaded in the tkeden modelling environment.

may offer, the time to obtain feedback is significantly slowed down.

In principle, any programming or modelling languages (or notations) can be interpreted. Therefore, languages that use interpreters may provide a certain degree of instant feedback. However, this potential further depends on the time to interpret the script (i.e. the interpretive overhead). Due to the interpretive overhead, the turnaround time can be significant if the script to be interpreted is huge. This becomes apparent, for instance, in database query languages (e.g. SQL) interpreters in which the turnaround time (i.e. the time to resolve a given query) is highly dependent on the size of the data set to be searched and the complexity of the query.

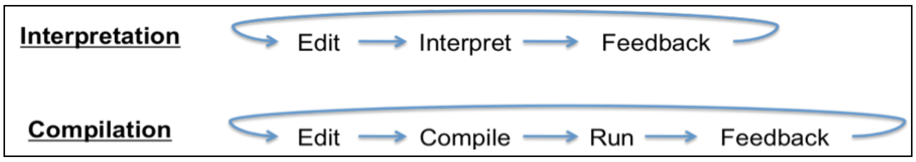
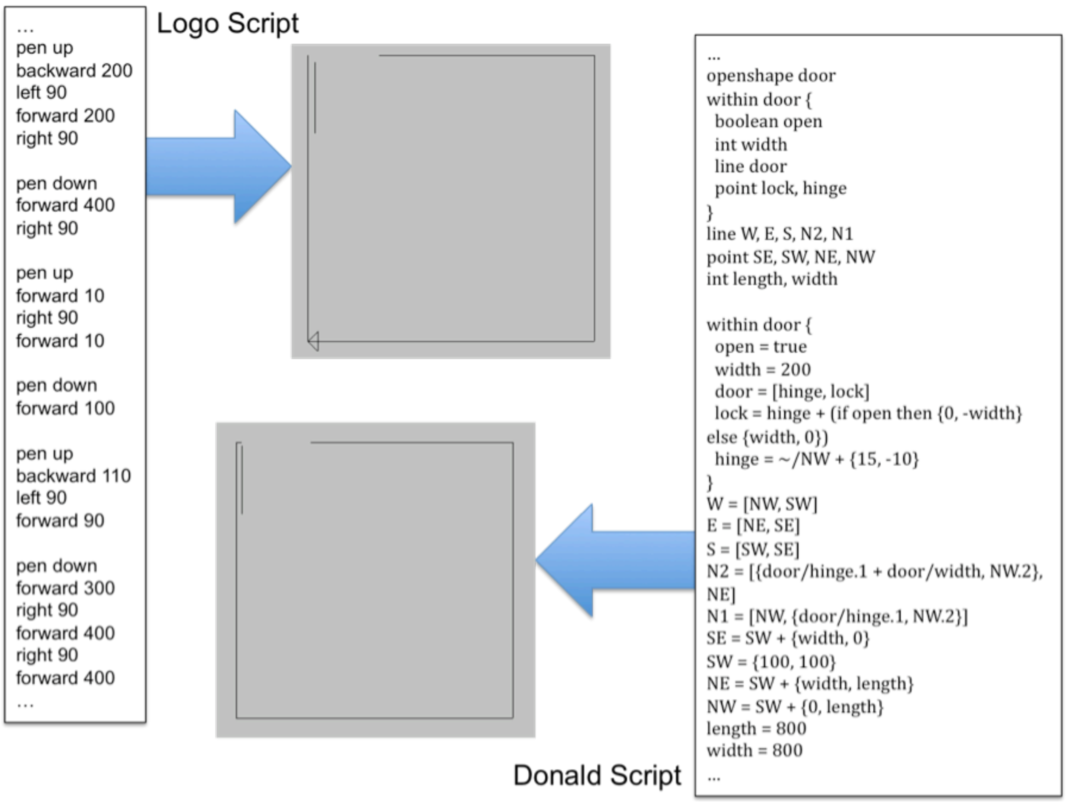


Figure 4.11 – The difference in feedback cycles between the interpretation and compilation



```
…
pen up
backward 200
left 90
forward 200
right 90

pen down
forward 400
right 90

pen up
forward 10
right 90
forward 10

pen down
forward 100

pen up
backward 110
left 90
forward 90

pen down
forward 300
right 90
forward 400
right 90
forward 400
…
```

```
...
openshape door
within door {
  boolean open
  int width
  line door
  point lock, hinge
}
line W, E, S, N2, N1
point SE, SW, NE, NW
int length, width

within door {
  open = true
  width = 200
  door = [hinge, lock]
  lock = hinge + (if open then {0, -width}
else {width, 0})
  hinge = ~/NW + {15, -10}
}
W = [NW, SW]
E = [NE, SE]
S = [SW, SE]
N2 = [{door/hinge.1 + door/width, NW.2},
NE]
N1 = [NW, {door/hinge.1, NW.2}]
SE = SW + {width, 0}
SW = {100, 100}
NE = SW + {width, length}
NW = SW + {0, length}
length = 800
width = 800
...
```

Figure 4.12 – Logo script vs. Donald script

The virtues of EM are not captured fully by merely considering the instant feedback that an interpreter offers. In this context, the special characteristics of EM are just as important. To illustrate this, consider, for instance, the small sequence of Logo commands that serves to define the outline floor plan of a room with a door with a Donald script that specifies the same drawing (cf. figure 4.12). Even though the sequence of Logo commands is interpreted, so that the modeller sees the line drawing being built up as the commands are entered, the sequence of commands is to be understood by the modeller as a recipe for drawing a floor plan. There is nothing comparable in the Logo script to the direct correspondence between observables in the Donald script and actual observables in the room. In modelling with the definitive script, the correspondence that is set up in the modeller's mind is then not between a recipe and the line drawing it creates, but between a dependency structure in which the elements (e.g. points, lines, attributes etc) have direct counterparts (e.g. as corners, walls, doors, widths etc) in the domain in which the line drawing is to be interpreted. This greatly enhances the semantic value of the instant feedback.

In addition, whereas the scale of revision necessary to adapt the line drawing recipe to make a meaningful change to the floor plan – e.g. to change the dimensions of the room, to make the door thinner, or open or close the door is quite hard to predict (and depends critically upon the particular drawing recipe developed), the redefinition of an observable is typically an update that is constrained in size. In this respect, EM encourages the modeller to make small changes incrementally rather than to make the batch changes, as is typically the norm in traditional programming due to the overhead in compiling or interpreting the program.

4.3.4 Dialogical interaction

The use of an interpreter not only potentially offers instant feedback; it changes how the modeller (or programmer) interacts with the model (or program). Because the use of an interpreter has shortened the interaction cycle (cf. figure 4.11), it encourages the modeller to interact with the model more seamlessly as in dialogical conversations in an everyday sense. As the use of interpreter removes significant time for compilation from the interaction cycle, the modeller switches between the development context and the use context more frequently than is the case with compilers. Furthermore, environments designed for interpreting programming languages often blur the distinction between the development context and the use context. For instance, a typical Logo environment offers the modeller direct access to the Logo script both for editing and running. In this respect, the use of interpreter promotes a conflated context of development and use. As observed by Bornat, this conflation of context is crucial for experimentation:

> *"In an environment where program design is constantly changing, where programmers are experimenting with the design of programs or where people are simply learning to program, an interpreter certainly saves human time and effort because it permits more effective interaction with the object program than is possible when the program is compiled." (Bornat, 1990, p. 356-357)*

Apart from the use of interpreter, the tkeden modelling environment provides a number of other features that promote dialogical interaction between the modeller and the model. In the first instance, these can be seen as stemming from technical advantages of modelling using spreadsheet-like definitive notations rather than program-like procedural commands:

Interactiveness of definitive notation – In practical EM, the unusual qualities of the dialogical interaction that is achieved arguably stem from the non-sequential nature of a script of definitions, that is to say, the fact that the order of the definitions is not significant. In contrast, the order of constructs in other programming paradigms, particularly in the procedural programming paradigm, can be significant. This flexibility of the notation structure affects how the modeller interacts with the model. For instance, 'redefining' a statement in an object in an object-oriented programming paradigm implies a reload of the containing object, if not a reload of the entire program. The significance of definitive notations in relation to flexibility will be discussed further in section 4.3.5.

Interactive prompt – The nature of the interactive prompt is similar to that of command shells in some operating systems (e.g. Linux, MacOS, DOS), in which users can create, interact, and amend 'objects' that are managed by the operating system. In practical EM, the modeller can manipulate the definitions of the EM model, interact with the EM model through redefinitions and, in addition, query the definitions of observables within the same modelling environment through the tkeden input panel (cf. figure 4.7). Interactive prompts are more common in databases, in which a database administrator can define, amend, and query data and the data model within the database through the interactive prompts without restarting the database. Some functional programming environments, e.g. Miranda, also provide an interactive prompt for the user to manipulate specifications interactively, and in the case of hybrid environments such as Mathematica allow such manipulation to be expressed visually through changes to graphics. In contrast, most programming environments for procedural programming languages do not provide an interactive prompt for the programmers to manipulate or query constructs on the fly, except with run-time debuggers. However, the programmers can only make changes to the program source code and reload the program to see the difference, or make changes to the objects which may not persist when the program is terminated in debuggers, rather than interact with the same program representation as in the development (i.e. non runtime) context.

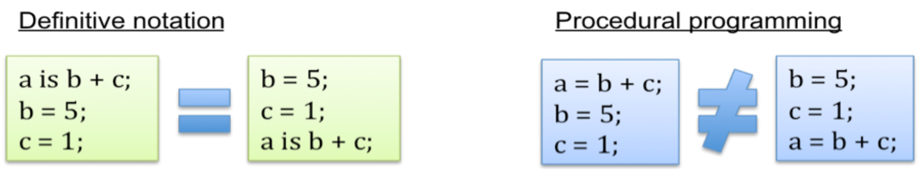
Interaction history – Interaction history potentially helps the modeller to make sense of a particular pattern of interactions with the model and to revisit a particular scene. For this reason, interactive environments, such as those for database query languages, may capture interaction history in log files. However, the interaction history that features in the use of EM is unusual in being of the kind which can be immediately fed back to the same interpreter for exploration[73]. In the tkeden modelling environment, interaction history consists of redefinitions which are either entered by the modeller through the input window or triggered by other agents in the model. Interaction history plays a crucial role in modelling with definitive script, as - reflecting a strong form of conflation of development with use - it can be viewed as

[73] To appreciate the advantages of the script, contrast the potentially wild effect of randomly choosing and re-executing a subset of commands from the Logo recipe in figure 4.12 (on page 124) with that of randomly choosing and re-interpreting a subset of definitions from the associated Donald script (which has no effect irrespective of the choice of subset and the order of interpretation).

itself part of the definitive script (cf. §4.1). In fact, the interaction history is the definitive script if the modeller initiates the modelling with an empty set of definitions rather than resumes a modelling session from a set of previously saved definitions. In tkeden, the interaction history is available to the modeller through the interaction history window (cf. figure 4.7).

4.3.5 Openness and flexibility

Openness to further exploration and interpretation (i.e. sense-making), and flexible structure are two interrelated characteristics of an EM model (cf. §4.1). In practical EM, the openness and flexibility of EM model are realised with the use of an interpreter and definitive notations in the tkeden modelling environment. In contrast to the use of compilers, where the program (or model) have to be compiled and/or encrypted into some form of low level human unreadable text, the use of an interpreter means that the script has to be *open source*[74] in principle. This makes the script always open to change.



```
a is b + c;
b = 5;
c = 1;
```

```
b = 5;
c = 1;
a is b + c;
```

```
a = b + c;
b = 5;
c = 1;
```

```
b = 5;
c = 1;
a = b + c;
```

Figure 4.13 – The difference between definitive notation and procedural programming

The virtue of a definitive notation is that the order of the definitions has no significance to the state of the model. As a simple example, figure 4.13 illustrates this character of definitive notation with two fragments of definitive scripts and two code fragments in procedural programming. The use of definitive notations in practical EM gives the modeller a flexible structure to construe a situation, in which, new definitions can be added to the definitive scripts and redefinitions can be made on the fly, or at any time during the modelling process. This can be done through the interactive prompt in tkeden modelling environment (cf. §4.3.1 and §4.3.4). Furthermore, the modeller can interact with the definitive script at the definition

[74] The term “open source” is here used in its literal sense to refer to the situation where the source code of a program (or model) is unencrypted and available to examination and modification by anybody who is interested.

level – a lower level of abstraction in contrast to, e.g. an object-oriented programming paradigm[75], where the data structure (or data model) and the methods for manipulating the data are often encapsulated in the same object. In such a case, it is not easy to make changes to individual data structure without affecting their common parents (i.e. classes). Indeed, the flexible structure in definitive notation eases the exploratory and experimental model-building activities in EM.

4.3.6 Experimentation

As mentioned at the beginning of this chapter, experimentation is one of the key motives for EM. Beynon and Russ (2008) argue that EM provides a holistic conceptual framework for *exploratory experiment*. In the journal paper *“Experimenting with Computing”*, Beynon and Russ (2008) distinguish two different kinds of experimental activity: post-theory and exploratory. In post-theory experiment, the context for observation is usually stable, and the outcomes can readily be observed when the key observables are changed. There is often a reliable basis for predicting criterion for successful outcome, whether drawn from theory or previous experience (Beynon and Russ, 2008). Despite the fact that there is some degree of human intelligence involved in setting up an experiment of this such kind, human interpretation is often a “marginal element” (Beynon and Russ, 2008). In contrast, the context for observation in exploratory experiment is preliminary and provisional, and human engagement in this context is crucial.

Beynon and Russ (2008) argue that it is in the context of exploratory experiment that the term ‘experiment’ has its authentic meaning, namely, “taking an action whose effect is unknown”. They distinguish such a context sharply from the context for post-theory experiment. In the context of exploratory experiment:

[75] The only exception is when the statement is enclosed in a procedure in the EDEN notation, in which case the modeller may have to redefine the entire procedure. This is due to the fact that EDEN is a hybrid notation that consists of both definitive and procedural constructs. However, there is no such restriction in other definitive notations such as DoNaLD or SCOUT.

"… the effect of an action cannot possibly be exactly predicted – indeed, it cannot even be uncontroversially identified or comprehensively registered." (Beynon and Russ, 2008, p.477)

Consequently, the support for exploratory experimentation is not straight forward in contrast to the support for post-theory experimentation, where the experiment can be set up with reference to suitable formalised procedures when certain key observables have been identified (Beynon and Russ, 2008).

Taken together, the various characteristics of tkeden discussed in the previous sub-sections (cf. §4.3.2 ~ 4.3.5) can be seen as supporting the use of EM for experimentation. Figure 4.14 depicts how these techniques and approaches support different characteristics of practical EM and how they give support for exploratory experimentation using EM.

Arguably, programming paradigms such as procedural, functional, and object-oriented fail to offer the qualities that definitive notations potentially afford for exploratory experimentation. For instance, procedural programming does not allow flexible structure (cf. §4.3.4); object-oriented programming maintains a high level of abstraction which may restrict effective interaction (cf. §4.3.4) and flexible structure (cf. §4.3.5); functional programming offers a poor correspondence between everyday observables and variables in the program which makes interaction and comprehension difficult (cf. Beynon and Russ, 2008).

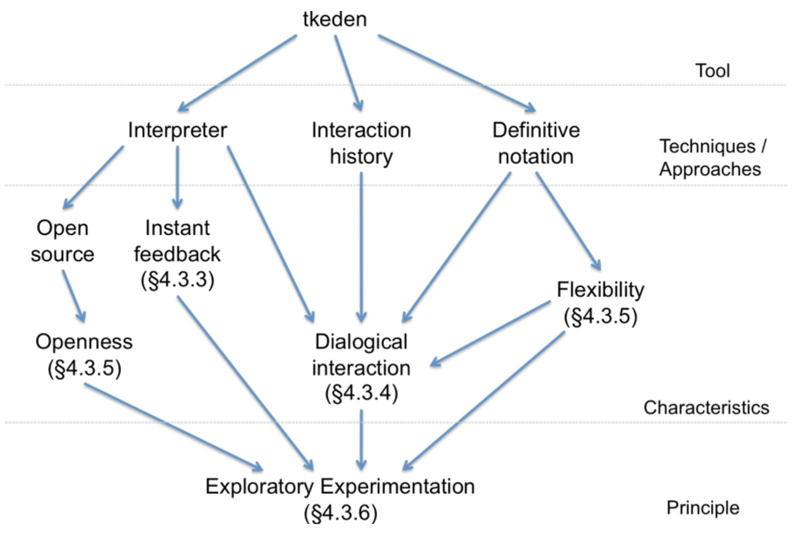


Figure 4.14 – Potential influence between instant feedback, dialogical interaction, experimentation, openness and flexibility

Experimentation not only needs support from programming paradigms, but also the support from the tool (or environment) as well. For instance, Javascript is an interpretable language, but it is very hard, if not impossible, to experiment with the script in the web browsers because they do not normally provide support for such interactions[76]. By contrast, spreadsheet environments provide much better support. A spreadsheet potentially offers instant feedback, openness and flexibility for creating numerical models. However, it does not provide an interaction history so that the modeller has no idea how he has reached the current situation.

In summary, I argue that current EM practice with the tkeden modelling environment potentially offers better support to exploratory experimentation. In the next chapter, I will discuss how EM can be practiced in collaborative modelling, an activity which I claim is central to groupware development (also cf. chapter 7).

[76] An interactive prompt is not a common feature and is not usually available in web browsers. Furthermore, the javascript console, if provided, is merely for displaying error messages, and not for taking input from the developer.

Chapter 5
Empirical Modelling for collaborative modelling

In the last chapter, I have discussed the philosophical foundation of Empirical Modelling (EM) and differences to other approaches to software systems development. So far, there has been little discussion on what kind of collaborative context may be benefited by adopting an EM approach and how EM principles such as the ODA framework can be exploited in collaborative work such as groupware development. Therefore, this chapter aims to explore the potential benefits that EM may offer to collaborative model construction[77] in general and the ideal level of tool support that is essential to unlock the potential benefit that EM may offer to collaborative modelling. The idea of applying EM principles to collaborative modelling as a framework for groupware development will not be discussed in this chapter, but in chapter 7 of this thesis.

In a broad sense, collaborative modelling refers to the process of model construction among two or more individuals with a common interest, whether or not computers are used. In a narrower sense, collaborative modelling may only refer to synchronized model construction among a group of individuals. It is hard to give a strict definition of collaborative modelling due to the very diverse ways in which the modellers can interact within the collaborative modelling process. This chapter begins with a discussion of three important aspects of collaborative modelling (cf. §5.1), namely, the degree of engagement, the relationship between the modellers and the agents in the model, and the modes of interaction. On the one hand, the idea of degree of engagement here is influenced by two models of

[77] i.e. the construction of an Interactive Situation Model (ISM) collaboratively.

collaboration in the literature, namely, (Borghoff and Schlichter, 2000) and (Kaptelinin and Nardi, 2006). On the other hand, the modes of interaction are inspired by the four paradigms for Distributed Empirical Modelling (DEM) (Sun, 1999). Though these aspects do not *define* collaborative modelling, it helps us to understand the types of interaction that may be involved in constructing a model collectively.

In section 5.2, the focus moves to the historical development in EM conceptual frameworks and tool support for collaboration. Previous research into EM has shown its potential in collaborative contexts such as concurrent engineering (Adzhiev et al., 1994a) and distributed work within organizations (Sun, 1999). Tools such as the ADM, LSD and dtkeden were developed for these contexts. However, these conceptual frameworks were based on theories for well-defined organization. They paid little attention to the diversified interaction and seamless moving between different modes of interaction and activity in collaborative contexts such as groupware development. In section 5.2, I also discuss why the existing conceptual frameworks and tool support are not good enough for collaborative modelling. In section 5.3, the focus is moved to the criteria for practising EM in the different types of collaborative activities that are mentioned in section 5.1, and collaborative modelling as a whole. Section 5.3 also argues that we should provide an integrated environment instead of isolated tool support for different aspects of collaborative modelling.

5.1 Defining collaborative modelling

The term “collaborative modelling” generally refers to a model construction process that requires a collaborative effort. Depending on what model is being constructed and in which context, collaborative modelling can be “carried out” in a wide range of domains from computer-based software design modelling (e.g. with UML), modelling of an eco-system (e.g. Salles and Bredeweg, 2003), to making conceptual models with pencil and paper. There are many ways in which different scenarios for collaborative modelling can be distinguished. On the one hand, the collaboration process can be carried out with or without computers. On the other hand, modellers in the collaborative process can interact diversely, e.g. asynchronously, semi-synchronously, or synchronously. It is hard to give a strict definition of collaborative modelling due to the general use of the term “modelling” and the diverse forms of “collaboration” that can appear in the collaborative modelling process.

Despite the ambiguity of the term, *collaborative modelling*, in this thesis, is used to refer to *the collaborative process for constructing an interactive computer-based model of a software system*. For this reason, collaborative modelling, in this thesis, should not be thought as a type of interaction or situation that may only take place in a particular configuration. It should be thought of as a process for which the aim is not an end product but a continual re-configuration of human practice. Therefore, the evolving and situated nature should be taken into account in this interpretation of collaborative modelling.

In this section, I describe three important aspects of collaborative modelling. My intention is to provide the reader with some handle on the kind of collaborative modelling that this thesis is concerned with and that will be used as the foundation for groupware development in chapter 7. These aspects play a crucial role in shaping the collaboration from time to time during the collaborative modelling process. Although they resemble a taxonomy of collaboration, they should not be viewed as one, as human collaboration cannot be “framed” by any means due to its dynamic characteristic.

As mentioned previously, the purpose of this chapter is to expose the potential of an EM approach to collaborative modelling. For this reason, the following discussion is oriented towards an EM perspective.

5.1.1 Five degrees of engagement in collaborative modelling

The idea of the degrees of engagement as discussed in this section is influenced by two models of collaboration in literature (Bair, 1989; Kaptelinin and Nardi, 2006).

Two models of collaboration

Borghoff and Schlichter (2000) describe four-level hierarchy of collaboration: inform, coordinate, collaborate, and cooperate. Their model is based on the earlier work of Bair (1989). Although the boundaries between one degree of engagement and another are not sharply defined, Bair’s (1989) research suggests that collaborations at different levels have different needs and require different communication media. Kaptelinin and Nardi (2006) describe another model of collaboration which draws on Raeithel's (1996) three-part scheme. Raeithel's hierarchy, in turn, is based on earlier work by Fichtner (1984). Unlike Borghoff and Schlichter, Kaptelinin and Nardi focus on the “creativity fissures” in collaboration.

Although these models of collaborative work seem compatible with each other, careful examination reveals that there are subtle differences. For instance, Borghoff and Schlichter (2000) focus on more low-level collaboration by including *informing*, which may not be considered as a form of collaboration in a strict sense. Borghoff and Schlichter (2000) suggest that people can coordinate without having a common goal (also cf. Bair, 1989), but that a common goal seems crucial to bind people together in Kaptelinin and Nardi’s model of collaboration. Similarly, Borghoff and Schlichter (2000) emphasize the demand for face-to-face meeting in co-operation, such synchronized interaction is not necessary, though it is favoured, in Kaptelinin and Nardi’s model of collaboration.

A unifying structure

Drawing on the above mentioned models of collaboration and our previous discussions on the evolutionary nature of group work (cf. §2.4) and the co-evolutionary phenomena in systems development (cf. §3.1), the degree of engagement in collaborative modelling can be classified into five levels: coordination, collaboration, cooperation, co-construction, co-evolution.

Coordination – Apart from *informing* as described in Borghoff and Schlichter’s (2000) model of collaboration, coordination has the lowest degree of interaction and coherence in collaborative work. In this case, individuals within the group are working alone but linked together through shared resources. For activity theorists, the coordination for the collective activity takes place outside the context of individual activities (Kaptelinin and Nardi, 2006). For instance, a group of salesmen may coordinate their use of, e.g. meeting room, on a shared schedule, and that is not included in the salesmen’s individual calendar.

Collaboration – In collaboration, individuals may work together in shared activity and with shared resources. However, the team may not have an explicit common goal. In this case, diverse perspectives are encouraged and some exploratory work may be required in order to shape a common goal for the collaborative work. For example, a preliminary brainstorming meeting for design ideas in a creative artifact development is a form of collaboration without an explicit concern for how the artifact would e.g. look and feel. It is worth stressing the importance of the unifying of diverse perspectives instead of imposing a perspective at some point from an individual either external to the group or a member of the group. While the former is the source of creativity in collaborative work, the latter could suppress creativity.

Cooperation – In cooperation, individuals need to be aware of other people's work in the team, and individual adjustments might be needed in order to work towards the overall shared objective of a more complex activity than in coordination. For instance, the early stages of the transformation process of qualitative data of traffic accidents into quantitative data as reported in Spinuzzi (2003) is a highly sophisticated cooperative work that cannot be done merely by coordinating the workers. In cooperation, individual goals may have to be sacrificed for the collective goal. Moreover, due to the potential conflicts, diverse perspectives are often not possible and may not be encouraged.

Co-construction – In co-construction, individuals do not just cooperate, but are also able to evolve and refine the objective of the activity during the collaboration (Kaptelinin and Nardi, 2006). This implies that individuals may have to develop shared understandings of the collective activity and of the artifacts under co-construction. For instance, groupware development is a form of co-construction among all stakeholders, developers, and future users. Effective groupware development involves not only cooperation, but will also develop a shared understanding throughout the collaboration process, and that shared understanding, in turn, will be used in the continuous refinement of the objectives for the groupware in order to cope with the changing work practices surrounding the use of the groupware.

Co-evolution – Co-evolution has the highest degree of engagement in the collaborative work, in which the individual understandings of all objects in the collaboration process (e.g. the objective, the artifacts under co-construction, etc.) and the artifacts themselves are all co-evolving throughout the collaboration process. The boundary between co-evolution and co-construction is blurred. The only difference between them lies in the individual and the collective understandings of all objects in the collaboration process: these understandings evolve to a greater degree in the case of co-evolution. In the context of computer-based systems development, however, co-evolution may only take place when the development team practises an evolutionary development approach in the co-construction process. In other words, co-evolution occurs through co-construction, not vice versa.

5.1.2 The relationship between modellers and agents in collaborative modelling

Artifacts play an essential role in facilitating collaboration between participants of the collaborative modelling process. Throughout the collaboration process, there is a close relationship between the kinds of interaction of the participants and the conceptual constructs in the artifact. When adopting an object-oriented approach to collaborative modelling, modellers engage and interact through sharing and co-constructing conceptual constructs (i.e. classes, objects, packages, etc). When practising an EM approach to collaborative modelling, modellers engage and interact through elaborating and negotiating observables, dependencies, and agents.

It seems self-evident that different relationships between the modellers and the conceptual constructs yield different kind of collaboration. I argue that, in fact, the relationship between modellers and agents characterises the collaboration in the context of agent-oriented computer-based modelling, such as EM. Figure 5.1 illustrates how this relationship can be roughly classified into four modelling scenarios in a two-dimensional matrix. It is the shaded areas in figure 5.1 that are classified as collaborative modelling.

For the sake of completeness, the following discussion includes “single modeller” modelling scenarios. Although such scenarios may involve a degree of collaboration between the modellers outside the modelling scenarios (e.g. through sharing the models), and may also involve discussion between modellers (cf. Harfield, 2008), “single modeller” modelling scenarios alone are not considered as collaborative modelling in the context of this thesis.

	Single modeller	**Multiple modeller**
Single agent	Single modeller single agent (SMSA)	Multiple modellers single agent (MMSA)
Multiple agent	Single modeller multiple agents (SMMA)	Multiple modellers multiple agents (MMMA)

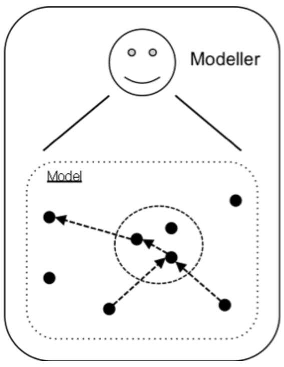
Figure 5.1 – Collaborative modelling scenarios

Single modeller single agent (SMSA)

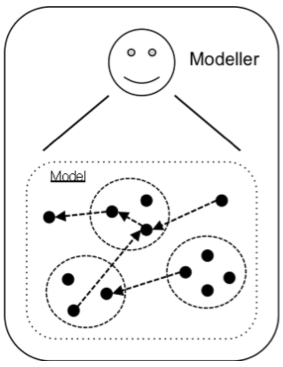
In this scenario, as shown in figure 5.2a, the modeller focuses on a particular agent. In EM, such a scenario requires the modeller to observe its referent and interact with the agent both in the role of an external observer ("externally") and in role of the particular agent being studied ("internally") (cf. figure 4.2). For instance, the modeller may investigate the relationship between the observables and dependencies within the agent; or the modeller may make internal observation to see how the agent interacts with other agents in the model. When such modelling is carried out as a part of a coordinated work activity, there is overlap between the observations made by different modellers and there is no real-time interaction among them. In the other words, a larger work may be divided into non-overlapping contexts in which individual modellers can carry out private modelling based on their personal observation.

Single modeller multiple agents (SMMA)

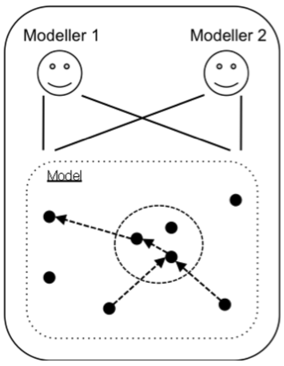
In this scenario, the modeller is managing multiple agents in the model and is mainly focusing on the interaction among the agents through external observation (cf. figure 5.2b). Such a modelling scenario is an embryonic form of the more complicated scenario – multiple modellers multiple agents – that will be described later in this section. In EM, the issue of agent identification is topical in this modelling scenario, because agency in EM is evolving in responds to the modeller's mode of observation and state-as-experienced (cf. §4.1). For instance, after the modeller has obtained further insight in the context of observation, a single agent may be decomposed into multiple agents, an agent may disappear, or the boundaries between agents may be changed.



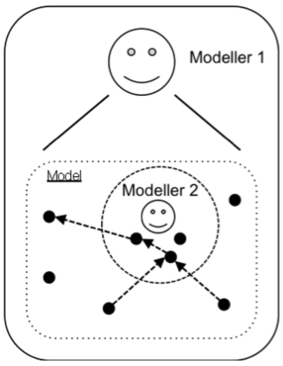
(a) Single modeller single agent



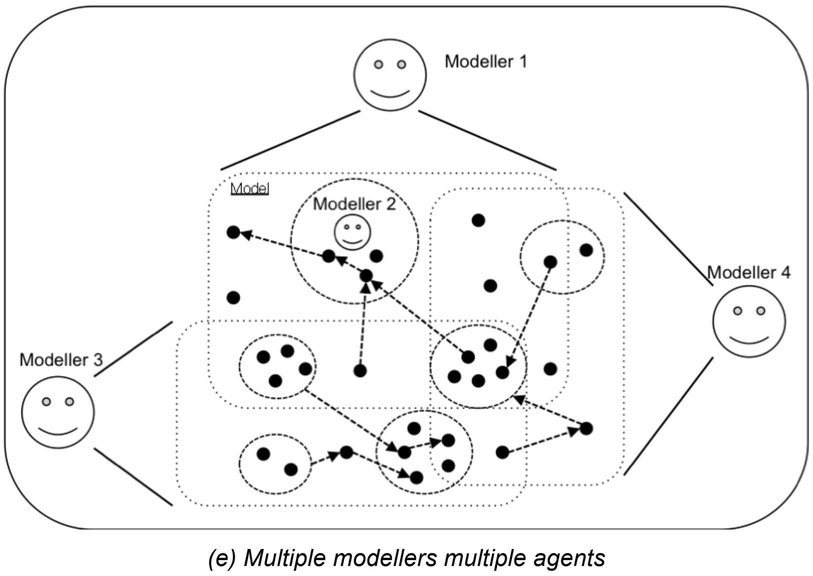
(b) Single modeller multiple agents



(c) Multiple modellers single agent (both modeller observe externally)



(d) Multiple modellers single agent (one modeller observes externally, one modeller observes internally)



(e) Multiple modellers multiple agents

Figure 5.2 – The relationships between modellers and agents in the model[78]

[78] The interpretation of the above diagrams follows the convention introduced in figure 4.1 (on page 103).

Multiple modellers single agent (MMSA)

Occasionally, multiple modellers may collaborate around the same agent in the model in the collaborative modelling process. As illustrated in figure 5.2c, collaboration of this kind may occur, for instance, in the context of pair modelling[79], where a pair of modellers focus on the same agent with the aim of to co-constructing, co-examining, or negotiating the properties[80] of the agent through external observation. Similar collaborations can occur among three or more modellers.

However, the more interesting situation in the MMSA relationship arises when modellers are collaborating around the same agent with diverse perspectives. In EM, this refers to the situation where one modeller makes internal observation and plays the role of the agent while the other modellers observe externally. Figure 5.2d, for simplicity, depicts this situation with two modellers – while modeller 1 is acting as an external observer, modeller 2 is exploring plausible definitions or redefinitions (cf. figure 4.4). This "diverse perspectives" MMSA, for instance, can be invoked when modeller 2 is demonstrating to modeller 1 how the agent might interact with other parts of the model (i.e. apprenticing), or when modeller 2 is performing exploratory experimentation while both modellers are making sense of the results of experimentation from different perspectives, one internally, one externally.

Multiple modellers multiple agents (MMMA)

The "multiple modellers multiple agents" scenario is the most interesting, but also the most complicated, relationship between the modellers and the agents in the model. In a simplistic view, when collaboration is ignored, the MMMA scenario can be thought as a scaffold scenario from the "single modeller" modelling scenarios – each of the modellers is in fact managing one (i.e. SMSA) or more (i.e. SMMA) agents in the model. However, the interaction between modellers through the agents in the model is not straightforward. When collaboration is taken into account, there can be many possible scenarios. For instance, as

[79] The term "pair modelling" is used by analogy with the notion of pair programming in extreme programming (Beck 1999). The term 'modelling' is used in place of 'programming' to emphasize the broader context of collaboration, in contrast to collaboration around a program in the programming context.

[80] Not to be confused with "the attributes of an object" in the sense of object-oriented modelling. *Properties* here means the characteristics, constituency, and boundary of the agent.

figure 5.2e depicts, it is possible that some modellers are observing a particular agent internally and playing its role in the model (cf. modeller 2), while others are observing or interacting with multiple agents externally (cf. modeller 1, 3, and 4). Moreover, modellers who are observing externally may focus on a different set of agents, whether many agents or a single agent, from time to time.

Although the above-mentioned scenarios are somehow similar to Sun's (1999) distributed modelling scenarios (cf. §5.2), the notion of collaborative modelling described in this thesis concerns not only playing the role of a particular agent through internal observing (I-modelling in Sun's sense), but the overall dynamic process of internal and external observation in such a MMMA situation.

5.1.3 The modes of interaction in collaborative modelling

The *modes of interaction* refer to communication or interaction patterns between modellers that relate specifically to interaction via the artifact (or model) and the computer-based tool support that is being developed throughout the collaborative process[81]. The concept, on the one hand, is inspired by the four communication paradigms that are closely associated with Sun's (1999) Distributed Empirical Modelling (DEM) framework. On the other hand, it is highly influenced by the possible *communication networks* in groups (Baron et al., 1992). Some of these patterns were incidentally found in a number of pilot experiments and case studies (cf. chapter 6) that are associated with the research work behind this thesis, and some were derived from the paradigms in DEM. With reference to Johansen's (1988) spatial-temporal classification, the modes of interaction introduced below can occur in all contexts of collaboration when the equipment is configured properly. Indeed, they are more generic modes of interaction which, in contrast to those described elsewhere (Sun, 1999; Beynon et al., 2003)[82], should not be viewed in association with the underlying technical

[81] What is not being considered is communication between modellers that can be carried out via email or other forms of communication that does not exploit or is not directly connected with the collaborative modelling environment.

[82] The interaction modes that are associated with the DEM framework are based on the client-server architecture of the dtkeden tool, and therefore, are limited to a star-like communication network (cf. Sun 1999).

architecture of the tool (e.g. a client-server architecture).

Private

In the private mode of interaction, modellers construct their own models in isolated contexts (i.e. in their private workspace). Though collaboration between modellers may exist, there is no interaction through the computer-based model among the modellers (i.e. collaboration takes place outside the modelling environment). In EM, this means each of the modellers constructs their own computer-based model on separate workstations. As shown in figure 5.3, though there is an overlapping area of observation and are potentially overlapping parts in the model, the modellers may not be aware of that potential overlapping with other modellers' models.

In EM, this means that the modellers do not share any observables, dependencies or agents between their models. In other words, in the private mode of interaction, modellers are either in SMSA or SMMA scenarios (as described in section 5.1.2). As an example, a couple of students sitting at workstations a short distance away in a classroom may build their own jugs EM model without any interaction within the modelling environment. However, they may discuss their progress from time to time through speaking loudly or other tools (e.g. instant messengers) outside the modelling environment.

Similar private modelling situations can be found in traditional software development, where developers are working in separated workspaces and integration only take place outside the individual workspaces. de Souza et al. (2004) observed that the application programming interface (API) has dual roles in collaboration in the software development process. On the one hand, APIs facilitate collaboration during the decomposition process because i) they serve as contracts between stakeholders and developers, and therefore a larger system can be broken down into smaller pieces and each of the pieces can be constructed by developers independently; ii) they enforce the organisational boundaries between software developer teams (deSouza et al., 2004). On the other hand, APIs limit the information available to the developers during the *recomposition* process (Grinter, 1998; also cf. deSouza et al., 2004).

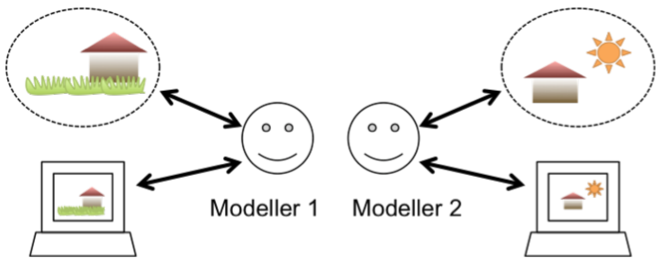


Figure 5.3 – Private mode of interaction

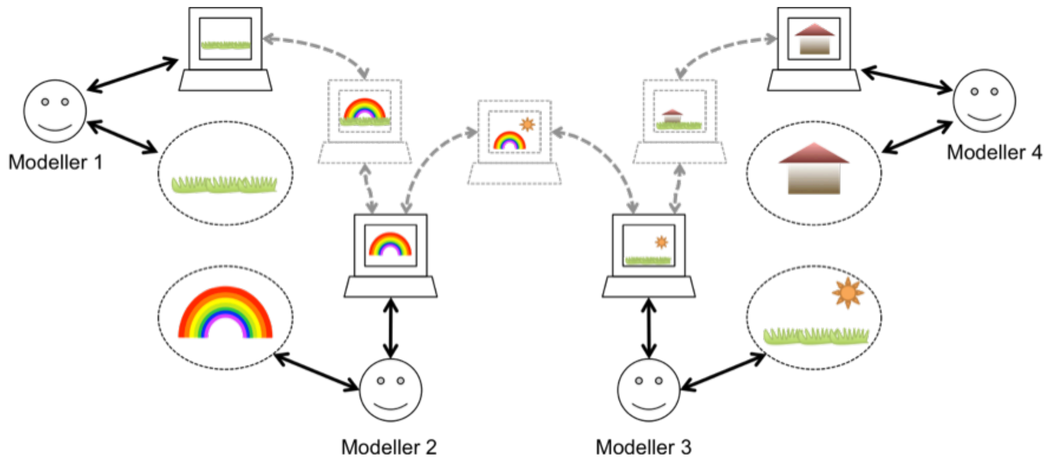


Figure 5.4 – Peer-to-peer mode of interaction

In the same sense, while the private mode of interaction may offer an isolated context for individual modellers to concentrate on their own model construction in an uninterruptable environment (as least in respect of the modelling environment), the private mode has little value when *recomposition* (Grinter, 1998) of individual models is required. In section 5.4, I will discuss how this issue may be overcome by practising an EM approach in the collaborative modelling context.

Peer-to-peer

In the peer-to-peer mode of interaction, modellers exchange parts of their model through "private channels" in the modelling environment (cf. figure 5.4). As Sonnenwald (1993) observed, peer-to-peer (or one-to-one) interaction often plays an important role in facilitating diverse perspectives in collaborative work. The term "peer-to-peer" reflects its fully decentralised characteristic. Such mode of interaction is commonly ad-hoc in nature, where interaction occurs spontaneously (i.e. on demand). Modellers are "connected" in pairs and each modeller may have multiple shared workspaces with other modellers. For instance, as shown in figure 5.4, modeller 2 is sharing her model with both modeller 1 and modeller 3, but in two separate shared spaces. It is worth noting that the sharing need not encompass the whole model, and parts need not be the same. For instance, in figure 5.4, modeller 3 shares “the sun” with modeller 2, but “the grass” with modeller 4; and neither “the sun” nor “the grass” is the whole model of modeller 3 currently constructing. The sharing between modeller 3 and modeller 4 echoes the MMMA scenario as mentioned in section 5.1.2. In EM, the peer-to-peer interaction implies that common observables, dependencies or agents only exist in the shared modelling space between the modeller pair. There are no overall authentic[83] observables, dependencies or agents among all modellers.

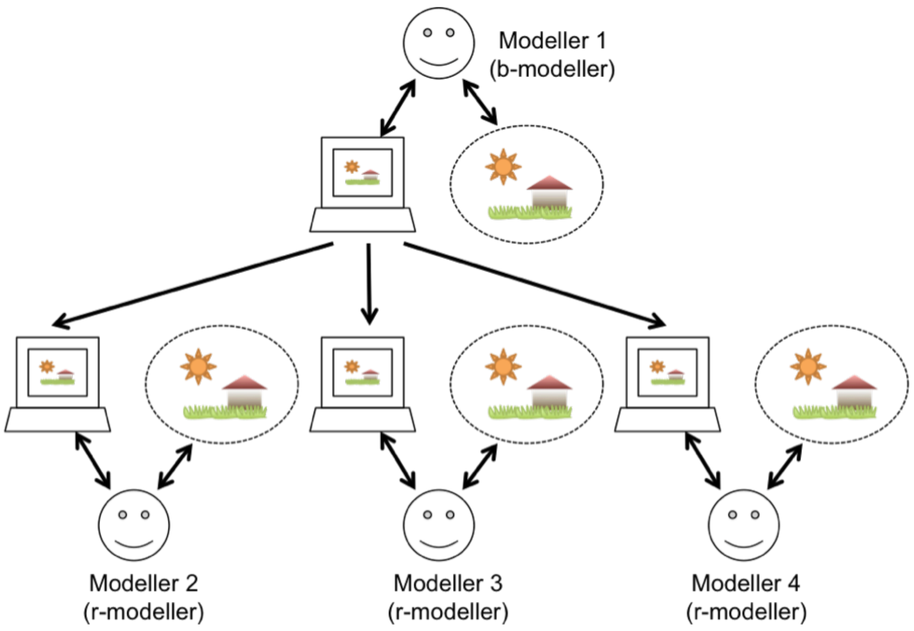


Figure 5.5 – Broadcast mode of interaction

[83] The word ‘authentic’ is used in the sense that is used in Sun’s (1999) thesis to refer to the instance of the observable that has the ‘objective’ value.

Broadcast

In the broadcast mode of interaction, modellers can be divided into two groups, namely the broadcasting modellers (referred to as *b-modeller* thereafter) and the receiving modellers (referred to as *r-modeller* thereafter). As shown in figure 5.5, the b-modeller broadcasts her model to all the r-modellers – all modellers are virtually seeing the same model through their workstation. Besides, all the r-modellers cannot modify the model and they can, in principle, only discuss the model outside the modelling environment. Because of this limitation, the r-modellers can be viewed as the subordinates of the b-modeller.

In the context of EM, the broadcast mode of interaction implies that the authentic observables, dependencies or agents are pushed from the b-modeller so that the r-modellers are not allowed to maintain private values for what they observed. In contrast to the broadcast mode associated with the DEM framework[84], the broadcast mode of interaction describes here is a mono-directional broadcast.

One example of the use of broadcast mode of interaction is in the classroom context, where the teacher (i.e. acting in the role of a b-modeller) demonstrates a pre-constructed model or constructs a model on the fly to the students (i.e. acting in the role of r-modellers), whether with or without feedback from the students (outside the modelling environment). One of the EM models, Virtual Electronic Laboratory (VEL)(EMPA: velShethDOrnellas1998), has exploited the broadcast mode of interaction, which will be discussed in detail in section 6.1. It is worth noting that modelling in the broadcast mode alone may not be considered to be "collaborative modelling" because the model in the public space, arguably, is merely the effort of a single modeller (i.e. the b-modeller).

[84] The broadcast mode in DEM is multi-directional and it allows individual modellers to have a private set of observables, dependencies and agents (Sun 1999; Beynon et al. 2003).

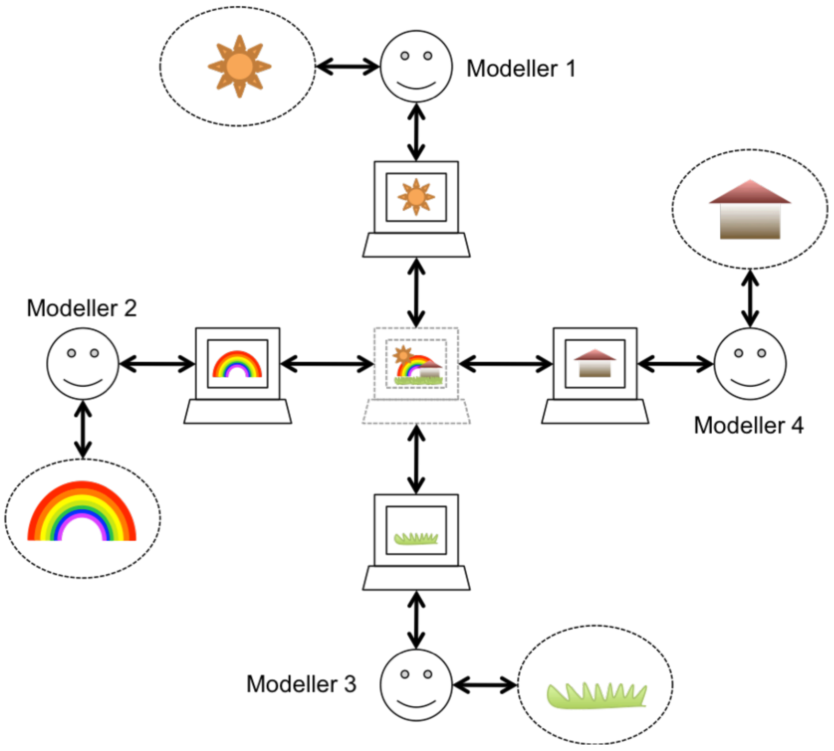


Figure 5.6 – Blackboard mode of interaction

Blackboard

The use of the term “blackboard” is similar to that described by Iain Craig (Craig, 1995) in his book *“Blackboard systems”*. The blackboard metaphor came from Newell (1962) in the context of group problem-solving activity (Craig, 1993). In the blackboard mode of interaction, modellers are interacting within the one and only one shared modelling space. Each modeller may freely contribute parts to the shared model based on their knowledge and personal experiences. As shown in figure 5.6, the shared model is composed from the parts from the individual modellers. Although individual modellers may only focus on their own parts on their workstations, the individual contributions are instantly integrated into the shared spaces. This resembles the situation where a group of students gather around a blackboard and sketch ideas, compose parts, or discuss the results on the blackboard at real-time.

In contrast to the broadcast mode, the blackboard mode can be thought of as multi-directional broadcast since it allows more than one contributor to the shared model. In contrast to the privileged mode (which will be described below), there is no “authentic values" or access control to the model, i.e. everyone can add, delete, or change parts that are currently in the shared model. In EM, modelling in blackboard mode means that the modelling space will only consist of public observables, dependencies and agents – there will be no "private" observables, dependencies and agents in the modelling space. The blackboard mode of interaction plays a crucial role in many collaborative modelling situations, as it allows modellers to interact freely. To show how EM may potentially facilitate this mode of interaction in collaborative modelling a case study on the collaborative jugs model will be discussed later in this thesis (cf. §6.3).

Interference

As in the broadcast mode of interaction, modellers in the interference mode are divided into two groups, namely, the senior modeller (denote as s-modeller thereafter) and the ordinary modeller (denote as o-modellers thereafter). The naming of the modellers reflects a status difference within the group of modellers who carry out the collaborative modelling. As shown in figure 5.7, the communication between o-modellers will be screened and filtered by the s-modeller, who has a higher status or authority in the group. In contrast to the blackboard mode, the collective model may not consist of all the parts from individual models. The advantage of modelling in this mode is that it avoids incompatibility between contributions between o-modellers. The integration of individual partial models can be negotiated among the s-modeller and o-members of the group (i.e. democratic strategy), or through the existing organisation structure of the group (i.e. dictatorial strategy). However, not all modellers may be satisfied with this integration arrangement. For instance, not every component of modeller 3's model is integrated into the shared model (cf. figure 5.7). Another disadvantage with the interference mode is that it requires full-time attention by the s-modeller. This implies that asynchronised collaboration is impossible.

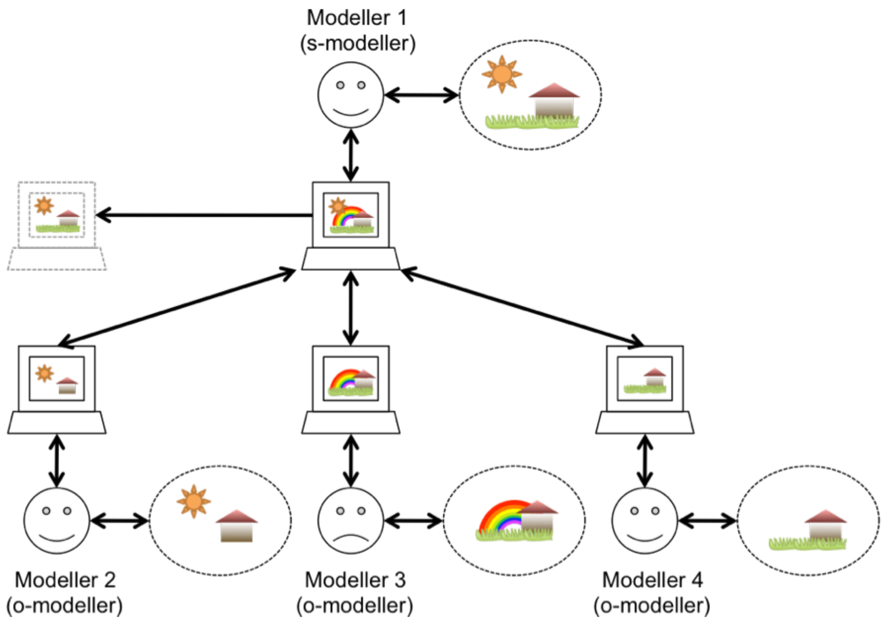


Figure 5.7 – Interference mode of interaction

In EM, modelling in the interference mode of interaction is associated with the situation that the collective model is composed through making explicit references to individual observables, dependencies and agents in the individual models, an intervention is carried out in the context of observation of the s-modeller.

One example of modelling in the interference mode is in the context of concurrent design (cf. Adzhiev et al., 1994a; 1994b), in which a single coherent account in the shared model is to be constructed from the diverse perspectives of modellers that require coordination and intervention from a more senior modeller. Such a mode of interaction may also be useful in relation to the creation of innovative artifacts, where such multi-faceted exploration between o-modellers is vital to improve the mutual understanding among the s-modeller and the o-modellers (cf. Sun, 1999; Sonnenwald, 1996).

Privileged

The privileged mode of interaction can be thought as a combination of the blackboard mode and the interference mode as described above. On the one hand, individual modellers (i.e. o-modellers) contribute their parts to the shared model as in the blackboard mode. On the other hand, o-modellers may carry out exploratory modelling in their private workspaces and are coordinated by the s-modeller as in the interference mode. In contrast to the blackboard mode, modelling in privileged mode allows co-existence of diverse perspectives over the same area of interest, and the modellers may possess “ownership” of certain components in the shared model. For this reason, access to the parts in the shared model from individual modellers is not unrestricted, but based on their privileges. In contrast to the interference mode, the privileged mode of interaction makes fewer demands on the attention of the s-modeller. This is because the coordination is handled by the agreed protocol – i.e. the privileges of the modellers. This also implies that asynchronised collaboration is possible in the privileged mode.

To some extent, the privileged mode of interaction resembles everyday interaction in the objective world. For instance, in terms of EM, modellers may have different perceptions of an observable *isLightOn*. One modeller observes that the light is on because she is in the same room as the light is in, and, for the same reason, this modeller is also privileged to be able to switch the light on or off. The other modeller, in contrast, believes that the light is off because she cannot see any lights coming out from the room while observing from behind a dark curtain.

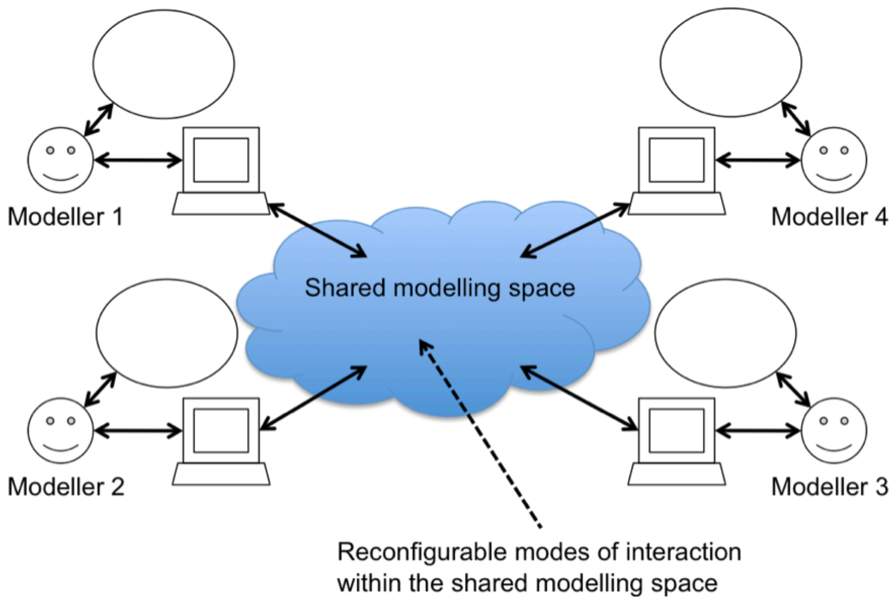

Modeller 1
Modeller 4
Shared modelling space
Modeller 2
Modeller 3
Reconfigurable modes of interaction within the shared modelling space

Figure 5.8 – Supporting diverse modes of interaction in collaborative modelling

5.1.4 The dynamic nature of collaboration

In the previous sections (§5.1.1 to §5.1.3), I have discussed different aspects of collaborative modelling. These include i) different degrees of engagement, ii) different relationships between modellers and agents, and iii) different modes of interaction. Whilst it may be possible to identify specific ways in which the modelling activity is configured with respect to (i), (ii), and (iii) over a short period of time, it may be a lot harder to characterise collaborative modelling that takes place over a prolonged period. For instance, in developing a 'jugs' model (cf. §4.1.1), the modellers could initially coordinate their individual observations, but later engage in collaboration when they counter overlapping interests amongst their observations. This could further develop into more intimate engagement which can be regarded as co-construction. Alternatively, the modellers could become less engaged if they employed some form of division of labour (i.e. might fall back into coordination activity). The relationships between modellers and agents might evolve in a similar fashion (e.g. from multiple SMSA situations initially to become a mix of MMMA and multiple MMSA situations) as might the mode of interaction (e.g. from interaction in broadcast mode, to peer-to-peer mode, then blackboard mode, and so forth). This is due to the fact that people interact in diverse ways for different needs. Furthermore, in a broader social context, people may engage in multiple collaborative modelling with various social groups. This dynamic, situated, and complex nature of human social nature of collaborative modelling motivates us to view the relationship among modellers in the collaborative process as a *cloud*[85]. Figure 5.8 illustrates how the "cloud" might be placed among the modellers. On the one hand, the cloud metaphor captures the complexity and difficulty in properly accounting for collaborative modelling moment by moment. On the other hand, the cloud metaphor recognises the need for reconfigurable support to address different aspects of collaborative modelling, which include different degrees of engagement, different relationships between modellers and agents, and different modes of interaction.

[85] The cloud metaphor is borrowed from cloud computing (Weiss 2007), which emerged in the IT industry recently.

5.2 Why is Distributed Empirical Modelling not good enough?

Previous sections explained what is meant by *collaborative modelling* in this thesis and how EM might be practised in such context. In this section, I argue that the previous EM conceptual frameworks for distributed modelling and the tool support derived from these frameworks might not be good enough to support the interaction in *collaborative modelling*.

5.2.1 Historical development in Empirical Modelling for collaboration

In EM, the term collaborative modelling first appeared in the COMICAL project[86], in which the term was used to refer to collaborative EM with the distributed EM framework and tool. Previous research into EM (e.g. Adzhiev et al., 1994a, Sun, 1999, Wong, 2003) shows the potential to apply EM principles in collaborative contexts. For instance, Adzhiev et. al. (1994a; 1994b) explained how EM might potentially be practised in the context of concurrent engineering. In Adzhiev et al.'s framework, the complex conceptual design of an engineering product is repeatedly decomposed into smaller pieces, along the hierarchical organization structure, for individual modellers to deal with. Each modeller is responsible for the coordination and composition of the modelling of their subordinate modellers. Beynon et. al. (Adzhiev et al., 1994b) argues that the roles of the design agents (i.e. the modelling contexts of the modellers) have to be arranged in a hierarchical manner due to the need for resolving conflicts in the concurrent design process. Hence, EM is recursively practised at an individual level and coordinated at the collective level throughout the organizational hierarchy in relation to the concurrent engineering design process.

The lack of networked distributed modelling support for developing multi-agent systems (cf.

[86] COMICAL stands for "Cognitive Observation-Oriented Modelling for Interactive Computer-aided Learning". It was a project which set out to examine the potential of the Empirical Modelling approach to Computer-Assisted Learning (CAL) in UK schools in 1999. The project was funded by Warwick's Research Teaching Initiative (RTI). It included a workshop that was hosted in the Department of Computer Science at the University of Warwick in the summer 1999, and the participants of that workshop were mainly school teachers in the UK.

Beynon and Russ, 1994; Beynon et al., 1990) became one of the main research motivations for the development of Distributed Empirical Modelling (DEM). Sun (Sun, 1999) argues that research into EM has been mainly focused on modelling activity that is centred around "*an external observer who can examine the system behaviour*, but [such modelling activity] has to identify the component agents and infer or construct profiles for their interaction [prior to the modelling activity being carried out]" (ibid, p.77, italic in original), or what he referred as *S1-modelling*. Despite the fact that such a *S1-modelling* strategy is still useful in the context of *collaborative modelling*, Sun (1999) argues that a more pertinent modelling strategy is to enable the modellers to observe from the perspectives of the component agents, i.e. observing from inside, or what he referred as *S2-modelling*.

To formulate a framework based on EM principles better suited to a distributed modelling environment, Sun further derives two distributed modelling strategies, *E-modelling* and *I-modelling*, that combine S1-modelling and S2-modelling. Sun argues that such a combination of strategies is potentially more useful in the context of distributed modelling in general, and software systems development in particular.

E-modelling and I-modelling, in fact, are quite similar – both maintain a hierarchical structure among modellers, where subordinate modellers are led by a single supervising modeller in the hierarchy[87]. The only differences between E-modelling and I-modelling are that:

1. In E-modelling, all modellers are observing externally, whilst in I-modelling, only the supervising modeller is observing externally; all subordinate modellers are observing internally.
2. In I-modelling, subordinate modellers are not only observing internally, but also able to "act as the agent from the perspective of an *actor*" (Sun, 1999, p.91, italic in original), performing what Sun referred to as *pretend play*.

According to Sun (1999), the DEM framework is based on ethnomethodology (Garfinkel, 1967) and distribution cognition (Hutchins, 1995). One the one hand, Sun (1999) argues that

[87] In the DEM framework, Sun (1999) called the supervising modeller the *S-modeller* and the subordinate modellers the *A-modellers*.

the pretend play by the subordinate modellers in I-modelling resembles two important beliefs in ethnomethodology: “each agent is capable of managing the world and of ‘being-in-the-world’” (Sun, 1999, p.91; Garfinkel, 1967). The modelling, therefore, is based on reflexivity, in which the subordinate modellers account for the agents’ interactions through reflecting on the interactions that they carried out on behalf of the agents (Sun, 1999). On the other hand, Sun (1999) argues that I-modelling resembles an environment for the subordinate modellers to interact with each other and with the artifact (i.e. the model pieces within the modelling environment) in a fashion that corresponds to social organisations such as are described by Hutchin’s (Hutchins, 1995) distributed cognition theory.

5.2.2 Issues with the DEM framework and its tool support

Since the introduction of DEM, a number of domain-specific conceptual frameworks have been built on the DEM framework. For instance, the SPORE[88] framework for requirements engineering (Sun, 1999), the participative business process re-engineering framework (Chen, 2001) and the framework for software integration and virtual collaboration in the financial sector (Maad, 2002). While DEM may have proved its potential in distributed modelling, it may not be good enough to support diverse modes of interaction and seamless transitions between different degrees of collaboration. In the rest of this section, I argue that the DEM framework may potentially hinder collaborative modelling in two aspects, namely, interaction and collaboration.

Interaction aspects

Flexible structure

To some extent, the two distributed modelling strategies that Sun (1999) described, i.e. E-modelling and I-modelling, are quite similar to the MMMA scenario described in section 5.1.2. Whether practising EM in E-modelling or I-modelling, there is always a supervising modeller to coordinate the contexts for observations and modelling of the subordinate modellers. This poses a structural constraint in the DEM framework which may limit its

[88] SPORE stands for “Situated Process of Requirement Engineering”.

potential use in wider collaborative work. Sun (1999) argued that this structural constraint makes it possible to synthesise Gruber and Sehl’s shadow-box experiment, where the supervising modeller acts as the “experimenter” and subordinate modellers act as “observers”. In this way, the supervising modeller can maintain full control of the knowledge construction process. For Sun (1999), the supervising modeller plays a crucial role in shaping the subordinate modellers’ knowledge of the situation in DEM, though the supervising modeller may be “transparent” to the subordinate modellers in such process. Despite the merits of this configuration, the DEM framework has paid little attention to the collaborative modelling that may take place outside formal structured social organizations. Such collaborations are usually dynamic and fluid, and may involve unstructured groups and ad-hoc collaborations.

Besides unstructured groups, the collaborating group may already have an existing structure, both formal and informal. It is not often a good idea to impose a particular structure on such a group as it may disturb the group dynamics and lead to the tool becoming redundant (cf. Grudin, 1994a). Because the tool support for the DEM (i.e. dtkeden) is a groupware, its adoption and adaptation is subject to socio-technical issues in groupware (cf. §2.2.2).

Furthermore, distributed cognition emphasizes the stability and the reproducibility of a socio-technical system rather than its emerging properties or changes (Kaptelinin and Nardi, 2006; Béguin and Clot, 2004). It is a theory that focuses on well-structured and organized groups, which “places the most emphasis on coordination” (Kaptelinin and Nardi, 2006, p.221). The adoption of distributed cognition theory in the DEM framework further proves that it was not developed for *collaborative modelling*, but merely for coordinated distributed modelling.

Flexible modes for observation

In the DEM framework, the external observation is assigned to the supervising modeller, and each subordinate modeller is assumed to enact the role of one particular agent at any given time. There is no provision for the subordinate modellers to construe through making observation from an external perspective in DEM. Sun (1999) argued that external observation alone is not suitable for DEM. Therefore, the DEM was developed in a way that does not allow the modellers to switch between external and internal observations.

In EM, agents and agency emerge and evolve during the course of the modelling process (cf. §4.1). Restricting a modeller to particular modes of observation and interaction with a particular agent restricts the modeller's ability to investigate the interactions between multiple agents and carry out the agentification process which is essential to EM (cf. §4.1; Wong, 2003). While this may be useful when refining the agents and agencies in the later stage of the modelling process, it potentially hinders the agentification process throughout the modelling process. From the collaborative modelling perspective, it is therefore inappropriate to frame the contexts for observation and for modelling through restricting the modellers to particular modes of observation.

Taking all these issues into consideration, I argue that the DEM framework is not appropriate for collaborative modelling where multiple modes of observation and seamless transition between these two modes of observations are essential.

Dynamic interaction and reconfigurable environment for interaction

In the DEM framework, modellers are forced to interact through the model. Sun (1999) argues that such configuration for distributed modelling "makes individual contexts combine with each other to create a social context corresponding to Hutchins's model of what actually happens in a human society" (ibid, p.102). Sun argues that only by interacting as being-participant-observer internal to the model and through the modelling environment will the modellers benefit from practising EM. In the other words, if the modellers have to interact and model in separate contexts, the advantages of enacting EM are largely reduced (Sun, 1999). After all, the conflict between agents will only surface through modelling in a common modelling space.

However, if modellers have to interact and construe in the same modelling space, DEM for collaborative modelling has to support diverse and reconfigurable modes of interaction, and seamless transitions between them. This is because human activities are situated (Suchman, 1987), and it is rather rare for them to stay in one particular form of interaction. For instance, in the Virtual Electronic Laboratory (D'Ornellas, 1998; Sheth, 1998), the students and the teachers may adopt various modes of interaction in different tasks throughout the laboratory. However, the experimentation in the laboratory is a continuous activity. It would be unreasonable to expect the students and the teachers to restart a new modelling space when they move from one mode of interaction to another. Since DEM inherits a star-like hierarchy (as mentioned earlier), it neither allows modellers to switch between different modes of interaction (cf. §5.1.3) nor allows modellers to reconfigure the interaction between the modellers and agents (cf. §5.1.2)

Collaboration aspects

Bridging the public and the private modelling spaces with a single shared modelling space

One of the issues with the DEM framework in the collaboration aspects is that DEM does not make an explicit distinction between the shared modelling space and the personal modelling space. Consequently, there is no support in the DEM tool (viz. dtkeden) for seamless transition between the shared modelling space and the personal modelling space.

One may argue that the roles of the supervising modeller and the subordinate modellers have already been fixed in the DEM framework (i.e. I-modelling). It is then impossible for a subordinate modeller to become confused because the supervising modeller has already fixed the contexts for observation by the time the subordinate modeller starts her modelling. Furthermore, the global context simply does not exist if all the subordinate modellers are acting as agents internal to the model, looking from inside the agents. However, from a collaborative modelling perspective, the DEM framework does pose a challenge – the subordinate modellers may not be aware that they are actually modelling in a shared space until there are mysterious clashes of their model. Even then, subordinate modellers may still think such clashes are interventions from the supervising modeller rather than overlapping or conflicting definition from other subordinate modellers. Indeed, the modelling experience can be very frustrating if the modeller cannot tell which space her modelling is going into. One way to mitigate such an issue is to bridge the public and the private modelling spaces with a single shared modelling space (cf. Greenberg and Roseman, 2003), such that all modellers, both supervising modeller and subordinate modellers are all modelling and interacting in the same modelling space, and the contributions (i.e. model pieces) from individual subordinate modeller are visible to all other modellers. To some extent, interacting within a single shared modelling space also stimulates individual modellers' "immediate experience" of the overall model.

Lack of support to awareness

In section 2.1, I mentioned five aspects of successful groupware, namely, awareness and coordination, articulation work, creativity, experimentation, knowledge sharing and knowledge construction. Since the DEM is the conceptual framework behind the distributed modelling tool – dtkeden – and the tool itself is a groupware, we expect the DEM framework to address these aspects at a conceptual level. In fact, the DEM does address most of these aspects, except awareness[89]. Due to the fact that DEM is based on the I-modelling strategy (Sun, 1999), information about other modellers' constructs (observables, dependencies, and agents) is often hidden away. Such information is only disclosed to the subordinate modeller at the discretion of the supervising modeller when they believe that information may be relevant to the subordinate modeller's modelling situation at hand. Even when the information is released, it is released as if the action is taken by the supervising modeller due to the star-like network architecture; the subordinate modellers have no way to detect who is responsible for the change. This low level of awareness of others' modelling work may lead to potentially conflicting or incompatible model pieces when subordinate modellers are interacting in the same modelling space but their contexts of observation are highly overlapping (e.g. the modelling is at the collaboration degree of engagement and in the MMMA scenario).

5.3 How EM might support collaborative modelling

In the previous section, I have argued that DEM was primarily developed for distributed modelling. When collaboration and interaction is at the centre of concern, rather than merely building an EM model in a distributed environment, DEM does not seem to be addressing these concerns very well. This may be in part because more recent research has given us a better understanding of collaboration. For instance, collaborative modelling is more complex

[89] This issue was discovered in a series of pilot experiments (unexpectedly) prior to the collaborative modelling case studies, as I will discuss in chapter 6.

than distributed modelling because the collaboration it involves may be both remote and collocated and both asynchronous and synchronous, depending on the situation at hand. All these considerations motivate the question: "How might EM potentially facilitate collaborative modelling, in relation to different styles of collaboration?"

In this thesis, *collaborative modelling* is viewed as the core activity for groupware development. While the link between collaborative modelling and groupware development will further be examined in chapter 7, it seems appropriate to devote this section to study those aforementioned questions in more detail. Precisely, this section investigates how EM may potentially be practised in the context of collaborative modelling.

As I discussed in section 5.1, there are three important aspects in collaborative modelling. One of these, the degree of engagement in collaboration is particularly interesting in the context of this thesis, as it offers a holistic view of the overall *configuration* of the modelling activity that takes account of the intimacy of the relationships amongst the modellers, the activity, the context, and the artifacts of the collaborative modelling process. Consequently, the research question stated above can be refined as "How might EM modellers coordinate, collaborate, cooperate, co-construct, co-evolve throughout the EM process?" and in particular, "How might EM modellers develop inter-subjective *construals* collaboratively through the model building process?" Therefore, it seems appropriate to examine the potential of EM at different degrees of engagement in the context of collaborative modelling.

5.3.1 Coordination

As mentioned in section 5.1.1, when collaborative modelling is at the lowest degree of engagement – coordination – there may be only sporadic interactions between modellers, and plausibly, most of these are carried out asynchronously. When practising an EM approach, this implies that the modellers are mostly working alone in their private workspace, either in the SMSA or SMMA situation (cf. §5.1.2), and the coordination of the individual model-building activities may be done through tools that are external to the modelling environment. For instance, modellers can coordinate asynchronously via exchanging definitive scripts through tools external to the modelling environment, e.g. email or script sharing on a file server. When synchronous interaction is needed, modellers can interact in the various modes discussed in section 5.1.3. However, due to the sporadic need to interact in coordination, modellers may prefer a peer-to-peer (ad-hoc) mode of interaction rather than the more sophisticated ones, e.g. interference or privileged.

5.3.2 Collaboration

When the degree of engagement in collaborative modelling is at collaboration (cf. §5.1.1), members of the team are bound together in a shared activity with shared resources. However, they may not have an explicit common goal for the modelling process. In this case, modellers may have to explore the same phenomena in the context of observation in different directions or from different perspectives. Therefore, the key concerns of such collaborative modelling activity are to facilitate diverse perspectives, to explicate different accounts, and eventually to come to negotiate a common unified perspective.

As mentioned in chapter 4, EM is an observational-based approach to modelling. Observation in EM is based on the modeller's subjective and personal experience of the situation being observed (cf. §4.1), and Hanson, a philosopher of science, has also argued that the observational process is personal and situated (cf. §4.2). Therefore, EM artifacts that are construed by individual modellers based on the same context of observation offer diverse perspectives. This satisfies the first concern – facilitating diverse perspectives. For the second concern – explicating different accounts: the observables, dependencies, and agents in an EM model offer a partial account of the situation being modelled and the modelling environment, tkeden, can be used to animate this partial account in different ways to reflect the experience of different modellers. For the last concern, modellers can negotiate for a common unified perspective though redefinitions in a shared modelling environment synchronously, e.g. in the blackboard mode of interaction (cf. §5.1.3). The purpose of these redefinitions can either be to add, modify, or replace "components" or "parts" in the EM model, and modellers can experience the differences between perspectives. All these potential styles of collaborative modelling will be demonstrated in a case study of cricket simulation in the chapter 6.

5.3.3 Cooperation

To some extent, cooperation can be thought of as "collaboration in a larger scale" in contrast to the lower degree of engagement of collaboration in the context of collaborative modelling. At this level of collaboration, people pursue a pre-defined shared goal – i.e. an explicit common goal in contrast to collaboration. For the purpose of productivity of the overall modelling activity, the larger modelling context is often broken down into smaller but interdependent tasks for individual modellers to tackle, i.e. division of labour. Meanwhile, individuals have to maintain awareness of other people's modelling to avoid incompatibilities with others' modelling and to improve the productivity of their own modelling. Therefore, the main concern in cooperation can be thought of as making the model-building activity easier through effective partitioning of the larger modelling context and integration of smaller individual pieces.

Although EM does not promote modelling cooperatively with a pre-defined goal, there is no reason why EM cannot be practised in such context. Sun (1999) suggested various strategies in which individual modellers can model cooperatively, where each individual modellers is assigned to a smaller context for observation. In *E-modelling*, each subordinate modeller act as "an external observer who observes the system as if from the perspective of a particular agent and can enact EM individually" (Sun, 1999, p87-88). In *I-modelling*, each subordinate modeller observes the system as an internal observer who plays the role of a particular agent, which Sun referred as *pretend play*. For Sun, both of these strategies enable a senior modeller to coordinate the subordinate modellers, both in E-modelling and I-modelling, in relation to the contexts of observation and to the scope of modelling. However, Sun (1999) argues that E-modelling is not suitable for the purpose of DEM, as the potential for *enacting EM* is diminished when subordinate modellers are forced to carry out EM in a context away from the one in which they normally interact with each other. However, in the context of this thesis, modelling through external observation is still considered within the scope of collaborative modelling, whilst both E-modelling and I-modelling are considered as particular configurations of the MMMA scenario in collaborative modelling (cf. §5.1.2). In section 6.1, modelling at the cooperative degree of engagement will be illustrated in the case study of the Virtual Electronic Laboratory (VEL) project (D'Ornellas, 1998; Sheth, 1998).

5.3.4 Co-construction

When the degree of engagement in collaborative modelling comes to co-construction, the modellers are not only concerned with what to do and how to do it, but also with communicating why they are doing it. Modelling in co-construction is not only concerned with the artifact, i.e. the model, but also with the shared understanding of the shared activity, the shared resources, and the artifacts that are co-constructed throughout the model-building activity. As the shared understanding emerges, there might be adjustments to the shared goal of the modelling.

Gruber and Sehl's shadow-box experiment reveals that shared understanding among modellers of the situation being observed can be obtained through exchanging tentative constructs (or construals) of their personal experience (Sun, 1999; Gooding, 1990). EM is concerned with the modellers' immediate experience throughout the modelling process. As I discussed in chapter 4, EM artifacts are cognitive artifacts (cf. §4.1.2 and §4.1.3). What is constructed in the modelling process is not merely the artifact (i.e. the EM model), but also the understanding of the situation. For this reason, EM advocates continuous reinterpretation of the situation being observed through continuous interaction with the EM model and continuous observation of the situation. These characteristics of EM are well-matched with the need to facilitate the changing perspectives in co-construction, and potentially ease the establishment of shared understanding through interacting with a shared EM model within the same modelling space.

Due to the diversity of interaction in co-construction, collaborative modelling at this level may be carried out through different modes of interaction throughout the modelling process. For instance, modellers may interact in the blackboard mode at one point in order to maintain visibility of their interactions to other modellers and interact in the peer-to-peer mode or privileged mode at other time for other types of investigations. In respect of the relationship between the modeller and the agent (cf. §5.1.2), co-construction involves modelling in both MMSA and MMMA scenarios, where modellers switch between these two scenarios seamlessly as their focus of attention shifts.

5.3.5 Co-evolution

At the highest degree of engagement in collaborative modelling, both the individuals' and the shared, understanding of the situation for modelling and of the artifact will co-evolve through the collective model-building activity. Meanwhile, the artifact itself will also evolve. Therefore, this co-evolution is multi-way. Consequently, the focus of support is on easing this co-evolution. One may argue that this co-evolution will happen anyway, regardless of the tools and the approach we are using in collaborative modelling due to our ability to adapt to less-than perfect situations. However, there are at least two objections to this idea:

i. The first objection is that the referent for modelling and the context of observation may evolve on some occasions. In fact, this is exactly the case for groupware development, where the work that is supported by the groupware is ever changing throughout the development process. That is to say, in the context of groupware development, the referent for modelling and the context of observation for collaborative modelling is a "moving target".

The second objection is that the co-evolution between individual understanding, individual artifact, and individual activity does not necessarily trigger coherent co-evolution in the collective context. In this cases, the co-evolutionary support of the tool and the approach arguably becomes crucial to effective collaborative modelling. And whilst it may be hard to dismiss the argument against the need to support co-evolution in collaborative modelling completely, there are reasons for supposing that adopting an EM approach can make a radical difference, and in due course provide a pragmatic justification. In any event, as with the co-evolutionary perspective in technology development (cf. §2.2.2; and also Ackerman, 2000), it is our choice whether to round the edges for greater convenience and productivity or to tolerate the discrepancy of the tools, methodology, etc for collaborative modelling.

As mentioned in chapter 4, EM has flexible and open characteristics in relation to model building (cf. §4.1.2 and §4.3.5). Modelling in definitive scripts enables the modeller to explore different possible configurations of observables, dependencies, and agents to represent the *referent*. From a holistic perspective, the modellers' understanding and the individual models co-evolve as the "continuous reinterpretation" takes place (as mentioned earlier in §5.2.4; also cf. §4.1). The intimate interplay between individual modellers' understanding and their individual model (or parts of the collective model) is similar to that suggested by Naur (1985a; also cf. §3.1). This eventually eases the overall co-evolution through the diverse interaction among the modellers through and around the artifact during the collaboration process. Indeed, EM does not presuppose a static context for observation nor a predefined goal in the modelling process (cf. §4.1.2).

To bridge the co-evolution at the individual level and the collective level, Sun (1999) proposed a framework (i.e. DEM) which enables collective co-evolution driven by individual co-evolution through what he referred as *evolutionary interaction* – modellers evolve the agents inside the model through playing the role of the agents and interact with other agents act as if they are the agents themselves inside the model. Apart from such "co-evolution from the inside", it is also possible to evolve in the scenario of MMMA through external observers or a mix of internal and external observers in a shared modelling space, in which modellers can evolve the agents from both inside and outside the agents. This eventually leads to co-evolution of modellers' understanding of the situation for modelling and of the artifact, both individually and collectively.

Chapter 6
Case studies: practising Empirical Modelling in collaborative contexts

In the previous chapters, I have shown that Empirical Modelling (EM), based on a different philosophical foundation from mainstream computing, potentially facilitates collaborative modelling particularly in respect of the three highest degrees of engagement (i.e. collaboration, co-evolution, and co-construction). I have also shown that the EM conceptual frameworks and the associated tool support that were conceived for concurrent engineering and distributed organizational work may not be expressive enough for the diverse and typically ill-structured activities that arise throughout the collaborative modelling process, e.g. groupware development. In these cases, it is preferable for the collaborative environment to allow modellers to interact using diverse modes of communication and to shift their roles seamlessly. While a full-fledged ideal collaborative environment is still under development, we still see benefits in studying the efficacy of EM for collaborative modelling in practice[90].

In this chapter, I discuss two case studies in software development projects where the core activity can be viewed as a collaborative modelling process throughout the development lifecycle, and two case studies in collaborative modelling that exploit an enhanced variant of the distributed EM environment developed by the author. The case studies in software development are designed to study collaborative modelling at the project level when practising an EM approach, while the case studies that exploit the enhanced variant of the collaborative EM environment are designed to study collaborative modelling at the

[90] Despite the imperfections in the tool support, EM may offer a better theoretical foundation for collaborative modelling.

interaction level.

In section 6.1, I present a case study on the Virtual Electronic Laboratory (VEL) project. This was a joint project between two MSc students (D'Ornellas, 1998; Sheth, 1998). Although the common goal for the students was to develop software to teach elementary electronics in a classroom environment, they had different perspectives in developing the VEL model. D'Ornellas focused on the simulation of electronics and showed the openness of the EM approach with an illustrative example in integrating the traffic light model (EMPA: trafficlightMendis1997) with the electronic simulation module of the VEL model. Sheth focused on a complementary objective: developing a multi virtual-agent interface and exemplifying the four interaction styles associated with the DEM framework in a classroom-teaching context. Unlike previous research by Maad (2002), this case study focuses on the co-construction of the VEL model and the discussion of the particular scenario adopted in relation to other possible scenarios that can be applied in a collaborative modelling approach.

In section 6.2, I discuss a case study on two collaborative modelling sessions of a distributed Jugs (d-Jugs) model, which is similar to the Jugs Model (EMPA: jugsBeynon1993) in the EM project archive. The purpose of this case study is to explore the interactions among a small group of modellers in ad-hoc synchronous collaboration[91] when practising an EM approach to collaborative modelling. This case study consists of two d-Jugs modelling sessions. Multiple data collection techniques were used in both modelling sessions. These techniques included participant observation (Spradley, 1980), video recording, screen video recording, and interaction history logging. The planning within the modelling process was kept to the minimum – neither of the modellers was briefed before the modelling sessions and the development time was very limited. This configuration gave rise to role-shifting behaviour of the modellers and revealed the dialogical character of the interactions involving script exchange to the observer of the modelling sessions.

In section 6.3, I discuss a case study of a collaborative modelling session of a Sudoku

[91] I also call ad-hoc synchronous collaboration "real-time" collaboration.

model. As in the case study of d-Jugs modelling, multiple data collection techniques were used in the case study of the collaborative Sudoku modelling. These techniques included participant observation (Spradley, 1980), video recording, and interaction history logging. The room configuration for the collaborative modelling session was similar to that in (Flor and Hutchins, 1992), where the modellers were sitting side-by-side and working on a shared model through separated but networked workstations. During the collaborative modelling process, the modellers were asked to develop a strategy to tackle a Sudoku puzzle collaboratively. They were encouraged to customise their collaborative environment when necessary. Despite the fact that the modellers were 'jump-started' with an existing model (EMPA: sudokuKing2005), they spent much of their time learning about the problem through interacting with the Sudoku model during the collaborative modelling session, due to their unfamiliarity with the problem context. Coincidentally, this made the modelling process similar to the early stage of traditional systems development, where developers are spending much of their time in learning the problem domain. This case study has exposed the diversity of human interaction and the efficacy of instant feedback[92] in synchronous collaborative modelling when using an EM approach.

In section 6.4, I study an undergraduate software development project on cricket simulation (Beynon, 1993). During the early stage of the software development process, the students were asked to practise an EM approach instead of a traditional approach to requirement elicitation, analysis and design. The aim of this case study is to reveal the potential benefits that EM may offer to collaborative modelling – where software development can be conceived as a specialization of collaborative modelling.

[92] Instant feedback is a feature provided by the principal EM tool (tkeden) and its variants. This includes state changes due to triggers, dependencies, and propagation of script changes from peers.

	Case Studies of Software Development Projects		Case studies that exploited the enhanced variant of the collaborative EM environment	
	Cricket (§6.4)	VEL (§6.1)	d-Jugs (§6.2)	Sudoku (§6.3)
Intended focus of the case study				
Project level	x	x		
Interaction level		x	x	x
Structure of the collaborative modelling process				
Structured		x		
Semi-structured	x		x	
Unstructured				x
Length of the collaborative modelling process				
Between 1 to 3 hours			x	x
Between 2 to 6 months	x	x		
Temporal character of the collaborative modelling process				
Asynchronous collaborations	x	x		
Synchronous collaborations			x	x
Spatial character of the collaborative modelling process				
Remote collaborations[93]	x	x		
Collocated collaborations			x	x

Table 6.1 – Considerations that the case studies have addressed

As table 6.1 depicts, the selection of case studies covered a range of frequently occurring situations and configurations that may be involved in collaborative modelling (cf. §5.1). The variety of factors considered in this selection include:

i) *Group size* – a range of number of collaborators from a pair of modellers to a medium size software development team with about a dozen developers.

ii) *Length of the collaboration* – a variation of the length of the collaborative modelling process ranging from a few hours to a few months.

[93] In both the cricket project and the VEL project, there is no 'firm' evidence to indicate that either remote or collocated collaboration was the only kind of collaboration. However, despite the fact that the modellers might have collaborated while collocated, I would argue that their collaboration was primarily remote. This is because:

1. At the time when the cricket project was carried out, there was no distributed (and collaborative) EM tool support available. Certainly, it is possible that the modellers discussed their models around a shared screen. Indeed, there were face-to-face project meetings, and collocated interaction between the modellers outside the cricket project was inevitable, as they were all enrolled in computing-related degree programmes at the same University. However, it is more likely that modellers were primarily working alone due to the nature of the coursework (despite the fact that the cricket project was a group project, one of their preliminary tasks was to submit an individual cricket model, cf. §6.4).

2. In the VEL project, the division of labour between the modeller pair was so clean that the modellers could work separately in isolated modelling spaces until late in the collaboration process, when integration of their individual models took place. The project work was planned in such a way that there was no strong need for collocated collaborations.

iii) *Structure of the collaboration* – different levels of structure in respect of the common goal.

iv) *Temporal character* – synchronous and asynchronous collaborations.

v) *Spatial character* – collocated and remote collaborations.

The motivation behind these case studies is:

i) To examine the effectiveness of the instant feedback provided by EM tools which potentially facilitates collaboration;

ii) To show that the interaction between modellers is similar to a dialogue, and has to be understood with reference to how their communication is enabled by their prior experience[94], as they negotiate the meaning of the evolving artifact through exchanging scripts;

iii) To explore the role-shifting behaviour of the modellers in synchronous collaborative modelling and to examine whether the conflation of the context of construction and use facilitates seamless transition of roles.

iv) To reveal the diverse interaction styles during the modelling process (in synchronous, semi-synchronous, and asynchronous collaborations) which stem from the dynamic nature of group work;

v) To assess the significance of tool support[95] in collaborative modelling and in particular whether or not we need tool support. If yes, what is needed in the tool support, and how can that be supported?

Through these case studies, we understand how EM principles may be applied in practice. Indeed, these case studies provide evidence of the potential of an EM approach to collaborative modelling. In the case of software development, the core activity can be viewed as a co-construction of an Interaction Situation Model (ISM). The idea of practising

[94] This includes domain knowledge, and both past and immediate experience in relation to the context of observation.

[95] The case studies reflect a paradigm shift in EM research, from traditional group work to interactive group work.

an EM approach to collaborative modelling has influenced the thesis author to develop a conceptual framework for participatory groupware development (see Chapter 7).

6.1 Case study I: the Virtual Electronic Laboratory project

There is evidence suggesting that the development and the use contexts are conflated when practising EM in a distributed systems development context. For instance, modellers in the Virtual Electronic Laboratory (VEL) project act as both the developer and the users (the role of teacher and student) at different times in the EM process. Previous accounts of the VEL project focused on its pedagogic potential in the context of collaborative learning (e.g. Beynon and Maad, 2002, Maad, 2002). In this section, I shed light on the VEL project from a collaboration perspective. In particular, this case study focuses on the interaction between the modellers.

6.1.1 The background of the case study

The Virtual Electronic Laboratory (VEL) is a distributed EM model that was collaboratively constructed by two MSc students, D'Ornellas (1998) and Sheth (1998), in a joint project. The motivation of this project was to examine and demonstrate the potential of EM in supporting learning in a distributed environment, in particular, supporting the learning of elementary electronics through experimentation in a virtual laboratory environment (D'Ornellas, 1998; Sheth, 1998). The development of the VEL was inspired by several typical real-life electronics laboratory scenarios[96] (Sheth (1998)). The project, in its very essence, was

[96] Sheth (1998) described four scenarios that are commonly found in real life electronic laboratory: (i) After the teacher and the laboratory assistant set up the equipment, they continue to set up the chosen circuit, with all the components on a circuit board. The students crowd around the teacher's table whilst the teacher first describes the circuit and its components. Then, the students are expected to observe and make notes while the teacher is changing the values of the components during the experimentation. In this scenario, the teacher is in control of experiment. (ii) Students are experimenting either in groups or individually. The students may either be given a pre-set circuit board with all the components on it, or be asked to build a given circuit on their own. In this case, the teacher does not have an overall control of the laboratory session: the teacher may walk around and assist individuals or groups one at a time. (iii) Similar to (i), but the teacher carries out the experiment with a computer tool instead. The advantage of computer simulation experiment is that the teacher can demonstrate drastic effects to the students, such as blowing the resistors up with an extreme voltage, which would normally be regarded as dangerous or wasteful when experimenting with real electronic components. (iv) Each student experiments with a computer tool, e.g. PSpice, on a private workstation. The students will be given the experimenting circuit and they

educational.

However, the virtue of the VEL project in the context of this thesis is not derived from its connection with computer-based learning, but its connection with collaborative modelling. At the time the study was carried out, the VEL was one of very few EM models in the EM archive (EMPA) that had fully exploited all four modes of interaction[97] that are closely associated with dtkeden, the tool for DEM:

i) In the broadcast mode, all the changes done at the teacher's workstation (i.e. s-modeller in DEM) within the modelling environment will be relayed to the students' workstations (i.e. a-modeller in DEM). The relayed components include the circuit and the model itself. The teacher and the laboratory assistant can prepare a circuit while the students are entering the laboratory. The teacher then explains the circuit and carries out the experiment in the same mode. Students watch the teacher's demonstration through their own workstations and make sense of it.

ii) After setting up the circuit in the broadcast mode, the teacher can choose to switch to the private mode. In this case, the teacher is letting the students carry out the experiment on their own.

iii) In the privileged mode, the teacher can assign access privileges on the components in the model to individual students or groups such that they can build or experiment a circuit cooperatively.

are responsible for 'drawing' the circuit within on their own workstation. Then, the students can start experimenting with the circuit, make observations, and learn from it. Similar to scenario (ii), the teacher may walk around the classroom and assist those who are struggling.

[97] It is worth noting that the four modes of interaction that are closely associated with the dtkeden and DEM are not exactly the same as those I described in chapter 5. For more details, please refer to section 5.1.3.

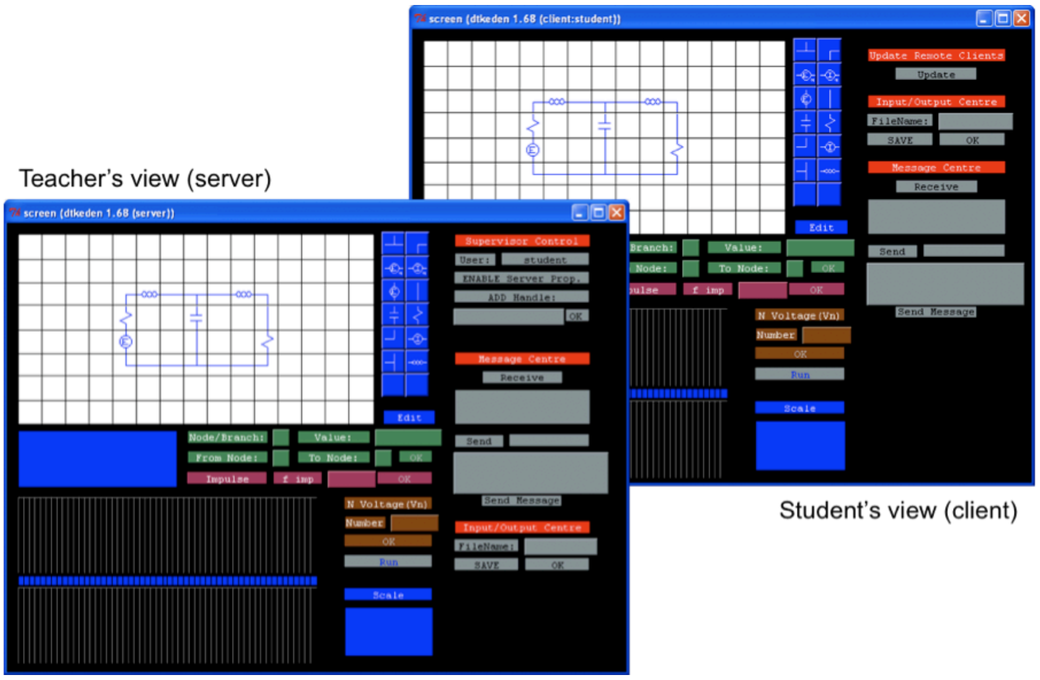


Figure 6.2 – A screen capture of the VEL model (EMPA: velShethDOrnellas1998) within the dtkeden modelling environment

iv) In the interference mode, the teacher selectively accepts redefinitions of the circuit components that are submitted by students. This can be used in, e.g. group discussion, where the group representatives (when students are working in groups) or individuals (when students are working alone) negotiate their contribution for integration into the public model.

As I will discuss later in section 6.1.5, the virtue of the VEL project is its use of the DEM framework for collaborative modelling. This in turn becomes a significant piece of evidence of the potential of EM for collaborative modelling at the cooperation degree of engagement.

Figure 6.2 shows a screen capture of the VEL model. Further discussion of the technical detail of the VEL model is beyond the scope of this study. For such details, the interested reader can refer to the MSc project reports written by D'Ornellas (1998) and Sheth (1998).

6.1.2 Method of study and approach to data analysis

This case study aims to reconstruct a description of the cooperation and coordination between the developers of the model. For this reason, the discussion is more appropriately regarded as an interpretive account that gives particular attention to the collaborative aspects of the project. Due to the age of the project, studying the project is far from straightforward; not only was it impossible to contact the developers, but the incomplete interaction history in the definitive scripts also posed a challenge. Consequently, this case study is largely based on the dissertations written by the developers and the archived definitive scripts of the VEL model (EMPA: velShethDOrnellas1998). This also makes the data analysis within the VEL case study similar to document analysis as described in literature in research methods such as (Duffy, 2005) and (Yin, 2009). Duffy (2005) suggests that documents can be classified as *primary* or *secondary*, *deliberated* or *inadvertent* sources. Following Duffy's classification of documents, the VEL case study that is discussed here involves both primary and secondary, and both deliberated and inadvertent document sources:

i. Both the developers' dissertations and the archived definitive scripts are primary source, and the other case studies of VEL by other researchers (e.g. (Maad, 2002); (Beynon and Maad, 2002)) are secondary source.

ii. Developers' dissertations, on the one hand, can be classified as *deliberated sources* because the developers might have attempted to preserve some "facts" about the development process of the VEL project. Archived definitive scripts, on the other hand, can be classified *inadvertent sources* because they were produced as one of the 'deliverables' of the VEL project rather than as a preservation of the development process. [98]

However, the archived definitive scripts are more than merely texts. This is because they can be interpreted in the EM modelling environment (viz. tkeden or dtkeden). Due to the nature of EM (cf. chapter 4), the archived definitive scripts can, to some extent, also be used to recapture the *state-as-experienced* by the modellers in developing the VEL model.

Before approaching the data, I expected this case study to shed some light on, though not

[98] Duffy (2005) classifies primary document sources into two catagories, namely, *deliberated sources* and or *inadvertent sources*. He explains that *deliberated sources* "are produced for the attention of future researchers" and these sources "involve a deliberate attempt to preserve evidence for the future, possibly for purposes of self-vindication or reputation enhancement." (ibid, p.126) Duffy also explains that *inadvertent sources* "are used by the researcher for some purpose other than that for which they were originally intended." (ibid, p.126)

be entirely devoted to, some research questions that I had in mind. For instance:

i. How did the developers collaborate in relation to the aspects of collaborative modelling described in chapter 5, namely, the degree of coherence, the relationship between the modellers and the agents, and the modes of interaction?

ii. Have the developers followed the interaction styles proposed in the DEM framework?

iii. Does EM hinder or facilitate the development process?

iv. What characteristic(s) of EM, if any, has the project demonstrated?

Since no similar type of analysis of research into EM had been done before this case study, it was unclear how the developers' dissertations might reveal. Taking this into account, I adopted a flexible approach to the data analysis in the VEL case study: firstly, consult the developers' dissertations, then, based on their description of the collaborative modelling process, analyse the archived definition scripts.

In order to find out how the developers approached the VEL joint project, I analysed their dissertations and highlighted the pieces and sections that describe the process of development and the cooperation between them. The original plan was that the analysis of the developers' dissertations would give a basic idea of how the developers might have collaborated, and the challenges (in relation to collaboration) that they faced and overcame during the VEL project. This information would then be used to guide the analysis of the archived definition scripts. Unfortunately, the dissertations neither explicitly discuss the degree of collaboration involved in the joint project, nor give details about how the developers collaborate in the joint project. The developers' dissertations were focused on the educational aspect and technical aspect. Despite the fact that the project was a collaborative effort, there is little information about the collaboration process between the developers apart from the declaration sections, the acknowledgement sections, and the following observation:

> *"This would be a two-part project in corporation with another MSc colleague, Hansel D'Ornellas" (Sheth, 1998, p.5) "… this is a joint project and another objective is to allow the teacher to draw the circuit diagram and then extract the data and place it into the wanted matrices format for manipulation by my colleague. The extraction will be dealing with issues such as node labels, component values and so on. A part of the interface would be to display graphs about the circuit." (ibid, p.5)*

Furthermore, the developers' dissertations indicated that there was little or no overlapping between the their contributions in the VEL joint project:

> *"… my part of the project consisted of designing and implementing a user interface so that the teachers and students may collaborate with the tool. The output from the circuit analysis was to be done by my colleague. It was necessary to extract the data from the circuit model drawn, and to represent it in the required matrix format, ready for processing. This was achieved and my colleague was hence able to provide output from the matrices." (Sheth, 1998, p.52)*

To find out how they cooperated in more detail, that is, to determine how they divided the labour in respect of model construction, I have compared their contributions in the VEL project archive (EMPA: velShethDOrnellas1998). The analysis of the archived definitive scripts, to a large extent, involves a kind of comprehension that is similar to program comprehension (cf. e.g. Beynon and Sun 1998).

It is clear that two different styles of structuring the definitions and two different styles of file naming were used in the definitive script of the VEL model. For instance, one set of files were named a suffix of ".lib.e", while the other set of files were named without the use of any suffix. One set of scripts seems to be feature-oriented (i.e. one triggered action per script), and the other set seems to be agent-oriented (i.e. big scripts which consist of all relevant definitions of agents). This somehow echoes the claims in the developers' dissertations that the project was a cooperation between the two developers.

In order to know what division of labour was adopted by them, I have analysed the ‘ownership’ of the observables, dependencies, and agents in the scripts. In contrast to traditional computer programming, EM models offer an unusual resource for such analysis: dependency. As mentioned in chapter 4, dependency in EM is not the same as dependency injection or the concept of dependency in Enterprise Java Beans. Through tracing the dependency structure within the joint VEL model, it reveals how dependencies are used to link observables, agents, and triggered actions together – this provides us with an insight into how the developers might have collaborated. The inspiration behind this analysis is de Souza’s (2004) work, where he suggests that the social dependencies among the developers are by and large captured in the artifacts they produce.

6.1.3 Limitations

The VEL case study is largely based on the analysis of archived documents. The major weakness of such an approach is that bias can easily be introduced either by the researcher in document selection (Yin, 2009) or by the document authors who may not declare implicit assumptions properly (Bell, 2005; Yin, 2009).

To some extent, these limitations have been addressed in the research process. For the first issue, i.e. document selection, this case study has included all the accessible documents that discuss the VEL project (cf. §6.1.1). While best effort has been made to report and analysis the VEL project as fairly as possible, this does not prevent unknown bias in the documents that was introduced by the documents’ authors. For instance, the developers’ dissertations might have reported their intended cooperation – but not their actual cooperation – in the development process. To verify whether the degree of the collaborative activity in the development process of the VEL model was genuinely limited to what I have classified as cooperation with a high level division of labour, I have also consulted the students’ project supervisor, Meurig Beynon, informally. According to Beynon, it was likely that the developers had limited opportunity to work together. He explained that the developers were living in two different sites and were working on the project during vacation time. Furthermore, I have analysed the archived definitive scripts from the VEL project archive (EMPA: velShethDOrnellas1998). However, the analysis of the definitive scripts does not tell the story by itself – it requires careful and skillful interpretation which involves knowledge of EM and of the EM process. This, in turn, is subject to possible bias in the interpretation. However, I would argue that any competent EM practitioner might arrive at similar, if not the same, conclusions if they analysed the documents in the way described in this case study.

6.1.4 Discussion

Initial coordination: division of labour

Due to the differences in their expertise, D'Ornellas and Sheth were observing from different perspectives and modelling from different contexts. On the one hand, D'Ornellas focused on the simulation of electronics underlying the graphical user interface due to his strong background in electronic engineering. On the other hand, Sheth focused on the graphical user interface (GUI) for both the teacher and the students. Although the two modellers were working as a team and modelling in separated spaces, there is no evidence to suggest that there are overlapping areas in their models – i.e. the coordination between the two was effective, if not perfect. Therefore, the next question in this case study is “How did they actually coordinate and cooperate?”

How did they actually coordinate and cooperate?

In traditional software development, developers often coordinate through interface components (deSouza et al., 2004), either well-defined in specifications or through mutual agreement between developers. In searching for the way that D'Ornellas and Sheth coordinated, I attempted to find the similar interface component in the VEL model. On this matter, Sheth (1998) provided a crucial hint on this “component”:

> *“… this is a joint project and another objective is to allow the teacher to draw the circuit diagram and then extract the data and place it into the wanted matrices format for manipulation by my colleague.” (ibid, p.5)*

The analysis of the definitive scripts of the VEL model reveals that D'Ornellas and Sheth were engaging in coordination. This was done through a two-dimensional matrix and semaphoric observables. The analysis also suggests that the two developers were coordinating and cooperating, and not practising any higher degree of engagement (cf. §5.1.1) despite the fact that they were co-constructing the VEL model.

Unlike other programming paradigms (e.g. object-oriented programming), modelling with definitive scripts means that agents are connected together with dependencies (cf. §4.1). The EM model cooperatively constructed between D'Ornellas and Sheth also revealed how dependencies (also known as triggered actions in this case) were used to connect various key agents between the GUI mode (construed by Sheth) and the circuit simulator model (construed by D'Ornellas). The GUI model gathers the components' properties on the circuit board (cf. figure 6.3) and transforms them into a two-dimensional matrix ('A'). When matrix *A* is updated, the values of the semaphoric observables (*start_sim* and *ready_status*) will be updated by the *superAgent*. The update of *A* also triggers the computation of matrices *Yn* and *Jn*. After *Yn* and *Jn* are computed, this triggers the computation of *Vn*, and so on and so forth. After the last matrix is computed, the *dataCollect* agent is triggered, which updates the magnitude graph and the phase graph respectively. Figure 6.4 shows how the agents are linked together with triggered actions. The use of dependencies, in this cooperative context, not only simplified the definitive scripts a great deal[99], but also potentially made the cooperation easier through easing model comprehension – it allows the other developers to *see* the "structure of the model" through the dependencies[100]. In other words, the use of dependencies potentially allows modellers to "*see through walls*" to know what is happening at their peers (deSouza et al., 2004).

[99] This is because the definitive scripts do not contain an excessive number of listeners or other polling mechanisms that check for the latest data. This potential merit of dependency is also identified by Harfield (2008) in a case study in which he compared the modelling of Jugs using EM and using an object-oriented programming paradigm.

[100] The application of dependencies, in fact, is not restricted to facilitating collaboration; it can also be applied in other parts of the model (cf. §4.1). For instance, the underlying data structure of the circuit (cf. figure 6.1) is in a tabular form that is similar to that is used in spreadsheets. This potentially allows the use of dependencies between 'cells' where the components are located.

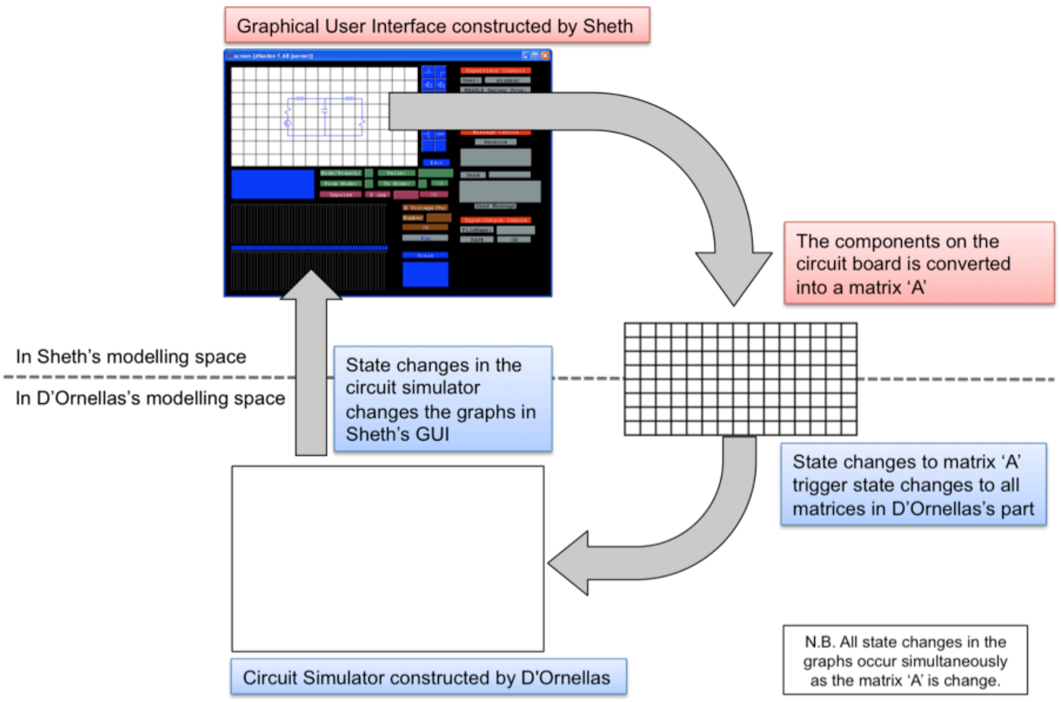

Graphical User Interface constructed by Sheth
The components on the circuit board is converted into a matrix 'A'
In Sheth's modelling space
In D'Ornellas's modelling space
State changes in the circuit simulator changes the graphs in Sheth's GUI
State changes to matrix 'A' trigger state changes to all matrices in D'Ornellas's part
N.B. All state changes in the graphs occur simultaneously as the matrix 'A' is change.
Circuit Simulator constructed by D'Ornellas

Figure 6.3 – Cooperation between Sheth and D'Ornellas in the VEL project

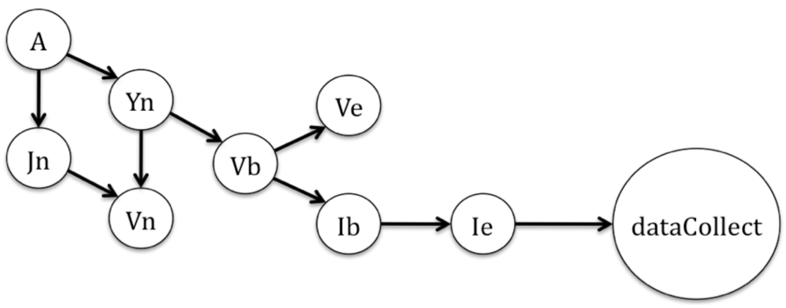


Figure 6.4 – Triggered actions graph after Matrix A is changed in the VEL model

Apart from revealing the cooperation, the analysis of the definitive scripts suggests that the two modellers may not have strictly followed the DEM framework during the modelling process – there are indications that they may have played the role of external observers, where in DEM, the subordinate modellers (or primitive modellers) are supposed to model agents through pretend play and internal observation (Sun, 1999). In the modelling process of the VEL model, there are only two modellers. However, there are more than two agents in both their individual models. It would not be possible to integrate all the agents together without external observation. This indicates that the modellers, at some point, must have observed the individual agents from an external perspective in order to investigate the interaction among those agents, either in the joint model, or in their individual models. Such a modelling process does not follow the I-modelling as proposed and advocated by the DEM framework, but more likely resembles the MMMA scenario I described in section 5.1.2. This, in turn, resonates with the call for better conceptual framework to apply EM in collaborative modelling context.

What potential of EM has the development process of the VEL demonstrated?

DEM allows a developer to play the role of an agent and make observations from a perspective inside the agent (Sun, 1999). In relation to the *pretend play* characteristic of DEM, two agents in the VEL deserve some attention in this analysis. In the VEL project, the teacher and the student are viewed as agents internal to the "VEL system". From an EM perspective, the role of the teacher and the student agents are not prescribed[101] before the modelling process; they emerge and are shaped through continuous observation throughout the EM process (cf. §4.1). The adoption of the DEM framework allows two modellers to play the role of the teacher and the student as if they were in the use context of the VEL, and to model from these perspectives when they gain new insight stemming from the experimentation with electronic circuits. In this way, the model will evolve through the interplay between the use of the model and the redefinition of it, within one and the same modelling and experimentation environment. This potentially conflates the contexts of development and use, which leads to the conflation of the roles of developer and user, as I discussed at length in chapter 3.

Apart from the potential for coordination and cooperation, the VEL project also demonstrates the flexibility and openness characteristics of EM. As mentioned in section 4.1, EM models are subject to refinement through further investigation and further insights of the modeller. In the VEL model, not only can the components (e.g. circuits) be changed on the fly, so that the laboratory environment supports the experimentation with electronic circuits, the laboratory environment, as a part of the EM model, is also open to change. For instance, D'Ornellas (1998) illustrates how the VEL model be integrated with Mendis's traffic lights model (EMPA:

[101] Strictly speaking, there is not even an initial presumption that the agents are to be played by humans, therefore 'users'. Only through the EM process does the role of these agents begin to be disclosed until it becomes clear how and why they should be played by humans, and therefore constitute a "context of use".

trafficlightMendis1997) for learning how a square wave generator can be used to control traffic lights at road junctions. Such a metamorphosis of the EM models (cf. Beynon and Sun, 1999) through connecting two originally unrelated EM models cannot be effected without the insight and ingenuity that the EM modeller has established through the modelling process.

6.2 Case study II: the development of a distributed Jugs model[102]

In section 5.3, I described how EM might support different degrees of engagement in collaborative modelling. In the previous section, the case study of the VEL project revealed how modellers might coordinate and cooperate throughout the EM process. Although the VEL case study revealed a number of implications of EM, the collaboration during the modelling process was nevertheless asynchronous[103]. Furthermore, in respect of the degree of engagement in collaborative modelling (cf. §5.1.1 and 5.3), the VEL case study only revealed EM's potential for coordination and cooperation,. This motivates us to study how modellers collaborate and co-construct through EM in synchronous (i.e. real-time) collaboration through the case study of the development of the "distributed jugs" (d-Jugs) model.

6.2.1 The background of the case study

The jugs model (EMPA: jugsBeynon1988) is a single-modeller constructed EM model which resembles the real world situation in which two jugs can be filled with liquid, emptied, or poured from one to another (cf. figure 4.6 on page 112). The modeller can either interact with the jugs through the graphical user interface or by making redefinitions through the input panel of the tkeden. The jugs model is simple yet one of the classic models that EM researchers use to demonstrate the principles of EM. For this reason, it seems appropriate

[102] The first distributed jugs modelling session was collaboratively carried out by the thesis author and Antony Harfield. Part of that data generated in this case study has been used by Antony Harfield (2008) in relation to his doctoral thesis. In addition, the second distributed jugs modelling session has been partially presented in the WPCCS 2006 (Chan 2006).

[103] Although the modellers in the VEL project were building a distributed model cooperatively, there is no evidence that they exploited real-time collaboration though the DEM tool support (i.e. dtkeden).

to study how a variant of the jugs model can be built collaboratively.

The target variant was a "distributed jugs" model in which modellers situated at different workstations could get access to either of the jugs in the model. However, this is not a precise "specification" of the meaning of "distributed jugs" – this was interpreted slightly differently in the two collaborative modelling sessions. Nevertheless, the development of the distributed Jugs (d-Jugs) model is quite similar to the single modeller's jugs model, except the fact that modellers have to construct the jugs collaboratively and concurrently.

The d-Jugs modelling has been carried out several times with different pairs of modellers. This case study reports two instances of such modelling[104]. The first collaborative modelling session was carried out by the thesis author with Anthony Harfield, who was interested in computer-supported learning with EM. The second collaborative modelling session was the first part of a "collaborative modelling workshop" that I carried out with two students. The second part of the workshop – collaborative Sudoku – will be discussed in section 6.3. The primary aim of the workshop was to explore how modellers might practise EM collaboratively through building a EM model collaboratively in real-time in a single shared modelling space where the configuration is similar to *single display groupware* (Stewart et al., 1999). The secondary aim was to give the volunteering students a flavour of collaborative modelling with an EM approach[105].

6.2.2 Method of study and approach to data analysis

Although the d-Jugs modelling was carried out twice with two different pairs of modellers, the methods used in both modelling sessions were similar:

> *Minimal planning activities* – in both modelling sessions, modellers were asked to keep the planning of the modelling to the minimum. Moreover, the modelling time was short (2 hours approximately in each case) and neither of the modellers was

[104] Prior to these two instances of successful collaborative modelling, the d-Jugs modelling was not so successful due to technical issues in the tool support – and this technical issue was one of the motivations to develop better support for collaborative EM.

[105] This was used as a "strategy" to attract volunteers to participate in our study due to the fact that neither DEM nor collaborative modelling with EM was part of the "Introduction to Empirical Modelling" course.

briefed before the collaborative modelling sessions. In this way, the interactions between modellers throughout the modelling process are more likely to be unstructured and unplanned. This makes modellers more likely to *collaborate* (in the sense described in section 5.1.1).

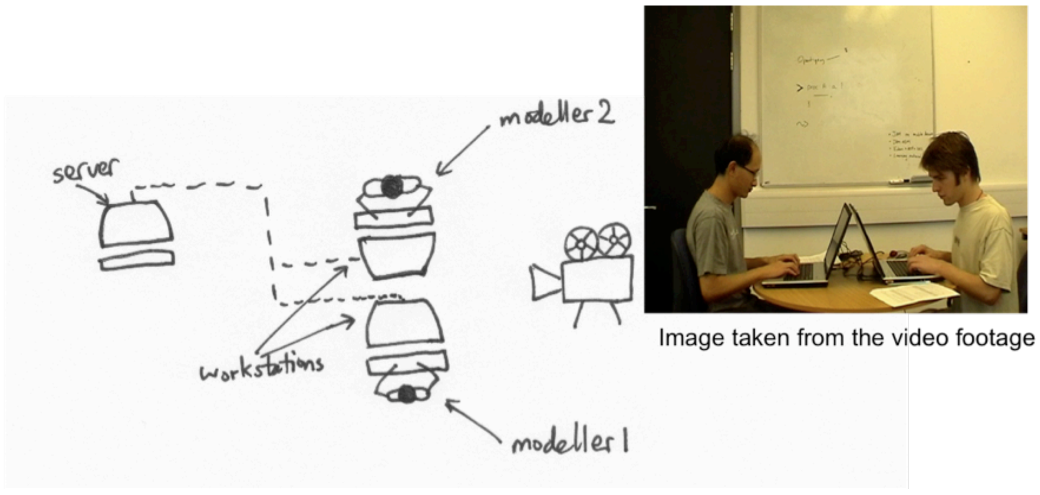


Figure 6.5 – The room configuration of the distributed jugs modelling case study

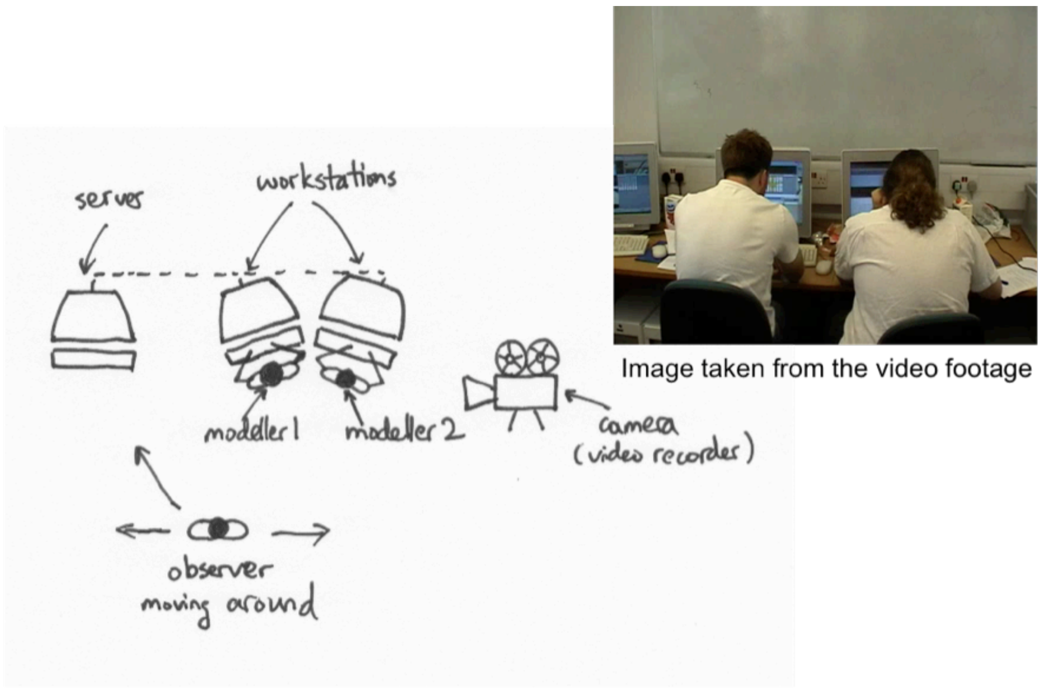


Figure 6.6 – The room configuration of the collaborative modelling workshop[106]

[106] Despite the fact that the camera was physically set up in the room, it was not used in the first part of the

Thinking aloud[107] – in both modelling sessions, modellers were encouraged to think aloud, i.e. to speak out what they are thinking when interpreting the meaning of what they see and their actions, why they are making such definitions or queries, etc. In this way, more information about the rationale or their state of mind can be obtained.

The data collection in both d-Jugs modelling sessions was informed by the participant observation approach (Spradley, 1980). Fieldnotes were taken on the spot during the modelling process. However, there was a significant difference in the degree of involvement between the first and the second d-Jugs modelling sessions. In the first d-Jugs modelling session, I was actively involved in the d-Jugs model construction while making observations to the modelling process. The degree of involvement was close to *complete participation* (Spradley, 1980). In contrast, in the second d-Jugs modelling session, I was merely involved in helping the modellers to troubleshoot the modelling environment when it was broken. Such a degree of involvement can be regarded as *passive participation* (Spradley, 1980).

Apart from the similarities in the study method, there are minor differences in the physical configurations and the tool support between the two versions of the d-Jugs modelling. In the first d-Jugs modelling session, the modellers sat face-to-face, with the workstations positioned in the middle between them (cf. figure 6.5). The modelling session were video recorded, screen captured (as video), and interactions within the modelling environment were logged.

In the second d-Jugs modelling session, the modeller pair were volunteering students selected from those who had attended the “Introduction to Empirical Modelling” course[108] in June 2006. The questionnaire that was used in the selection is attached in Appendix A. The main concern in the selection of students was availability and their proficiency in using tkeden, the primary tool support for individual EM. Neither of them had collaborative modelling experience with either dtkeden or dtkeden-cm prior to the d-Jugs modelling. The

collaborative modelling workshop (viz. the second distributed jugs modelling case study). It was only used in the second part of the collaborative modelling workshop (viz. the collaborative Sudoku case study, cf. 6.3).

[107] Cf. (Lewis and Rieman, 1994)

[108] “Introduction to Empirical Modelling (CS405)” is an optional module for the four-year MEng degree programme in Computer Science at University of Warwick.

physical configuration of the second d-Jugs modelling session was similar to that described in (Flor and Hutchins, 1992), where the modellers were sitting side-by-side (cf. figure 6.6). In contrast to the first modelling session, only interaction history was logged in the second modelling session. That is, there was no video recording and screen capturing. Two variants of dtkeden were used in the d-Jugs modelling sessions: in the first modelling session, the dtkeden-blackboard[109] was used; in the second modelling session, the dtkeden-cm[110] was used. The major difference between them is the mode of interaction. While the former supports the blackboard mode of interaction, the latter supplies an enhancement of dtkeden in "normal mode" with improved readability in the log files and different use scenarios (in contrast to the four closely associated with the DEM conceptual framework).

Despite the differences in the data collection techniques and physical configurations, both d-Jugs modelling sessions were analysed in a similar way. The motivation behind this strategy (i.e. analysing both modelling sessions in a similar way) was to make the results more comparable, so that cross-case analysis (cf. Yin, 2009) become possible. Both analyses can be roughly divided into four steps:

1. Reconstruct the modelling process by synthesizing a time-sequence log from multiple data sources.
2. Add annotations to the time-sequence log.
3. Identify events in the modelling process and seek explanation for the events.
4. Identify characteristics of the modelling process and seek explanation for the characteristics.

In the analysis of the first d-Jugs modelling, the time-sequence log was synthesized from

[109] This modified version of dtkeden was developed by the thesis author and Antony Harfield collaboratively for the purpose of the distributed jugs modelling. In dtkeden-blackboard, the virtual agency is partially removed so that it gives a single shared modelling space across all workstations and the server. This was later known as the "blackboard mode", as it resembles the blackboard metaphor (also cf. §5.1.3). Technically, the dtkeden-blackboard is equivalent to dtkeden in 'normal' mode with both automatic propagation and the virtual agency feature disabled, and some minor user interface changes.

[110] This enhancement gives the following features: (i) larger text in the input window (originally developed by Karl King); (ii) timestamps for the redefinitions logged in the interaction history file; and (iii) the distinction between definitions in the public space and in the private spaces was made explicit.

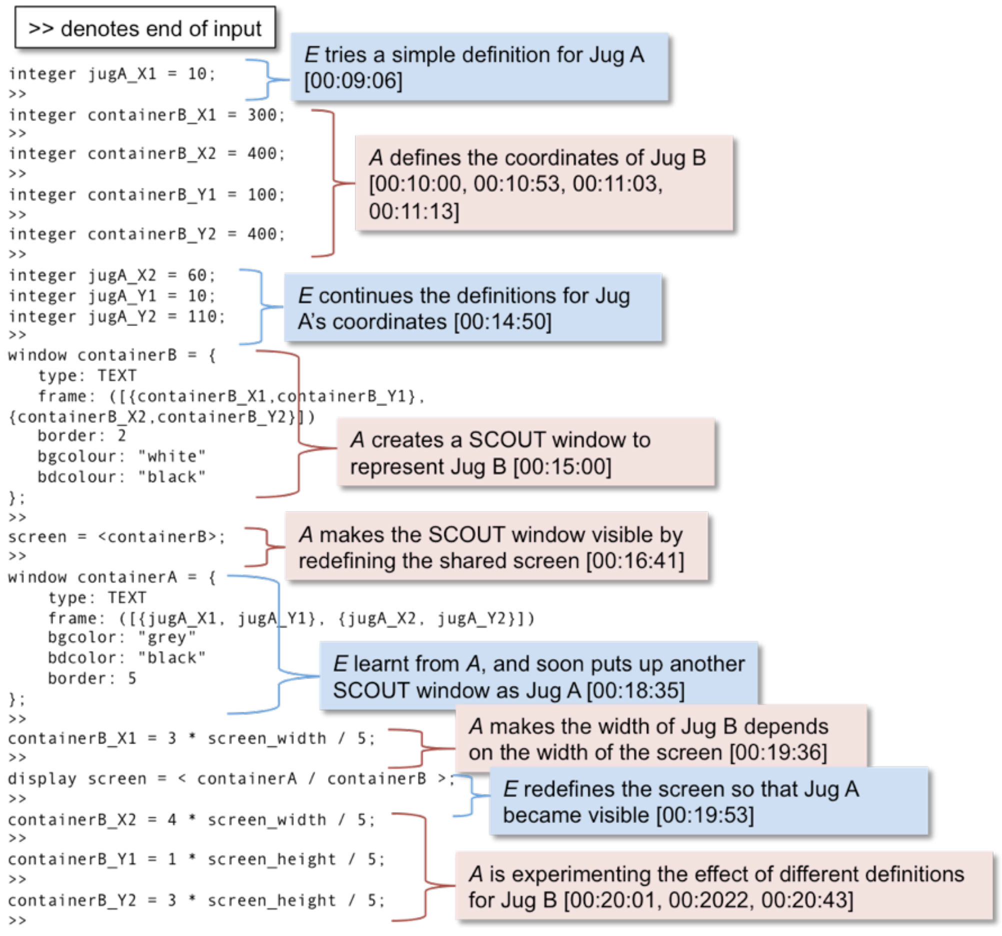
three data sources, namely, video footage, screen video capture, and the interaction history (in the form of definitive scripts). However, the synthesis of the data sources was far from straightforward. This was due to the fact that the interaction history has no timestamps. That is to say, I could gain no information about when a definition was input into the modelling environment from the interaction history alone. It was the screen video capture, after synchronising with the video footage, which provided the missing link between the video footage and the interaction history. In the second step, fieldnotes were edited and used to annotate the time-sequence log. Annotations were not by any means complete and they merely provided a starting point for further studying "what is going on?" in the modelling process. In the third step, the modelling process was broken down into events (or segments). The beginning timestamp, the ending timestamp, and the relative definitive scripts in the interaction history for the events were identified. Through multi-directional tracing between data sources, and with reference to the EM principles (cf. chapter 4), descriptions of the events and an explanation of the interactions between the modellers were derived in the context of collaborative modelling (cf. chapter 5). Attention was also given to the social context in which the collaborative modelling took place, e.g. the use of and the switching between the private and the public modelling spaces. To avoid bias in the analysis of the events, consideration were given to whether the modellers were actually practising an EM approach. In the fourth step, I looked at the modelling process at the holistic level. Attention was given to the role of EM and its tool support in the collaborative modelling process.

Similar analysis techniques were used in the analysis of the second d-Jugs modelling. However, there were two significant differences between the analysis of the first and the second d-Jugs modelling. Firstly, the time-sequence log of the second d-Jugs modelling was synthesized from two data sources, namely, video footage and the interaction history. This was because the screen video capture was not possible due to a technical limitation. Secondly, the data set for analysis was much larger than in the first d-Jugs modelling session. This was due to the fact that the modelling process in the second d-Jugs modelling session was much longer.

6.2.3 Limitations

In the distributed jugs case study, both d-Jugs modelling sessions were informed by participant observation (Spradley, 1980). In the first d-jugs modelling session, complete participant observation was adopted as one of the techniques for data collection. It enabled me, as a researcher, to obtain deep insight into the process of collaborative modelling with an EM approach. Such first-person experience would be otherwise difficult, if not impossible, to obtain without being a participant inside the activity.

The major criticism for participant observation is that the study may be shaped by the researcher’s bias, whether intentionally or unintentionally. It is hard to avoid such criticism even if the researcher has consciously tried to avoid bias. To address this criticism, a reduced degree of involvement (i.e. passive participant observation) was adopted in the second d-Jugs modelling session. That is, help was only offered to the modellers when they struggled with the tools and were unable to overcome an issue in a timely manner. Moreover, other means of data collection were also used in both d-Jugs modelling sessions. Having multiple sources of data is an advantage of case study as a research method (Yin, 2009). To some extent, this has enabled data triangulation (Yin, 2009), so that the findings of the case study do not rely solely on the fieldnotes that were generated from participant observation.



>> denotes end of input

```
integer jugA_X1 = 10;
>>
integer containerB_X1 = 300;
>>
integer containerB_X2 = 400;
>>
integer containerB_Y1 = 100;
>>
integer containerB_Y2 = 400;
>>
integer jugA_X2 = 60;
integer jugA_Y1 = 10;
integer jugA_Y2 = 110;
>>
window containerB = {
   type: TEXT
   frame: ([{containerB_X1,containerB_Y1},
{containerB_X2,containerB_Y2}])
   border: 2
   bgcolour: "white"
   bdcolour: "black"
};
>>
screen = <containerB>;
>>
window containerA = {
    type: TEXT
    frame: ([{jugA_X1, jugA_Y1}, {jugA_X2, jugA_Y2}])
    bgcolor: "grey"
    bdcolor: "black"
    border: 5
};
>>
containerB_X1 = 3 * screen_width / 5;
>>
display screen = < containerA / containerB >;
>>
containerB_X2 = 4 * screen_width / 5;
>>
containerB_Y1 = 1 * screen_height / 5;
>>
containerB_Y2 = 3 * screen_height / 5;
>>
```

E tries a simple definition for Jug A [00:09:06]

A defines the coordinates of Jug B [00:10:00, 00:10:53, 00:11:03, 00:11:13]

E continues the definitions for Jug A's coordinates [00:14:50]

A creates a SCOUT window to represent Jug B [00:15:00]

A makes the SCOUT window visible by redefining the shared screen [00:16:41]

E learnt from *A*, and soon puts up another SCOUT window as Jug A [00:18:35]

A makes the width of Jug B depends on the width of the screen [00:19:36]

E redefines the screen so that Jug A became visible [00:19:53]

A is experimenting the effect of different definitions for Jug B [00:20:01, 00:2022, 00:20:43]

Figure 6.7 – A glimpse of the interaction between two modellers in a distributed jugs modelling[111]

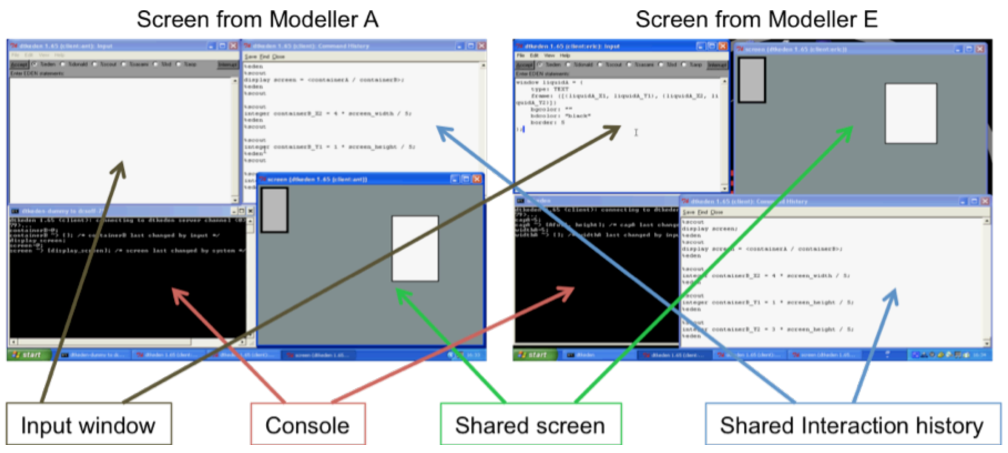


Figure 6.8 – Collaborative modelling of a pair of jugs at the early stage

[111] This figure is modified from figure 5.24 in (Harfield 2008, p.170), with an improved description of the actions by the modellers and timestamps added (for cross referencing with figure 6.9)

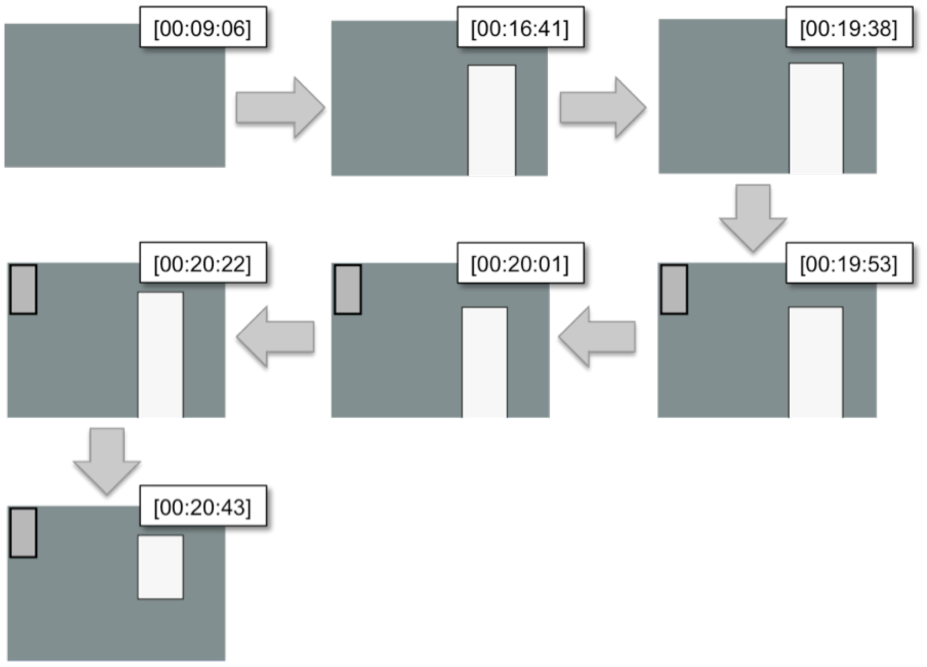


Figure 6.9 – A glimpse of the development of the distributed jugs in a collaborative modelling environment

6.2.4 The collaborative modelling process

In the first d-Jugs modelling session, modellers did not begin the collaborative modelling from scratch. After some discussion, the text-based jugs model[112] was adopted as the foundation for the d-Jugs model. Indeed, the aim of the modelling session was to study the collaborative modelling process, not to construct the product (i.e. a working, polished jugs model). Despite the fact that the modellers in this modelling session were very experienced in using tool support for EM, they were struggling with different sorts of problems (e.g. invalid syntax, being unaware of the current notation) during the first 9 minutes of the collaborative modelling process. Instead of making complex definitions, which these modellers normally do when they practise EM on their own, they merely made simple one-line definitions such as *"capA = 5;"* and *"Aempty is contentA==0;"* initially. Later, modeller *E* made a breakthrough by making a simple definition for the coordinates of Jug A (cf. figure 6.7). Soon, taking up modeller *E*'s suggestion, modeller *A* made a few definitions for the

[112] A variant of the original jugs model (EMPA: jugsBeynon1988) that was used in a laboratory of the "CS405 Introduction to Empirical Modelling" course in the academic year 2006/07.

coordinates of Jug B. Then, modeller *E* completed the definition for the coordinates of his jug. After the initial success, modeller *A* went back to what he did initially – defining a SCOUT window to represent a jug – and made that visible on the shared screen. At this moment, modeller *E* got excited – since he saw a jug appearing on the shared screen. He created another SCOUT window to represent his jug, by referring to the definition of Jug B[113] created by modeller *A* a moment ago (cf. figure 6.7 and the second screen in figure 6.9). On the other side, modeller *A* was not quite satisfied with the appearance of his jug, and experimented with the effect of different definitions. Figure 6.9 highlights the crucial moments – in relation to visual experience on the shared screen – in this short collaborative modelling session.

In the second d-Jugs modelling session, modellers were asked to extend the jugs model (a model used in a laboratory of the “Introduction to Empirical Modelling” course in 2006)[114] into three jugs. Their collaborative modelling process can be summarised as follows[115]:

1. In contrast to the first d-jugs modelling session, the modellers were testing the collaborative modelling environment with simple definitions during the first 15 minutes (c.f. Appendix D1). This is probably due to the fact that the dtkeden-cm environment was new to them, and the modellers wished to find out the difference between “Accept Locally” and “Send to Public” (cf. Appendix B).
2. After the initial testing, the modellers soon switched to composing their scripts in an external editor, copying and pasting across from the editor to the modelling environment. For instance, at one point, modeller *R* repeatedly copied and pasted a similar script a few times from his external editor to the modelling environment (c.f. Appendix D2)[116].This is another style of modelling,

[113] The SCOUT definition of Jug B was displayed in the shared interaction history window located in the lower right-hand corner of his screen (cf. figure 6.7)

[114] A variant of the original jugs model (EMPA: jugsBeynon1988) that was used in a laboratory of the “CS405 Introduction to Empirical Modelling” course in the academic year 2006/07.

[115] The collaborative modelling process is reconstructed mainly based on the interaction history logged on the workstations.

[116] The similarities among (re)definitions and the short time intervals between (re)definitions suggests that modeller

unlike that used by the modellers in the first d-Jugs modelling session. In contrast, their "interaction" within the modelling environment became "repetitive" because of their copy-and-paste.

3. Then the modellers began to work in their private workspaces and occasionally put their mature pieces into the public space (cf. Appendix D2 and D3). For instance, modeller *S* experimented for about 10 minutes in his private space with different sizes and colours for the SCOUT window *fillA* (which represents a button in this model) before he put that button, together with another two buttons (*pourAtoBbutton* and *pourAtoCbutton* – developed by cloning and modifying the *fillA* button – into the public space (cf. 10:42:54 to 10:54:52 Appendix D3).

4. For the first 25 minutes after initial testing, modeller *R* had been working privately in an editor (external to the modelling environment). Then, he turned to test his piece in his private modelling environment. When that piece (including some definitions for Jug C, all the button agents, pouring agents and the animation agent) matured, he put it into the public space [117].

5. Modeller *R* tested his piece in the private space again, making sure the buttons were working[118]. He then put all the dependencies for guarding the button in the public space simultaneously. Again, modeller *R* put his piece first into the private space (to make sure that will not cause any harm in the public space),

R was not typing the (re)definitions but modifying them in an external editor and copying-and-pasting them to the modelling environment.

[117] No (re)definitions of Jug C were sent to the public space between 10:30:39 (the last redefinition from modeller S that was captured on the dtkeden-cm server) and 10:54:38 (the first definition that modeller S sent to the public modelling space after the initial testing). The latter definition indicates the end of the initial testing (cf. Appendix D1). Meanwhile, the definition that modeller R sent to his private modelling space at 10:55:43 indicates that modeller R was beginning to test his contribution to the (re)definitions of Jug C in his private modelling space (cf. Appendix D2).

[118] Modeller R did not define the SCOUT windows for the buttons – it was modeller S who defined the buttons (SCOUT windows) *fillA*, *pourAtoBbutton*, and *pourAtoCbutton*. (cf. Appendix D3). Rather, modeller R defined the trigger procedures behind the buttons (cf. 10:58:02 to 10:59:49 in Appendix D2). The interaction history between 10:58:02 and 10:58:03 (cf. Appendix D2) indicates mouse button clicks on the SCOUT windows: *fillA*, *pourAtoBbutton*, and *pourAtoCbutton*. Though it was not 'formal' software testing, these mouse clicks suggests that modeller R was experimenting or 'playing around' with the buttons after he sent the Jug C definitions to the public modelling space.

then into the public space (cf. 10:59:44 and 10:59:49 in Appendix D2).

6. While modeller *R* was busy in putting Jug C into the shared modelling space, modeller *S* was testing the other two jugs – Jug A and Jug B – in his private modelling space. The interaction history between 10:55:43 and 10:58:03 shows that modeller *R* was concentrating on making the basic definitions for Jug C and making the pouring buttons work (cf. (2) above and Appendix D2). Meanwhile, the interaction history between 10:42:54 and 11:06:15 shows that modeller *S* was concentrating on making definitions for the SCOUT windows that represents various buttons for the filling function for Jug A and Jug B, and the pouring actions between them (cf. Appendix D3 and D4).

7. Soon, it became apparent that the construction was pretty much dominated by modeller S, and modeller *R* shifted into a more passive role. This is to say, the collaborative modelling of the distributed jugs was largely driven by modeller S. This is suggested by the observation that modeller *R* did not make any contribution to the public modelling space between 10:59:49 and 11:30:25 (cf. Appendix 5).

8. At around 11:40, the dtkeden-cm client on modeller *R* was malfunctioning. The pair restarted the dtkeden-cm server and clients, and reloaded all the scripts that they had backed up in their private editors.

9. Mature versions of Jug A, Jug B, Jug C, and all the GUI agents were developed after 85 minutes of collaborative modelling (cf. Appendix D6).

The data excerpt in appendix D6 shows the major contributions from the modellers which have shaped the final version of their d-Jugs model in the modelling session. Despite the fact that their d-Jugs model could perform some basic operations (e.g. pouring from one jug to another), there are some ‘bugs’ in their model. In order to make their d-Jugs model function as expected, the definitions that are listed in appendix D7 would be required. Figure 6.10 shows a screen capture of the resulting d-Jugs model, i.e. the final version of the d-Jugs model constructed by modellers R and S with the definitions in appendix D7 added.

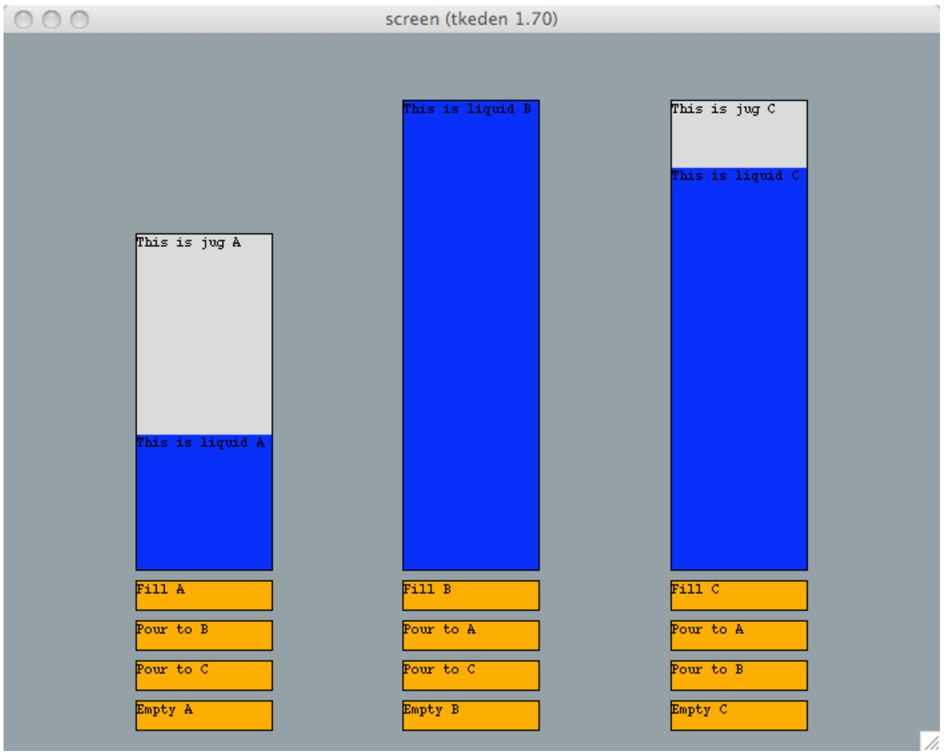


Figure 6.10 – Screen capture of the distributed Jugs model constructed in the second distributed jugs modelling session

6.2.5 Discussion

Although both d-Jugs modelling sessions are trivial and short, they do illustrate some issues and potential benefits of practising EM in collaborative modelling.

Coordination in the shared space

In the first d-Jugs modelling session, both modellers are experienced in tool support for EM. Both modellers are familiar with the text-based jugs model (EMPA: jugsBeynon1988). Consequently, there was no planning at all; responsibility for the observables, dependencies, and agents were negotiated *in situ*. For instance, at the beginning of the modelling process, modeller *E* said that he would like to construct the graphical representation of Jug A on the screen. Modeller *A* responded to modeller *E* that he would create the other jug (Jug B). Modeller *A* and modeller *E* had a short conversation about where to put their jugs before they introduced their first definitions relating to their jugs. In the second d-Jugs modelling session, the modellers simply avoided conflict by working in their private editors and only using the shared modelling space when their pieces were mature.

Despite working in a shared modelling environment, modellers in both d-Jugs modelling sessions have actively avoided the potential conflicts in changing others' work, through different strategies. In the first case, modellers were using preceding verbal communication and queries to the identifier in question[119]. The dtkeden-blackboard (and dtkeden-cm) modelling environment responds to queries of identifiers promptly[120]. For instance, Modeller A queried the *screen* observable[121] before he made a definition in relation to it.

To some extent, this mode of working may facilitate collaborative modelling effectively but unobtrusively. In the second case, modellers exploited the private and public spaces. Careful analysis suggests that these are two different kinds of collaborative modelling; while both can be regarded as exemplifying the MMMA scenario, the former is *collaboration* and the latter is *cooperation* (cf. the discussion in section 5.1.1). On the one hand, this seems analoguous to the natural avoidance of entering another's "personal territory" in spatial interaction (cf. Scott et al., 2004; Tse et al., 2004), e.g. using whiteboards or tabletop groupware. On the other hand, this suggests that EM coped well with two degrees of engagement in collaborative modelling.

Experimentation

In the first d-Jugs modelling session, modeller *A* was not satisfied with the appearance of his jug. He made use of the experimental characteristics of the EM tool support to *try-and-feel* the effect of different definitions. Similar *try-and-feel* actions are also found in the second d-Jugs modelling session, where both modeller *S* and *R* experimented with their pieces before making them available to the other modeller via the shared modelling space. This suggests that the experimental characteristics of EM may play an active role in real-time collaborative modelling, where modellers want to verify their perceptions (i.e. their observation of the referent in the sense of EM) either through public experimentation – in the first case – or through private experimentation – in the second case. Apart from potential conflicts, when

[119] That is, to check whether it has been used and, if so, consult its current definition before making any changes. An identifier in tkeden and all its variants (including dtkeden-blackboard and dtkeden-cm that were used in the modelling sessions in the case study) can be an observable, a dependency, or an agent.

[120] This feature is in fact inherited from tkeden, the primary tool support for EM.

[121] An observable which is often used to denote the default SCOUT screen in most EM tool support.

well coordinated, the only difference between public and private experimentation is its visibility.

Role-shifting

In chapter 3, I argued that multiple roles and shifting roles of participants feature in the systems development process. If we view the collaborative modelling process as a miniature form of a systems development process, and attribute roles to modellers in relation to their task in the situation and their current focus of attention, it is not difficult to imagine a modeller may have multiple roles and shifting roles throughout the collaborative modelling process. In other words, a modeller can be thought as a developer, a learner, a user, a teacher, a demonstrator, etc. The d-Jugs modelling sessions seem in line with this phenomenon. In the first d-Jugs modelling session, modeller *E* observed, learnt and cloned SCOUT definitions of a jug from modeller *A* (cf. figure 6.7 on page 187, 00:18:35). In that moment, modeller *E* was in a learner role rather than merely a developer. In the same spirit, modeller R, in the second d-Jugs modelling session, has turned into a learner and modeller *S* became a teacher (or demonstrator) in the later stages of collaborative modelling process. In addition, both modeller *S* and modeller *R* verified their definitions in their private modelling spaces before they made them public. Such behaviour can be thought as *using* a component of the model in their private space. Taking all the above into consideration, it is plausible to claim that EM does not obstructed the role-shifting phenomenon and, to some extent, its open, flexible, experimental, and instant feedback characteristics facilitate seamless shifts.

Dialogical and diverse interaction

As I argued in section 4.3, the interaction between the modellers and the interaction between an individual modeller and the agents within the model (whether in the private or public modelling space) can be thought of as dialogical. As shown in figure 6.7 (on page 187), the interaction between modellers *A* and *E* more or less resembles a dialogue (or a conversation). This resemblance stems from the pattern of interaction, whereby A and E take it in turns to contribute to the definitions of Jug A and Jug B, and the common theme of their interactions.

The effect of this dialogue is to negotiate the meaning of “distributed jugs” (cf. §6.2.1) through redefining the SCOUT screen. For instance, modeller *A* could have defined:

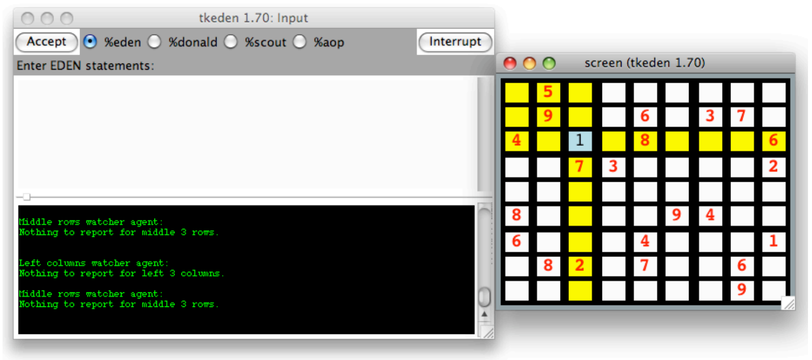
```
screen = <containerA / containerB / liquidA / liquidB>
```

but instead, he actually defined:

```
screen = <containerB>
```

overlooking the fact that there should be several SCOUT windows on the screen. It was modeller *E* who spotted the issue a moment later when he could not see two jugs on the screen. Even then, *E* did not consider the need for more SCOUT windows (viz. *liquidA* and *liquidB*), which he would have to add subsequently. Similar dialogical interaction can be found in the second d-Jugs modelling session, where modeller *S* and modeller *R* exchanged definitions through the public modelling space and interacted with their local model in their private modelling space through redefining observables. The interaction among all parties (i.e. the modeller, and the agents in the model) suggests that collaborative modelling, when practising an EM approach, is dialogical.

In both d-Jugs modelling sessions, modellers interacted through different channels, which included the evolving EM model, gestures around the screen, and verbal communication. They also switched between different means of communication seamlessly to: i) exchange their perspectives; ii) negotiate when conflicting perspectives arise; iii) coordinate their work; iv) inform about and diagnose each other's progress. The virtue of the blackboard mode is that it allows diverse modes of interaction, which as mentioned earlier also enables the modeller to shift their roles seamlessly.



```
Middle rows watcher agent:
Nothing to report for middle 3 rows.

Left columns watcher agent:
Nothing to report for left 3 columns.

Middle rows watcher agent:
Nothing to report for middle 3 rows.
```

Figure 6.11 – A screen capture of King's Sudoku model (EMPA: sudokuKing2006) with the tkeden modelling environment

6.3 Case study III: collaborative Sudoku

Like the second d-Jugs modelling session described in the previous section, the collaborative Sudoku modelling was part of the collaborative modelling workshop. However, unlike the case study of the d-Jugs modelling, the focus for the model building was more complicated than in the previous one, and related to computer support for human solving of Sudoku puzzles.

6.3.1 The background of the case study

Sudoku is a popular puzzle game in newspapers and magazines[122]. The first EM model of the Sudoku puzzle was constructed by Karl King (EMPA: sudokuKing2006) in relation to his Masters thesis (King, 2007). For King, the modelling of the Sudoku was:

> *"an exercise in embodying personal understanding within a computational artefact in conjunction with skill development" and "it was to explore ways in which the puzzle solving exercise could be usefully supported by the computer, and how the skills of the human solver could be embodied in a computer-based artefact" (ibid, p.48).*

As depicted in figure 6.11, King developed a number of visual aids to assist the human solver of the original Sudoku puzzle game. King's Sudoku model was enhanced by Efstathiou (2006) for studying his definitive notation for representing combinatorial graphs (C-Graph) (also cf. EMPA: sudokuEfstathiou2006) and later enhanced with colours by Harfield for tackling Sudoku with a novel technique (EMPA: sudokucolourHarfield2007). The collaborative Sudoku modelling was based on King's Sudoku model, and it was made available to the modellers at the beginning of the modelling session.

6.3.2 Method of study and approach to data analysis

As a part of the collaborative modelling workshop, the collaborative Sudoku modelling has the same physical configurations of the equipment, where modellers were adjacently seated and working on a shared model through separated but networked workstations (cf. figure

[122] http://www.guardian.co.uk/media/2005/may/15/pressandpublishing.usnews

6.6). The modeller pair was the same pair who participated in the first part of the collaborative modelling workshop. The modelling session was video recorded and interaction within the modelling environment was logged. As in the first part of the collaborative modelling workshop, dtkeden-cm was used in the modelling session (cf. §6.2.2). Due to instability of the tool support, the observer had an active role in supporting the modellers when the modelling environment broke down[123], though this was a rare occurrence.

During the collaborative modelling process, the modellers were asked to develop a collaborative strategy to tackle any given Sudoku puzzle collaboratively. Our intention was to encourage the modellers to make use of observables, dependencies, agents, and agencies. For instance, the modellers were told that they could represent the way in which the value in one square affects another using dependencies, and that through negotiating and validating the dependencies the solution may emerge. They were also given prototypes for the agents that would exploit such dependencies to implement rules of the form "if digit *i* is placed in cell *x* then digit *i* must also by placed in cell *y*". The nine agents for the digits (viz. *i* equals 1 to 9) were called *'checkrules1'* to *'checkrules9'* respectively.

In addition, they were encouraged to make use of the benefit of having a separate private space for their local experimentation when necessary. Apart from those guidelines, the modellers were free to make any changes within the modelling environment, free to work in any style, and free to switch to any modes of interaction when necessary.

The analysis strategy for the collaborative Sudoku case study was similar to the four-step analysis strategy that was adopted in the distributed jugs case study (cf. §6.2.2). The topical research question for the analysis was "how do the modellers practise an EM approach to collaborative modelling collocated at real-time?" Therefore, the main focus was on building an explanation for such a modelling situation. In comparison with the distributed jugs modelling case study, collaborative Sudoku generated significantly more data for analysis. Despite the fact that timestamps were logged in the interaction history (an improvement in

[123] To avoid unnecessary interference to the modeller, we only provide support when there was a breakdown in using the tool support and when they could not remedy the situation during the modelling session.

data collection over the distributed jugs case study), the synthesis of the data sources in the analysis of the collaborative Sudoku modelling was not as trivial as expected. This was due to the fact that the interaction history were not originally designed solely for research purposes. Consequently, they were scattered in different locations and overlapped in some respects. As in the analysis used in the distributed jugs modelling case study (cf. §6.2.2), an annotated time-sequence log was generated after the second step of the analysis. Then, events in and characteristics of the modelling process were identified based on the annotated time-sequence log, the video footage, and the interaction history.

Apart from the similarities, the analysis for the collaborative Sudoku case study was different from the four-step analysis strategy that was adopted in the distributed jugs case study in the following ways:

i. *Dialogue transcription from the video footage* – Due to the complex interaction that was observed in the events, all the dialogues in the video footage were also transcribed literally. These transcribed dialogues were used as supporting evidence for the claims in the discussion section (see §6.3.4 for details).

ii. *Informal interview at the end of the modelling session* – During the collaborative Sudoku modelling, I observed that the modellers were struggling to collaborate efficiently. At that point, it was unclear to me what caused the issue. For this reason, I asked the modellers to give their feedback at the end of the collaborative Sudoku modelling.

6.3.3 Limitations

The collaborative Sudoku modelling session was the second part of the Collaborative Modelling workshop, which aimed to reveal how EM modellers approach and collaborate in a more complex problem situation. As in the second d-Jugs modelling session, the collaborative Sudoku case study adopted passive participant observation as one of the techniques for data collection. Likewise, other means of data collection (viz. video capture, and interaction history) were used to avoid some of the pitfalls of participant observation. The main reason that participant observation was not rejected is that it gives an opportunity for the researcher to rectify minor problems, such as tool stability in our case (Yin, 2009). It would otherwise be difficult for the researcher to rectify the problems as an external observer.

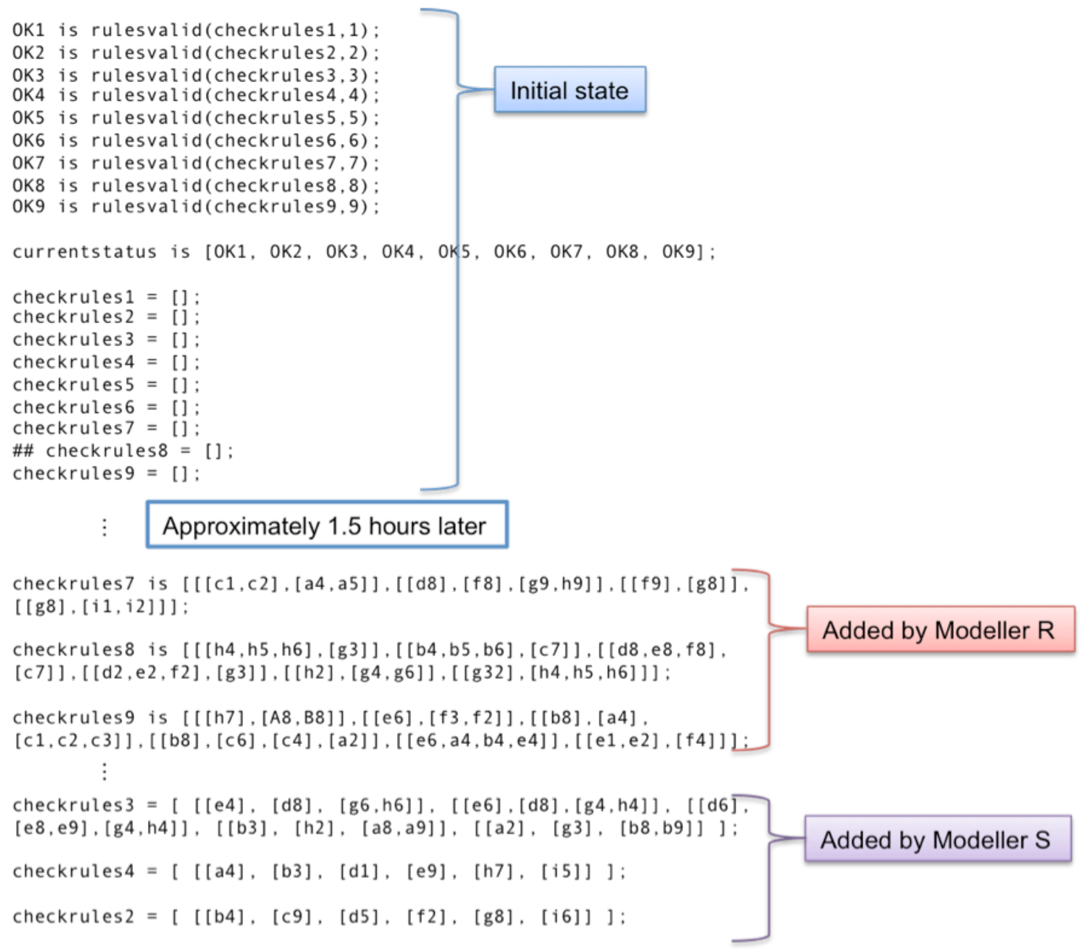
6.3.4 Discussion

The original intention of the collaborative Sudoku modelling was to reveal how modellers might carry out EM collaboratively with two modelling spaces – their private modelling space and a shared modelling space – at real-time. That is, to allow the modellers to improve the Sudoku model collaboratively through reflecting on their own experience[124] in relation to the game, and through interaction with the model and the other modeller. The idea was to reveal the interplay between the developing artifact and the modellers' understanding throughout the modelling process.

The pair of modellers in the collaborative Sudoku modelling was the same pair who participated in the second d-Jugs modelling session (cf. §6.2). As in the d-Jugs modelling session, the modellers in the collaborative Sudoku modelling exhibited similar role-shifting behaviour, dialogical interaction, and diverse interaction behaviour. However, the result of the collaborative Sudoku modelling was quite different in relation to the artifact – there was no "working product" out of the collaboration process. Despite the fact that Sudoku was so popular, the modellers had not heard of the Sudoku. In other words, they had no previous experience to draw on. It became a "collaborative comprehension exercise", and incidentally made the modelling process similar to the early stage of systems development where participants are spending much of their time in learning the problem domain[125]. It turned out to be an interesting scenario because the modellers were not only co-constructing an understanding of that problem situation, but because that understanding and the goal of the activity also co-evolved through the modelling process.

[124] That is, the puzzle is printed on a paper and solved using a pencil and a rubber.

[125] Apart from the scale, our situation here is similar to the situation in Danielsson's (2004) mobile learning systems development project, where the students (i.e. future users) of the system had no knowledge of what could be done with the technology and therefore the students became learners in the early stage of the systems development process.

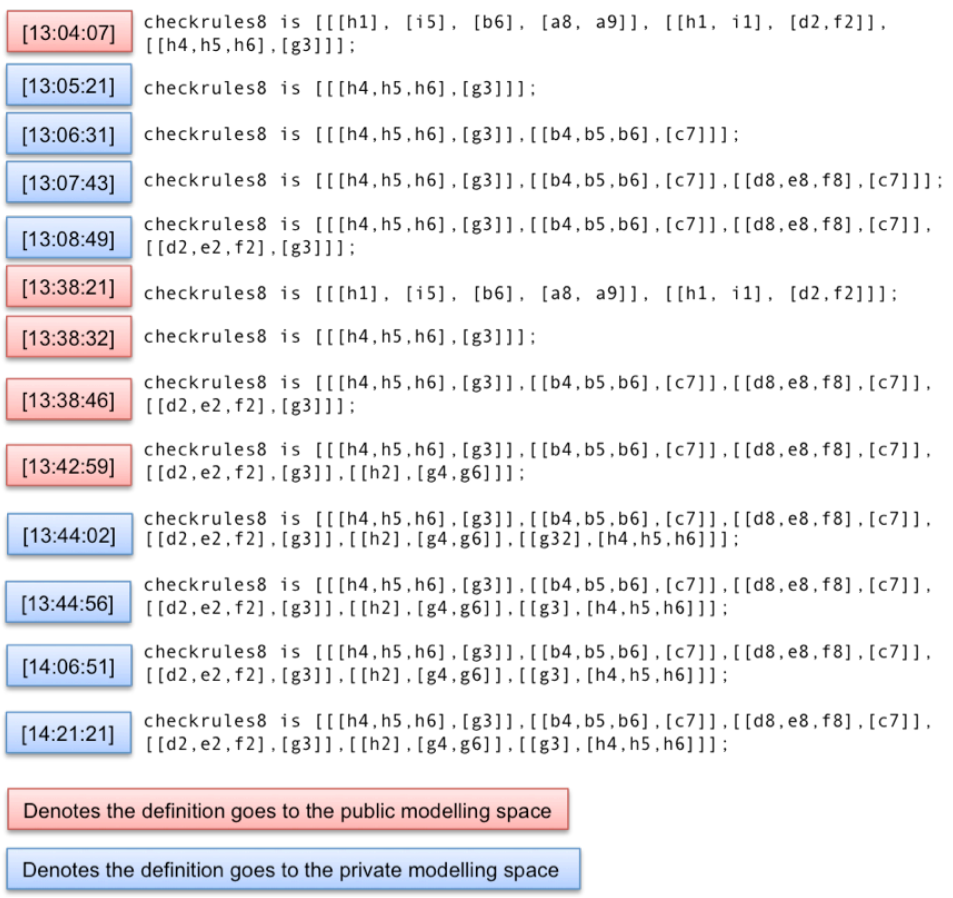


```
OK1 is rulesvalid(checkrules1,1);
OK2 is rulesvalid(checkrules2,2);
OK3 is rulesvalid(checkrules3,3);
OK4 is rulesvalid(checkrules4,4);
OK5 is rulesvalid(checkrules5,5);
OK6 is rulesvalid(checkrules6,6);
OK7 is rulesvalid(checkrules7,7);
OK8 is rulesvalid(checkrules8,8);
OK9 is rulesvalid(checkrules9,9);

currentstatus is [OK1, OK2, OK3, OK4, OK5, OK6, OK7, OK8, OK9];

checkrules1 = [];
checkrules2 = [];
checkrules3 = [];
checkrules4 = [];
checkrules5 = [];
checkrules6 = [];
checkrules7 = [];
## checkrules8 = [];
checkrules9 = [];

        ⋮

checkrules7 is [[[c1,c2],[a4,a5]],[[d8],[f8],[g9,h9]],[[f9],[g8]],
[[g8],[i1,i2]]];

checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],
[c7]],[[d2,e2,f2],[g3]],[[h2],[g4,g6]],[[g32],[h4,h5,h6]]];

checkrules9 is [[[h7],[A8,B8]],[[e6],[f3,f2]],[[b8],[a4],
[c1,c2,c3]],[[b8],[c6],[c4],[a2]],[[e6,a4,b4,e4]],[[e1,e2],[f4]]];
        ⋮
checkrules3 = [ [[e4], [d8], [g6,h6]], [[e6],[d8],[g4,h4]], [[d6],
[e8,e9],[g4,h4]], [[b3], [h2], [a8,a9]], [[a2], [g3], [b8,b9]] ];

checkrules4 = [ [[a4], [b3], [d1], [e9], [h7], [i5]] ];

checkrules2 = [ [[b4], [c9], [d5], [f2], [g8], [i6]] ];
```

Figure 6.12 – Co-construction of the check rules in the collaborative Sudoku model

Co-construction and co-evolution

The first one and a half hours of the collaborative Sudoku modelling can be thought of as a co-construction through EM of a shared understanding (e.g. what Sudoku is about, and what is the collaboration strategy for tackling the puzzle) between the modellers. In this period, the modellers gradually understand what the "check rules" can do, as a mechanism to tackle the puzzle, and what is its limitation. They also managed to define the "check rules" for some numbers and at least agreed that the check rules for digit 9 "seems fine". As shown in figure 6.12, the pair co-constructed some "check rules" after approximately 1.5 hours of collaboration. However, the modellers did not leap from the initial set of check rules into the final set at once. Taking the set of check rules for the number eight (i.e. *'checkrules8'* agent) as an example, modeller *R* refined *'checkrules8'* in all 13 times throughout the modelling process (cf. figure 6.13). Similar updates were made to all the other *'checkrule'* agents in the model. As we discussed in chapter 4, all agents co-evolve with the modeller's understanding of the situation in EM. Indeed, any redefinitions of '*checkrules8*' would also cause immediate redefinition of other observables in the model (e.g. *'currentstatus'*) and that in turn affected modeller S's action in redefining other *'checkrules'*. This interplay between modellers and redefinition exemplified one small aspect in the negotiation of the meaning of possible solution (of the puzzle).



```
🟥 [13:04:07] checkrules8 is [[[h1], [i5], [b6], [a8, a9]], [[h1, i1], [d2,f2]],
              [[h4,h5,h6],[g3]]];
🟦 [13:05:21] checkrules8 is [[[h4,h5,h6],[g3]]];
🟦 [13:06:31] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]]];
🟦 [13:07:43] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]]];
🟦 [13:08:49] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]],
              [[d2,e2,f2],[g3]]];
🟥 [13:38:21] checkrules8 is [[[h1], [i5], [b6], [a8, a9]], [[h1, i1], [d2,f2]]];
🟥 [13:38:32] checkrules8 is [[[h4,h5,h6],[g3]]];
🟥 [13:38:46] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]],
              [[d2,e2,f2],[g3]]];
🟥 [13:42:59] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]],
              [[d2,e2,f2],[g3]],[[h2],[g4,g6]]];
🟦 [13:44:02] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]],
              [[d2,e2,f2],[g3]],[[h2],[g4,g6]],[[g32],[h4,h5,h6]]];
🟦 [13:44:56] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]],
              [[d2,e2,f2],[g3]],[[h2],[g4,g6]],[[g3],[h4,h5,h6]]];
🟦 [14:06:51] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]],
              [[d2,e2,f2],[g3]],[[h2],[g4,g6]],[[g3],[h4,h5,h6]]];
🟦 [14:21:21] checkrules8 is [[[h4,h5,h6],[g3]],[[b4,b5,b6],[c7]],[[d8,e8,f8],[c7]],
              [[d2,e2,f2],[g3]],[[h2],[g4,g6]],[[g3],[h4,h5,h6]]];
```

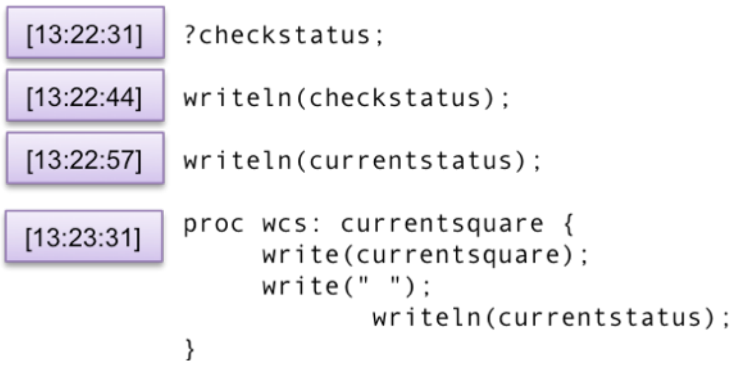
🟥 Denotes the definition goes to the public modelling space

🟦 Denotes the definition goes to the private modelling space

Figure 6.13 – Evolution of a set of check rules in the collaborative Sudoku model

```
proc wcs: currentsquare {
     write(currentsquare);
     write(" / ");
     write(currentstatus);
     write(" / ");
     writeln(current_region // current_row // current_column );
}
```

Figure 6.14 – A little helper agent 'wcs' used in the collaborative Sudoku modelling



```
[13:22:31]  ?checkstatus;
[13:22:44]  writeln(checkstatus);
[13:22:57]  writeln(currentstatus);
[13:23:31]  proc wcs: currentsquare {
                 write(currentsquare);
                 write(" ");
                        writeln(currentstatus);
            }
```

Figure 6.15 – Modeller S queried the 'checkstatus' observable before defining the initial version of the little helper agent 'wcs'

Despite the short time span of the collaborative Sudoku modelling, there is some evidence that the goal of the modelling process had evolved from "developing a collaboration strategy for tackling the puzzle" to "tackle the puzzle first!" The goal shifting was first observed when modeller *S* focused on tackling the puzzle rather than further development of the check rules[126]. He seemed to be enjoying the game play but the goal shifting (as well as role-shifting – from a developer of the model to a user of the model) became apparent when modeller *R* noticed that his colleague has changed his role, while he was struggling with making progress for a set of check rules for the number 9. Then, modeller *R* joined in the solving process, and at that point (1.5 hours after the collaborative modelling began), the shared goal became "tackle the puzzle first!"

Evolving collaboration support

There is evidence that the modellers not only co-constructed the check rules for the Sudoku model, but also co-evolved the collaboration support for the quest. For instance, modeller *S* made an agent (in the form of a triggered procedure) to show the coordinate of the square and the region where the mouse is located. A moment later, modeller *S* shared that agent with modeller *R* upon his request. Then, modeller *S* further developed the little agent '*wcs*' for the next 7 minutes (with input from modeller R) before he finalised it as shown in figure 6.14. This was all initiated by the conversation:

[126] It was the moment right after modeller R's workstation crashed.

```
Modeller S: have you got any cheap and nasty way to show whether the
            status has changed?

Modeller R: [delay a few seconds – because he was busy doing
            something] Err.. Whether what status has changed?

Modeller S: The … is it, is it checkstatus went to the main yea …
            [making a query “?checkstatus” to dtkeden-cm (cf. figure
            6.15)] enable the ‘1’s there … what sort of list of the
            list of ones?

Modeller R: Err… yea, write line on check status.

Modeller S: [entering “writeln(checkstatus);” to dtkeden-cm (cf.
            figure 6.15)] Humm … hu-er hu-er … [got “@” (undefined)
            from dtkeden-cm leaning over to richard’s screen] just
            check, just check …

Modeller R: ok, that? … current status.

Modeller S: current status.

Modeller R: Right do.

Modeller R: Umm…

Modeller S: Right, ok.

Modeller R: Can you pop in there … ?

Modeller S: What I’ll do is I’ll pop that in a bit with mouse over.
```

Indeed, the little helper agent (cf. figure 6.14) played an important role in the collaboration – it provided a shared reference for the modeller to communicate[127]. On another occasion, modeller *S* redefined all the agents behind the squares, so that any changes to the values of the squares would become a public redefinition, whether it is made in the public or private modelling space. This was in response to the need for better collaboration support. More importantly, this provides evidence that the instant feedback characteristic of EM can be easily converted into real-time collaboration support. In addition, such ad-hoc evolution might be difficult in the context of traditional software development, but requires relatively little effort when practising EM.

Collaboration and conflation of the contexts of development and use

As mentioned in chapter 4, the practice of EM potentially conflates the contexts of development and use. In the collaborative Sudoku modelling, it is hard to distinguish the

[127] In principle, it is possible that such a helper agent might evolve into an awareness support agent, which provides information about the current focus of attention of the other modeller.

context of use from the context of development, as both modellers are actively ‘using’ the Sudoku model like a normal player while thinking about how the check rules or other agents can be developed as described earlier.

Issues with the tool support

The collaborative Sudoku modelling also highlights several issues in the current tool support for collaborative modelling. Firstly, it seems that the tool itself is not stable enough for an extended collaborative modelling exercise. The server crashed twice and the clients were required to restart several times due to malfunctioning during the collaborative Sudoku modelling. Instead of the input panel provided by the tool support, both modellers used external text editors as their temporary workspaces. They believed that their approach would give extra reliability (i.e. to keep their script safe) and security (i.e. to reload the script in a timely way when the tool crashed). Secondly, the transition between private and public workspace is not without issues. For instance, modellers often need to interrogate the modelling environment (i.e. to make a series of queries concerning the same observable) to determine whether the definition of the observable is part of the model in the public modelling space or part of the model in the private modelling space. Besides, they often asked the other modeller to share his work in the public space. To some extent, this hinted that the blackboard mode might be better for synchronised collaboration due to its single shared space[128]. Thirdly, it is impossible to switch to other modes of interaction from the client side. The tools that I used in the collaborative modelling sessions (i.e. the d-Jugs modelling and the collaborative Sudoku modelling) implemented one mode of interaction at a time. For instance, dtkeden-blackboard is for blackboard mode, dtkeden-cm is a combination of the blackboard mode and the private mode. Nevertheless, the current architecture of the dtkeden requires the switching to be done on the server. At the moment, dynamic and client-determined modes of interaction are not possible without re-implementation of the tool.

[128] However, as discussed in chapter 2, people’s information sharing behaviour can be quite subtle. Hence, the single shared space may not work for all groups.

The difficulty in coordination

At the end of the collaborative modelling workshop, I had an informal discussion with the modellers about their feeling of the difficult situation they had faced in the collaborative Sudoku modelling. They believed that tackling a Sudoku puzzle, as well as developing a strategy for it, is a difficult problem situation because there is no easy way to partition the problem. This is reflected in the following conversation:

```
Modeller S: It's difficult with this one to know what you can … add to
            the model which is going to significantly to help you
            which is kind of worth the effort.

Modeller R: The problem with the check rules was that you could have …
            it was, it wasn't care whether number is correct. There's
            usually a problem, is it? Like you could place them, you
            could place how nine each of the numbers so they match
            themselves fairly easily. It's one of the check rules,
            kind of the crossover, that we're having … problem.

Modeller S: Being split into two, I am not sure that helps that much.
            Because I was saying like, I'd like to put them there.

Modeller R: Yeah.

Modeller R: Yea, that worked really well with the jugs, cause I was
            able to do the adding or changing easily, which was just
            come back together. But not Sudoku … [laughing] … No.

Modeller S: It's such a high degree of integration with all those
            things, isn't it …

Modeller R: Yea.

Modeller S: it makes it really difficult to do this.
```

Although there might be many possible partitioning strategies for solving the Sudoku puzzle whereby each person takes partial responsibility for solving the puzzle (either based on digits or regions on the grid, for instance), there is no doubt that the underlying dependencies between squares are complicated. To some extent, the coordination problem in collaborative Sudoku modelling highlights the difficulties in supporting highly-dependent collaborative modelling. While one may argue that such situations are better dealt with individually, the collaborative Sudoku modelling shed some light on how well an EM approach might work for difficult-to-coordinate problem situations.

6.4 Case study IV: the cricket project[129]

This section discusses the cricket project (Beynon, 1993). The primary focus of this case study is how the teams practised an EM approach without the recently developed collaboration tool support. Whereas the previous case studies looked closely at the interaction between modellers, the case study of the cricket project examines collaboration through an EM approach at the project level. While the coursework was aimed at practising EM in the early stages of a systems development life-cycle, this case study seeks to explain the significance of integrating different perspectives contributed by individuals within a systems development context. Therefore, the aim of the study is to determine how the student modellers made the EM model for cricket simulation through integrating different perspectives within a group.

6.4.1 The background of the project

The cricket project (Beynon, 1993) was a second year undergraduate software development project. It was the main piece of assessed coursework in the "Introduction to Software Engineering" module. The students were asked to build a cricket simulation system for a fictitious county cricket club using an agent-oriented approach (i.e. EM). There were 122 students in the class. The students were allocated into 11 groups of 11-person teams[130] based on their knowledge and skills in 9 areas[131]: i) Unix familiarity; ii) workstation experience; iii) programming skills; iv) mathematical ability; v) oral communication skills; vi) technical writing ability; vii) leadership qualities; viii) enthusiasm for the project; ix) knowledge of cricket. To begin with, the students were given a booklet on the cricket rules

[129] This case study is based on the presentation "The co-evolution of system and developer's understanding in practicing an Empirical Modelling approach to systems development: a case study on cricket simulation" in WPCCS 2008 (Chan 2008), and on the technical research report CS-RR-444 (Beynon and Chan 2009), with new ideas introduced. In particular, table 6.16, and figure 6.17 and 6.18 appeared in the presentation slides of WPCCS 2008 (Chan 2008), and table 6.19 appeared in the research report CS-RR-444.

[130] Team 9 was an exception, which had 12 students. In addition, there were visiting students in both team 3 and team 6, one in each team. The choice of an 11-person team was more or less influenced by the constitution of a cricket team. The purpose behind this arrangement was to make it feasible for each member of the team to play the role of a player agent within a cricket game and make internal observations as described in (Adzhiev et al. 1994a, Sun 1999).

[131] The students were asked to rate their knowledge and skills in 9 areas in a survey prior to project. The allocation, therefore, was based on the results of the survey.

(MCC, 1990), a sample score sheet, a sample match record and a simple EM model of a cricket match. The project lasted 10 weeks during the spring term in the academic year 1993/94. Each team had a third-year undergraduate student as tutor for the project. The tutors provided consultancy to the students and monitored their progress throughout the project. The students also had access to UNIX workstations, a version control management system (viz. SCCS), and the principal EM tool support – tkeden. However, there was no real-time collaboration tool support, as the distributed version of tkeden had not been developed. This configuration reveals how EM might be carried out in a larger team (compared to the teams in other case studies in this chapter) with asynchronous collaboration tool support. It also gives clues as to what might be needed in the tool support for asynchronous collaborative modelling.

Technically, the simple cricket model is made up out of *agents* and *observables*, and they are connected by *dependencies* (cf. §4.1). The significance of using dependencies in place of listeners has been discussed elsewhere (Heron, 2002). In some aspects, the modelling paradigm used in this modelling exercise is similar to event-driven modelling, in that agents are made to respond to changes in observables (cf. §4.3).

The simple cricket model ("urchin cricket") was constructed from a commentator perspective on a primitive form of cricket match, that is, as a SMMA model (in the sense as described in section 5.1.2). The course of a cricket simulation (when using the model) is, by and large, driven by human interaction with the model. That is to say, the modeller drives the simulation through playing the role of the agents as described in section 4.1. This is significantly different from most computer simulations, which often take parameters at the beginning, compute a sequence of scenes, and visualise or render the resulting computation. This simple model has various purposes in this project:

i) To serve as a means of developing a requirement specification through animation;

ii) To be an artifact for the students to experiment with in deepening their understanding;

iii) To be a basic prototype for the students to begin with.

The main tasks for each team were to carry out activities that are predominantly in the early stages of systems development, e.g. requirement elicitation, understanding the problem domain, and project planning and management. They were also asked to generate an agent-oriented specification of the cricket simulation and to construct a detailed model of cricket based on the given simple cricket model using an EM approach. Although the focus of the exercise was on the early stages rather than on a full lifecycle of systems development, each team still had to develop a quality assurance strategy to assure the quality of the artifacts they were producing. This included carrying out review meetings and the use of quality checklists (as in a typical systems development project).

During the course of the development, the students were encouraged to enrich the model through observation of a cricket game as it happens in the real world taking account of many different *modes of observation* (Adzhiev et al., 1994a; Ness et al., 1994). The project was not aimed at full-lifecycle software development; the primary focus of the project was on the activities in the early stage of the development. To some extent, the context of the cricket project resembles the preparation of a tender bid for a software project in the public domain.

6.4.2 Method of study and approach to data analysis

Since the project was carried out more than 15 years ago, it would be difficult, if not impossible, for the participants to recall their experience of the project through interviews or questionnaires. Consequently, the analysis in this case study was primarily document based. The analysis has consulted the following documents:

i) *Preliminary Team Report* (PTR) – The PTR was the first report that the students had to submit in the project. In most cases, it consists of a feasibility study of the problem, a project plan, the individual responsibilities, and the quality assurance mechanism that the team has adopted.

ii) *Preliminary Requirement Analysis Report* (PRA) – The PRA is most appropriately viewed as an interim report of the project. There were 2 versions of the PRA. The individual PRA usually contains the detailed cricket model and its rationale. The teams' PRAs are similar to the individual PRAs, but the perspective on the model was collective and integrated the common and good features from the individuals' models.

iii) *Final Team Report* (FTR) – The FTR was the last deliverable and the concluding report of the project. Most FTRs had documentation in the form of a project schedule, minutes, a preliminary design, an implementation plan, and the definitive scripts of the team's EM model of the cricket simulation.

iv) *Project Handouts* – These included the project guidance, the exercise sheets, and a couple of other documents that were given out during the project (Beynon, 1993).

v) Archives of emails and newsgroup discussions between the students and the module organiser.

Although the analysis in this case study was primarily based on documents, this should not have much impact on the quality of the case study. It is believed that the collaboration process was largely document-centric. This is to say, the collaboration process might be well captured and described, either directly or indirectly, by the documents such as project reports, meeting minutes, and the definitive scripts that they have generated in the process. This is because real-time collaboration EM tool support (e.g. dtkeden) was not available and therefore the students were forced to collaborate through other means such as face-to-face subgroup meetings (which in turn were reported in meeting minutes).

In order to develop a preliminary understanding of how the project was managed and how the teams approached to the project, I performed a preliminary analysis on the project handouts that were given out by the module organiser during the project and the archives of emails and newsgroup discussions between the students and the module organiser. The preliminary analysis did not provide conclusive results but provided essential information that was found useful in the later stage of the case study and has contributed to the discussion of the background of the project (cf. §6.4.1) and the discussion of this case study (cf. §6.4.3 and §6.4.4). These included:

i. Important milestones (i.e. dates) within the project.
ii. The composition and expertise distribution of the teams.

iii. What was the expected outcome of the project (e.g. what was the intended scope of the development process).

iv. The context of the development process (e.g. what tools were available and what were not)

Since the exploratory model construction process is the centre of the development process when practising an EM approach, the cricket simulation models become the primary object for the study analysis in this case study and the project reports are used as a supplementary resource. By glancing through all the team's reports, individual's reports, and the team's cricket simulation model that is embedded in the team's report, I found that most teams had come up with similar observations and similar structure within their models. They more or less followed Beynon's suggestions (Beynon, 1993) when practising EM and using tkeden to construct their models. However, larger variations were detected when I compared the individual models against each other and against the corresponding team's model. This motivated us to have a longitudinal study of the development process within one chosen team to study how they collaborated throughout the project in detail. By studying both individuals' and teams' models, various aspects of collaborative work that exploits an EM approach can be revealed, such as how their knowledge and personal interest influenced task allocation, their roles and the quality of their observations, how the team took advantage of diverse perspectives, and how the team resolved conflicting perspectives. This also helps us to understand the potential of an EM approach for collaborative work such as that involved in the early stages of systems development.

In the next subsection, I discuss in more detail how the analysis on the cricket simulation model construction for a particular team was carried out, and in particular, how their definitive scripts for the cricket model were analysed.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
Cricket Expert	-	-	-	-	Y	Y	-	-	-	Y	Y
Systems Analysis	-	Y	-	Y	-	-	Y	-	Y	-	-
Non-cricketer	-	-	Y	Y	-	-	Y	Y	Y	-	-
Technical Expert	-	-	Y	-	-	-	-	Y	Y	-	-
General Programmer	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Background Research	Y	-	Y	-	Y	V	Y	-	-	Y	-

Table 6.16 – The distribution of expertise in team 11 (extracted from the minutes attached in Team's 11 FTR).

6.4.3 Team 11's cricket simulation project

After the preliminary investigation of the team's model, I randomly picked a team – team 11 – for the detailed investigation of how the team collaborated throughout the project. According to their FTR, team 11 started their project with an introductory meeting on 19 January 1993. As in many commercial projects, the first task they decided to do was to find out their expertise and their area of interests in relation to the direction for investigation of the cricket simulation. Table 6.16 shows the distribution of their expertise. In a subsequent meeting, the team divided into four sub-groups to study aspects of the problem, such as "how do we simulate the cricket game?", and "how is the game actually played?" For this reason, each sub-group had been assigned a cricket expert for sub-group coaching and learning about the situation to be modelled.

After some brief study of the problem context, the team re-formed the sub-groups in relation to particular areas of concern in the cricket game – there were sub-groups for non-technical issues relating to cricket as the subject domain such as observing i) the bowlers and the umpires, ii) batsmen, iii) fielders, and sub-groups for technical issues such as score-keeping[132] and graphical simulation. As the deadline for submitting preliminary reports approached, the team re-formed again and focused on the technical issues rather than the modelling of the cricket game (what they regarding as the "control issue").

[132] Score-keeping can be a technical issue, and a non-technical issue. On the one hand, when it is treated as a technical issue, how the agent that presents the 'data' in a tabular format as in the scoreboard or a score-sheet is 'implemented' becomes significant. On the other hand, when it is viewed as a non-technical issue, determining what real-world agent awards the score and analysing the observational protocol they follow become a significant feature in the modelling context.

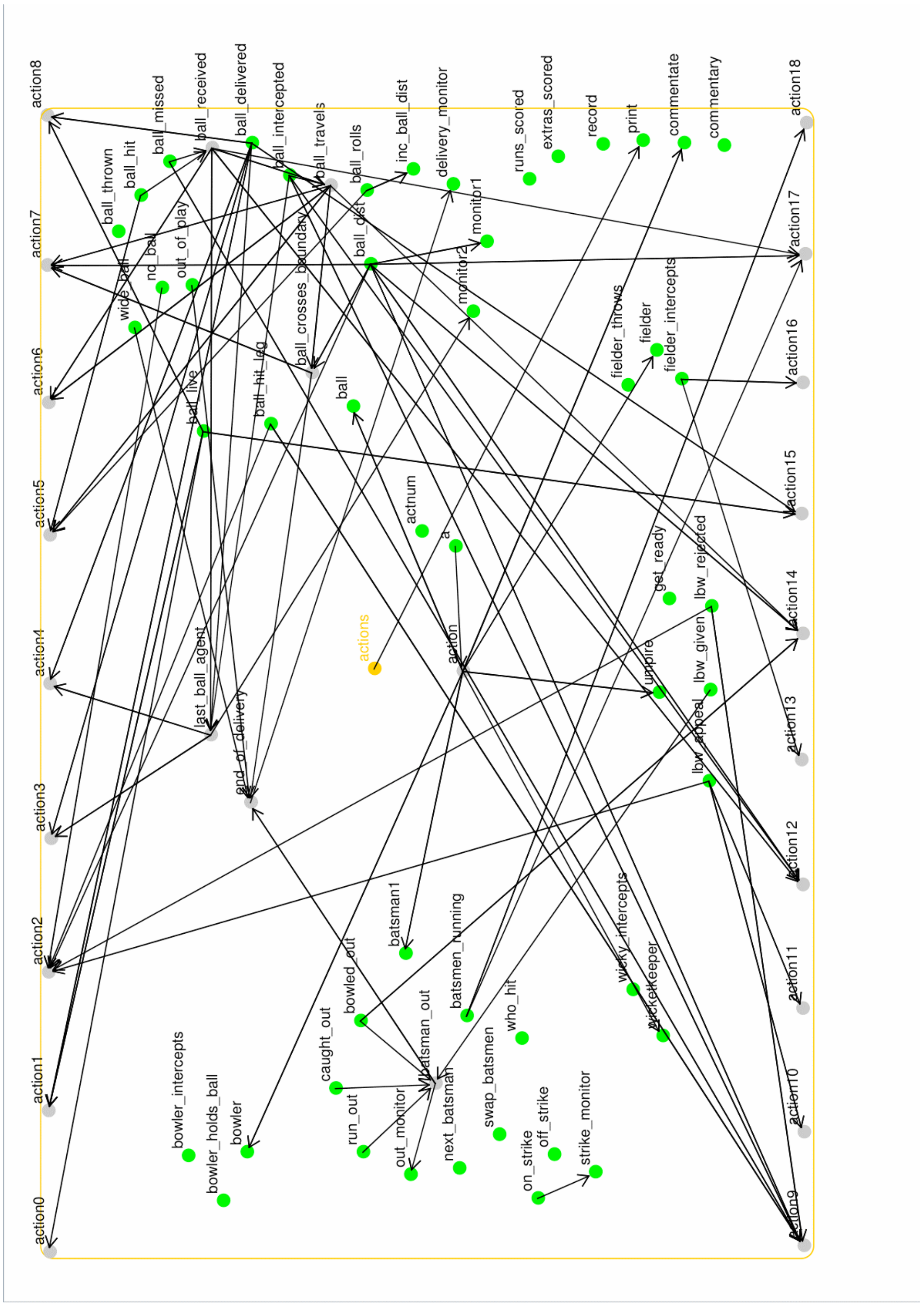


Figure 6.17 – The dependency graph for the team model of Team 11[133]

[133] The dependency graph is plotted using the Dependency Maintenance Tool (DMT) developed by Wong (2003). The green dots correspond to the observables in the model, and the grey dots correspond to the actions from which

To understand how the agents are connected within the team's model, I have plotted a dependency graph (cf. figure 6.17) based on the key observables and agents in the model using the Dependency Maintenance Tool (DMT) developed by Wong (2003). In the dependency graph, relevant observables were placed close together. The green dots correspond to the observables in the model, and the grey dots correspond to the actions from which a commentator agent can select. The orange border corresponds to the current scope of the model as defined by the total set of observables and agent actions currently modelled. The 19 actions (viz. action0 – action18) which are located on the edge of the orange boundary are the most important "key observables". These correspond to the definitions that shape how the cricket simulation can be animated. In EM terms, these definitions in fact manifest *dependencies*. For instance, action0 is defined as:

```
action0 is [ "Umpire says ready for play", 0,!ball_live ];
```

which denotes that action0 can be performed when the ball is not live. The dependency graph has provided crucial clues for understanding how a cricket game might be simulated through the team's model. The dependency graph analysis has also revealed that the team has pretty much followed Beynon's (1993) suggestion (cf. §6.4.1) of observing, modelling, and simulating a cricket game from a commentator perspective (or as an external observer of a cricket game). During the simulation of a cricket game with the EM model, the modeller plays the role of the commentator agent in the model. In order to progress the simulation, the modeller would have to make state changes to the observable "action" (as depicted in the middle of figure 6.17, below the observable "actions" in orange colour). Note that it would be misleading to think of the cricket model as a "product" in the sense of traditional systems development. At no stage is there a preconceived fixed interaction pattern with the EM model (cf. chapter 4). As I discuss in the next section, the modellers of the cricket project not only co-constructed the simulation, but also a shared understanding of the game.

---

a commentator agent can select. There are 19 actions in total (0-19) which are located on the circumference of the diagram. The orange border corresponds to the current scope of the model as defined by the total set of observables and agent actions currently modelled.

As I discussed in chapter 5, one of the key concerns in collaborative modelling (at the collaboration degree of engagement) is to facilitate diverse perspectives. Consequently, to answer the question “how did the team collaborate?” (the aim of this case study), we need to know how they facilitated diverse perspectives. To some extent, individual perspectives on the cricket simulation are embedded in the individual models (in the sense of EM construals; cf. chapter 4). Therefore, a more practical question would be “how did they negotiate the team’s model as an integration of individual models?” However, apart from the work break down captured in the minutes, there was no discussion about how the team integrated individual models – it remains a mystery in the project documentation. For this reason, I analysed the differences among the team’s model and the individuals’ models. This analysis was informed by the grounded theory (Strauss, 1987; Strauss and Corbin, 1998). The coding strategy used in this analysis was similar to the one that is advocated in grounded theory. During the initial coding stage (i.e. open coding), the clustering in the dependency graph (cf. figure 6.17) was used to inform (but not restrict) the coding. Observables and definitions in the team’s model that can be derived from the observations of different aspects of a cricket game were identified. For instance, the observables *‘lbw_appeal’*, *‘lbw_given’*, and *‘lbw_rejected’* can be classified as “Leg Before Wicket (LBW) and its appeal”. Likewise, the definitions

```
action2 is [ "Fielder appeals for LBW" , 2, !lbw_appeal && ball_dist < 1
&& ball_hit_leg && !no_ball && !lbw_rejected];
action10 is [ "Umpire says batsman out for LBW", 10, lbw_appeal ];
action11 is [ "Umpire says batsman not out for LBW" , 11, lbw_appeal ];
```

fall into the same category. After the coding of the team’s model, I used the same coding scheme for the individuals’ models. However, the original coding scheme does not reflect the subtle differences between the individuals’ models and the team’s model. In the later stage of the coding (i.e. axial coding), the coding categories co-evolved with the analysis as I “played” with the data further (cf. Yin, 2009). For instance, the LBW was later split into two categories, namely, “considered LBW”, and “considered LBW and appeal to umpire”. The vital difference between these two categories is that the former merely captured the act of giving the batsmen out LBW, and the latter captured the LBW appeal process. This difference also highlights the quality of observation that is captured in the model. For a model to capture the LBW appeal process, the model must have captured the LBW situation. Therefore, I reviewed the observables and definitions that were codified in each category. Thirty-nine (39) categories were grouped into thirteen (13) scenarios. The final coding is depicted in figure 6.18. The definitive scripts of the teams' model (on the leftmost in the figure) and the individual models were listed side by side during the coding. Different parts (key observables and definitions in the definitive scripts) of the models were tagged in different colours according to the categories in the evolving coding scheme.

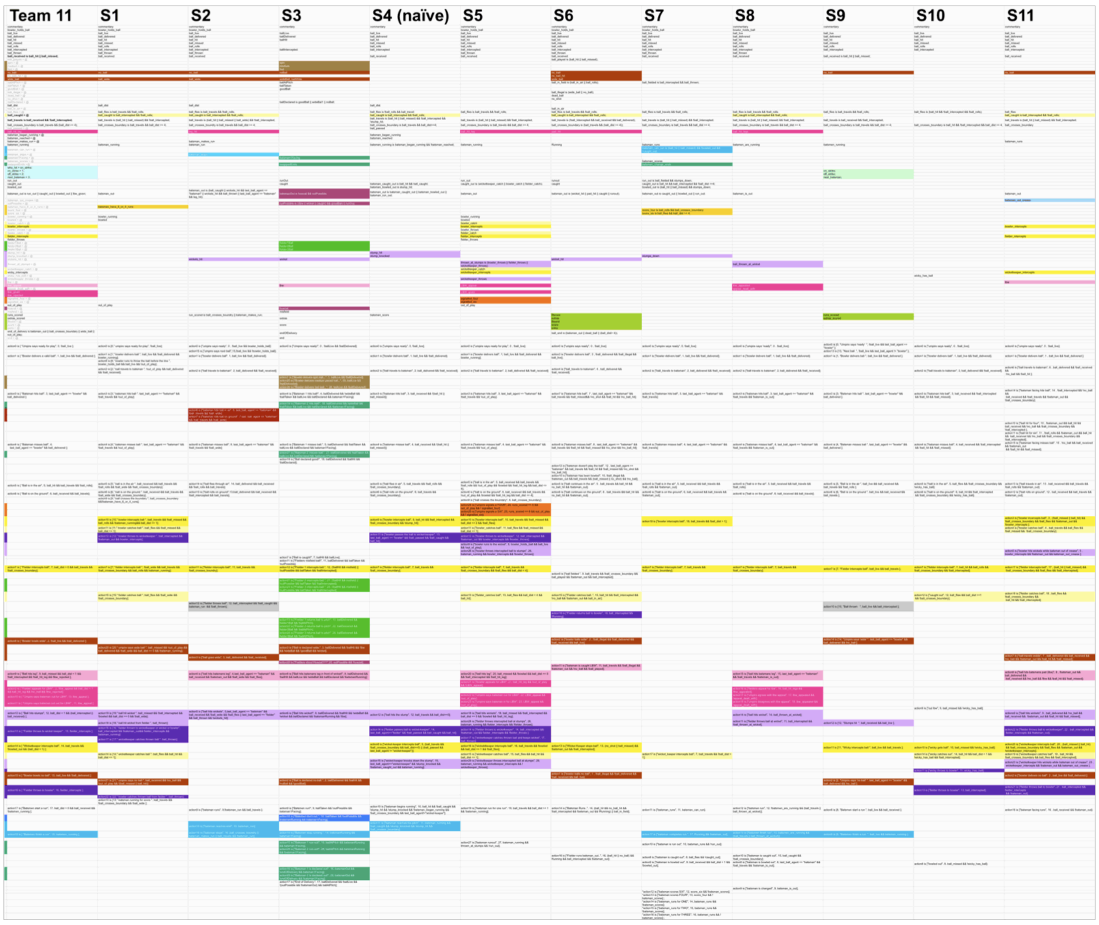


Figure 6.18 – Colour coded team's model and individuals' models listed side by side[134]

[134] The text (i.e. definitions in the models) in this diagram is not intended to be readable. The purpose of this diagram is to show how the analysis was carried out.

Scenarios	Team	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
a. How the ball is delivered	-	-	-	1	-	-	-	-	-	-	-	-
b. Abnormal deliveries	1	2*	/	2*	-	-	2	-	-	2*	-	1
c. How the ball is hit	-	-	1	-	-	-	-	-	-	-	-	-
d. How batsmen run after hitting or missing the ball	2	-	1	2	1	-	-	1	1	1	-	-
e. LBW and LBW appeals	2	-	1	1*	-	2	1	-	2	-	-	1
f. How batsmen are dismissed	-	-	-	1	-	-	-	-	-	-	-	-
g. How and by whom the wicket was broken	/	2	2	1	2	4	/	/	1	1	-	3
h. How the ball is intercepted	2	3	1	1	3	3	2	2	1	2	2	3
i. How the ball is caught	1d	3	1d	/	3d	3	2	2d	d	2p	/	3d
j. Modelling the transition between balls	1	3*	/	-	2	2*	1	-	-	/	2	2
k. Multiple batsmen and fielders	2	-	-	3	-	-	-	2	-	2	-	-
l. Scoring when the ball crosses boundary	1	-	1	-	1	1	1	2	/	1	/	-
m. Keeping scores	-	1	1	1	1	1	2	1	-	1	1	-

Table 6.19 – Contribution by the members in Team 11

After the coding, the next step in the analysis was to identify and to evaluate individual modeller's contribution to the team model. As shown in table 6.19, individual models were awarded with a numerical rating according to the quality of observation they captured in respect of each scenario (i.e. how much detail their model captured from the cricket simulation situation). These numerical ratings also reflect the degree of attention that individual modellers have given to those scenarios. Thus, a higher rating signifies a greater degree of attention (i.e. more details from the scenario have been captured), and '-' denotes that no attention was given to the scenario at all. For instance, the models that have merely captured the LBW situation have a rating of 1, while the models that have captured both the LBW situation and the LBW appeals would have a rating of 2 (cf. scenario (e) in table 6.19). In another example, '1' has been awarded to S2, S3, and S8 because their model only captured one possibility: the ball is retrieved by the fielders, while '2' has been awarded to the team, S6, S7, S9 and S10 because their model has captured two possibilities: retrieval by the fielders or by the wicketkeeper. In addition, the symbol '/' refers to a more passive approach in modelling the situation in which the fact that they had given attention to a scenario might almost go unnoticed. For example, in the team's model, no observable or definitions were found to be directly related to how and by whom the wicket was broken (i.e.. scenario (g)). The definition:

```
action12 is [ "Ball hits stumps", 12, ball_dist < 1 && (ball_intercepted
|| ball_received) ];
```

models the scenario more passively (in terms of observable, dependency, agent and agencies in EM) when compare to S5's model, in which observables are linked up by dependencies:

```
thrown_at_stumps is (bowler_throws || fielder_throws ||
wicketkeeper_throws);
action26 is ["bowler throws intercepted ball to stumps", 26,
batsman_running && bowler_intercepts && !bowler_throws];
action27 is ["batsman runout", 27, batsman_running && thrown_at_stumps
&& !run_out];
action28 is ["fielder throws intercepted ball at stumps", 28,
batsman_running && fielder_intercepts && !fielder_throws];
action29 is ["wicketkeeper throws intercepted ball at stumps", 29,
batsman_running && wicketkeeper_intercepts && ! wicketkeeper_throws];
```

Moreover, the same scenario can be modelled in many different ways (e.g. by observables complemented with triggered actions, or solely by a set of dependencies). To distinguish different uses of definitions, letters are appended to the numerical ratings in table 6.19 – 'd' indicates that the scenario was modelled without the use of any procedural action, and 'p' indicates the use of procedural constructs. Unusual contributions are denoted with asterisks. For instance, in modelling the transition between one ball and the next, S1 not only considered the throwing of the cricket ball between cricket players (viz. fielder, wicketkeeper, and bowler), but also the catching of the cricket ball by the cricket players. By comparing the team's and individuals' model side-by-side, similarities between them can also be identified. Contributions were attributed to an individual model if the parts were close in semantics. For example, the observable *'ball_hit_legs'* introduced by S8 is viewed as significant, as the observable *'ball_hit_leg'* is a contribution to *'ball_hit_leg'* in the team's model in the aspect of the LBW appeal process (cf. scenario (e) in table 6.19). In most cases, significant individual contributions can easily be identified because they are taken into the team's model without modifications. In some scenarios, however, it is difficult to tell who is the main contributor – the team simply integrated the common perspective that was shared by the majority of individuals (e.g. scenario). In table 6.19, shaded squares mark the individual perspectives that were adopted in the team's perspective.

This analysis seems to suggest that the team's model was not integrated by one person[135], but through negotiation of diverse perspectives embedded in the individual models. It also suggests that the collaboration between the students was close to *genuine participation* in the participatory development model that I described in section 3.4. In such a development process, participants gain insight about the artifact (i.e. the simulation of the cricket game with an EM model), as well as contributing freely with different perspectives and with different knowledge. In the next subsection, I discuss the results of this analysis in relation to the EM approach.

6.4.4 Limitations

There are at least three research limitations in the cricket case study. The first limitation comes from the data collection technique. As in the VEL case study (cf. §6.1), the cricket case study was mainly based on archived documents. While the documents are readily available to be reviewed and analysed, they can only be trusted to a limited extent due to the unknown bias of the documents' authors. This is to say, the documents might have been drafted to give an illusion that the model were developed in the way that they described (cf. Duffy, 2005). For instance, it could be the case that the documents were written to give an impression that the project was a collaborative effort when in fact it was not. However, it would be difficult for the students to perpetrate this deception due to the structure of the cricket project: the students had to submit both individual and team reports at various stages of the project (cf. §6.4.1).

The second limitation comes from the fact that only the final definitive scripts were available. That is, no intermediate definitive scripts were available for further analysis. Despite the fact that there are traces of the development process in their meeting minutes and individual reports, it is difficult to be sure that the definitive scripts of the cricket simulation models were not created at the last minute.

The third limitation comes from the coding techniques used in the analysis of the definitive

[135] For example, by a chief software architect in the traditional systems development.

script. There can be endless options for coding and comparing the same set of data (Flick 1998). Where the quality of observation in the model construction is concerned, it seems natural to code according to the aspects in a cricket match that the cricket model captures, rather than to the features and functions it implements as in a cricket simulation program. Certainly, this does not preclude the possibility of coding against features and functions in future studies.

6.4.5 Discussion

Neither the project documentation nor the models made this project interesting; it was the collaboration process behind the construction of these artifacts that made this case study interesting. The lack of synchronous tool support in the cricket project had forced the modellers to collaborate asynchronously most of the time, and this makes it stand out from the other case studies in this chapter. In the rest of this sub-section, the virtues of an EM approach to collaborative modelling are examined by drawing on the analysis of Team 11's cricket project, and with reference to figure 6.17, 6.18, and table 6.19.

Facilitating and integrating diverse perspectives

As Beynon pointed out to the students at the beginning of the cricket simulation project, EM should not be used as a hacking tool for 'quick and dirty' prototyping, but as an *instrument* (Beynon et al., 2000) to facilitate the observation that is the analogue of doing experiments in other scientific disciplines (Beynon and Russ, 2008) throughout the modelling process:

> *"Its [EM approach] focus is not upon 'hacking a program till it works', but upon thoughtfully correlating the behaviour of variables in the computer model with the real-world observations they represent. In spirit, this can be regarded as following the pattern of the experimental method in science, and should be subject to the same discipline." (Beynon, 1993)*

As I discussed in chapter 4, the EM artifacts are construals that are based on modeller's subjective and personal experience of the situation being observed. As was the case in team 11, different modellers (i.e. members of the team) adopted different modes of observation of the relevant situations (i.e. scenarios in a cricket game) at different times. For instance, a modeller may begin with external observation of the cricket ball, and trace a sequence of states that follows the trajectory of the ball, to see what kind of state changes the ball may cause in other agents as in a cricket match in the real world. In other times, modellers may make observation from the perspective of an internal agent, e.g. to see what types of state changes the umpire (i.e. an agent) can make in the context of a cricket game.

Different modes of observation may result in different degrees of attention to the details within the context of the cricket game (i.e. the problem domain of the simulation), and that leads to diverse perspectives on the same problem domain. Our analysis of the team's individual models is in line with this claim. As shown in table 6.19, individuals have different ratings for different scenarios. This reflected the fact that different degrees of attention were given to different aspects in the problem domain, even by the same individual modeller.

Despite the potential they offer for a richer understanding of the problem domain, such diverse perspectives posed a challenge in integrating the individual models. The analysis of all teams' FTR seemed to indicate that, in most cases, the team chose to address this issue by selecting one or the other, instead of venturing to introduce dependencies to reconcile variants of the same event or situation. This is corroborated by the fact that numerous individual contributions have been discarded (e.g. scenario (b), (g), (h), (i), (j), (k), and (l) in table 6.19)[136], and contributions in some scenarios were entirely ignored (e.g. (a), (c), (f), (m)). It may be that the issues in the integration of diverse perspectives stem from potential conflicts of scenarios or subtle group dynamics, but this cannot be fully verified in either case. Despite these integration issues, allowing diverse perspectives potentially offers a richer understanding of the problem domain and a higher degree of individual engagement, and this makes collaboration closer to *genuine participation* (cf. §2.5.2 and §3.4).

Learning and co-construction of the problem situation

Our analysis seems to indicate that in some scenarios students with less sophisticated knowledge of cricket were the more successful in identifying the most appropriate way to

[136] For example, the team's model does not integrate all contributions from S6's individual model for scenario (b), and does not consider S5's or S11's modelling for scenario (g).

model the possible actions of agents in a cricket game, based purely on their observation and immediate experience. For example, S1 and S9 showed a higher degree of attention to abnormal deliveries of the cricket ball (i.e. scenario b), whilst S3 and S4 had the most detailed model for scenarios (d) and (h, i) respectively and these were subsequently taken into the team's model (cf. table 6.16 and table 6.19). One explanation is that the novice cricketer exploited the EM process for a learning purpose. This idea has been explored in *constructivist computing* (Harfield, 2008; Beynon, 2008). Another explanation is that the students with more experience of cricket found it harder to analyse their 'intuitive' knowledge but felt they would be able to retrieve this as necessary. For instance, as one such student said:

> *"I decided to give a minimilistic [sic.] simulation just to show I understand how to program [model] the requirements. The design can be developed quite easily into whichever level [is] wanted." (S11's PTA)*

As mentioned in chapter 5, *co-construction* in collaborative modelling involves the co-construction of shared understanding of the shared activity and of the problem situation. Hence, if we view the development of the simulation of a cricket game as a collaborative modelling process, the co-construction of the team's model would be the shared activity, and the co-construction of the understanding of how to simulate a cricket game would be the problem situation. In this sense, the learning through the construction of the individual cricket model seems to have played a crucial role in developing a shared understanding of the problem situation. It enables individual understandings of the problem situation (i.e. embedded in the individual models) to become a shared understanding (i.e. embedded in the team's model) through negotiating diverse perspectives into a unified shared perspective (i.e. the integration process).

6.5 Concluding Remarks[137]

Even though research into EM has progressed tremendously over the past two decades, there have been very limited studies and documentation of the EM process through research techniques that are common in social science experimental practice. In particular, there have been virtually no attempts at gathering data about modeller interaction in collaborative modelling with an EM approach through observational techniques such as video recording and participant observation. At present, there is insufficient expertise in EM available to carry out larger scale of observational studies in collaborative EM that involves many modellers. It would therefore be difficult at this stage to attempt rigorous experimental analysis of the practice of EM using the kind of experimental studies that we would expect in investigating a well-established practice.

The case studies presented in this chapter represented some of the very first attempts at ethnographical studies of the EM process. From a holistic perspective, these case studies have the following characteristics:

i. Relying on the analysis of the archived documentation of collaborative projects that had practised an EM approach without access to the participants in those projects (viz. the VEL case study discussed in §6.1 and the cricket project case study discussed in §6.3).

ii. Documentation of a collaborative modelling process that practised an EM approach from a first-person experience[138] (viz. the first distributed jugs discussed in §6.2).

iii. Investigating the distinctive characteristics of EM through multiple data sources. For instance, the examination of the interaction history in conjunction with other data sources, such as video footage, screen video capture, and archived documentation.

[137] This section highlights some backgrounds and limitations concerning the empirical study of collaborative modelling with an EM approach using observational research techniques that are common in social science experimental practice. This is a topical issue which is fruitful for further research into EM. The discussion in this section is a result of intensive discussion between myself and my thesis supervisor, Dr Meurig Beynon, which was based on the thesis examination committee's comments on this topic.

[138] Due to the fact that the modeller's experience is one of the primary focuses in EM (cf. Beynon 2009), it is appropriate to include the description of the engagement in EM from a first-person perspective, no matter how difficult it might be to reconcile this with the need for objectivity in the results of the case studies.

Apart from the collaborative projects presented in the case studies, many hundred students have practised EM over the last two decades. Their reported experience seems to indicate that modelling with definitive scripts in EM has some distinctive characteristics that requires more detailed investigation in future research projects. For instance:

1. The interaction history in EM is an unusual resource that offers insight into the progression of the model-construction process. As the analysis in the case studies have demonstrated, the interaction history can be a fruitful resource for the reconstruction and investigation of the entire modelling process. Such use of interaction history in reconstructing part of the modelling process of Care's planimeter modelling can be found in Harfield's (2009) doctoral thesis. In fact, the interaction history is not only useful for the researchers who are interested in the modelling process, but also for the modeller him/herself. This is due to the fact that the interaction history, to some extent, can be used to 'revisit' the *state-as-experienced* as encountered by the modeller stage-by-stage in the modelling process. This potentially allows the modeller to retro-fix certain problems in the model – a feature which is not typically supported in the conventional systems development paradigm.

2. The role-shifting behaviour discussed in the case studies is itself "matter-of-factly" exploited by many EM practitioners. The modes of interaction underlying "interacting as a developer" and "interaction as a user" are virtually the same. The sole difference between these two styles lies in the perspective that the modeller has adopted to support that interaction and its interpretation. For example, "interacting as a user" typically refers to the fact that the modeller, in contrast to "interacting as a developer", is presuming little knowledge and experience of the observables that can help to frame the meaning of redefinitions. This highlights some logistic issues which may benefit from further empirical studies. For instance, "in what context does this role-shifting behaviour occur?", "how consciously does this role-shifting behaviour occur?", "how does this role-shifting affect the collaborative modelling process when practising EM?"

Chapter 7 Efficacious groupware development through Empirical Modelling

In the previous chapters, I pointed out the difficulties in developing groupware efficaciously and the issues surrounding the notion of participation in groupware development. In chapter 2, I argue that groupware must be able to evolve along with the work practices and human activities that it supports (cf. §2.4). Unfortunately, groupware is not trivial to develop due to its socio-technical nature. Even worse, the development of groupware is highly situated due to the fact that the group dynamics in each group are unlikely to be the same: this poses serious challenges to groupware developers (cf. §2.2). Consequently, groupware development often incorporates ethnographical and participatory design (PD) techniques in order to understand how people actually work in a particular situation. To increase the likelihood of groupware acceptance and to ease the adoption of groupware, user participation[139] has been thought as a remedy. However, the user involvement in the development process of any systems development, including groupware development, can sometimes be a tricky business (cf. 2.5.2). As Gasson (2003) observed, the nature of the development – its technological-oriented[140] focus – may remain the same even when practising a PD approach. In this respect, the culture in groupware development remains more or less user-centred or group-centred, but is not human-centred in the sense that I

[139] The terms “user” and “user participation” are far from perfect. As I have argued in chapter 3 (by citing various sources), the choice of these words hints at a technological focus. It may also mislead the reader into supposing that a “user” is technology-blind and has no technical competence.

[140] The technology-oriented focus can also be understood as a developer-oriented (or developer-driven) approach to systems development.

have elaborated earlier in this thesis (cf. §2.5.1 and §3.2).

As I have pointed out in chapter 3, the main issue behind participatory approaches is how we view the conception of roles and participation in the development process. If we set hard boundaries for the roles that the participants may play in the development process, any role-shifting behaviour becomes irrational. This not only affects the potential actions that the participants may carry out in the process, but also potentially inhibits advances in research into human-centred development. Even though this issue arises in contexts other than groupware development, it is the main obstacle to moving towards *efficacious groupware development* – a holistic perspective on groupware development (and evolution) that I set out in chapter 2 (cf. §2.5). Consequently, I argue for an alternative conceptual framework that may potentially be better aligned with the vision for efficacious groupware development.

The idea of practising an EM approach in the context of groupware development was inspired by various potential applications of EM, namely, requirements engineering (Sun, 1999), business process reengineering (Chen, 2001), financial sector (Maad 2002), and ubiquitous systems development (Wong, 2003). Throughout chapter 4, 5 and 6, I have discussed Empirical Modelling (EM) and examined its potential for collaborative modelling. So far in this thesis, the discussion of EM has been not directly connected with groupware development and the issue of participation. In this chapter, I argue that EM potentially facilitates efficacious groupware development.

In section 7.1, I argue that the core activity of groupware development can be thought of as a collaborative modelling process. This is to say, a shared understanding of the group activity is co-constructed through the construction of a model, and that model (which consists of shared pieces and private pieces) will gradually evolve into a stable artifact that can be regarded as a groupware. In section 7.2, I examine how EM may address the issues in participatory development (cf. chapter 3) which present the main obstacle in moving towards efficacious groupware development. Based on this perspective, I synthesize a conceptual framework – GroupPIE – for an EM approach to efficacious groupware development in section 7.3. Like other human-centred approaches, this framework is participatory. The framework facilitates participants with different backgrounds, expertise and interests to co-construct and co-evolve the groupware and to co-evolve their understanding within a shared information space during the lifespan of the groupware from conception to disposal. In section 7.4 I discuss the limitations of the GroupPIE framework. In section 7.5, I argue that such a framework potentially moves towards “efficacious groupware development”.

7.1 Groupware development as collaborative modelling

In chapter 5, I defined *collaborative modelling* (CM) as *the collaborative process of constructing an interactive computer-based model of a software system*. Viewed from the CM perspective, the core activity in groupware development is a two-fold collaborative modelling process: co-constructing a shared understanding of the group activity through the co-construction of a model consisting of shared pieces and private pieces that will gradually evolve into a stable artifact that can be regarded as groupware. In fact, such embodiment of the shared understanding of the group activity in the development process of groupware is consonant with Naur’s (Naur, 1985a) account of the co-evolution between the developer’s understanding and the construction of a program (cf. §3.1).

As with the *collaborative modelling* I described in chapter 5, the development process of groupware does not stay within the conceptual scope of one particular form of collaboration. Its construction involves different degrees of engagement, different relationships between the modeller and the agents, different modes of interaction in different situations and at different times. In relation to the construction of “models”, the groupware development process generates two “models”: i) a *conceptual model* of the group activity (and work practices), and ii) a *computer-based model* of the groupware. In this way, the construction of the latter model would lead the stakeholders to a better understanding of the group activity (i.e. the former model) and of how the groupware may fit into the context of that group activity.

It is worth noting that the groupware development as collaborative modelling (GD-as-CM) is a holistic view of the collaboration process in groupware development. It is not associated with, nor does it preclude, any particular form of participation. With this orientation, it is plausible to practise an EM approach to groupware development by practising an EM approach to collaborative modelling (cf. chapter 5). In the next section, based on the discussion in section 2.5 and with EM in mind, I sketch a plausible process model for efficacious groupware development.

7.2 EM for human-centred development

In section 2.5, I argued that a human-centred focus is a pre-requisite for efficacious groupware development. In this section, I revisit human-centred development (HCD) through the lens of Empirical Modelling (EM) – an alternative approach to computing – that I introduced and discussed throughout chapter 4, 5 and 6. As mentioned in section 2.5, user participation is seen as a remedy for reducing the tensions among all stakeholders, particularly between the technical and the non-technical participants in the development, and as a prerequisite for establishing a suitable context for practising HCD. With this goal of HCD in mind, EM seems to offer a condition for such a high degree of participation to take place (cf. §7.2.1). In fact, there are many common characteristics between EM and PD (cf. §8.2), e.g. how they handle changes, diverse perspectives, etc throughout a systems development project. Apart from the potential for genuine participation, EM also has the potential to address three other HCD concerns which are particularly relevant in the context of groupware development: i) integrating the human and the technological (cf. §7.2.2), ii) enabling flexibility and evolution (cf. §7.2.3), iii) supporting diverse interaction and communication (cf. §7.2.4)[141].

[141] Beynon and Chan (2006, 2009) had already explored these three concerns in some detail in the context of Distributed Participatory Design (DPD). The discussion here is more oriented to the context of groupware development.

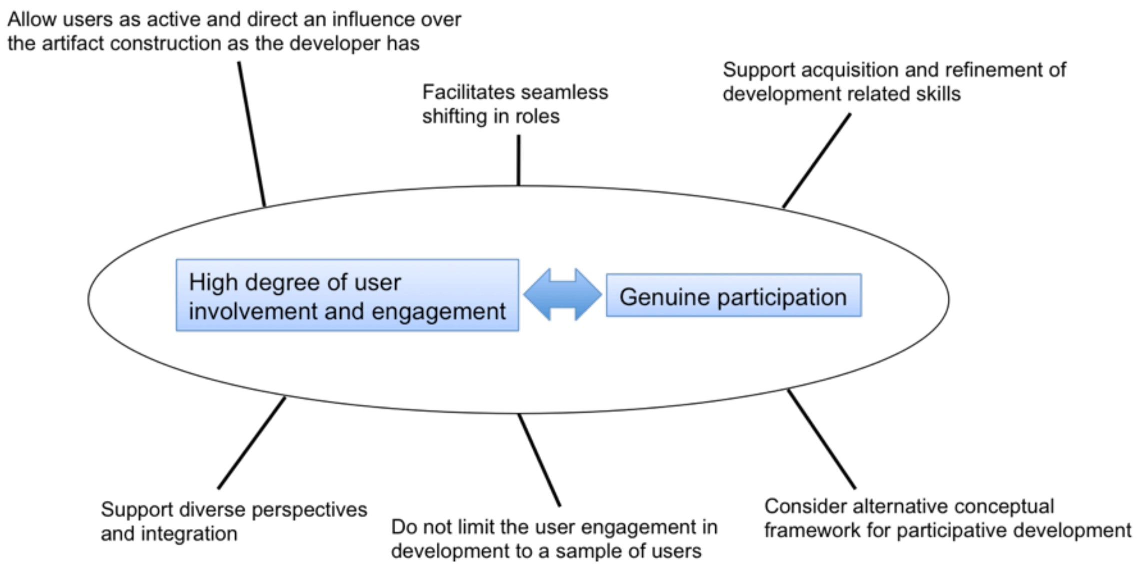

Allow users as active and direct an influence over the artifact construction as the developer has
Facilitates seamless shifting in roles
Support acquisition and refinement of development related skills
High degree of user involvement and engagement
Genuine participation
Support diverse perspectives and integration
Do not limit the user engagement in development to a sample of users
Consider alternative conceptual framework for participative development

Figure 7.1 – Factors surrounding participatory development

7.2.1 EM for genuine participation

Earlier in this thesis, I made seven suggestions to address the participation problem in human-centred groupware development (cf. §3.5). As figure 7.1 illustrates, the intention behind these suggestions is to find a balance between a high degree of user involvement and engagement and the notion of *genuine participation* for the group of practitioners who will eventually use the groupware. In the following, I argue that EM potentially addresses most of these suggestions and is therefore potentially well-suited for human-centred groupware development.

Consider an alternative conceptual framework for participatory development

As I discussed throughout chapter 4, EM is fundamentally different from the traditional systems development approach (e.g. such as is described in Sommerville 2007). EM promotes modelling through observation, experimentation, and immediate experience. In relation to model construction, the EM process can be thought of as construing through identifying observables, dependencies, agents, and agencies (ODA). In contrast to a traditional systems development approach, EM emphasizes the construction of an understanding of a situation (and the activity) through and throughout the construction of the model. Due to this focus in the modelling process, an EM model (or construal) should not be considered as a complete end product in the traditional sense of systems development.

Allow users as active and direct an influence over the artifact construction as the developer has

As I argued in chapter 2 and 3, groupware may be resisted if its use disrupts the normal operation of the group. If a result of groupware development is a system that is based heavily on what the developers (including the ethnographers) can observe and interpret, but not on the future users' perspectives, the future users may be reluctant to adopt and adapt to the groupware. To make groupware and its users co-adapt to each other efficiently and effectively, it is important that the users of the developing groupware have as active and direct an influence as the developers.

Modelling with definitive scripts in EM potentially addresses this concern by blurring the boundaries of development and use, and the roles of developers and users. As I discussed in chapter 4, the semantic meaning of a (re)definition to the EM model is entirely at the modeller's discretion (also cf. Beynon, 2008). Take the jugs model (EMPA:jugsBeynon2008) as an example. There may be no syntactic difference between two semantically different redefinitions. For instance, the definition “contentA = 0” could be the result of a user's interaction “emptying jugA”, but it could also be a developer's action “initialising a condition for jugA”. Both developer and non-developers share a common interface: they can both make and trigger the same redefinition through the input window and through interaction with the “buttons” (i.e. the SCOUT windows) on the screen (cf. figure 4.7). That is to say, the state-changing effects within the model associated with these two different species of interaction (‘development’ and ‘use’) are expressed in one and the same space[142]. This affords an unusually literal meaning to the idea that the developer and the user interact in a shared modelling space when practising an EM approach to collaborative modelling (cf. chapter 5 and 6).

Taking all the above into account, EM not only blurs the boundary between developers and users, but also facilitates collaboration between the developers and the users in shaping the patterns of interaction among agents within the model. In other words, EM potentially

[142] Contrast the distinction between editing a line of code in a procedural program and executing the recipe that the programming code prescribes.

facilitates active and direct influence by the users of the groupware.

Support seamless shifting of roles

There is evidence that participants in two of the collaborative modelling case studies shifted their roles (viz. learner, user, developer, demonstrator/teacher) seamlessly from time to time during the collaborative modelling process (cf. §6.2 and §6.3). According to the video footage and the interaction history (i.e. part of the definitive scripts), this seamlessness of change may potentially be due to the instant feedback of the EM tool support – which is also a characteristic of EM (cf. §4.3). The role shifting phenomena almost went unnoticed both by the researcher and the modellers themselves (i.e. the participants in the case studies) because the modellers have to *do* virtually nothing in order to shift their roles in the collaborative modelling process.

In fact, such seamless shifting in roles may be viewed as a conflation of roles such as I discussed in chapter 3 (cf. figure 3.4 on page 90 and §3.4.3) if we step back and view the modeller as a human being who is interacting with the model with her native knowledge and skills rather than playing preconceived roles in the modelling (or development) process (cf. §3.2 and §3.4).

Support acquisition and refinement of development related skills

In the collaborative modelling case studies (cf. §6.2, §6.3, and §6.4), there are indications that the modellers learnt from each other through collaborative EM (i.e. collaborative modelling with an EM approach). For instance, modeller *E* cloned the definition for a SCOUT window from modeller *A*, through the shared interaction history. Although it can be argued that modeller *E* could have figured how out to make such a definition without cloning from A, the collaborative modelling would have been much less effective and productive. Indeed, *E*'s action can be viewed as *learning through example* as a novice systems developer. In the collaborative Sudoku (cf. §6.3), a similar learning situation occurred when modeller *R* asked modeller *S* how to define a set of check rules exhibiting a disjunctive relationship (i.e. logical OR). In the cricket project (cf. §6.4), the learning seems not to be limited to transfer of technical knowledge (e.g. how to use to tool support, syntax of the notations); there is evidence that the students who knew little about cricket (i.e. domain-specific knowledge) have gradually acquired that knowledge through the collaborative EM process[143].

One may argue that the non-developers (e.g. users, managers, ethnographers) may not wish to acquire development related skills (e.g. making definitions in the EM process). However, as I argued in §3.2, users may invest in such learning if they believe such effort worthwhile (e.g. to increase productivity for future tasks).

Support diverse perspectives and integration

Both the VEL case study (cf. §6.1) and the case study of the cricket project (cf. §6.4) suggest that an EM approach to collaborative modelling is capable of facilitating diverse individual perspectives and integrating them into a shared perspective. In the case study of VEL, two modellers apparently were coming from different trainings (one from electronic engineering, one from computer science). Although the modelling process between them was by and large coordinative rather than collaborative in respect of the degree of engagement (cf. §5.1.1), the practice of EM nevertheless demonstrated how two different yet interdependent agent perspectives integrated into one shared perspective on the “virtual electronic laboratory”. In the case study of cricket, the facilitation and integration of diverse perspectives were more apparent. Students modelled the game of cricket with different modes of observation. Such diverse perspectives on the same situation for investigation have their counterpart in groupware development where stakeholders often have different perspectives of how the groupware ought to be, e.g. what feature it should support, how it should behave, what communication structure the groupware should support, etc.

Do not limit the user engagement in development to a sample of users

Although there is provision for *pretend play* (Sun, 1999) by the other human agents, EM does not encourage modelling by *proxy*. This is due to the fact that the referent situation in the real-world cannot be fully apprehended by any person other than the human agent who in that situation. That is to say, only the real users can reflect on and relate to their personal

[143] This potential for EM to integrate learning activities relating to many different subjects and goals was remarked in (Boyatt et al. 2006).

experience. This becomes apparent when the situation involves a large degree of tacit knowledge. However, collaborative modelling with all the stakeholders involved and engaged can be difficult in practice. As mentioned in chapter 3, user participation is impractical in some situations (e.g. there is no user in the product development, the users are too busy in their own activities (cf. Bannon, 1992b)). Furthermore, user participation may sometimes take the form of a token gesture or a political ruse rather than an approach for improving the fitness of the resulting system (cf. §2.5.2). Indeed, user sampling is more or less a socio-technical issue rather than a purely technical issue. Consequently, the potential for an EM approach could be diminished if user sampling takes place in the context of systems development that is highly sensitive to the specific situation, such as is the case in groupware development.

Potential for genuine participation: foundation for human-centred groupware development

Drawing on the above discussion, EM potentially allows active and direct influence by non-developers, supports seamless shifting in participants' roles, facilitates the participants to acquire and to progress their development related skills, and facilitates and integrates diverse perspectives. In the light of the discussion in chapter 3, these in turn support the co-evolution between the participants' understanding and the developing artifact, offering an alternative conceptual framework that takes account of the conflation of and shifting of roles of participants. Indeed, these are the enabling characteristics of genuine participation (cf. figure 7.1 and §3.5). Taking all the above into account, EM for collaborative modelling potentially supports the notion of genuine participation (cf. §2.5.2 and §3.4). Therefore, EM is potentially better suited for human-centred groupware development – since I have already argued earlier in this thesis (cf. §2.5) that genuine participation could be a remedy for human-centred systems development, and groupware development is one special branch of systems development.

7.2.2 Integrating the human and the technological

One of the topical concerns in developing socio-technical systems is how to integrate the technology into the human context so that the use of technology becomes an extension of the human body and transparent in the human activities (cf. Heidegger 1927; Dourish, 2001). In doing so, tool development has to be human-centred and the tool (i.e. groupware) has to be *grown* in the context of that human activity. As I discussed earlier in this thesis, the essence of HCD can be digested into six principles (cf. §2.5.1). Four of these principles (ii, iv, v, vi) are relevant to the integration of the human and the technological. EM addresses these principles accordingly:

- In EM, modellers continuously negotiate the best configuration (or disposition) through continuous observation, interaction and experimentation (cf. §4.1). These characteristics of EM allow the modellers to examine what is technically possible and to examine whether a particular configuration (and the particular state-as-experienced) is socially desirable. This, in turn, fulfils the HCD principle (iv) and (v).
- In EM, human intervention is possible throughout the process of MwDS (Rungratannaubol 2002). As we seen in the case studies (cf. §6.2 and 6.3), human communication and collaboration took priority over the technological-mediated communication and interaction. This suggests that EM fulfils the HCD principle (ii).
- EM does not distinguish between the context of development and the context of use. This not only facilitates the continuous refinement of the artifact in the context in which the work is carried out, but also avoids the drawbacks that may stem from the separation of the two – e.g. the deskilling of the workers due to formalised procedures of work (i.e. the lack of flexibility in handling exceptions) (cf. Evans and Beynon, 2001). Furthermore, it is possible to alter the actions of a machine agent through human intervention in EM. This all suggests that EM fulfils the HCD principle (vi).

Taking all the above into account, it is appropriate to claim that EM facilitates the integration between the human and the technological. Such a claim contributes to the overall argument that EM supports HCD, which is a prerequisite for supporting the efficacious groupware development perspective.

7.2.3 Enabling flexibility and evolution

As mentioned in chapter 2, HCD needs to cope with the flexible and evolving nature of human activities (cf. §2.4 and §2.5.1). As I discussed in chapter 4, EM artifacts are flexible in structure and always open to changes (cf. §4.1 and §4.3). EM artifacts (or construals) co-evolve in a fashion that is intimately linked with the modellers' personal understanding and experience of the artifact, shaped by the modellers' continuing observation of the referent, and always subject to change (or evolve) in response to the new insights of the modeller. Modelling with definitive scripts (MwDS) in EM neither necessarily follows a predetermined structure nor necessarily follows a predefined path (cf. §4.1.2 and §4.3.5). In fact, EM facilitates evolution at all times because it does not distinguish the development time and the use time (cf. §4.1.2 and §4.3.4). As the analysis of the case studies suggests (cf. chapter 6), the openness, flexibility, and evolving characteristics of EM are preserved when it is practised in a collaborative modelling environment. These characteristics of EM facilitate ad-hoc collaborative modelling (i.e. minimal planning) and *growing instead of building* the artifact in the situation (Brooks, 1987). Taking all these points into consideration, it should be evident that EM facilitates the development of flexible computer technologies that cope with people and allow them to manage and to shape their work, i.e. satisfying the HCD principle (i) in section 2.5.1.

7.2.4 Supporting diverse interaction and communication

When they come to work together, people make use of all the many different interaction and communication channels available to them:

> *"… they [People] take each others' past, present and prospective activities into account in planning and conducting their own work; they gesture, talk, write to each other, and so on, and they mesh these interactional modalities dynamically and seamlessly." (Schmidt and Simone, 1996, p.159)*

HCD not only needs to take the heterogeneous use of interaction and communication into account, but must also address the use of explicit (or rule-based, formal) and implicit (or tacit, informal) knowledge. As Gasson (2003) pointed out, computer technologies that ignore the implicit knowledge counterpart of the explicit knowledge may be unable to play a significant role in supporting human activities.

EM potentially addresses the use of diverse interaction and communication by facilitating the modellers to reflect on the state-as-experienced at real-time around the developing shared artifact. As the case studies on distributed jugs and collaborative Sudoku reveal, EM modellers interact around and through the artifact in many different modalities – they talked, gestured, leaned towards each other's screen, communicating with and exchanging definitive scripts within the modelling environment.

EM is fundamentally concerned with the use of tacit knowledge in the modelling process. Firstly, the modelling of a referent progresses through practical and tacit knowledge of the referent and the developing artifact. As I mentioned in chapter 4, EM artifacts are not *programs* and are more appropriately regarded as *construals* in the sense of Gooding, which are open for diverse interpretation based on the modeller's experience (cf. §4.1.3). Secondly, EM facilitates modellers who interact and communicate through reflecting on tacit knowledge. For example, in the collaborative Sudoku modelling case study (cf. §6.3), the interaction between modeller *R* and *S* regarding the '*currentstatus*' evidently involved tacit knowledge about the practical use of the observable.

7.3 GroupPIE – a framework for efficacious groupware development

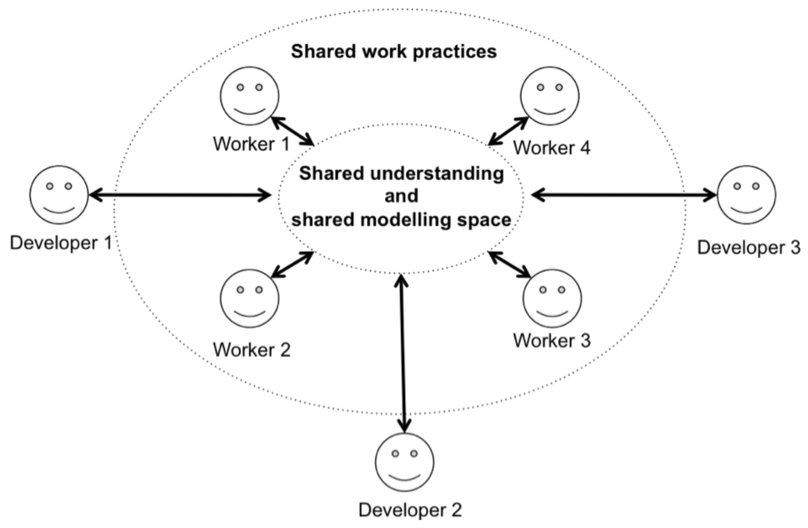
Earlier in this chapter, I have argued that EM is potentially better suited for human-centred development through facilitating *genuine participation*. As I argued in chapter 3, in genuine participation, the roles of participants (developers, users, etc.) become conflated. In fact, the users may demand a high degree of participation as they are deeply engaged in the development process, and contemporary approaches to systems development are not well-suited to this. This motivates us to propose EM as an alternative approach to systems development where human concerns should be given the highest priority, such as groupware development in the context of this thesis. In the previous section (cf. §7.1), I argued that the core activity in groupware development could be viewed as a collaborative modelling process. Viewed from this perspective, it allows us to practise an EM approach to groupware development, as if co-constructing an EM model collaboratively as I discussed in chapter 5 and 6.

Drawing on the conceptual discussion in chapter 3, I define participants as any people who are actively involved in the development of the groupware. Furthermore, I attribute roles to participants according to their actual work in the groupware development process at a particular time. This is to say, participants do not have a dedicated role in our framework. This is to acknowledge and to facilitate the shifting of roles in genuine participation that I discussed in chapter 3. As I already argued in chapter 3, the traditional conceptions of developers and users are loaded with implicit assumptions and are therefore misleading. However, for the purpose of the following discussion, it is helpful to distinguish participants by referring to their main roles respectively. I define *developers* as participants who are systems developers and consultants (e.g. ethnographers, information systems experts), users as workers (i.e. those who mainly carry out work using the groupware), and managers (i.e. those who manage and fund the groupware development).

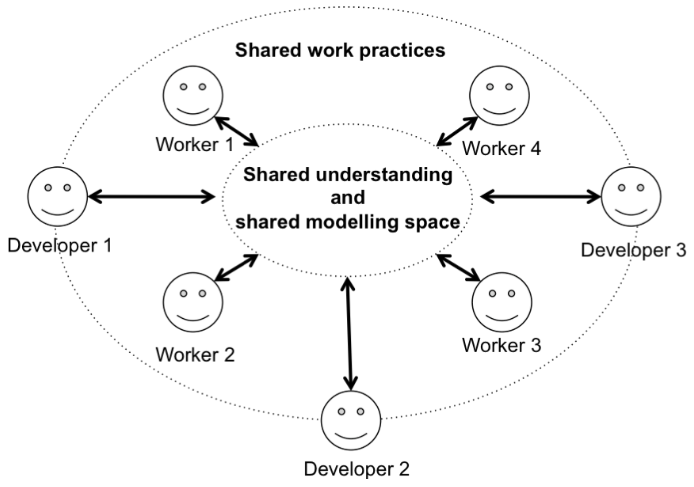
In the following, I present a conceptual framework for practising an EM approach in the context of groupware development. Rather than presenting our approach in a unified process model, I describe it from three perspectives, namely, i) participation, ii) interaction, iii) evolution – hence, GroupPIE. Unless otherwise stated, the groupware development process I describe below follows an EM approach.

The participation perspective

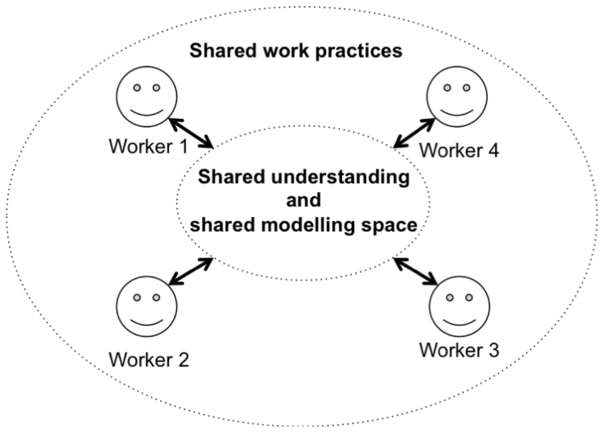
Viewed from the participation perspective, the focus is on the relationship between the developers and the workers. Although this relationship is changing throughout the development process, the changing relationship can be roughly divided into three scenarios. In the initial stage of the development project, only the workers would be familiar with the current work practices and context of work. Although they may interact within a shared modelling space, the developers nevertheless have little or no knowledge about the work practices that surrounding the groupware. In this scenario, irrespective of the purpose of the development and whether it is developer-oriented or user-oriented, the developer is still considered to be the an outsider in the workers' world (cf. figure 7.3a).



(a) Developers contributing from outside of the work practices



(b) Developers observing and working closely with the workers



(c) Developers left the scene and the workers took charge of the system

Figure 7.3 – The participation perspective in the GroupPIE framework

When the groupware development has progressed further, the developers move closer to the workers. The developers may observe closely how the workers carry out the work within the existing work practices. In this scenario, the developers and the workers can jointly develop the groupware and experiment with the developing groupware for the most suitable configuration for carrying out their (future) work practice. With an EM approach, this means that the modellers (i.e. participants of the groupware development) explore and negotiate through and within the model to find an *efficacious* configuration that reflects the state-as-experienced by way of co-constructing and co-evolving a shared EM model through the EM process (in the sense of co-evolution as it was described in chapter 5, and studied through the case studies in chapter 6). The motive behind this collaborative modelling with EM is to facilitate *genuine participation*. The virtue of collaborating and experimenting within a shared modelling space and workspace is that the participants (or modellers) can reflect upon their immediate experience when interacting with developing groupware (i.e. the EM artifact) in the shared modelling space (i.e. the EM environment). As in PD, developers and workers would mutually learn each other's practices through a high level of participation and engagement within this collaborative modelling process. However, as figure 7.3b depicts, the developers can never entirely enter into the workers' world, they can merely observe and collaborate with the workers at the boundary of the workers' shared work practices.

The motive behind such a high level of participation and facilitation of engagement is to shift the ownership of the groupware and the development process to the workers. This is based on the belief that workers are the owners of the problem (i.e. the activity which the groupware is to support), of the groupware, and of the groupware development process. In keeping with the idea of *efficacious groupware development* (cf. §2.5), the development process for groupware should encompass its lifespan from *conception of the groupware* to *disposal of the groupware*. This implies that the workers have to take charge of the *groupware development* at some stage (cf. figure 7.3c). Where 'customisation' in groupware development has been conceptualised in the past (e.g. design-in-use (Henderson and Kyng, 1992), tailoring (Wulf, 1999), evolution (cf. §2.4.2), etc.), it has been rare for the groupware developers to stay within the customisation process (whatever it is called) in the traditional conception of systems development. For this reason, I argue for *genuine participation* through collaborative modelling with EM in the GroupPIE framework. As argued in section 7.1, EM is potentially better suited for *genuine participation*, which allows the shifting in ownership and the shifting of roles (i.e. from users to developers in the sense of the traditional conception) to take place seamlessly.

The interaction perspective

Viewed from the interaction perspective, the focus is on facilitating diverse modes of interaction among the participants (or modellers) within the groupware development process (or collaborative modelling process). In chapter 5, I discussed a number of modes of interaction that may take place during a collaborative modelling process and I also argued that the modes of interaction among the modellers might evolve from time to time, due to different needs in the current activity or task at hand (cf. §5.1.3 and §5.1.4).

As in collaborative modelling, it is impossible to prescribe a framework that covers all the potential patterns of interaction that may take place in the context of efficacious groupware development due to the dynamic, situated, and complex nature of human interaction. One way to conceptualise such diverse modes of interaction in the context of groupware development, is to recognise it with a *cloud*[144]. The notion of cloud here is similar to what McLoughlin et al. (2009) refer to as the “interaction cloud” in a recent workshop in ECSCW 2009. McLoughlin et al. (2009) explains that the notion of interaction cloud reflects “the complex patterns of interaction, sharing and exchange that unfold around assemblies of mobile and embedded technologies”, in which “technology is emerging that allows people to shift between multiple contexts (mobile/embedded), facilitated by a connection to remote computing resources (clouds)”.

As figure 7.4 depicts, interaction among the participants within the development of groupware should be *reconfigurable* according to the situation. With an EM approach, the participants of the groupware development process may interact, observe, and experiment in diverse modes. In the light of the participation perspective (as discussed above), the

144 The cloud metaphor is borrowed from cloud computing (Weiss 2007), which has recently emerged in the IT industry.

workers will interact, observe, and experiment with the developing artifact during the early stage of development (viz. figure 7.3a to figure 7.3b) as if the developers are transparent in the process. The workers will continue to interact in the same way, and observe and experiment in a similar way when the developers withdraw from the process.

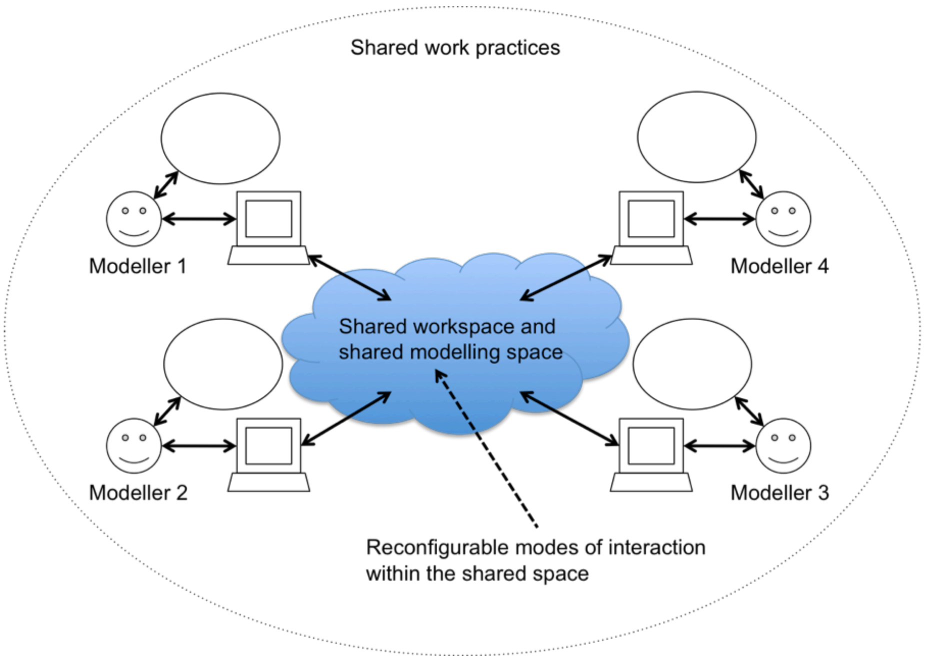

Shared work practices
Modeller 1
Modeller 4
Shared workspace and shared modelling space
Modeller 2
Modeller 3
Reconfigurable modes of interaction within the shared space

Figure 7.4 – The interaction perspective in the GroupPIE framework

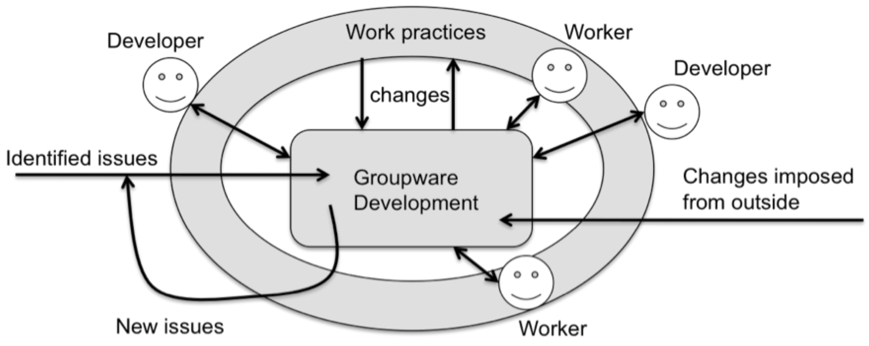


Figure 7.5 – The evolution perspective in the GroupPIE framework[145]

[145] This diagram takes its inspiration from the SPORE process in (Sun 1999, p.233)

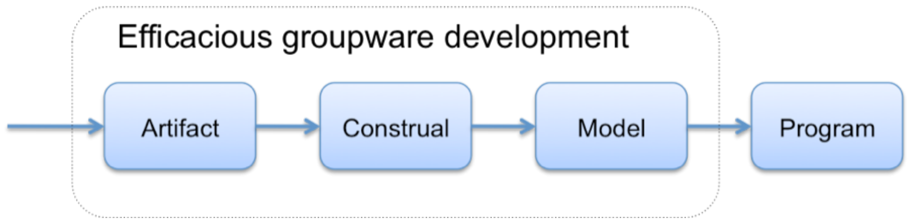


Figure 7.6 – The scope of efficacious groupware development in relation to ACMP

The evolution perspective

Viewed from the evolution perspective, developing groupware with the GroupPIE framework can be thought of as *growing* groupware *in situ*. That is, the groupware is co-evolved with the group of workers that eventually takes charge of it (and the developers) within the environment which the work is taking place. Figure 7.5 – though it might be overly simplified from a real world groupware development situation – illustrates the interplay between the key entities (e.g. participants, factors, etc.) around and surrounding the evolutionary process in the GroupPIE framework.

In chapter 4, I described how an EM artifact can be viewed as a construal, a model, or a program when the observables, dependencies, agents, and agencies (ODA) achieve a certain degree of stability. In the context of efficacious groupware development, however, the EM construal may never be regarded as a program. This is due to the fact that the context for the groupware operation (viz. context of the work, the work practices, the group itself, the individuals within the group, etc.) is ever-changing and evolving at all times. Subsequently, ODA in this context may never be considered as finalised in the sense of traditional programming (Beynon et al., 2006). Viewed from the evolution perspective of the GroupPIE framework, the groupware is always *growing* like an organic process with an extended lifetime. Figure 7.6 depicts the scope for efficacious groupware development from an EM perspective.

7.4 Limitations for the GroupPIE framework

The formulation of the GroupPIE framework is based on previous research into EM and the research into EM for collaborative modelling that I discussed in chapter 5 and 6. Although I have argued the potential of EM in the collaborative modelling context and have shown the potential of EM in the groupware development context (when practising in the GroupPIE framework), there are at least three limitations for the GroupPIE framework:

1. Firstly, I have only considered collaborative modelling and groupware development with EM in small groups which have no more than a dozen of participants (or modellers). It is intended to be an approach for small teams in small and medium sized enterprises (SMEs) to *grow* groupware. The case studies that I discussed in chapter 6 were all aimed at this context. Certainly, larger teams may face a similar socio-technical challenge when developing software systems support. However, the scale of the case studies cannot indicate whether an EM approach, with the GroupPIE framework, can be practised in larger teams. Furthermore, it can be argued that software systems for a larger population of ‘users’ are not groupware but multi-user enterprise systems.
2. Secondly, the GroupPIE framework has not taken safety critical properties of the developing system into account. In the safety critical domains (e.g. healthcare, space mission, avionics, automobile, etc.), priority in the development is given to ensuring that the resulting system is safe, secure, and reliable (cf. Sommerville, 2004), rather than to the social aspects surrounding the development process. Failure of the system can be disastrous and often costs human lives. It would be risky to adopt the GroupPIE framework for groupware development in safety critical domains.
3. Thirdly, the GroupPIE framework merely affords the potential for the participants of the groupware development process to grow the groupware and to negotiate for an *efficacious disposition* from time to time (cf. figure 7.5), it does not explicitly focus on the existing political issues in the group. It is possible that the research described in this thesis could be extended to study the political impact of the GroupPIE framework, but such research is nevertheless out of the scope of this thesis.

7.5 Towards efficacious groupware development: vision for GroupPIE

Earlier in this thesis (cf. chapter 2), I argued for an *efficacious disposition* perspective for groupware development (i.e. efficacious groupware development), which views the previously separated development, use and evolution processes as *a single, holistic, organic, and situated process* that continuously arranges existing ingredients in the situation to find a suitable configuration in the evolving context throughout its lifespan. This perspective takes account of the fact that the design and development of groupware goes over the preconceived boundary between the context of development and use. Rather than drawing a boundary between design, development, and use, this perspective conflates all preconceived contexts. Subsequently, the roles of participants are also conflated (cf. chapter 3). Due to the holistic view of contexts and participant roles, efficacious groupware development is different from other perspectives on groupware development and evolution, e.g. meta-design (Fischer and Scharff, 2000). Furthermore, such a "lifelong" perspective is controversial in the context of traditional systems development, but not in the light of lifelong human activities, e.g. lifelong learning (cf. Beynon and Harfield, 2007).

In this chapter, I have proposed a plausible agenda for practising an EM approach to efficacious groupware development. This agenda consists of two parts: i) orienting groupware development as collaborative modelling (cf. §7.1); ii) a framework (viz. GroupPIE) that sketches how EM might be practised collaboratively in the context of groupware development as it were in a collaborative modelling context (cf. §7.3). Central to the GroupPIE framework is the notion of *genuine participation*. Earlier in this chapter, I argued that EM is potentially better-suited for human-centred development and for efficacious groupware development, because it potentially fulfils seven recommendations in facilitating the notion of genuine participation (cf. chapter 3). In genuine participation, participants are treated as human beings who are acting with their knowledge rather than persons who have preconceived roles as in traditional systems development. Hence, the context for participation and action can be thought as a single, shared, situated, and conflated modelling space. As argued in chapter 2, the issues in the socio-technical challenge are intertwined so that they cannot easily be tackled independently. Insofar as EM supports genuine participation, this allows the issues to be considered and to be addressed as a whole, rather than independently. What is more, these issues can be addressed when they arise throughout the EM process. To some extent, collaborative modelling through an EM approach is similar to collaborative prototyping (e.g. Stavash and Chadh, 2000). However, an EM approach to systems development is different from prototyping in at least three ways. Firstly, a conventional prototype may be thrown away, but this is not the case in the EM approach that I am proposing for efficacious groupware development. Secondly, prototyping maintains a developer-user conceptual gap: prototypes are constructed by the developers and then sent to the users for evaluation. Users then give feedback to the developers. Sun (1999) refers to this process as *backward feedback*. Thirdly, EM allows the modellers to interact with and through the EM model in an open and flexible manner (cf. chapter 4).

Since EM is a human-centred approach to computer-based modelling, it does not follow a prescription paradigm for systems development (Beynon et al., 2006, Wong, 2003, Sun, 1999). As Harfield (2008) writes

> *"… one of EM's powers is that within a model 'everything is up for grabs' at any time during construction and use [i.e. the EM process – EM does not distinguish between construction and use]" (ibid, p.223)*

The direction for developing the artifact (i.e. the groupware) is determined by the modellers' past and immediate experience of continuous interaction with and observation of the referent (i.e. the mental model of the groupware in the modellers' mind) and of the developing artifact (i.e. the actual groupware evolving in the modelling environment), which cannot be predicted and framed in advance. Consequently, the GroupPIE framework sketches three perspectives (viz. participation, interaction, and evolution) – which cover different levels of activities – in the context of efficacious groupware development when practising an EM approach, rather than explicit and step-by-step guidelines for practising EM.

The GroupPIE framework is based on the potential of practising an EM approach to collaborative modelling. In this framework, the participants co-evolve the EM model at different degrees of engagement, different participation schemes among the modellers, and different modes of interaction through a shared modelling space, as if they were practising an EM approach to collaborative modelling (cf. chapter 5). The case studies in chapter 6 strengthen this argument as they have shown that EM models can be co-constructed and co-evolved in different collaborative modelling contexts.

Efficacious groupware development is a continuous disposition of the groupware (through arranging the agents internal and external to the groupware) within the evolving human context that includes the work practices and the group activities, which surround the groupware and are to be supported by the groupware, throughout its lifespan. With this in mind, the GroupPIE framework facilitates *efficacious disposition* in the following respects:

> *Initial efficacious disposition* – the developers (i.e. participants including ethnographers and experts other than the workers) and the workers co-determine whether the development should begin from scratch (i.e. no definitions in the modelling environment prior to the development) or be based on an existing model. As I demonstrated in the collaborative Sudoku case study in section 6.3, it is possible to reuse an existing EM model to kick-start the collaborative modelling[146]. On the one hand, it is more efficient to begin the development from an existing EM model. On the other hand, reusing an EM model can be risky because there will no guarantee the model would capture the experience of the modeller as the links between observables and their contexts of observation may be broken[147] – hence, the benefit of practising an EM approach may be partially lost. Since the initial stage in the GroupPIE framework is a selection between two possibilities, it can then be viewed as *finding an efficacious disposition (or a condition) for the groupware to grow*.

[146] The EM model used for kick starting can be thought as a ‘seed’ EM model (Sun, 1999; Chen, 2001).

[147] If the modeller is not the *owner* of the reusing model, the model can merely be regarded as a traditional program rather than an EM construal.

Participative efficacious disposition **–** based on the initial model, the participants co-evolve the EM model. The participants continuously observe, interact, and experiment with the evolving artifact collaboratively at different degrees of engagement, different participation schemes among the modellers, and different modes of interaction in a shared modelling space. Since the participants are improving the configuration continuously, it can be viewed as *seeking an efficacious disposition so that the groupware and the group activities will work together harmoniously*.

Evolutionary efficacious disposition – as the developers withdraw from the efficacious groupware development process, the workers take over the process and continue to co-evolve the groupware (i.e. making redefinitions to the EM artifact) according to their needs and in response to the evolving work practices in the rest of the groupware's lifespan. This can be viewed as *a continuous seeking for an efficacious disposition so that the groupware and the group activities continue to work harmoniously*.

Despite the potential that the GroupPIE framework may entail, realising efficacious groupware development in practice is still challenging because of the following issues:

Lack of good EM tool support for collaboration – the tool support for DEM (viz. dtkeden) and its variants that I used to carry out the collaborative modelling case studies in relation to this thesis (viz. dtkeden-blackboard, dtkeden-cm) were still in an experimental stage. None of this tool support allows dynamic switching of the interaction modes faultlessly. Furthermore, as we observed in the collaborative modelling case studies, the modelling was disrupted a couple of times due to the instability of the tool. Apart from the lack of collaboration support[148], the modellers were often distracted by the error messages caused by mistaken definitions input in the shared modelling space.

[148] However, the tool support for collaboration could be added on-the-fly in the modelling process as needed. As one of the collaborative modelling case studies demonstrated, the modellers can develop and evolve such support in a situated manner.

Motivation for participants – GroupPIE may suffer similar issues to other participatory approaches for socio-technical systems[149] development that involve a high-degree of user participation, e.g. PD (Schuler and Namioka, 1993), meta-design (Fischer and Scharff, 2000), informed participation (Brown et al., 1994, Fischer and Ostwald, 2002). For instance, Fischer and Scharff (2000) pointed out that stakeholders (i.e. workers, managers, etc.) must be motivated and rewarded for the extra work that they may incur through their participation. This echoes Grudin's (1993) observation in groupware development. A similar motivation-reward issue is also found in relation to the acquisition of development-related skills. Blackwell (2002; 2004) argued that the users (or workers) would invest resource to acquire such skill if they believe that the potential benefit of that investment is greater than the outcome of not investing. A common issue in these participatory approaches is that they are more or less developer-oriented – a problem that I highlighted in section 2.5.1. In contrast, GroupPIE is a hybrid approach that combines developer-oriented and user-oriented development. It is a framework specially designed for practising an EM approach in the context of groupware development. The ownership of the process is gradually shifted from the developer to the workers through *genuine participation* (cf. §7.3). By shifting the ownership to the workers, the responsibility is also shifted to the workers. In this way, the workers will have to take care of their own groupware, just as a house owner has full ownership and responsibility for taking care of their own house when she gets the key from the house builder[150]. Nevertheless, it is not too clear how far the GroupPIE can go in relation to the lifelong aspect of efficacious groupware development.

[149] Groupware is an instance of a socio-technical system, but the size, the scope, and the users of groupware are on a smaller scale. For instance, Facebook (http://www.facebook.com) can be classified as a socio-technical system but not as a groupware because the size of the groups on Facebook is usually more than a hundred.

[150] This analogy is only valid for the ownership and responsibility handover. It is not perfect because the house builder is not likely to transfer their building skills to the house owner.

Groupware development in the economic culture – in my own industrial experience[151], most systems development projects (whether socio-technical or not) prioritise cost and time over quality (in terms of the factors surrounding the project triangle (cf. Lewis, 2005)) due to the existing economic culture. As argued earlier in this thesis, it is unreasonable to facilitate a reduced degree of participation and to expect the benefit of a high degree of participation (cf. chapter 2 and chapter 3). If participatory approaches do not cope with the demands of such a cruel economic culture, their promise, future, and success in groupware development (and in generic systems development) could be diminished. It is not clear, at this stage, whether radical participatory approaches, such as the GroupPIE framework (cf. §7.3), will flourish in the *cultures of participation* (Fischer, 2008; 2009).

Despite the issues mentioned above, it should be evident that EM is promising for efficacious groupware development. The study of EM for collaborative modelling (cf. chapter 5) and the GroupPIE framework nevertheless has set out a plausible direction for further development for collaborative EM and for realising efficacious groupware development. With the GroupPIE framework, the possibility of practising an EM approach to efficacious groupware development is taken one step forward.

[151] Prior to my postgraduate studies, I worked in the IT industry as a systems developer and a systems administrator between 2000 and 2003.

Chapter 8
Conclusions

As I mentioned in the beginning of this thesis, the aim of this research is to explore the respects in which EM can provide an alternative and potentially a better conceptual framework for groupware development that accommodates the dynamic nature of the context of group work that the groupware aims to support. This thesis consists of three sections. The first section (chapter 2 and 3) reviews research into groupware and the approaches to its development. The second section (chapter 4, 5, and 6) explores EM and its potential in collaborative modelling. The last section (chapter 7) puts EM into the context of groupware development, arguing that the potential of EM in collaborative modelling can be exploited in groupware development. Chapter 7 also argues that such orientation potentially yields a better conceptual framework that gives a holistic account of groupware development from the emergence of conception to its eventual disposal.

8.1 Research summary

In chapter 2, I stated that the groupware development is challenging due to: i) the socio-technical issues surrounding its development (cf. §2.2.2), and ii) the difficulties that typical software development faces (because groupware is a species of software)(cf. §2.2.3). These issues are intertwined and are often hard to infer from the formal description of the group activities (especially the work practices) so that they cannot easily be addressed independently. To expose and address these issues, contemporary groupware development is usually iterative, ethnographical, and participatory (cf. §2.3). Due to the evolving nature of the group activities, successful groupware has to co-evolve with the human/group activities that it supports (cf. §2.4). One promising way to alleviate these problematic issues is to adopt a human-centred focus for the development process, as this takes the evolving and situated nature of human activities into account (cf. §2.5.1). However, contemporary systems development approaches do not respect human-centred design principles entirely, though participatory approaches (e.g. PD) come closest (cf. Gasson, 2003). Furthermore, the need for evolution in groupware development also leads to a conceptual disorientation in respect of the perceived contexts of development and use for the groupware – the increased user participation alone does not resolve this conceptual disorientation properly (cf. §2.5.2). For this reason, I endeavour to establish a unified organic process view of groupware development – efficacious groupware development – which does not distinguish contexts and roles (cf. §2.5.3). This view is influenced by the notion of *efficacious disposition* – Jullien's interpretation of the ancient Chinese philosophical notion of *shi* in seeing the propensity of things (also cf. Jullien, 1995).

The analysis in chapter 2 indicates that conceptual issues surrounding participation are at the centre of efficacious groupware development. For this reason, chapter 3 is devoted to a deeper examination of the issues surrounding participation. I argue that the contexts of development and use, the roles of stakeholders (in particular developer and user), the nature of participation (natural to the roles of participants) are interdependent concepts. These concepts are intertwined: re-conceptualising one of them may result in the need to re-conceptualise the rest. In short, chapter 3 argues:

i) a high degree of user participation problematizes the perceived contexts and roles (cf. §3.4);

ii) the roles of the participants begin to shift as they become deeply engaged in the development process (cf. §3.3);

iii) user participation may cause a tension between stakeholders because of the imbalance of influence over the construction of the artifact (cf. §3.1); and

iv) reconceptualising the conception of roles is not trivial (§3.2).

Human-centred development promotes a high degree of user participation that is close to *genuine participation*. In fact, the nature of participation is arguably *genuine participation* when the contexts and roles are conflated, which is the case in efficacious groupware development. In closing chapter 3, I made seven suggestions for facilitating *genuine* *participation* (cf. §3.5).

In chapter 4, I discussed EM principles (§4.1), the philosophical outlook of EM compared with some key ideas in philosophy of science (cf. §4.2), and the practical tool support for EM (§4.3). EM is a modelling approach which is based on experience, interaction, observation, and experimentation. The idea behind modelling through observation is similar to the idea of theory building through observation in philosophy of science (§4.2.1), and the idea of an experience-based approach shares a similar perspective with James's Radical Empiricism. The key elements in an EM model are the observable, dependency, and agent (cf. §4.1.1). The EM process is open and flexible. In contrast to traditional systems development, the EM process does not necessarily follow a pre-defined path or goal. Central to the EM process are two important concepts: modelling *state-as-experienced* and *modelling with definitive scripts* (MwDS) (cf. §4.1.2). The meaning of an EM model (or artifact) cannot be apprehended through the examination of the artifact itself – its meaning is closely associated with the possible interactions and possible experience it offers to its observer. These qualities are always open for new interpretation and subject for change in response to continuous observation throughout the EM process. In the sense of Gooding's construal, EM artifacts are provisional, situated, tacit, interactive, and ostensive (cf. §4.1.3). In section 4.3, I discussed how the EM principles and modelling process are currently supported, though far from perfectly, by the principal tool (viz. tkeden).

There are many different forms of collaborative modelling (CM), different degrees of engagement, different relationships between modellers and agents, different modes of interaction (cf. §5.1). In section 5.2, I argue that Sun's (1999) Distributed Empirical Modelling (DEM) framework has limited capacity for collaborative modelling. This can be attributed to the fact that DEM was developed for structured organisations rather than collaborative work which is typically dynamic and situated, and often involves changing and diverse modes of interaction, communication, and structure. I argue that EM has in fact more potential for facilitating the heterogeneous nature of collaborative modelling (cf. §5.3).

In chapter 6, I investigated how EM can be practiced in collaborative modelling contexts through four case studies, namely, i) virtual electronic laboratory; ii) distributed jugs, iii) collaborative Sudoku, and iv) the cricket project. These case studies are meant to cover a broad range of collaborative modelling scenarios, from a pair of modellers to not more than a dozen modellers, from asynchronous to synchronous collaborations, from unstructured to structured goals for the modelling. The case studies firmly support the potential of an EM approach to collaborative modelling. In addition, the case studies reveal that collaborative EM facilitates some key properties that we are looking for in supporting the notion of genuine participation and human-centred development: supporting different degrees of engagement (cf. §6.1, §6.2, §6.3, and §6.4), conflation of contexts (cf. §6.1, §6.2, and §6.3), conflation and shifting of roles (cf. §6.2 and §6.3), modelling through experimentation (§6.2), supporting dialogical and diverse interaction (cf. §6.2), co-evolution between the modellers understanding and the artifact (cf. §6.3), integrating diverse perspectives (cf. §6.4), and learning through co-construction (cf. §6.4).

Drawing on the discussions from previous chapters, chapter 7 argued that EM is potentially a conceptual framework better-suited for guiding efficacious groupware development. In order to show the potential of practising an EM approach in groupware development, firstly (cf. §7.1), I view groupware development as a collaborative modelling process that co-constructs a shared understanding of the group activity (i.e. a conceptual model) through construction of the groupware (i.e. the actual artifact). Through the lens of efficacious groupware development, such a modelling process is in fact a co-evolution without a pre-defined *end product*. Secondly (cf. §7.2), I argued that EM is potentially suited to providing the two prerequisites for efficacious groupware development: human-centred development and genuine participation. This claim is justified by the potential fulfilment of the seven recommendations that were set out in chapter 3 and the human-centred design principles discussed in chapter 2. Thirdly (cf. §7.3), I proposed a *GroupPIE* framework that allows EM to be practised in the groupware development context through practising EM for collaborative modelling in the way I discussed in chapter 5 and 6. Instead of giving precise step-by-step guidelines on how EM is practised (as EM does not encourage a preconceived path for modelling), the GroupPIE framework sketches the plausible scenarios when EM is introduced into the groupware development context at three levels: **P**articipatory, **I**nteraction, and **E**volution. Fourthly, I discussed the limitations of the GroupPIE framework (cf. §7.4).

Finally (cf. §7.5), I sketched a vision, the potential issues and obstacles in realising efficacious groupware development with EM.

8.2 Further work

In this thesis, I have shown how EM could be practised in a collaborative modelling context. Although I only justify its potential in the groupware development context, the potential of EM is in principle more far-reaching than the scope of this thesis. In this section, I sketch a few possible directions for further work based on the research described in this thesis.

Approaches / Characteristics	Empirical Modelling	Participatory Design
i. Development strategy	Growing incrementally through continuous observation	Iterative, design-by-doing (Bødker 1987, Greenbaum and Kyng)i
ii. Problem solving	Bottom-up	Bottom-up
iii. Integrating diverse perspectives	Encourages diverse perspectives through different modes of observation	Enables workspace democracy and takes account of diverse perspectives of all stakeholders
iv. Diverse interaction	Does not presume a particular mode of interaction	Multiple channels (Mambrey and Pipek, 1999})
v. Participation	Encourages genuine participation	Workspaces democracy through active and direct user participation
vi. Structural constraints	None, but encourages genuine participation	None, at least not during the PD project

Table 8.1 – Similarities between Empirical Modelling and Participatory Design

Participatory design and distributed participatory design

Participatory design (PD) is a body of practices, techniques, methodologies, policies, and philosophy that advocates a high degree of workspace democracy through active and direct user participation. In chapter 2, I briefly discussed participatory design in the context of groupware development. Due to the scope of this thesis, the full potential of EM for PD has yet to be explored in the broader context. Although there is so far no direct connection between EM and PD, previous research has already hinted at the potential of EM in participative systems development (cf. Sun, 1999, Chen, 2001, Wong, 2003). As I argued in chapter 7, EM potentially facilitates the notion of *genuine participation* (cf. chapter 3) – a degree of participation (and collaboration) that transcends the notion of *genuine user participation* as advocated in PD (cf. Bødker et al., 2004). Furthermore, as shown in table

8.1, EM and PD share a number of similar characteristics. This potentially means that EM can be elaborated to support PD activities in the wider context.

Distributed participatory design (DPD) is an emerging field of research that is dedicated to the distributed challenge in PD projects (cf. Danielsson et al., 2006; 2008a; 2008b). Gumm (2006) suggests that DPD projects are distributed in three dimensions: physical, temporal, and organisational. The distributed challenge in DPD, thus, can be thought as *to maintain workspace democracy through active and direct user participation in highly distributed settings*. As the case studies (cf. chapter 6) revealed, EM facilitates different degrees of engagement and diverse modes of interaction in a collaborative modelling context. With proper tool support, it is possible to transform EM principles and the GroupPIE framework into principles and frameworks that can be used to inform DPD projects.

Collaborative software development

With the ongoing trend of globalisation, members of a software development team are usually spread across the world. Collaborative software development (CSD) is a growing research field that often utilises groupware technologies and research from research fields such as CSCW and HCI. CSD faces a similar challenge as does DPD in coping with distributed work. However, being more inclined towards the traditional software engineering paradigm, CSD puts its focus on facilitating the collaboration and coordination of larger developer teams over the Internet rather than promoting active and direct user participation (cf. Hildenbrand et al., 2008). So far in this thesis, I have only considered collaborative modelling and groupware development with EM in relation to small groups having no more than a dozen of participants (or modellers). With the emerging need to collaborate over the Internet and the recent development in *Web EDEN* – a web-based support tool for EM developed by Richard Myers (2008), it is worthwhile to explore how the work in this thesis can be further developed to cope with larger scale collaborative modelling over the Internet.

Collaborative modelling tool support

As I mentioned earlier in this thesis (cf. §5.2.2 and §6.3.5), there are at present a number of problematic issues in the EM tool support for collaborative modelling.

Firstly, the current collaborative modelling (CM) tool support does not allow switching between modes of interaction (cf. §5.1.3) – any reconfiguration of the modelling environment means restarting the collaborative modelling session. Although it is possible to mimic different modes of interaction by exploiting existing modes supported by the modified tool (cf. §6.2, §6.3, and §6.4), it is possible that the modellers would like to *formalise* a set of ritualised and situated modes of interaction that emerged during the collaborative modelling process. Due to the situatedness of the group dynamics, it is not possible to prescribe a generic universal set of modes of interaction. One plausible way of improving the collaborative modelling environment is to develop a definitive notation which enables the modellers to *grow* and experiment with the appropriate utilities for support *in situ* – similar to the 'little helper agent' grown out of the modellers' needs in respect of communication and collaboration discussed in section 6.3.5 (cf. figure 6.14 on page 202).

Secondly, the transition between private and public workspace is not without issues. Throughout the collaborative modelling case studies, it became clear that some form of basic awareness support is necessary. This is an issue in the user interface of the tool support. A good starting point is to integrate tool support models, such as the dependency viewer (EMPA: dmtHarfield2006) and visual symbol table (EMPA: vstKing2005), into the collaborative modelling environment.

Thirdly, there is no doubt that the modelling environment is not yet stable enough. However, improving the stability of the modelling environment is challenging without sophisticated knowledge about how and why the tool has been built in the way that it has – in Naur's (1985b) sense, it is challenging because the original authors of the tool left with the "theory" of the tool.

Indeed, the collaboration tool support features that I mentioned above is not unique. To some extent, contemporary collaboration tools have addressed these issues, e.g. (Sonnenwald et al. 2003).

Activity theory

Sun's (1999) Distributed Empirical Modelling (DEM) framework is draws on ethnomethodology and distributed cognition. As I argued in chapter 5, distributed cognition may not be fully compatible with the need for flexible structure and interactions in collaborative modelling due to its emphasis on fluent cognitive function (or operation) in structured organisation (or teams). Recently, another social theory, *activity theory*, has received much attention in the CSCW and HCI community. Activity is the basic unit of analysis in activity theory:

> *"Activity is considered the most basic category; analysis of activities opens up a possibility to properly understand both subjects and objects [of the activity]. This idea may appear counter-intuitive. Traditional analytical thinking, typical, for instance, of natural sciences, would assume that to understand an activity it is necessary to understand the subject and the object separately and then make an inference about their interaction. Activity theory challenges this assumption. It claims that this apparently flawless logic can be misleading." (Kaptelinin and Nardi, 2006, p.31)*

Since activity theory focuses on the relationship between the subject and the object and the possibilities for changes, activity theory potentially gives a better account of the co-evolution between participants' understandings and the evolving artifact in the collaborative modelling process (in the sense as described in §5.1.1). So far in this thesis, I have only drawn on activity theory in explaining the evolving nature of human activity (cf. §2.4), but have yet to establish firm connections with activity theory.

Activity theory has a long history that originated from Vygotsky's historical and cultural psychology (cf. Kaptelinin and Nardi, 2006). It would be interesting to know whether activity theory can inform the collaborative modelling process when it is practised using an EM approach and to identify the main points of contact between EM and activity theory. However, there are many varieties of activity theory (Rogers, 2008). It is not clear, at this stage, which strand of activity theory this work should attach to. Moreover, although there seems to be a natural fit for activity theory in explaining collaborative modelling (as an activity), it seems that activity theory conceptualises human agency differently from EM, as Kaptelinin and Nardi (2006) write:

> *"Activity theory conceptualizes the potency of human agency in part through the principle of mediation: tools empower in mediating between people and the world ... things have agency, because if they did not, they could not act as mediators." (ibid, p.248)*

For this reason, further research is necessary before any strong links to activity theory can be made.

8.3 Concluding remarks

This thesis argues that Empirical Modelling (EM) potentially offers a better conceptual framework for groupware development that accommodates the dynamic nature of the group work it supports. The main conclusions of this thesis can be summarised as follow:

1. The entire lifecycle of groupware should be conceived as a continuous development process. This is because groupware, unlike other species of software, has to co-evolve with the context for the group work it supports. This means that, viewed from a holistic perspective, the development never terminates. There is so far no evidence that computer artifacts can evolve themselves according to human needs – the evolution of a computer artifact involves human agency.
2. The *efficacious groupware development* perspective provides a unified organic process view of groupware development consonant with Jullien's interpretation of ancient Chinese philosophy in "*The Propensity of Things*".
3. EM promotes a high degree of participatory development and potentially facilitates *genuine participation*. EM also potentially fulfils the human-centred design principles. This makes EM potentially better suited for human-centred groupware development when practised in the *GroupPIE* framework.
4. The *GroupPIE* framework provides a promising foundation for practising an EM approach to groupware development. It potentially fosters the notion of *efficacious groupware development*.

As I pointed out in chapter 7 (cf. §7.5), there are a number of factors, both technical and social, that may hamper the realisation of *efficacious groupware development*. These include, amongst other things, the limitations of the current tool support for collaboration using EM, the difficulties of motivating genuine participation, and the economic culture. These issues are not trivial to tackle and, arguably, the latter two go beyond the scope of the computer science discipline. In order to tackle these issues properly, interdisciplinary research is required – but "true" interdisciplinary research is not easy to achieve (Rogers et al., 2003).

Bibliography

Ackerman, M. S. (2000). The Intellectual Challenge of CSCW: The Gap Between Social Requirements and Technical Feasibility. *Human-Computer Interaction*, 15(2/3), 181-203.

Ackerman, M. S. and Malone, T. W. (1990). *Answer Garden: a tool for growing organizational memory*. In Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems, 31-39, New York, NY, USA. ACM.

Ackerman, M. S. and McDonald, D. W. (1996). *Answer Garden 2: merging organizational memory with collaborative help*. In CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work, 97-105, New York, NY, USA. ACM.

Adzhiev, V. D., Beynon, W. M., Cartwright, A. J., and Yung, Y. P. (1994a). *A Computational Model for Multiagent Interaction in Concurrent Engineering*. In Proc. CEEDA'94, 227-232, Bournemouth University.

Adzhiev, V. D., Beynon, W. M., Cartwright, A. J., and Yung, Y. P. (1994b). *A New Computer-Based Tool for Conceptual Design*. In Proc. Workshop Computer Tools for Conceptual Design, 171-188, University of Lancaster.

Alter, S. (2000). Same words, different meanings: are basic IS/IT concepts our self-imposed Tower of Babel?. *Commun. AIS*, 2.

Amabile, T. M., Conti, R., Coon, H., Lazenby, J., and Herron, M. (1996). Assessing the Work Environment for Creativity. *The Academy of Management Journal*, 39(5), 1154-1184. (Also available at: http://www.jstor.org/stable/256995)

Anderson, R. (1994). Representation and Requirements: The Value of Ethnography in System Design. *Human-Computer Interaction*, 9(2), 151-182.

Andriessen, J. H. E., Hettinga, M., and Wulf, V. (2003). Introduction to Special Issue on Evolving Use of Groupware. *Comput. Supported Coop. Work*, 12(4), 367-380.

Andriessen, J. H. E. (2003). *Working with Groupware: Understanding and Evaluating Collaboration Technology*. Springer.

Bair, J. H. (1989). *Supporting cooperative work with computers: Addressing meeting mania*. In COMPCON Spring '89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers, 208-217, San Francisco, CA.

Bannon, L. J. (1992a). *Perspectives on CSCW: From HCI and CMC to CSCW*. In Proceedings International Conference on Human-Computer Interaction (EW-HCI'92), St. Petersburg, Russia.

Bannon, L. J. (1992b). *From human factors to human actors: the role of psychology and human-computer interaction studies in system design*. In Greenbaum, J. and Kyng, M. (Eds.), *Design at work: cooperative design of computer systems*, 25-44. Hillsdale, NJ, USA: L. Erlbaum Associates Inc..

Bannon, L. J. and Schmidt, K. (1989). *CSCW: four characters in search of a context*. In Proceedings of the First European Conference on Computer Supported Cooperative Work (ECSCW '89), 358-372, Gatwick, London.

Barki, H. and Hartwick, J. (1989). Rethinking the concept of user involvement. *MIS Q.*, 13(1), 53-63.

Baron, R. S., Kerr, N. L., and Miller, N. (1992). *Group process, group decision, group action*. Buckingham: Open University Press.

Béguin, P. (2003). Design as a mutual learning process between users and designers. *Interacting with Computers*, 15, 709-730.

Béguin, P. and Clot, Y. (2004). Situated Action in the Development of Activity. *Activites*, 1(2), 50-63. (Also available at: http://www.activites.org/v1n2/html/beguin.eng.html)

Bentley, R. and Dourish, P. (1995). *Medium versus mechanism: supporting collaboration through customisation*. In ECSCW'95: Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work, 133-148, Norwell, MA, USA. Kluwer Academic Publishers.

Beringer, J., Fischer, G., Mussio, P., Myers, B., Paterno, F., and de Ruyter, B. (2008). *The next challenge: from easy-to-use to easy-to-develop. are you ready?*. In CHI '08: CHI '08 extended abstracts on Human factors in computing systems, 2257-2260, New York, NY, USA. ACM.

Bertelsen, O. W. (2006). *Tertiary Artefactness at the Interface*. In Fishwick, P. (Ed.), *Aesthetic Computing*, 357-368. The MIT Press.

Beynon, W. M., Angier, D., Bissell, T., and Hunt, S. (1986). *DoNaLD: A Line-drawing System Based on Definitive Principles*. Research Report CS-RR-086, Department of Computer Science, University of Warwick. 1986.

Beynon, W. M., Norris, M. T., Orr, R. A., and Slade, M. D. (1990). *Definitive specification of concurrent systems*. In Proc UKIT'90, 52-57. IEE Conference Publications.

Beynon, W. M., Cartwright, R., Sun, P. H., and Ward, A. (1999). *Interactive Situation Models for Information Systems Development*. In Proceedings of SCI'99 and ISAS'99, Orlando, USA.

Beynon, W. M., Ward, A., Maad, S., Wong, A., Rasmequan, S., and Russ, S. (2000). *The Temposcope: a Computer Instrument for the Idealist Timetabler*. In Proceedings of the 3rd international conference on the practice and Theory of Automated Timetabling, 153-175, Konstanz, Germany.

Beynon, W. M., Boyatt, R., and Ward, A. (2003). *The dtkeden manual - distributed communication features for tkeden*. Retrieved 23 July 2005 from http://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/eden/dtkeden

Beynon, W. M., Boyatt, R. C., and Russ, S. B. (2006). *Rethinking Programming*. In ITNG '06: Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06), 149-154, Washington, DC, USA. IEEE Computer Society.

Beynon, W. M., Boyatt, R., and Chan, Z. E. (2008). *Intuition in Software Development Revisited*. In Proceedings of 20th Annual Psychology of Programming Interest Group Conference.

Beynon, W. M. (1983). *A Definition of the ARCA Notation*. Research Report 54, Department of Computer Science, University of Warwick. Coventry, United Kingdom. 1983.

Beynon, W. M. (1993). *Handouts for the cricket simulation project in CS223 Introduction to Software Engineering (1992/93)*. Department of Computer Science, University of Warwick. Coventry, United Kingdom. 1993.

Beynon, W. M. (2005). Radical Empiricism, Empirical Modelling and the nature of knowing. *Pragmatics and Cognition*, 13(3), 615-646.

Beynon, W. M. (2008). *Lecture Notes for CS405 Introduction to Empirical Modelling (2008/9)*. http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405/

Beynon, W. M. (2009). *A glossary for Empirical Modelling*. Retrieved 15 April 2009 from http://www2.warwick.ac.uk/fac/sci/dcs/research/em/intro/glossary

Beynon, W. M. and Cartwright, R. I. (1995). *Empirical Modelling Principles for Cognitive Artefacts*. In Proc. IEE Colloquium: Design Systems with Users in Mind: The Role of Cognitive Artefacts.

Beynon, W. M. and Chan, Z. E. (2006). *A conception of computing technology better suited to Distributed Participatory Design*. In Distributed Participatory Design Workshop in NordiCHI 2006.

Beynon, W. M. and Chan, Z. E. (2009). *Computing for construals in distributed participatory design - principles and tools*. Research Report CS-RR-444, Department of Computer Science, University of Warwick, United Kingdom. June, 2009.

Beynon, W. M. and Harfield, A. (2007). Lifelong Learning, Empirical Modelling and the Promises of Constructivism. *Journal of Computers*, 2(3), 43-55.

Beynon, W. M. and Maad, S. (2000). *{Integrated Environments for Virtual Collaboration: an Empirical Modelling Perspective}*. In Proceedings of the Fifth World Conference On Integrated Design & Process Technology, Texas.

Beynon, W. M. and Russ, S. B. (1994). *Empirical Modelling of requirements*. Research Report CS-RR-277, Department of Computer Science, University of Warwick. Coventry, United Kingdom. 1994.

Beynon, W. M. and Steve, B. R. (2008). Experimenting with computing. *Journal of Applied Logic*, 6(4), 476 - 489. (Also available at: http://www.sciencedirect.com/science/article/B758H-4TJX1TR-1/2/b11d253dbcea524885ecd96f51dd00c2)

Beynon, W. M. and Sun, P. H. (1998). *Interactive Situation Models for Program Comprehension*. Research Report CS-RR-352, Department of Computer Science, University of Warwick. Coventry, United Kingdom. September, 1998.

Beynon, W. M. and Sun, P. H. (1999). *Computer-mediated communication: a Distributed Empirical Modelling perspective*. In Proceedings of CT'99, San Francisco.

Biddle, B. J. and Thomas, E. J. (1966). *Role theory : concepts and research*. New York: Wiley.

Blackwell, A. F. (2002). *First Steps in Programming: A Rationale for Attention Investment Models*. In HCC '02: Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02), 2, Washington, DC, USA. IEEE Computer Society.

Blackwell, A. F. (2004). End-user developers at home. *Commun. ACM*, 47(9), 65-66.

Bødker, K., Kensing, F., and Simonsen, J. (2004). *Participatory IT Design: Designing for Business and Workplace Realities*. Cambridge, MA, USA: MIT Press.

Bødker, S. (2006). *When second wave HCI meets third wave challenges*. In NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction, 1-8, New York, NY, USA. ACM.

Boehm, B. (1986). A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4), 14-24.

Bohr, N. (1958). *Atomic physics and human knowledge*. Wiley.

Borghoff, U. M. and Schlichter, J. H. (2000). *Computer-Supported Cooperative Work: introduction to distributed applications*. Springer.

Bornat, R. (1990). *Understanding and writing compilers: a do-it-yourself guide*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc..

Boyatt, R., Harfield, A., and Beynon, W. M. (2006). *Learning about and through Empirical Modelling*. In ICALT '06: Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies, 662-666, Washington, DC, USA. IEEE Computer Society.

Brand, S. (1995). *How Buildings Learn: What Happens After They're Built*. New York: Penguin Books.

Brooks, Jr., F. P. (1987). No Silver Bullet Essence and Accidents of Software Engineering. *Computer*, 20(4), 10-19.

Brooks, Jr., F. P. (1995). *The mythical man-month (anniversary ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc..

Brooks, I. (1999). *Organisational behaviour : individuals, groups and the organisation*. Financial Times Pitman.

Brown, J. S., Duguid, P., and Haviland, S. (1994). Toward Informed Participation: Six Scenarios in Search of Democracy in the Information Age. *The Aspen Institute Quarterly*, 6(4), 49-73.

Cannon-Bowers, J. A. and Salas, E. (2001). Reflections on shared cognition. *Journal of Organizational Behavior*, 22(2), 195-202.

Carroll, J. M., Rosson, M. B., Convertino, G., and Ganoe, C. H. (2006). Awareness and teamwork in computer-supported collaborations. *Interact. Comput.*, 18(1), 21-46.

Carter, B. (2000). *Sasami project report: A 3D display engine for the EDEN modelling environment*. Final year undergraduate project report, Department of Computer Science, University of Warwick. Coventry, United Kingdom. 2000.

Cartwright, A. (1994). *Application of Definitive Scripts to CACD*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. July, 1994.

Cartwright, R. (1999). *Geometric Aspects of Empirical Modelling: Issues in Design and Implementation*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. May, 1999.

Cavaye, A. L. M. (1995). User Participation in System Development Revisited. *Information and Management*, 28, 311-323.

Chalmers, M. (2002). Awareness, Representation and Interpretation. *Computer Supported Cooperative Work (CSCW)*, 11(3 - 4), 389 - 409.

Chan, Z. E. (2005). *Distributed Cognition in Distributed Empirical Modelling*. Poster for Warwick Postgraduate Colloquium in Computer Science (WPCCS `05). Department of Computer Science, University of Warwick, United Kingdom. June, 2005.

Chan, Z. E. (2006). *Groupware Development: an Empirical Modelling approach*. Presentation for Warwick Postgraduate Colloquium in Computer Science (WPCCS `06). Department of Computer Science, University of Warwick. Coventry, United Kingdom. June, 2006.

Chan, Z. E. (2008). *The Co-evolution of system and developer's understand in practising an Empirical Modelling approach to systems development: a case study on cricket simulation*. Presentation for Warwick Postgraduate Colloquium in Computer Science (WPCCS `08). Department of Computer Science, University of Warwick. Coventry, United Kingdom. June, 2008.

Checkland, P. B. (1981). *Systems Thinking, Systems Practice*. Chichester: John Wiley & Sons.

Chen, Y. (2001). *Empirical Modelling for Participative Business Process Reengineering*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. December, 2001.

Clark, A. (1997). *Being there: Putting brain, body, and world together again*. Cambridge, MA: The MIT Press.

Cluts, M. M. (2003). *The evolution of artifacts in cooperative work: constructing meaning through activity*. In GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work, 144-152, New York, NY, USA. ACM.

Constantine, L. L. (1989). *Teamwork paradigms and the structured open team*. In Freeman, M. (Ed.), Embedded Systems Conference, San Francisco, Califonria.

Constantine, L. L. (1991). *Building Structured Open Teams to Work*. In Software Development '91 Proceedings, San Francisco. Miller Freeman.

Constantine, L. L. (1993). Work organization: paradigms for project management and organization. *Commun. ACM*, 36(10), 35-43.

Cooley, M. (1987). *Architect Or Bee?: The Human Price Of Technology*. Hogarth Press Eondon.

Cox, B. (1990). There Is a Silver Bullet. *Byte*, 209-218.

Crabtree, A. (2003). *Designing Collaborative Systems: a pratical guide to ethnography*. Springer.

Craig, I. D. (1993). *A New Interpretation of the Blackboard Architecture*. Research Report CS-RR-254, Department of Computer Science, University of Warwick. Coventry, United Kingdom. October, 1993.

Craig, I. (1995). *Blackboard systems*. Norwood, N.J.: Ablex Publishing Corporation.

Danielsson, K., Naghsh, A. M., and Dearden, A. (2006). *Distributed Participatory Design Workshop in NordiCHI 2006*. http://extra.shu.ac.uk/paperchaste/dpd/nordichi.html

Danielsson, K., Naghsh, A. M., Warr, A., and Gumm, D. (2008a). *Distributed Participatory Design Workshop in CHI 2008*. http://extra.shu.ac.uk/paperchaste/dpd/chi2008/index.html

Danielsson, K., Naghsh, A. M., Gumm, D., and Warr, A. (2008b). *Distributed participatory design*. In CHI '08: CHI '08 extended abstracts on Human factors in computing systems, 3953-3956, New York, NY, USA. ACM.

Danielsson, K. (2004). *The shift from user, to learner, to participant: An inevitable development of (just a) mere coincidence*. In Proceedings of the Participatory Design Conference (PDC-04), Toronto, Canada. CPSR.

Davis, G. B. (1984). *Caution: User Developed Systems Can be Dangerous to Your Organization*. MIS Research Center, School of Management, University of Minnesota. Minneapolis, USA. 1984.

DePaula, R. (2004). *Lost in translation: a critical analysis of actors, artifacts, agendas, and arenas in participatory design*. In PDC 04: Proceedings of the eighth conference on Participatory design, 162-172, New York, NY, USA. ACM Press.

de Souza, C. R. B., Redmiles, D., Cheng, L., Millen, D., and Patterson, J. (2004). *Sometimes you need to see through walls: a field study of application programming interfaces*. In CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work, 63-71, New York, NY, USA. ACM.

de Vreede, G. and Guerrero, L. A. (2006). Editorial: Theoretical and empirical advances in groupware research. *International Journal of Human-Computer Studies*, 64(7), 571-572.

Divitini, M., Sale, G. O., Pozzoli, A., and Simone, C. (1993). *Supporting the dynamics of knowledge sharing within organizations*. In COCS '93: Proceedings of the conference on Organizational computing systems, 178-183, New York, NY, USA. ACM.

D'Ornellas, H. P. (1998). *Agent Oriented Modelling for Collaborative Group Learning*. MSc Project Report, Department of Computer Science, University of Warwick. 1998.

Dourish, P. (1995). Developing a reflective model of collaborative systems. *ACM Trans. Comput.-Hum. Interact.*, 2(1), 40-63.

Dourish, P. (2001). *Where the action is: the foundations of embodied interaction*. Cambridge, MA, USA: The MIT Press.

Dourish, P. (2003). The Appropriation of Interactive Technologies: Some Lessons from Placeless Documents. *Comput. Supported Coop. Work*, 12(4), 465-490.

Dourish, P. (2006). *Implications for design*. In CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, 541-550, New York, NY, USA. ACM.

Dourish, P. and Bellotti, V. (1992). *Awareness and coordination in shared workspaces*. In CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work, 107-114, New York, NY, USA. ACM Press.

Dourish, P. and Button, G. (1998). On "Technomethodology";: foundational relationships between ethnomethodology and system design. *Hum.-Comput. Interact.*, 13(4), 395-432.

Duffy, B. (2005). The analysis of documentary evidence. In Bell, J. (Ed.), *Doing your Research Project*. Open University Press.

Efstathiou, G. (2006). *C-GRAPH: A case study in the design, implementation and application of a definitive notation*. MSc thesis, Department of Computer Science, University of Warwick. Coventry, United Kingdom. 2006.

Ellis, C. A., Gibbs, S. J., and Rein, G. (1991). Groupware: some issues and experiences. *Communications of the ACM*, 34(1), 39-58.

EMPA (1987-2009). *Empirical Modelling Projects Archive*. Retrieved 28 July 2009 from http://www2.warwick.ac.uk/fac/sci/dcs/research/em/projects/

Evans, M., Beynon, W. M., and Fischer, C. N. (2001). *Empirical Modelling for the Logistics of Rework in the Manufacturing Process*. In Proc 16th Brazilian Congress of Mechanical Engineering, 226-234.

Farooq, U., Carroll, J. M., and Ganoe, C. H. (2005). *Supporting creativity in distributed scientific communities*. In GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work, 217-226, New York, NY, USA. ACM.

Fischer, G., McCall, R., Ostwald, J., Reeves, B., and Shipman, F. (1994). *Seeding, evolutionary growth and reseeding: supporting the incremental development of design environments*. In CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems, 292-298, New York, NY, USA. ACM.

Fischer, G., Giaccardi, E., Eden, H., Sugimoto, M., and Ye, Y. (2005). Beyond binary choices: integrating individual and social creativity. *Int. J. Hum.-Comput. Stud.*, 63(4-5), 482-512.

Fischer, G. (1999). *Symmetry of igorance, social creativity, and meta-design*. In C&C '99: Proceedings of the 3rd conference on Creativity & cognition, 116-123, New York, NY, USA. ACM Press.

Fischer, G. (2005). *Social creativity: making all voices heard*. In Proceedings of Human Computer Interaction International Conference (HCII), Las Vegas.

Fischer, G. (2008). Rethinking software design in participation cultures. *Automated Software Engg.*, 15(3-4), 365-377.

Fischer, G. (2009). *End-User Development and Meta-design: Foundations for Cultures of Participation*. In IS-EUD '09: Proceedings of the 2nd International Symposium on End-User Development, 3-14, Berlin, Heidelberg. Springer-Verlag.

Fischer, G. and Ostwald, J. (2002). *Transcending the Information Given: Designing Learning Environments for Informed Participation*. In ICCE '02: Proceedings of the International Conference on Computers in Education, 378, Washington, DC, USA. IEEE Computer Society.

Fischer, G. and Scharff, E. (2000). *Meta-design: design for designers*. In DIS '00: Proceedings of the 3rd conference on Designing interactive systems, 396-405, New York, NY, USA. ACM.

Fjuk, A., Smørdal, O., and Nurminen, M. I. (1997). *Taking Articulation Work Seriously - an Activity Theoretical Approach*. Turku Centre for Computer Science. 1997.

Flick, U. (1998). An Introduction to Qualitative Research. SAGE Publications.

Flores, F., Graves, M., Hartfield, B., and Winograd, T. (1988). Computer systems and the design of organizational interaction. *ACM Trans. Inf. Syst.*, 6(2), 153-172.

Flor, N. V. and Hutchins, E. (1991). *Analysing Distributed cognition in software teams*. In Jurgen Koenemann-Belliveau, Thomas G. Moher, S. P. R. (Ed.), Proceedings of Empirical Studies of Programmers: fourth workshop, Norwood, N. J.. Ablex Publishing.

Floyd, C., Reisin, F., and Schmidt, G. (1989). *STEPS to Software Development with Users*. In ESEC '89: Proceedings of the 2nd European Software Engineering Conference, 48-64, London, UK. Springer-Verlag.

Floyd, C., Keil-Slawik, R., Budde, R., and Zullighoven, H. (1992). *Software Development and Reality Construction*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.. Illustrator-C. Weiler-Kuhn.

Fraser, S. D., Brooks, Jr., F. P., Fowler, M., Lopez, R., Namioka, A., Northrop, L., Parnas, D. L., and Thomas, D. (2007). *"No silver bullet" reloaded: retrospective on "essence and accidents of software engineering"*. In OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, 1026-1030, New York, NY, USA. ACM.

Friis, S. (1984). *Prototyping and User Developed Requirements Specifications*. In M., S. (Ed.), *Report of the Seventh Scandinavian Research Seminar on Systemeering*. Helsinki, Finland: Helsinki School of Economics.

Friis, S. (1988). Action research on systems development: case study of changing actor roles. *SIGCAS Comput. Soc.*, 18(1), 22-31.

Garfinkel, H. (1967). *Studies in Ethnomethdology*. New York: Prentice Hall.

Gasson, S. (1998). *Framing design: a social process view of information system development*. In ICIS '98: Proceedings of the international conference on Information systems, 224-236, Atlanta, GA, USA. Association for Information Systems.

Gasson, S. (1999). The reality of user-centered design. *J. End User Comput.*, 11(4), 5-15.

Gasson, S. (2003). Human-centered vs. user-centered approaches to information system design. *JITTA : Journal of Information Technology Theory and Application*, 5(2), 29-46.

Gibbs, S. J. (1989). *LIZA: an extensible groupware toolkit*. In CHI '89: Proceedings of the SIGCHI conference on Human factors in computing systems, 29-35, New York, NY, USA. ACM Press.

Gill, K. S. (1991). Summary Of Human-Centred Systems Research In Europe, Part 1. *Systemist, the journal of the UK Systems Society*, 13(1), 7-27.

Goffman, E. (1961). *The presentation of self in everyday life*. New York: Anchor-Doubleday.

Goffman, E. (1971). *Relations in public*. New York: Basic Books.

Gooding, D. (1990). *Experiment and the making of meaning*. Kluwer Academic Publishers.

Greenbaum, J. (1993). *A Design of One's Own: Towards Participatory Design in the United States*. In Schuler, D. and Namioka, A. (Eds.), *Participatory Design: Principles and Practices*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc..

Greenberg, S. and Roseman, M. (2003). *Using a Room Metaphor to Ease Transitions in Groupware*. In M. Ackerman, V. Pipek, V. W. (Ed.), *Sharing Expertise: Beyond Knowledge Management*, 203-256. Cambridge, MA: MIT Press.

Greif, I. (1988). *Computer-supported cooperative work: a book of readings*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Grinter, R. E. (1998). *Recomposition: putting it all back together again*. In CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work, 393-402, New York, NY, USA. ACM.

Grinter, R. E. (2000). Workflow Systems: Occasions for Success and Failure. *Comput. Supported Coop. Work*, 9(2), 189-214.

Grønbaek, K., Kyng, M., and Mogensen, P. (1993). CSCW challenges: cooperative design in engineering projects. *Commun. ACM*, 36(6), 67-77.

Grudin, J. and Grinter, R. E. (1995). Ethnography and design. *Comput. Supported Coop. Work*, 3(1), 55-59.

Grudin, J. (1988). *Why CSCW applications fail: problems in the design and evaluation of organization of organizational interfaces*. In CSCW '88: Proceedings of the 1988 ACM conference on Computer-supported cooperative work, 85-93, New York, NY, USA. ACM Press.

Grudin, J. (1990). *Interface*. In CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work, 269-278, New York, NY, USA. ACM.

Grudin, J. (1991a). *CSCW: the convergence of two development contexts*. In CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems, 91-97, New York, NY, USA. ACM.

Grudin, J. (1991b). Interactive Systems: Bridging the Gaps Between Developers and Users. *Computer*, 24(4), 59-69.

Grudin, J. (1993). *Obstacles to Participatory Design in Large Product Development Organizations*. In Schuler, D. and Namioka, A. (Eds.), *Participatory Design: Principles and Practices*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc..

Grudin, J. (1994a). Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, 37(1), 92-105.

Grudin, J. (1994b). Computer-Supported Cooperative Work: History and Focus. *Computer*, 27(5), 19-26.

Grudin, J. and Grinter, R. E. (1995). Ethnography and design. *Comput. Supported Coop. Work*, 3(1), 55-59.

Gumm, D. C. (2006). *Distributed Participatory Design: An inherent Paradoxon?*. In Proceedings of IRIS29, Helsingoer, Denmark.

Gutwin, C. (1997). *Workspace awareness in real-time distributed groupware*. Ph.D. dissertation, University of Calgary. 1997.

Gutwin, C., Penner, R., and Schneider, K. (2004). *Group awareness in distributed software development*. In CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work, 72-81, New York, NY, USA. ACM Press.

Hansen, M. T., Nohria, N., and Tierney, T. (1999). What's Your Strategy for Managing Knowledge?. *Harvard Business Review*.

Hanson, N. R. (1958). *Patterns of Discovery: An Inquiry into the Conceptual Foundations of Science*. Cambridge University Press.

Harel, D. (1992). Biting the Silver Bullet: Toward a Brighter Future for System Development. *Computer*, 25(1), 8-20.

Harfield, A., Ward, A., and Chan, Z. E. (2005). *Recreating an experience of the Clayton Tunnel railway accident in 1861*. Poster for EU TEL Kaleidoscope Network of Excellence Showcase 2005 in Oberhausen. 2005.

Harfield, A. (2002). *Agent-oriented parsing wih Empirical Modelling*. Final year undergraduate project report, Department of Computer Science, University of Warwick. Coventry, United Kingdom. 2002.

Harfield, A. J. (2008). *Empirical Modelling as a new paradigm for educational technology*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. January, 2008.

Harper, R. H. R. (2000). The Organisation in Ethnography -A Discussion of Ethnographic Fieldwork Programs in CSCW. *Comput. Supported Coop. Work*, 9(2), 239-264.

Hayashi, K., Hazama, T., Nomura, T., Yamada, T., and Gudmundson, S. (1999). *Activity awareness: a framework for sharing knowledge of people, projects, and places*. In ECSCW'99: Proceedings of the sixth conference on European Conference on Computer Supported Cooperative Work, 99-118, Norwell, MA, USA. Kluwer Academic Publishers.

He, J. and King, W. (2008). The Role of User Participation in Information Systems Development: Implications from a Meta-Analysis. *J. Manage. Inf. Syst.*, 25(1), 301-331.

Henderson, A. and Kyng, M. (1992). *There's no place like home: continuing design in use*. In *Design at work: cooperative design of computer systems*, 219-240. Hillsdale, NJ, USA: L. Erlbaum Associates Inc..

Heron, T. (2002). *Programming with Dependency*. Masters thesis, Department of Computer Science, University of Warwick. Coventry, United Kingdom. September, 2002.

Hildenbrand, T., Rothlauf, F., Geisser, M., Heinzl, A., and Kude, T. (2008). *Approaches to Collaborative Software Development*. In CISIS '08: Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems, 523-528, Washington, DC, USA. IEEE Computer Society.

von Hippel, E. (1986). Lead users: a source of novel product concepts. *Manage. Sci.*, 32(7), 791-805.

Howcroft, D. and Wilson, M. (2003). Participation: `bounded freedom' or hidden constraints on user involvement. *New Technology, Work and Employment*, 18(1).

Hughes, J., King, V., Rodden, T., and Andersen, H. (1994). *Moving out from the control room: ethnography in system design*. In CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work, 429-439, New York, NY, USA. ACM.

Hughes, J., King, V., Rodden, T., and Andersen, H. (1995). The role of ethnography in interactive systems design. *interactions*, 2(2), 56-65.

Hummes, J. and Merialdo, B. (2000). Design of Extensible Component-Based Groupware. *Computer Supported Cooperative Work*, 9(1), 53-74.

Hutchins, E. (1995). *Cognition in the Wild*. MIT Press.

Iivari, N. (2006). 'Representing the User' in software development-a cultural analysis of usability work in the product development context. *Interact. Comput.*, 18(4), 635-664.

Ishii, H. and Kobayashi, M. (1992). *ClearBoard: a seamless medium for shared drawing and conversation with eye contact*. In CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems, 525-532, New York, NY, USA. ACM.

Jackson, M. (2001). *Problem frames: analyzing and structuring software development problems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc..

James, W. (1912). *Essays in radical empiricism*. Longmans, Green and Co..

Janis, I. L. (1982). *Groupthink: Psychological studies of policy decisions and fiascoes* (2nd). Boston: Houghton Mifflin.

Johansen, R. (1988). *GroupWare: Computer Support for Business Teams*. New York, NY, USA: The Free Press.

Johnson-Lentz, P. and Lentz, T. J. (1982). *Groupware: The process and impacts of design choices*. In Kerr, E. B. and Hiltz, S. R. (Eds.), *Computer-Mediated Communication Systems: Status and Evaluation*. New York, N. Y.: Academic Press.

Jullien, F. (1995). *The Propensity of Things: Toward a History of Efficacy in China*. New York: Zone Books. (Translated by Janet Lloyd).

Kaptelinin, V., Nardi, B., B\odker, S., Carroll, J., Hollan, J., Hutchins, E., and Winograd, T. (2003). *Post-cognitivist HCI: second-wave theories*. In CHI '03 extended abstracts on Human factors in computing systems, 692-693, New York, NY, USA. ACM.

Kaptelinin, V. and Nardi, B. A. (2006). *Acting with Technology: Activity Theory and Interaction Design*. The MIT Press.

Karasti, H. (2001). Bridging Work Practice and System Design: Integrating Systemic Analysis, Appreciative Intervention and Practitioner Participation. *Comput. Supported Coop. Work*, 10(2), 211-246.

Kensing, F. and Blomberg, J. (1998). Participatory Design: Issues and Concerns. *Comput. Supported Coop. Work*, 7(3-4), 167-185.

King, K. G. (2004). *Timetabling with Empirical Modelling Principles and Tools*. Final year undergraduate project report, Department of Computer Science, University of Warwick, United Kingdom. Coventry, United Kingdom. 2004.

King, K. G. (2007). *Uncovering Empirical Modelling*. Masters thesis, Department of Computer Science, University of Warwick. Coventry, United Kingdom. January, 2007.

Kirsch-Pinheiro, M., de Lima, J. V., and Borges, M. R. S. (2003). A framework for awareness support in groupware systems. *Comput. Ind.*, 52(1), 47-57.

Kuhn, S. (1996). Design for people at work. , 273-294.

Kuutti, K. (2001). *Hunting for the lost user: From sources of errors to active actors - and beyond*. Paper written for the Cultural Usability seminar, Media Lab, University of Art and Design Helsinki, 24 April 2001. 2001.

Lamb, R. and Kling, R. (2003). Reconceptualizing Users as Social Actors in Information Systems Research. *MIS Quarterly*, 27(2), 197-235.

Lave, J. and Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge, UK: Cambridge University Press.

Lee, C. P. (2005). *Between chaos and routine: boundary negotiating artifacts in collaboration*. In ECSCW'05: Proceedings of the ninth conference on European Conference on Computer Supported Cooperative Work, 387-406, New York, NY, USA. Springer-Verlag New York, Inc..

Letondal, C. and Mackay, W. E. (2004). *Participatory programming and the scope of mutual responsibility: balancing scientific, design and software commitment*. In PDC 04: Proceedings of the eighth conference on Participatory design, 31-41, New York, NY, USA. ACM.

Lewis, J. (2005). *Project Planning, Scheduling & Control: A Hands-On Guide to Bringing Projects in on Time and on Budget* (4th Edition). McGraw-Hill.

Lewis, C. and Rieman, J. (1994). *Task-Centered User Interface Design: A Practical Introduction*. University of Colorado, Boulder, USA. (Also available at: http://hcibib.org/tcuid/)

Maad, S. (2002). *An Empirical Modelling Approach to Software System Development in Finance: Applications and Prospects*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. March, 2002.

Mackay, W. E. (1990). *Users and customizable software: a co-adaptive phenomenon*. Ph.D. dissertation, MIT. 1990.

Mambrey, P. and Pipek, V. (1999). *Enhancing participatory design by multiple communication channels*. In Proceedings of the HCI International '99 (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction, 387-391, Mahwah, NJ, USA. Lawrence Erlbaum Associates, Inc..

Lynne Markus, M. and Connolly, T. (1990). *Why CSCW applications fail: problems in the adoption of interdependent work tools*. In CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work, 371-380, New York, NY, USA. ACM.

MCC (1990). *Cricket* (10th Edition). London, UK: Marylebone Cricket Club, A & C Black Publishers.

McCarthy, J. and Wright, P. (2004). *Technology as Experience*. The MIT Press.

McGrath, J. E. (1984). *Groups: Interaction and Performance*. Prentice Hall.

McLoughlin, M., Ciolfi, L., Fraser, M., Hornecker, E., and Bowers, J. (2009). *Bridging "Interaction Clouds": Exploring collaborative interaction across assemblies of mobile and embedded technology (Workshop at ECSCW 2009)*. . 2009.

Merton, R. K. (1968). *Social theory and social structure*. Collier-Macmillan.

Meyer, B. (1997). *Object-oriented software construction*. Prentice Hall.

Mørch, A., Stiemerling, O., and Wulf, V. (1998). Tailorable groupware: issues, methods, and architectures report of a workshop held at GROUP'97, Phoenix, AZ, Nov. 16th, 1997. *SIGGROUP Bull.*, 19(1), 4-7.

Mullen, B., Johnson, C., and Salas, E. (1991). Productivity Loss in Brainstorming Groups: A Meta-Analytic Integration. *Basic and Applied Social Psychology*, 12(1), 3-23.

Muller, M. (2001). *A participatory poster of participatory methods*. In CHI '01: CHI '01 extended abstracts on Human factors in computing systems, 99-100, New York, NY, USA. ACM.

Mumford, E. (1983). *Designing Participatively*. Manchester UK: Manchester Business School.

Mumford, E. (1993). The ETHICS approach. *Commun. ACM*, 36(6), 82.

Myers, B. A., Burnett, M. M., Wiedenbeck, S., and Ko, A. J. (2007). *End user software engineering: CHI 2007 special interest group meeting*. In CHI '07: CHI '07 extended abstracts on Human factors in computing systems, 2125-2128, New York, NY, USA. ACM.

Myers, B. A., Ko, A. J., and Burnett, M. M. (2006). *Invited research overview: end-user programming*. In CHI '06: CHI '06 extended abstracts on Human factors in computing systems, 75-80, New York, NY, USA. ACM.

Myers, R. (2008). *Web EDEN*. Final year undergraduate project report, Department of Computer Science, University of Warwick. 2008.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y. (2002). *Evolution patterns of open-source software systems and communities*. In IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution, 76-85, New York, NY, USA. ACM Press.

Nardi, B. A. (1993). *A small matter of programming: perspectives on end user computing*. Cambridge, MA, USA: The MIT Press.

Nardi, B. A. and Miller, J. R. (1990). *The spreadsheet interface: A basis for end user programming*. In INTERACT '90: Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction, 977-983, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co..

Naur, P. (1985a). *Intuition in software development*. In Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT) on Formal Methods and Software, Vol.2: Colloquium on Software Engineering (CSE), 60-79, New York, NY, USA. Springer-Verlag New York, Inc..

Naur, P. (1985b). Programming as Theory Building. *Microprocessing and Microprogramming*, 15, 253-261.

Naur, P. (2007). Computing versus human thinking. *Commun. ACM*, 50(1), 85-94.

Nemeth, C. J. and Nemeth-Brown, B. (2003). *Better than Individuals? The Potential Benefits of Dissent and Diversity for Group Creativity*. In Paulus, P. B. and Nijstad, B. A. (Eds.), *Group creativity : innovation through collaboration*. New York: Oxford University Press.

Ness, P. E., Beynon, W. M., and Yung, Y. P. (1994). *Applying Agent-oriented Design to a Sail Boat Simulation*. In Proc. ESDA 1994, 1-8.

Ness, P. E. (1997). *Creative Software Development: An Empirical Modelling Framework*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. October, 1997.

Newell, A. (1962). *Some problems of the basic organization in problem-solving programs*. In Yovits, M.C., J. G. T. and Goldstein, G. D. (Eds.), Proc. Second Conference on Self-Organizing Systems, 393-423. Spartan Books.

Newman, B. M. and Newman, P. R. (2007). *Theories of human development*. Mahwah, N.J.: Lawrence Erlbaum Associates.

Bernard A. Nijstad, E. F. R. and Stroebe, W. (2006). *Four Principles of Group Creativity*. In Leigh L. Thompson, H. C. (Ed.), *Creativity and innovation in organizational teams*. Mahwah, N.J.: Lawrence Erlbaum Assoicates.

Noble, J., Taivalsaari, A., and Moore, I. (2001). *Prototype-Based Programming: Concepts, Languages and Applications*. Springer.

Norman, D. A. (2002). *The design of everyday things*. Basic Books.

Norman, D. A. and Draper, S. W. (1986). *User Centered System Design: New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc..

ODS (2009). *International Workshop on Open Design Spaces supporting User Innovation (ODS '09)*. Retrieved 1 March 2009 from http://www.cwe-projects.eu/pub/bscw.cgi/d1493593-2/*/*/Home.htm#motivation

OED (2008). *Oxford English Dictionary (Online Edition)*. Oxford Press. Retrieved 1 August 2008 from http://www.oed.com/

Olson, M. and Ives, B. (1981). User Involvement in Systems Design: an Empirical Test of Alternative Approaches. *Information and Management*, 4(4), 183-196.

Orlikowski, W. and Hofman, D. (1997). An improvisational model of change management: the case of groupware technologies. *Sloan Manag Review*, 38(2), 11-22. (Also available at: Available at: http://ccs.mit.edu/papers/CCSWP191/CCSWP191.html)

Ostwald, J. L. (1996). *Knowledge construction in software development: the evolving artifact approach*. Ph.D. dissertation, University of Colorado. Boulder, CO, USA. 1996.

Palen, L. A. (1997). *Groupware adoption & adaptation*. In CHI '97: CHI '97 extended abstracts on Human factors in computing systems, 67-68, New York, NY, USA. ACM.

Parnas, D. L. (1978). *Designing software for ease of extension and contraction*. In ICSE '78: Proceedings of the 3rd international conference on Software engineering, 264-277, Piscataway, NJ, USA. IEEE Press.

Paulus, P. B., Nakui, T., and Putman, V. L. (2006). *Group Brainstorming and Teamwork: Some Rules for the Road to Innovation*. In Leigh L. Thompson, H. C. (Ed.), *Creativity and innovation in organizational teams*. Mahwah, N.J.: Lawrence Erlbaum Assoicates.

PDC (2008). *Participatory Design Conference: experiences & challenges*. Retrieved 28 January 2009 from http://www.pdc2008.org/

Peirce, C. S. (1931-1958). *Collected Papers of Charles Sanders Peirce*. Cambridge, MA: Harvard University Press.

Pekkola, S., Kaarilahti, N., and Pohjola, P. (2006). *Towards formalised end-user participation in information systems development process: bridging the gap between participatory design and ISD methodologies*. In PDC '06: Proceedings of the ninth conference on Participatory design, 21-30, New York, NY, USA. ACM.

Penichet, V. M. R., Marin, I., Gallud, J. A., Lozano, M. D., and Tesoriero, R. (2007). A Classification Method for CSCW Systems. *Electron. Notes Theor. Comput. Sci.*, 168, 237-247.

Petre, M. and Winder, R. (1990). On languages, models and programming styles. *Comput. J.*, 33(2), 173-180.

Poole, M. S. and DeSanctis, G. (1990). *Understanding the use of group decision support systems: The theory of adaptive structuration*. In Fulk, J. and Steinfeld, C. (Eds.), *Organizations and Communication Technology*. Newbury Park, CA: Sage.

Quintana, C., Carra, A., Krajcik, J., and Soloway, E. (2001). . In Carroll, J. M. (Ed.), *Learner-centered design: Reflections and new directions*, 605-626. New York: ACM Press.

Raeithel, A. (1996). *From Coordinatedness to Coordination via Cooperation and Co-construction*. In Workshop on Work and Learning in Transition, San Diego.

Rasmequan, S. (2001). *An Approach to Computer-based Knowledge Representation for the Business Environment using Empirical Modelling*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. November, 2001.

Rettig, M. (1990). Practical programmer: software teams. *Commun. ACM*, 33(10), 23-27.

Rittel, H. J. and Webber, M. M. (1973). "Dilemmas in a General Theory of Planning". *Policy Sciences*, 4, 155-169.

Roe, C. (2003). *Computers for Learning: An Empirical Modelling Perspective*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. November, 2003.

Rogers, Y., Scaife, M., and Rizzo, A. (2003). *Interdisciplinarity:an Emergent or Engineered Process?*. Cognitive Science Research Paper 556, School of Cognitive and Computing Sciences, University o Sussex. Sussex, Brighton, United Kingdom. 2003.

Rogers, Y. (1994). *Exploring obstacles: integrating CSCW in evolving organisations*. In CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work, 67-77, New York, NY, USA. ACM.

Rogers, Y. (2008). Discussion: 57 Varieties of Activity Theory. *Interact. Comput.*, 20(2), 247-250.

Rönkkö, K. (2007). Interpretation, interaction and reality construction in software engineering: An explanatory model. *Inf. Softw. Technol.*, 49(6), 682-693.

Rungrattanaubol, J. (2002). *A Treatise on Modelling with Definitive Scripts*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. April, 2002.

Ryle, G. (1949). *The Concept of Mind*. Penguin.

Sachs, P. (1995). Transforming work: collaboration, learning, and design. *Commun. ACM*, 38(9), 36-44.

Salles, P. and Bredeweg, B. (2003). *A Case Study of Collaborative Modelling: building qualitative models in ecology*. In Hoppe, U., Verdejo, F., and Kay, J. (Eds.), Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies, 245-252, Osaka, Japan. IOS-Press/Ohmsha.

Scaffidi, Christopher, Shaw, Mary, and Myers, Brad (2005). *Estimating the Numbers of End Users and End User Programmers*. In VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, 207-214, Washington, DC, USA. IEEE Computer Society.

Schmidt, K. and Bannon, L. (1992). Taking CSCW Seriously: Supporting Articulation Work. *Computer Supported Cooperative Work*, 1, 7-40.

Schmidt, K. and Simone, C. (1996). Coordination mechanisms: towards a conceptual foundation of CSCW systems design. *Comput. Supported Coop. Work*, 5(2-3), 155-200.

Schön, D. A. (1983). *The reflective practitioner: how professionals think in action*. New York: Basic Books.

Schuler, D. and Namioka, A. (1993). *Participatory Design: Principles and Practices*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc..

Scott, S. D., Sheelagh, M., Carpendale, T., and Inkpen, K. M. (2004). *Territoriality in collaborative tabletop workspaces*. In CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work, 294-303, New York, NY, USA. ACM.

Shapiro, D. (1994). *The limits of ethnography: combining social sciences for CSCW*. In CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work, 417-428, New York, NY, USA. ACM.

Sheth, C. R. (1998). *An Investigation into the Application of the Distributed Definitive Programming Paradigm in a Teaching Environment: The Development of a Virtual Electrical Laboratory*. MSc Project Report, Department of Computer Science, University of Warwick. Coventry, United Kingdom. September, 1998.

Smith, B. C. (1987). Two lessons of logic. *Computational Intelligence*, 3(1), 214-218.

Soloway, E., Guzdial, M., and Hay, K. E. (1994). Learner-centered design: the challenge for HCI in the 21st century. *interactions*, 1(2), 36-48.

Sommerville, I. (2004). *Software Engineering* (7th Edition). Addison-Wesley.

Sonnenwald, D. H. (1993). *Communication in design*. Ph.D. dissertation, Rutgers University. New Brunswick, NJ, USA. 1993.

Sonnenwald, D. H. (1996). Communication roles that support collaboration during the design process. *Design studies*, 17, 277-301.

Sonnenwald, D. H. (2003). *Expectations for a scientific collaboratory: a case study*. In GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work, 68-74, New York, NY, USA. ACM.

Sonnenwald, D. H., Whitton, M. C., and Maglaughlin, K. L. (2003). Evaluating a scientific collaboratory: Results of a controlled experiment. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(2), 150--176.

Spinuzzi, C. (2003). *Tracing Genres through Organizations: A Sociocultural Approach to Information Design*. The MIT Press.

Spradley, J. P. (1980). *Participant observation*. Holt, Rinehart and Winston.

Stahl, G. (2002). *Computer Support for Collaborative Learning: foundations for a CSCL community*. In Proceedings of CSCL 2002 Boulder, Colorado, USA, Hillsdale, N.J.. Lawrence Erlbaum Associates.

Stahl, G. (2006). *Group Cognition: Computer Support for Building Collaborative Knowledge (Acting with Technology)*. The MIT Press.

Star, S. L. and Griesemer, J. (1989). Institutional Ecology, "Transla-tions," and Boundary Objects. *Social Studies of Science*, 19(3), 387-420.

Star, S. L. and Strauss, A. (1999). Layers of Silence, Arenas of Voice: The Ecology of Visible and Invisible Work. *Comput. Supported Coop. Work*, 8(1-2), 9-30.

Stavash, J. P., Welsh, J., and Chadha, B. (2000). *Collaborative Prototyping and Product Development on the Web*. In Lokeheed Martin Mission Critical Enterprise systems (MCES) Symposium, Orlando, FL.

Stewart, J., Bederson, B. B., and Druin, A. (1999). *Single display groupware: a model for co-present collaboration*. In CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems, 286-293, New York, NY, USA. ACM.

Stiemerling, O., Kahler, H., and Wulf, V. (1997). *How to make software softer--designing tailorable applications*. In DIS '97: Proceedings of the 2nd conference on Designing interactive systems, 365-376, New York, NY, USA. ACM.

Stiemerling, O. and Cremers, A. B. (1998). *Tailorable Component Architectures for CSCW-Systems*. In Proceedings of 6th Euromicro Workshop on Parallel and Distributed Processing, 302-308. IEEE Press.

Strauss, A. (1985). Work and the Division of Labor. *The Sociological Quarterly*, 26(1), 1-19.

Strauss, A. L. (1987). *Qualitative Analysis For Social Scientists*. Cambridge University Press.

Strauss, A. L. (1993). *Continual Permutation of Actions*. Aldine de Gruyter.

Strauss, A. and Corbin, J. M. (1998). *Basics of Qualitative Research: Techniques and Procedure for Developing Grounded Theory* (2nd Edition). SAGE Publications.

Suchman, L. A. (1987). *Plans and situated actions: the problem of human-machine communication*. New York, NY, USA: Cambridge University Press.

Suchman, L. A. (2007). *Human-Machine Reconfigurations: Plans and Situated Actions*. New York, NY, USA: Cambridge University Press.

Sun, P. (1999). *Distributed Empirical Modelling and its Application to Software System Development*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. July, 1999.

Tam, J. and Greenberg, S. (2004). *A Framework for Asynchronous Change Awareness in Collaboratively-Constructed Documents*. In *Lecture Notes in Computer Science* (LNCS 3198). Springer.

Tang, A., Boyle, M., and Greenberg, S. (2004). *Display and presence disparity in Mixed Presence Groupware*. In AUIC '04: Proceedings of the fifth conference on Australasian user interface, 73-82, Darlinghurst, Australia, Australia. Australian Computer Society, Inc..

Tang, J. C. (1991). Findings from Observational Studies of Collaborative Work. *International Journal of Man-Machine Studies*, 34(2), 143-160.

Teege, G., Kahler, H., and Stiemerling, O. (1999). Implementing tailorability in groupware: WACC '99 workshop report. *SIGGROUP Bull.*, 20(2), 57-59.

Teufel, S., Sauter, C., Mühlherr, T., and Bauknecht, K. (1995). *Computerunterstüzte Gruppenarbeit*. Bonn: Addison-Wesley. (in German).

Thomke, S. H. (1998). Managing experimentation in the design of new products. *Management Science*, 44(6), 743-762. (Also available at: Retrieved January 18, 2009, from ABI/INFORM Global database. (Document ID: 32306220))

Thomke, S. H. (2003). . In *Experimentation Matters: Unlocking the Potential of New Technologies for Innovation*. Harvard Business School Press.

Thomsett, R. (1990). Effective project teams: A dilemma, a model, a solution. *American Programmer*.

Törpel, B. (2005). *Participatory design: a multi-voiced effort*. In CC '05: Proceedings of the 4th decennial conference on Critical computing, 177-181, New York, NY, USA. ACM.

Tse, E., Histon, J., Scott, S. D., and Greenberg, S. (2004). Avoiding interference: how people use spatial separation and partitioning in SDG workspaces. , 252-261.

Turski, W. M. and Maibaum, T. S. E. (1987). *The Specification of Computer Programs*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc..

van der Aalst, W. M. P. (2007). Exploring the CSCW spectrum using process mining. *Adv. Eng. Inform.*, 21(2), 191-199.

Vergnaud, G., R. M. (2000). De Renault d'Allonnes à une théorie du schème aujourd'hui.. *Psychologie Française*, 45(1), 35-50.

Wallace, R. M. (1999). *Learners as Users, Users as Learners: What's the Difference?*. Short paper on Learner-Centered Design, prepared for HCIC, February 3-8, 1999. 1999.

Ward, A. T. (2004). *Interaction with Meaningful State: Implementing Dependency on Digital Computers*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. May, 2004.

Wertheimer, M. (1966). *Productive thinking*. London: Associated Book Publishers; Tavistock Publications.

Wilson, P. (1991). *Computer Supported Cooperative Work: An Introduction*. Kluwer Academic Publishers.

Wong, A. K. T. (2003). *Before and Beyond Systems: An Empirical Modelling Approach*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. January, 2003.

Woods, S. K. (2002). *Relationships of user participation, satisfaction, influence, and system acceptance in implementing new computer technology*. Ph.D. dissertation, . West Lafayette, IN, USA. 2002.

Wulf, V. (1999). *"Let's see your search-tool!"--collaborative use of tailored artifacts in groupware*. In GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work, 50-59, New York, NY, USA. ACM.

Wulf, V. and Rohde, M. (1995). *Towards an integrated organization and technology development*. In DIS '95: Proceedings of the 1st conference on Designing interactive systems, 55-64, New York, NY, USA. ACM.

Xiao, L., Merkel, C. B., Nash, H., Ganoe, C., Rosson, M. B., Carroll, J. M., Shon, E., Lee, R., and Farooq, U. (2005). *Students as Teachers and Teachers as Facilitators*. In HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 1, 4.1, Washington, DC, USA. IEEE Computer Society.

Yin, R. K. (2009). *Case Study Research: Design and Methods* (4th Edition). SAGE Publications.

Yung, Y. W. (1987). *EDEN: evaluator of definitive notations*. Final year undergraduate project report, Department of Computer Science, University of Warwick. Coventry, United Kingdom. 1987.

Yung, Y. P. (1988). *Design & Implementation of SCOUT (a definitive notation for describing SCreen layOUT) in EDEN*. Final year undergraduate project report, Department of Computer Science, University of Warwick. Coventry, United Kingdom. 1988.

Yung, Y. W. (1990). *EDEN: An Engine for Definitive Notations*. Masters thesis, Department of Computer Science, University of Warwick. Coventry, United Kingdom. September, 1990.

Yung, Y. P. (1993). *Definitive Programming - a Paradigm for Exploratory Programming*. Ph.D. dissertation, Department of Computer Science, University of Warwick. Coventry, United Kingdom. January, 1993.

Yung, Y. P. (1996). *Agent-Oriented Modelling for Interactive Systems*. Post-doctoral EPSRC Project Report, Department of Computer Science, University of Warwick. Coventry, United Kingdom. July, 1996.

Yung, Y. P. and Yung, Y. W. (1988). *The EDEN handbook*. Department of Computer Science, University of Warwick. 1988.

Appendix A

Questionnaire on Collaborative Modelling with dtkeden

The purpose of this questionnaire is to collect student's perspectives on Collaborative Modelling with dtkeden in order to improve the teaching materials for CS405 next year. We are also looking for volunteers to participate in a Collaborative Modelling study which will likely be organised near the end of term or in the early summer vacation.

Please circle your answer.

1 During CS405 lab sessions, do you normally:

a. Work alone

b. Work in a small group

c. Work with your closest neighbour

2 Have you had any experience with dtkeden?

Yes / No

Please go to Question 3 if you select No.

2.1 How would you rate your skill in using dtkeden?

a. Novice

b. Intermediate

c. Expert

2.2 Do you think dtkeden allows groups to build models more effectively?

3 The study will be about 3 to 4 hours which will be held in one day. Are you interested in participating in a Collaborative Modelling study?

Yes / No

If yes, when would be the most suitable time for you?

Term 3			**July**		
Week 8	**Week 9**	**Week 10**	**late**	**Mid**	**late**

4 Contact Detail

Name:

Email / Phone:

Appendix B

Workshop on Collaborative Modelling with dtkeden

In this workshop, you will first spend an hour in building an extension of the jugs model. Then, you will spend rest of the time to tackle Sudoku puzzles collaboratively with the Sudoku model.

1. A Taste of Collaborative Modelling – The Jugs Model (1 hour)

Similar to exercise in lab 3, you need to construct additional jugs in this exercise, but one additional jug for one user. For example, if there are 3 people in your group, you need to make 3 additional jugs. The additional jugs must be labelled with the name of the human agent, e.g., Ant, Charlie, and Eric. There should be separate Pour buttons for each additional jug, but no fill buttons. Therefore, filling these labelled jugs must be through pouring from Jug A or B.

Hints: Try to make use of the private modelling space to test out your jug and "Send to Public" when you are happy with your result.

2. Solving Sudoku puzzles (2~3 hours)

The major part of this workshop is to try to solve Sudoku puzzles collaboratively with the Sudoku model. The Sudoku model was originally built by Karl King to help to master this popular game. It was intended to be a single user model. If you look into the definitive script carefully, you will realise that all observables, dependencies, triggers and procedures are defined without using marcos. i.e. a flattened structure.

To begin with,

i. Setup the dtkeden environment, (see the attached note).

ii. Load the Sudoku model (Run.e) into the environment.

    Note: You only need to do once on one workstation by using Send to Public

Possible steps to solve a Sudoku puzzle

It is not clear if there is a systematic strategy to solve any Sudoku puzzle collaboratively. A possible starting point is to looking for assertions for a particular digit. Next, look for the influences and formulae them into rules (cf. attached article).

Tasks you may also be interested

As you solving Sudoku puzzles with the model, you may find that the model itself does not provide enough aids for you to solve the puzzles or to observe what is going on. You and your colleague are free to decide whether you should do something to improve the situation. Below are some suggestions.

1. Customizing your environment. You may customize the interface of the Sudoku model on your workstation without interfering others by sending the redefinitions to the local environment. Things that you may wish to customize, e.g.
    - Square Colour
    - Font Size
    - Grid size
2. Improvements for observation. You may find that the initial situation is hard to observe in some sense – much information is available for queries but they are not immediate available on the screen. You may define new triggers, scout windows, etc. to improve this situation when it is necessary.
3. Improving awareness and communication. Do you know who is working on the other side? You can do some tricks on the server to make the CLIENT_LIST observable visible to all clients. You can also use some triggers or better interface to monitor changes to some key observables.
4. Making interface for easy rules input.

Useful information about the sudoku model

Here is a list of observables and functions in the Sudoku model you may find handy when reasoning the model.

Name		**Description**
currentsquare	-	The current focused square in a list format. E.g. [“D”, “3”]
currentgrid	-	The current focused region grid. E.g. 5 (the middle grid)
current_column	-	All values in the current focused column.
current_row	-	All values in the current focused row.
current_region	-	All values in the current focused region.
DFfont	-	Default font style
SD_relevant_colour	-	Highlight colours for the current focused row, column, and region.
SD_bgcolour		Background colour for the squares
SD_fgcolour		Foreground colour for the squares
SD_bdcolour		Border colour for the squares
SD_fixed_fgcolour		Foreground colour for the fixed digits
SD_focus_colour		Background colour for the current focused square
SD_relief		Relief policy for the squares
SD_font		Font style for the squares
save_puzzle()	-	Saving the current state of the puzzle. Only digits, no definitions will be saved.

Appendix C

A Note to dtkeden novice users

Dtkeden is an extension of tkeden which supports multiple human agents to build an interactive situation model (ISM) collaboratively. The major difference you may experience in a dtkeden environment is that you are not building model alone. Definitions that you have just entered may affect what your colleague's observation and experiment on another workstation. A dtkeden model is a hybrid of a shared model and individual models.

Setting up the dtkeden modelling environment

A dtkeden modelling environment consists of servers and clients. In the simplest configuration, there is at least one server and one client. Although it is possible to run the server and the client on the same machine, we recommend that you run the server and the client on separate machines to avoid confusions of the outputs, e.g. scout screen.

To start the dtkeden server,

Run *dtkeden -s -l /dcs/emp/echan/lib-dtkeden* in a command prompt.

To start the dtkeden client,

i. Run *dtkeden -a -l /dcs/emp/echan/lib-dtkeden* in a command prompt.
ii. Enter a unique name for the client when prompted.

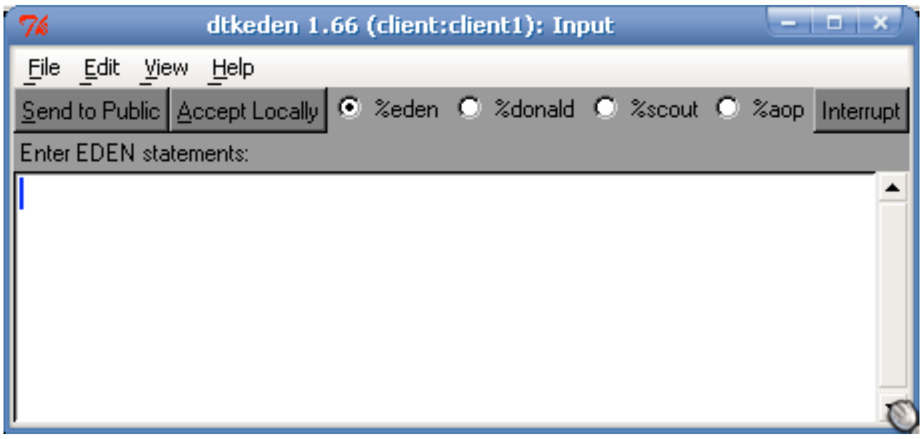


Fig 1 – A dtkeden client input window

Public and Local Workspaces

It is necessary to distinguish the public workspace and the local workspace in mind when using dtkeden. It is also worth to note that the local workspace is not private; definitions in the local workspace can be changed by public definitions. For example, if you enter “a = 10;” and accepted locally. A few seconds later, your colleague make a definition “a is b + c;” and send it to public. The public definition “a is b +c;” will then take over your local definition of a. You can track redefinitions in the command history window as you normally do in tkeden.

To make a definition public:

i) put the definition in the input window

ii) Click the *Send to Public* button.

To make local definition:

i) put the definition in the input window

ii) Click the *Accept to Local* button.

You can also make a definitive script public by using the sendToPublic() function

Syntax: sendToPublic (*statement*);

Example: sendToPublic (“writeln(\”Hello Server.\”);”);

Built-in Functions

Built-in functions which are available in tkeden are also available in dtkeden. However, you need to aware that some functions behave differently in the local workspace and the public workspace. These include:

Function Name	**Behaviour when 'Send to Public'**
include	Since the current file path on the local file system may not be the same, sending include statements to public is risky.
execute	The execute statement will be sent to all dtkeden clients and server, and embedded statement will be executed within the local workspace. i.e. After sending *execute("a is b + c;");* to public, the value of *a* may not be the same on every clients. This is because the value of *b* and *c* may not be the same from one client to another.
todo	Same as execute
edenclocks	Similar to execute. edenclocks are maintained in the local workspace technically.

Appendix D

Excerpts from the interaction history in the second distributed jugs modelling case study

This appendix contains data excerpts from the interaction history that was captured in the second distributed jugs modelling session on 20th June 2009. The interaction history is generated by dtkeden-cm, a tool to support collaborative modelling with EM. In simple terms, an interaction history captures interactions of two main types, namely, the (re)definitions that are input by the modeller through the input window, and the (re)definitions that are triggered by the triggered actions.

There were 3 sets of interaction history files. One of them merely captured the interactions that took place in the public modelling space, others captured both interactions in the public modelling spaces and interactions in the private modelling spaces. This makes it difficult for novice EM modellers or researchers who have no EM background to make sense of the data excerpts. For this reason, the interaction history data excerpts in this Appendix are presented in the following format:

```
 Timestamp    [<From-To>]  Definition(s)
```

where *Timestamp* refers to the time at which the (re)definitions were captured in the interaction history (i.e. the time almost immediately after the action was performed either by the modeller or by a triggered action in the model), *From* refers to the server or the workstation (as identified by the modeller accessing the machine) at which the (re)definitions were captured, *To* refers to the modelling space to which the (re)definitions were sent, and *Definition(s)* refers to the (re)definitions in the definitive script that were sent either by the modeller or by a triggered action in the model. Due to anonymity requirements in researching with human subjects, the letters R and S are used to represent the modellers who that particiapated in the case study instead of their real names. For instance, the following line:

```
10:20:01     [R-Private]  x = 10;
```

means that modeller R has sent a redefinition of *“x = 10;”* to his private modelling space at the time 10:20:01. Occasionally, the modellers encountered errors returned from the modelling tool. The interaction history captured these errors and they are reformatted as follows:

```
Timestamp    [ErrorMessage]
```

where *Timestamp* refers to the time at which the (re)definitions were captured in the interaction history (as discussed above), and *ErrorMessage* is the error message that was thrown from the dtkeden-cm modelling environement. For instance, the following line:

```
10:49:45     [## ERROR number 3: SCOUT: parse error, encountered
`UNKNOWN', expecting `ST...]
```

refers to an error that was thrown to the modellers at 10.49.45. It is worth noting that such errors are visible to all modellers.

D1 Interaction history for the first 15 minutes

```
10:16:53     [Server]     ?CLEINT_LIST;
10:17:01     [Server]     ?CLIENT_LIST;
10:20:01     [R-Private]  x = 10;
10:20:03     [S-Public]   writeln("Hello Mullet\n");
10:20:15     [S-Private]  writeln("Hello Si\n");
10:20:20     [R-Public]   y is x + 5;
10:20:28     [R-Private]  ?y;
10:20:34     [S-Private]  ?y;
10:21:00     [R-Public]   x = 10;
10:21:09     [S-Private]  x = 25;
```

```
10:21:13     [S-Private]  ?y;

10:21:24     [R-Private]  ?y;

10:21:39     [R-Public]   x = 10;

10:21:50     [S-Private]  ?y;

10:25:46     [S-Public]   %eden
                          include("/dcs/emp/echan/public/jugs/jugs.e");
                          include("/dcs/emp/echan/public/jugs/jugs.dependen
                          cy.e");
                          include("/dcs/emp/echan/public/jugs/jugs.s");
10:25:46     [## ERROR number 1: createtimer is not defined (expecting a
             built-in): if th...]

10:30:23     [R-Public]   %eden
                          capB = 9;

10:30:29     [R-Public]   %eden
                          capB = 7;

10:30:39     [S-Public]   %eden
                          capB = 5;
```

D2 Copy and Paste / experimentation in the private space before launching definitve script to the public modelling space

```
10:55:43     [R-Private]  ## the basics of jug c

                          capC = 7;
                          contentC = 0;
                          widthC = 5;
                          Cfull is capC==contentC;
                          Cempty is contentC==0;

                          #redefined fills

                          proc FillA { action = 1; }
                          proc FillB { action = 2; }
                          proc FillC { action = 3; }
                          proc EmptyA { action = 4; }
                          proc EmptyB { action = 5; }
                          proc EmptyC { action = 6; }

                          ## additional constants for pouring
                          ATOC = 2;
                          BTOC = 3;
                          CTOA = 4;
                          CTOB = 5;

                          # modified pour functions for buttons
                          proc PourATOB {
                             if (!(Bfull || contentA==0))
                                pourdirection = ATOB;
                             action = 7;
                          }

                          proc PourATOC {
                             if (!(Cfull || contentA==0))
                                pourdirection = ATOC;
                             action = 7;
                          }
```

```
proc PourBTOA {
   if (!(Afull || contentB==0))
      pourdirection = BTOA;
   action = 7;
}

proc PourBTOC {
   if (!(Cfull || contentB==0))
      pourdirection = BTOC;
   action = 7;
}

proc PourCTOA {
   if (!(Afull || contentC==0))
      pourdirection = CTOA;
   action = 7;
}

proc PourCTOB {
   if (!(Bfull || contentC==0))
      pourdirection = CTOB;
   action = 7;
}

proc update : clocktick {
   switch(action) {
      case 1:
         contentA++;
        if (Afull) action = 0;
        break;
      case 2:
         contentB++;
        if (Bfull) action = 0;
        break;
      case 3:
         contentC++;
        if (Cfull) action = 0;
        break;
      case 4:
         contentA--;
        if (Aempty) action = 0;
        break;
      case 5:
         contentB--;
        if (Bempty) action = 0;
        break;
      case 6:
         contentC--;
        if (Cempty) action = 0;
        break;
      case 5:
        if (pourdirection == ATOB) {
           contentA--;
           contentB++;
           if (Bfull || Aempty) action = 0;
        }
        if (pourdirection == ATOC) {
           contentA--;
           contentC++;
           if (Cfull || Aempty) action = 0;
        }
        if (pourdirection == BTOA) {
           contentB--;
           contentA++;
           if (Afull || Bempty) action = 0;
        }
        if (pourdirection == BTOC) {
           contentB--;
```

```
                                   contentC++;
                                   if (Cfull || Bempty) action = 0;
                                }
                                if (pourdirection == CTOA) {
                                   contentC--;
                                   contentA++;
                                   if (Afull || Cempty) action = 0;
                                }
                                if (pourdirection == CTOB) {
                                   contentC--;
                                   contentB++;
                                   if (Bfull || Cempty) action = 0;
                                }
                                break;
                              default: break;
                           }
                        }

10:55:43     [## ERROR number 2: parse error, encountered `'#'', expecting
             `$' or `error'...]

10:56:49     [R-Private]   ## the basics of jug c
                           … (Similar to the script entered above,
                           with syntax error corrected.)

10:56:16     [R-Private]   ?PourATOB;

10:56:49     [R-Public]    ## the basics of jug c
                           … (Similar to the script entered above,
                           some blank spaces are removed.)

10:58:02     [R-Private]   ~fillA_mouse_1 = [1,4,0,77, 21];
10:58:02     [R-Private]   ~fillA_mouse_1 = [1,5,256,77, 21];
10:58:03     [R-Private]   ~pourAtoBbutton_mouse_1 = [1,4,0,59, 13];
10:58:03     [R-Private]   ~pourAtoBbutton_mouse_1 = [3,4,256,59, 13];
10:58:03     [R-Private]   ~pourAtoBbutton_mouse_1 = [1,5,1280,59, 13];
10:58:03     [R-Private]   ~pourAtoBbutton_mouse_1 = [3,5,1024,59, 13];
10:58:03     [R-Private]   ~pourAtoCbutton_mouse_1 = [1,4,0,51, 22];
10:58:03     [R-Private]   ~pourAtoCbutton_mouse_1 = [3,4,256,51, 22];
10:58:03     [R-Private]   ~pourAtoCbutton_mouse_1 = [3,5,1280,51, 22];
10:58:03     [R-Private]   ~pourAtoCbutton_mouse_1 = [1,5,256,51, 22];

10:59:44     [R-Private]   valid1 is !Afull;
                           valid2 is !Bfull && !Aempty;
                           valid3 is !Cfull && !Aempty;

                           valid4 is !Bfull;
                           valid5 is !Afull && !Bempty;
                           valid6 is !Cfull && !Bempty;

                           valid7 is !Cfull;
                           valid8 is !Afull && !Cempty;
                           valid9 is !Bfull && !Cempty;

10:59:49     [R-Public]    valid1 is !Afull;
                           valid2 is !Bfull && !Aempty;
                           valid3 is !Cfull && !Aempty;
                           valid4 is !Bfull;
                           valid5 is !Afull && !Bempty;
                           valid6 is !Cfull && !Bempty;
                           valid7 is !Cfull;
                           valid8 is !Afull && !Cempty;
                           valid9 is !Bfull && !Cempty;
```

D3 Modeller S experimenting with different definitions for the *fillA* button

```
10:42:54     [S-Public]    %scout
                           window fillA = {
                                  type: TEXT
                                  string: "Fill"
                                  frame: ([{jugA_X1, jugA_Y2 + 40}, {jugA_X1
                           + widthA * 20, jugA_Y2 + 10}])
                           };

10:43:09     [S-Private]   %scout
                           screen=<liquidA/containerA/liquidB/containerB/fil
                           lA>;

10:43:30     [S-Private]   %scout
                           screen=<fillA/liquidA/containerA/liquidB/containe
                           rB>;

10:44:07     [S-Private]   %scout
                           screen=<fillA/liquidA/containerA>;

10:44:22     [S-Private]   %scout
                           screen=<fillA/liquidA/containerA/liquidB/containe
                           rB>;

10:44:49     [S-Private]   %scout
                           window fillA = {
                                  type: TEXT
                                  string: "Fill"
                                  frame: ([{10, 40}, {100, 100])
                           };
10:44:49     [## ERROR number 2: SCOUT: parse error, encountered `']'',
             expecting `OR' or...]

10:44:56     [S-Private]   %scout
                           window fillA = {
                                  type: TEXT
                                  string: "Fill"
                                  frame: ([{10, 40}, {100, 100}])
                           };

10:46:12     [S-Private]   %scout
                           window fillA = {
                                  type: TEXT
                                  string: "Fill"
                                  frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                           + widthA * 20, jugA_Y2 + 50}])
                           };

10:47:06     [S-Private]   %scout
                           window fillA = {
                                  type: TEXT
                                  string: "Fill"
                                  frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                           + widthA * 20, jugA_Y2 + 50}])
                                  border: 2
                                  bgcolour: "orange1"
                                  fgcolour: "black"
                                  bdcolour: "black"
                           };

10:47:20     [S-Private]   %scout
                           window fillA = {
```

```
                                      type: TEXT
                                      string: "Fill"
                                      frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                                      + widthA * 20, jugA_Y2 + 20}])
                                      border: 2
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                              };

10:47:32      [S-Private]     %scout
                              window fillA = {
                                      type: TEXT
                                      string: "Fill"
                                      frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                                      + widthA * 20, jugA_Y2 + 30}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                              };

10:49:45      [S-Private]     %scout
                              window pourAtoBbutton = {
                                      type: TEXT
                                      string: "Pour to B"
                                      frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1
                                      + widthA * 20, pourAtoB_Y2}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                              };
10:49:45      [## ERROR number 3: SCOUT: parse error, encountered `UNKNOWN',
              expecting `ST...]

10:49:56      [S-Private]     %scout
                              integer fillA_Y2 = jugA_Y2 + 30;
                              integer pourAtoB_Y1 = fillA_Y2 + 10;
                              integer pourAtoB_Y2 = pourAtoB_Y1 + 30;

                              window fillA = {
                                      type: TEXT
                                      string: "Fill"
                                      frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                              + widthA * 20, fillA_Y2}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                              };

                              window pourAtoBbutton = {
                                      type: TEXT
                                      string: "Pour to B"
                                      frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1
                              + widthA * 20, pourAtoB_Y2}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                              };

10:50:13      [S-Private]     %scout
                              screen=<fillA/pourAtoBbutton/liquidA/containerA/l
                              iquidB/containerB>;

10:50:41      [S-Private]     %scout
```

```
                                  integer pourAtoB_Y2 = pourAtoB_Y1 + 20;

10:52:08      [S-Private]        %scout

                                  integer fillA_Y2 = jugA_Y2 + 30;
                                  integer pourAtoB_Y1 = fillA_Y2 + 10;
                                  integer pourAtoB_Y2 = pourAtoB_Y1 + 20;
                                  integer pourAtoC_Y1 = pourAtoB_Y2 + 10;
                                  integer pourAtoC_Y2 = pourAtoC_Y1 + 20;

                                  window fillA = {
                                         type: TEXT
                                         string: "Fill"
                                         frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                                         + widthA * 20, fillA_Y2}])
                                         border: 1
                                         bgcolour: "orange1"
                                         fgcolour: "black"
                                         bdcolour: "black"
                                  };

                                  window pourAtoBbutton = {
                                         type: TEXT
                                         string: "Pour to B"
                                         frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1
                                         + widthA * 20, pourAtoB_Y2}])
                                         border: 1
                                         bgcolour: "orange1"
                                         fgcolour: "black"
                                         bdcolour: "black"
                                  };

                                  window pourAtoCbutton = {
                                         type: TEXT
                                         string: "Pour to B"
                                         frame: ([{jugA_X1, pourAtoC_Y1}, {jugA_X1
                                         + widthA * 20, pourAtoC_Y2}])
                                         border: 1
                                         bgcolour: "orange1"
                                         fgcolour: "black"
                                         bdcolour: "black"
                                  };

10:52:18      [S-Private]        %scout
                                  screen=<fillA/pourAtoBbutton/pourAtoCbutton/liqui
                                  dA/containerA/liquidB/containerB>;

10:54:15      [S-Private]        %scout
                                  window fillA = {
                                         type: TEXT
                                         string: "Fill"
                                         frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                                         + widthA * 20, fillA_Y2}])
                                         border: 1
                                         bgcolour: "orange1"
                                         fgcolour: "black"
                                         bdcolour: "black"
                                         sensitive: ON
                                  };

                                  window pourAtoBbutton = {
                                         type: TEXT
                                         string: "Pour to B"
                                         frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1
                                         + widthA * 20, pourAtoB_Y2}])
                                         border: 1
                                         bgcolour: "orange1"
                                         fgcolour: "black"
```

```
                                      bdcolour: "black"
                                      sensitive: ON
                               };

                               window pourAtoCbutton = {
                                      type: TEXT
                                      string: "Pour to B"
                                      frame: ([{jugA_X1, pourAtoC_Y1}, {jugA_X1
                                      + widthA * 20, pourAtoC_Y2}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                                      sensitive: ON
                               };

… (Some interaction with the model between 10:54:17 and 10:54:22
are omitted here)

10:54:38     [S-Public]        %scout
                               integer fillA_Y2 = jugA_Y2 + 30;
                               integer pourAtoB_Y1 = fillA_Y2 + 10;
                               integer pourAtoB_Y2 = pourAtoB_Y1 + 20;
                               integer pourAtoC_Y1 = pourAtoB_Y2 + 10;
                               integer pourAtoC_Y2 = pourAtoC_Y1 + 20;
                               window fillA = {
                                      type: TEXT
                                      string: "Fill"
                                      frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                                      + widthA * 20, fillA_Y2}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                                      sensitive: ON
                               };
                               window pourAtoBbutton = {
                                      type: TEXT
                                      string: "Pour to B"
                                      frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1
                                      + widthA * 20, pourAtoB_Y2}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                                      sensitive: ON
                               };
                               window pourAtoCbutton = {
                                      type: TEXT
                                      string: "Pour to B"
                                      frame: ([{jugA_X1, pourAtoC_Y1}, {jugA_X1
                                      + widthA * 20, pourAtoC_Y2}])
                                      border: 1
                                      bgcolour: "orange1"
                                      fgcolour: "black"
                                      bdcolour: "black"
                                      sensitive: ON
                               };

10:54:52     [S-Public]        %scout
                               screen=<fillA/pourAtoBbutton/pourAtoCbutton/liqui
                               dA/containerA/liquidB/containerB>;
```

D4 The development of the SCOUT windows for triggering the filling and pouring functions in the distributed jugs model

```
10:58:54     [S-Private]  %eden
                          // This is jug A
                          %scout

                          integer fillA_Y2 = jugA_Y2 + 30;
                          integer pourAtoB_Y1 = fillA_Y2 + 10;
                          integer pourAtoB_Y2 = pourAtoB_Y1 + 20;
                          integer pourAtoC_Y1 = pourAtoB_Y2 + 10;
                          integer pourAtoC_Y2 = pourAtoC_Y1 + 20;

                          window fillA = {
                                 type: TEXT
                                 string: "Fill"
                                 frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                          + widthA * 20, fillA_Y2}])
                                 border: 1
                                 bgcolour: "orange1"
                                 fgcolour: "black"
                                 bdcolour: "black"
                                 sensitive: ON
                          };

                          window pourAtoBbutton = {
                                 type: TEXT
                                 string: "Pour to B"
                                 frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1
                          + widthA * 20, pourAtoB_Y2}])
                                 border: 1
                                 bgcolour: "orange1"
                                 fgcolour: "black"
                                 bdcolour: "black"
                                 sensitive: ON
                          };

                          window pourAtoCbutton = {
                                 type: TEXT
                                 string: "Pour to C"
                                 frame: ([{jugA_X1, pourAtoC_Y1}, {jugA_X1
                          + widthA * 20, pourAtoC_Y2}])
                                 border: 1
                                 bgcolour: "orange1"
                                 fgcolour: "black"
                                 bdcolour: "black"
                                 sensitive: ON
                          };

                          %eden
                          // This is jug B
                          %scout

                          integer fillB_Y2 = jugB_Y2 + 30;
                          integer pourBtoA_Y1 = fillB_Y2 + 10;
                          integer pourBtoA_Y2 = pourBtoA_Y1 + 20;
                          integer pourBtoC_Y1 = pourBtoA_Y2 + 10;
                          integer pourBtoC_Y2 = pourBtoC_Y1 + 20;

                          window fillB = {
                                 type: TEXT
                                 string: "Fill"
                                 frame: ([{jugB_X1, jugB_Y2 + 10}, {jugB_X1
                          + widthB * 20, fillB_Y2}])
```

```
                                    border: 1
                                    bgcolour: "orange1"
                                    fgcolour: "black"
                                    bdcolour: "black"
                                    sensitive: ON
                             };

                             window pourBtoAbutton = {
                                    type: TEXT
                                    string: "Pour to A"
                                    frame: ([{jugB_X1, pourBtoA_Y1}, {jugB_X1
                             + widthB * 20, pourBtoA_Y2}])
                                    border: 1
                                    bgcolour: "orange1"
                                    fgcolour: "black"
                                    bdcolour: "black"
                                    sensitive: ON
                             };

                             window pourBtoCbutton = {
                                    type: TEXT
                                    string: "Pour to C"
                                    frame: ([{jugB_X1, pourBtoC_Y1}, {jugB_X1
                             + widthB * 20, pourBtoC_Y2}])
                                    border: 1
                                    bgcolour: "orange1"
                                    fgcolour: "black"
                                    bdcolour: "black"
                                    sensitive: ON
                             };


                             screen=<fillA/fillB/pourAtoBbutton/pourAtoCbutton
                             /pourBtoAbutton/pourBtoCbutton/liquidA/containerA
                             /liquidB/containerB>;
                             screen=<fillA/pourAtoBbutton/pourAtoCbutton/liqui
                             dA/containerA/liquidB/containerB>;
```

10:58:54   [## ERROR number 4: parse error, encountered `SLASH_SLASH', expecting `$' or...]

```
10:59:06      [S-Private]    [Similar to 10:58:54]
```

10:59:06   [## ERROR number 5: parse error, encountered `FORMULA', expecting `'?'' or `...]

```
10:59:15      [S-Private]   %scout

10:59:17      [S-Private]   [Similar to 10:59:06]
```

10:59:17   [## ERROR number 6: SCOUT: parse error, encountered `UNKNOWN', expecting `ST...]

```
10:59:32      [S-Private]   [Similar to 10:59:17]
```

10:59:32   [## ERROR number 7: SCOUT: parse error, encountered `UNKNOWN', expecting]

```
11:00:30      [S-Private]   [Similar to 10:59:32]
```

11:00:30   [## ERROR number 8: SCOUT: parse error, encountered `UNKNOWN', expecting `ST...]

```
11:00:44      [S-Private]   [Similar to 11:00:30]
```

11:00:44   [## ERROR number 9: SCOUT: parse error, encountered `UNKNOWN', expecting `ST...]

```
11:01:19      [S-Private]  %scout
                           ?jugB_X1;

11:01:41      [S-Private]  %scout
                           ?pourBtoA_Y1;

11:01:47      [S-Private]  %eden

11:01:50      [S-Private]  %eden
                           ?pourBtoA_Y1;

11:02:12      [S-Private]  %eden
                           ?jugB_X1;

11:02:28      [S-Private]  ~pourAtoBbutton_mouse_1 = [1,4,0,70, 15];
11:02:28      [S-Private]  ~pourAtoBbutton_mouse_1 = [1,5,256,70, 15];
11:02:28      [S-Private]  ~pourAtoCbutton_mouse_1 = [1,4,0,65, 3];
11:02:28      [S-Private]  ~pourAtoCbutton_mouse_1 = [1,5,256,65, 3];
11:02:28      [S-Private]  ~pourAtoBbutton_mouse_1 = [1,4,0,71, 12];
11:02:28      [S-Private]  ~pourAtoBbutton_mouse_1 = [1,5,256,71, 12];
11:02:29      [S-Private]  ~fillA_mouse_1 = [1,4,0,73, 13];
11:02:29      [S-Private]  ~fillA_mouse_1 = [1,5,256,73, 13];
11:02:29      [S-Private]  ~pourAtoBbutton_mouse_1 = [1,4,0,62, 16];
11:02:29      [S-Private]  ~pourAtoBbutton_mouse_1 = [1,5,256,62, 16];
11:02:30      [S-Private]  ~pourAtoCbutton_mouse_1 = [1,4,0,51, 22];
11:02:30      [S-Private]  ~pourAtoCbutton_mouse_1 = [1,5,256,51, 22];

11:02:56      [S-Private]  [Similar to 11:00:44]
```

11:02:56   [## ERROR number 10: parse error, encountered `FORMULA', expecting `'?'' or `...]

```
11:02:58      [S-Private]  %scout

11:03:00      [S-Private]  [Similar to 11:02:56]
```

11:03:00   [## ERROR number 11: SCOUT: parse error, encountered `UNKNOWN', expecting `ST...]

```
11:03:41      [S-Private]  [Similar to 11:03:00]
```

11:03:41   [## ERROR number 12: SCOUT: parse error, encountered `UNKNOWN', expecting `ST...]

```
11:03:56      [S-Private]  [Similar to 11:03:41]
```

11:03:56   [## ERROR number 13: SCOUT: parse error, encountered `UNKNOWN', expecting `ST...]

11:03:58 ~ 11:04:03 (Some interactions with the model, similar to those between 11:02:28 and 11:02:30)

```
11:05:14      [S-Pubic]    [Similar to 11:03:56]
```

11:05:14   [## ERROR number 14: SCOUT: parse error, encountered `UNKNOWN', expecting `ST...]

```
11:06:15      [S-Pri vate] %scout
                           integer fillA_Y2 = jugA_Y2 + 30;
                           integer pourAtoB_Y1 = fillA_Y2 + 10;
                           integer pourAtoB_Y2 = pourAtoB_Y1 + 20;
                           integer pourAtoC_Y1 = pourAtoB_Y2 + 10;
                           integer pourAtoC_Y2 = pourAtoC_Y1 + 20;
                           window fillA = {
                                  type: TEXT
                                  string: "Fill A"
                                  frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1
                           + widthA * 20, fillA_Y2}])
```

```
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window pourAtoBbutton = {
       type: TEXT
       string: "Pour to B"
       frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1
+ widthA * 20, pourAtoB_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window pourAtoCbutton = {
       type: TEXT
       string: "Pour to C"
       frame: ([{jugA_X1, pourAtoC_Y1}, {jugA_X1
+ widthA * 20, pourAtoC_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
integer fillB_Y2 = jugA_Y2 + 30;
integer pourBtoA_Y1 = fillB_Y2 + 10;
integer pourBtoA_Y2 = pourBtoA_Y1 + 20;
integer pourBtoC_Y1 = pourBtoA_Y2 + 10;
integer pourBtoC_Y2 = pourBtoC_Y1 + 20;
window fillB = {
       type: TEXT
       string: "Fill B"
       frame: ([{jugB_X1, jugA_Y2 + 10}, {jugB_X1
+ widthB * 20, fillB_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window pourBtoAbutton = {
       type: TEXT
       string: "Pour to A"
       frame: ([{jugB_X1, pourBtoA_Y1}, {jugB_X1
+ widthB * 20, pourBtoA_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window pourBtoCbutton = {
       type: TEXT
       string: "Pour to C"
       frame: ([{jugB_X1, pourBtoC_Y1}, {jugB_X1
+ widthB * 20, pourBtoC_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
```

```
                                  screen=<fillB/fillA/pourAtoBbutton/pourAtoCbutton
                                  /pourBtoAbutton/pourBtoCbutton/liquidA/containerA
                                  /liquidB/containerB>;
```

D5 Contributions to the public modelling space made by modeller R between 10:30:23 and 11:46:16

```
10:30:23      [R-Public]      %eden
                              capB = 9;

10:30:29      [R-Public]      %eden
                              capB = 7;

10:56:49      [R-Public]      %eden
                              ## the basics of jug c
                              capC = 7;
                              contentC = 0;
                              widthC = 5;
                              Cfull is capC==contentC;
                              Cempty is contentC==0;
                              ## redefined fills
                              proc FillA { action = 1; }
                              proc FillB { action = 2; }
                              proc FillC { action = 3; }
                              proc EmptyA { action = 4; }
                              proc EmptyB { action = 5; }
                              proc EmptyC { action = 6; }
                              ## additional constants for pouring
                              ATOC = 2;
                              BTOC = 3;
                              CTOA = 4;
                              CTOB = 5;
                              ## modified pour functions for buttons
                              proc PourATOB {
                                 if (!(Bfull || contentA==0))
                                    pourdirection = ATOB;
                                 action = 7;
                              }
                              proc PourATOC {
                                 if (!(Cfull || contentA==0))
                                    pourdirection = ATOC;
                                 action = 7;
                              }
                              proc PourBTOA {
                                 if (!(Afull || contentB==0))
                                    pourdirection = BTOA;
                                 action = 7;
                              }
                              proc PourBTOC {
                                 if (!(Cfull || contentB==0))
                                    pourdirection = BTOC;
                                 action = 7;
                              }
                              proc PourCTOA {
                                 if (!(Afull || contentC==0))
                                    pourdirection = CTOA;
                                 action = 7;
                              }
                              proc PourCTOB {
                                 if (!(Bfull || contentC==0))
                                    pourdirection = CTOB;
                                 action = 7;
                              }
                              proc update : clocktick {
                                 switch(action) {
                                    case 1:
                                       contentA++;
                                      if (Afull) action = 0;
                                      break;
                                    case 2:
                                       contentB++;
                                      if (Bfull) action = 0;
                                      break;
                                    case 3:
                                       contentC++;
                                      if (Cfull) action = 0;
                                      break;
                                    case 4:
                                       contentA--;
                                      if (Aempty) action = 0;
                                      break;
                                    case 5:
                                       contentB--;
                                      if (Bempty) action = 0;
                                      break;
                                    case 6:
                                       contentC--;
                                      if (Cempty) action = 0;
                                      break;
                                    case 5:
                                      if (pourdirection == ATOB) {
                                         contentA--;
                                         contentB++;
                                         if (Bfull || Aempty) action = 0;
                                      }
                                      if (pourdirection == ATOC) {
                                         contentA--;
                                         contentC++;
                                         if (Cfull || Aempty) action = 0;
                                      }
                                      if (pourdirection == BTOA) {
                                         contentB--;
                                         contentA++;
                                         if (Afull || Bempty) action = 0;
                                      }
                                      if (pourdirection == BTOC) {
                                         contentB--;
                                         contentC++;
                                         if (Cfull || Bempty) action = 0;
                                      }
                                      if (pourdirection == CTOA) {
                                         contentC--;
                                         contentA++;
                                         if (Afull || Cempty) action = 0;
                                      }
                                      if (pourdirection == CTOB) {
                                         contentC--;
                                         contentB++;
                                         if (Bfull || Cempty) action = 0;
                                      }
                                      break;
                                    default: break;
                                 }
                              }

10:59:49     [R-Public]      %eden
                              valid1 is !Afull;
                              valid2 is !Bfull && !Aempty;
                              valid3 is !Cfull && !Aempty;
                              valid4 is !Bfull;
                              valid5 is !Afull && !Bempty;
                              valid6 is !Cfull && !Bempty;
```

```
                        valid7 is !Cfull;
                        valid8 is !Afull && !Cempty;
                        valid9 is !Bfull && !Cempty;

11:30:25     [R-Public] %eden
                        contentC = 0;
```

D6 Major contributions that constitutes the final distributed jugs model

Due to the complexity of the data excerpt in this section, annotations were added and are printed in bold font.

```
(Modeller S reloads the given scripts to the public modelling
space)
11:42:31     [S]    %eden
                    include("/dcs/emp/echan/public/jugs/jugs.e");
                    include("jugs.dependency.e");
                    include("jugs.s");

(Modeller S sets the width of Jug C to 1/7 of the screen width
(dependency))
11:46:48     [S]    %scout
                    integer widthC = (screen_width/7);


(Modeller R transfers his contribution from his private editor
directly to the public modelling space, the PourXToY agents are
now affecting the public modelling space)
11:52:37     [R]    %eden
                    ## the basics of jug c
                    capC = 7;
                    contentC = 0;
                    Cfull is capC==contentC;
                    Cempty is contentC==0;
                    ## redefined fills
                    proc FillA : fillA_mouse_1 { action = 1; }
                    proc FillB : fillB_mouse_1 { action = 2; }
                    proc FillC : fillC_mouse_1 { action = 3; }
                    proc EmptyA : emptyA_mouse_1 { action = 4; }
                    proc EmptyB : emptyB_mouse_1 { action = 5; }
                    proc EmptyC : emptyC_mouse_1 { action = 6; }
                    ## additional constants for pouring
                    ATOC = 2;
                    BTOC = 3;
                    CTOA = 4;
                    CTOB = 5;
                    ## modified pour functions for buttons
                    proc PourATOB : pourAtoBbutton_mouse_1 {
                       sendToPublic("%eden\npourdirection = ATOB;
                       action = 7;");
                    }
                    proc PourATOC : pourAtoCbutton_mouse_1 {
                       sendToPublic("%eden\npourdirection = ATOC;
                       action = 7;");
                    }
                    proc PourBTOA : pourBtoAbutton_mouse_1 {
                       sendToPublic("%eden\npourdirection = BTOA;
                       action = 7;");
                    }
                    proc PourBTOC : pourBtoCbutton_mouse_1 {
                       sendToPublic("%eden\npourdirection = BTOC;
                       action = 7;");
                    }
                    proc PourCTOA : pourAtoBbutton_mouse_1 {
                       sendToPublic("%eden\npourdirection = CTOA;
                       action = 7;");
                    }
                    proc PourCTOB : pourCtoBbutton_mouse_1 {
                       sendToPublic("%eden\n
                       pourdirection = CTOB;
                       action = 7;
                      ");
                    }
                    proc update : clocktick {
                       switch(action) {
                          case 1:
                             contentA++;
                            if (Afull) action = 0;
                            break;
                          case 2:
                             contentB++;
                            if (Bfull) action = 0;
                            break;
                          case 3:
                             contentC++;
                            if (Cfull) action = 0;
                            break;
                          case 4:
                             contentA--;
                            if (Aempty) action = 0;
                            break;
                          case 5:
                             contentB--;
                            if (Bempty) action = 0;
                            break;
                          case 6:
                             contentC--;
                            if (Cempty) action = 0;
                            break;
                          case 7:
                            if (pourdirection == ATOB) {
                               contentA--;
                               contentB++;
                               if (Bfull || Aempty) action = 0;
                            }
                            else if (pourdirection == ATOC) {
                               contentA--;
                               contentC++;
                               if (Cfull || Aempty) action = 0;
                            }
                            else if (pourdirection == BTOA) {
                               contentB--;
                               contentA++;
                               if (Afull || Bempty) action = 0;
                            }
                            else if (pourdirection == BTOC) {
                               contentB--;
                               contentC++;
                               if (Cfull || Bempty) action = 0;
                            }
                            else if (pourdirection == CTOA) {
                               contentC--;
                               contentA++;
                               if (Afull || Cempty) action = 0;
                            }
                            else if (pourdirection == CTOB) {
                               contentC--;
                               contentB++;
                               if (Bfull || Cempty) action = 0;
```

```
                                }
                                break;
                              default: break;
                           }
                        }
                        valid1 is !Afull;
                        valid2 is !Bfull && !Aempty;
                        valid3 is !Cfull && !Aempty;
                        valid4 is !Bfull;
                        valid5 is !Afull && !Bempty;
                        valid6 is !Cfull && !Bempty;
                        valid7 is !Cfull;
                        valid8 is !Afull && !Cempty;
                        valid9 is !Bfull && !Cempty;
```

(Modeller R modifies the fills and empty triggered actions so that they make redefinitions in the public modelling space instead of private modelling space)

```
11:55:18     [R]    %eden
                    proc FillA : fillA_mouse_1 {
                    sendToPublic("%eden\naction = 1;"); }
                    proc FillB : fillB_mouse_1 {
                    sendToPublic("%eden\naction = 2;"); }
                    proc FillC : fillC_mouse_1 {
                    sendToPublic("%eden\naction = 3;"); }
                    proc EmptyA : emptyA_mouse_1 {
                    sendToPublic("%eden\naction = 4;"); }
                    proc EmptyB : emptyB_mouse_1 {
                    sendToPublic("%eden\naction = 5;"); }
                    proc EmptyC : emptyC_mouse_1 {
                    sendToPublic("%eden\naction = 6;"); }
```

(Modeller R redefines the action numbers for the pouring directions)

```
11:58:09     [R]    %eden
                    ATOB = 1;
                    ATOC = 2;
                    BTOC = 3;
                    BTOA = 4;
                    CTOA = 5;
                    CTOB = 6;
```

(Modeller S attempts to fix the sendToPublic agent)

```
12:07:26     [S]    %eden
                    proc sendToPublic {
                            para stmt;
                      execute(stmt);
                    }
```

(Modeller S refreshs his contributions in the public modelling space – he resends all the definitions he has in his private editor)

```
12:18:01     [S]    %scout
                    integer capA;
                    integer contentA;
                    integer widthA = (screen_width/7);
                    integer jugA_X1 = (screen_width/7);
                    integer jugA_Y2 = 400;
                    integer capB;
                    integer contentB;
                    integer widthB = (screen_width/7);
                    integer jugB_X1 = jugA_X1+2*(screen_width/7);
                    integer capC;
                    integer contentC;
                    integer widthC = (screen_width/7);
                    integer jugC_X1 = jugA_X1+(4*(screen_width/7));
                    integer fillA_Y2 = jugA_Y2 + 30;
                    integer pourAtoB_Y1 = fillA_Y2 + 10;
                    integer pourAtoB_Y2 = pourAtoB_Y1 + 20;
                    integer pourAtoC_Y1 = pourAtoB_Y2 + 10;
                    integer pourAtoC_Y2 = pourAtoC_Y1 + 20;
                    integer emptyA_Y1 = pourAtoC_Y2 + 10;
                    integer emptyA_Y2 = emptyA_Y1 + 20;
                    integer fillB_Y2 = jugA_Y2 + 30;
                    integer pourBtoA_Y1 = fillB_Y2 + 10;
                    integer pourBtoA_Y2 = pourBtoA_Y1 + 20;
                    integer pourBtoC_Y1 = pourBtoA_Y2 + 10;
                    integer pourBtoC_Y2 = pourBtoC_Y1 + 20;
                    integer emptyB_Y1 = pourBtoC_Y2 + 10;
                    integer emptyB_Y2 = emptyB_Y1 + 20;
                    integer fillC_Y2 = jugA_Y2 + 30;
                    integer pourCtoA_Y1 = fillC_Y2 + 10;
                    integer pourCtoA_Y2 = pourCtoA_Y1 + 20;
                    integer pourCtoB_Y1 = pourCtoA_Y2 + 10;
                    integer pourCtoB_Y2 = pourCtoB_Y1 + 20;
                    integer emptyC_Y1 = pourCtoB_Y2 + 10;
                    integer emptyC_Y2 = emptyC_Y1 + 20;
                    window emptyA = {
                           type: TEXT
                           string: "Empty A"
                           frame: ([{jugA_X1, emptyA_Y1}, {jugA_X1 + widthA,
                    emptyA_Y2}])
                           border: 1
                           bgcolour: "orange1"
                           fgcolour: "black"
                           bdcolour: "black"
                           sensitive: ON
                    };
                    window fillA = {
                           type: TEXT
                           string: "Fill A"
                           frame: ([{jugA_X1, jugA_Y2 + 10}, {jugA_X1 +
                    widthA, fillA_Y2}])
                           border: 1
                           bgcolour: "orange1"
                           fgcolour: "black"
                           bdcolour: "black"
                           sensitive: ON
                    };
                    window pourAtoBbutton = {
                           type: TEXT
                           string: "Pour to B"
                           frame: ([{jugA_X1, pourAtoB_Y1}, {jugA_X1 +
                    widthA, pourAtoB_Y2}])
                           border: 1
                           bgcolour: "orange1"
                           fgcolour: "black"
                           bdcolour: "black"
                           sensitive: ON
                    };
                    window pourAtoCbutton = {
                           type: TEXT
                           string: "Pour to C"
                           frame: ([{jugA_X1, pourAtoC_Y1}, {jugA_X1 +
                    widthA, pourAtoC_Y2}])
                           border: 1
                           bgcolour: "orange1"
                           fgcolour: "black"
                           bdcolour: "black"
                           sensitive: ON
                    };
                    window emptyB = {
                           type: TEXT
                           string: "Empty B"
                           frame: ([{jugB_X1, emptyB_Y1}, {jugB_X1 + widthB,
                    emptyB_Y2}])
```

```
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window fillB = {
       type: TEXT
       string: "Fill B"
       frame: ([{jugB_X1, jugA_Y2 + 10}, {jugB_X1 +
widthB, fillB_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window pourBtoAbutton = {
       type: TEXT
       string: "Pour to A"
       frame: ([{jugB_X1, pourBtoA_Y1}, {jugB_X1 +
widthB, pourBtoA_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window pourBtoCbutton = {
       type: TEXT
       string: "Pour to C"
       frame: ([{jugB_X1, pourBtoC_Y1}, {jugB_X1 +
widthB, pourBtoC_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
########################################################
##################
window emptyC = {
       type: TEXT
       string: "Empty C"
       frame: ([{jugC_X1, emptyC_Y1}, {jugC_X1 + widthC,
emptyC_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window fillC = {
       type: TEXT
       string: "Fill C"
       frame: ([{jugC_X1, jugA_Y2 + 10}, {jugC_X1 +
widthC, fillC_Y2}])
       border: 1
       bgcolour: "orange1"
       fgcolour: "black"
       bdcolour: "black"
       sensitive: ON
};
window pourCtoAbutton = {
       type: TEXT
       string: "Pour to A"
       frame: ([{jugC_X1, pourCtoA_Y1}, {jugC_X1 +
widthC, pourCtoA_Y2}])
```

```
        border: 1
        bgcolour: "orange1"
        fgcolour: "black"
        bdcolour: "black"
        sensitive: ON
};
window pourCtoBbutton = {
        type: TEXT
        string: "Pour to B"
        frame: ([{jugC_X1, pourCtoB_Y1}, {jugC_X1 +
widthC, pourCtoB_Y2}])
        border: 1
        bgcolour: "orange1"
        fgcolour: "black"
        bdcolour: "black"
        sensitive: ON
};
#####################################################
##################
window containerA = {
         type: TEXT
         string: "This is jug A"
         frame: ([{jugA_X1,jugA_Y2 - 50*capA},
{jugA_X1+widthA,jugA_Y2}])
                                bgcolour: "lightgrey"
                                bdcolour: "black"
                                border: 1
};
%eden
liquidAheight = jugA_Y2 - max(50*contentA, 0);
%scout
integer liquidAheight;
window liquidA = {
         type: TEXT
         string: "This is liquid A"
         frame: ([{jugA_X1,liquidAheight},
{jugA_X1+widthA,jugA_Y2}])
                                bgcolour: "blue"
};
window containerB = {
         type: TEXT
         string: "This is jug B"
         frame: ([{jugB_X1,jugA_Y2 - 50*capB},
{jugB_X1+widthB,jugA_Y2}])
                                bgcolour: "lightgrey"
                                bdcolour: "black"
                                border: 1
};
%eden
liquidBheight = jugA_Y2 - 50* max(contentB, 0);
%scout
integer liquidBheight;
window liquidB = {
         type: TEXT
         string: "This is liquid B"
         frame: ([{jugB_X1,liquidBheight},
{jugB_X1+widthB,jugA_Y2}])
                                bgcolour: "blue"
};
window containerC = {
         type: TEXT
         string: "This is jug C"
         frame: ([{jugC_X1,jugA_Y2 - 50*capC},
{jugC_X1+widthC,jugA_Y2}])
                                bgcolour: "lightgrey"
                                bdcolour: "black"
                                border: 1
};
```

```
                    %eden
                    liquidCheight = jugA_Y2 - 50*max(contentC,0);
                    %scout
                    integer liquidCheight;
                    window liquidC = {
                            type: TEXT
                            string: "This is liquid C"
                            frame: ([{jugC_X1,liquidCheight},
                    {jugC_X1+widthC,jugA_Y2}])
                                                    bgcolour: "blue"
                    };
                    screen=<fillB/fillA/fillC/emptyA/emptyB/emptyC/pourAtoB
                    button/pourAtoCbutton/pourBtoAbutton/pourBtoCbutton/liq
                    uidA/containerA/liquidB/containerB/liquidC/containerC/p
                    ourCtoBbutton/pourCtoAbutton>;

(The pair resets the contents of Jug A, B and C)
12:18:05     [On the Server]     %eden
                                 contentA=0;
                                 contentB=0;
                                 contentC=0;

(Modeller R fixes the update agent)
12:18:13     [R]    %eden
                    proc update : clocktick {
                       switch(action) {
                          case 1:
                             contentA++;
                            if (Afull) action = 0;
                            break;
                          case 2:
                             contentB++;
                            if (Bfull) action = 0;
                            break;
                          case 3:
                             contentC++;
                            if (Cfull) action = 0;
                            break;
                          case 4:
                             contentA--;
                            if (Aempty) action = 0;
                            break;
                          case 5:
                             contentB--;
                            if (Bempty) action = 0;
                            break;
                          case 6:
                             contentC--;
                            if (Cempty) action = 0;
                            break;
                          case 7:
                            if (pourdirection == ATOB) {
                               if (Bfull || Aempty)
                                   action = 0;
                               else {
                                   contentA--;
                                   contentB++;
                               }
                            }
                            else if (pourdirection == ATOC) {
                               if (Cfull || Aempty)
                                   action = 0;
                               else {
                                   contentA--;
                                   contentC++;
                               }
                            }
                            else if (pourdirection == BTOA) {
                               if (Afull || Bempty)
                                   action = 0;
                               else {
                                   contentB--;
                                   contentA++;
                               }   }
                            else if (pourdirection == BTOC) {
                               if (Cfull || Bempty)
                                   action = 0;
                               else {
                                   contentB--;
                                   contentC++;
                               }
                            }
                            else if (pourdirection == CTOA) {
                               if (Afull || Cempty)
                                   action = 0;
                               else {
                                   contentC--;
                                   contentA++;
                               }
                            }
                            else if (pourdirection == CTOB) {
                               if (Bfull || Cempty)
                                   action = 0;
                               else {
                                   contentC--;
                                   contentB++;
                               }
                            }
                            break;
                          default: break;
                       }
                    }
```

(Modeller S makes some changes to SCOUT window that represents the liquid of Jug A)

```
12:19:18     [S]    %eden
                    liquidAheight = jugA_Y2 - 50*contentA;
                    %scout
                    integer liquidAheight;
                    window liquidA = {
                            type: TEXT
                            string: "This is liquid A"
                            frame: ([{jugA_X1,liquidAheight},
                    {jugA_X1+widthA,jugA_Y2}])
                            bgcolour: "blue"
                    };
```

(Modeller S thinks that the eden was stopped, and therefore tries to re-enable it)

```
12:22:35     [S]    %eden
                    unpauseedenclocks();
```

D7 Definitions that will make the distributed jugs model function ‘properly’

```
%scout

integer jugB_Y2 = jugA_Y2;
integer jugC_Y2 = jugA_Y2;

window liquidA = {
        type: TEXT
        string: "This is liquid A"
        frame: ([{jugA_X1,jugA_Y2 - 50*contentA},
                 {jugA_X1+widthA,jugA_Y2}])
        bgcolour: "blue"
};

window liquidB = {
        type: TEXT
        string: "This is liquid B"
        frame: ([{jugB_X1,jugB_Y2 - 50*contentB},
                 {jugB_X1+widthB,jugA_Y2}])
        bgcolour: "blue"
};

window liquidC = {
        type: TEXT
        string: "This is liquid C"
        frame: ([{jugC_X1,jugC_Y2 - 50*contentC},
                 {jugC_X1+widthC,jugA_Y2}])
        bgcolour: "blue"
};

### Fix the mistake that modeller R made at 11:52:37
### (He redefined pourAtoBbutton_mouse_1 instead of pourCtoAbutton_mouse_1)
%eden
proc PourCTOA : pourCtoAbutton_mouse_1 {
   sendToPublic("%eden\npourdirection = CTOA;
   action = 7;");
}


### Fix the edenclock that was broken by modeller S at 12:22:35
%eden
setedenclock(&clocktick,200);

### Reset action
action = 0;

### Reset contentA, B, C
contentA = 0;
contentB = 0;
contentC = 0;
```