Chapter 7
Distributed Empirical Modelling for Requirements Engineering

It is increasingly recognised that one of the most intricate and important tasks in software system development (SSD) is to understand the needs of users. No matter how well designed they are, software systems that fail to satisfy the users' needs will disappoint users and bring grief to developers. Accordingly, Requirements Engineering (RE), which aims to solve this problem by developing requirements in a systematic fashion, has recently attracted widespread attention in the software community. Unfortunately, since RE is located at the intersection of formal and informal, objective and subjective, and technical and non-technical approaches, the RE process (REP) is very difficult to understand fully [Fin94, Bub95]. The development of requirements remains the most crucial, labour-intensive and expensive part of SSD.

7.0 Overview

This chapter seeks to apply the framework of DEM to effective requirements development. In Section 7.1, a brief overview of RE and the difficulties of enacting the REP are given. These difficulties, arising from the inability to provide adequate support for the interpersonal interaction and take sufficient account of context (i.e. the real-world environment in which requirements are developed and the system is used), have given rise to the wide gap between research and practice [BL98].

Section 7.2 attempts to bridge this gap by reengineering the REP. Firstly, to reflect the situatedness of requirements [Gog94], a new definition of requirements that draws more attention to the context in which the requirements are developed and used is given. To correspond to this definition, the need for requirements to be incrementally formulated through interpersonal interaction is identified. This section then reengineers the REP by regarding it as a problem-solving process in which human agents interact with each other in order to develop requirements within their context. In this way, requirements are cultivated in order to solve problems as they are identified in the real world. Hence, the REP must closely intertwine with SSD in a symbiotic manner rather than simply being bolted on as a front-end to SSD. Section 7.2 also discusses the influence of the interactive relationship between human agents on the interaction of the REP, and consequently proposes that a computer-based, collaborative interaction environment is useful for cultivating requirements.

In response to the basic characteristics of the REP, Section 7.3 proposes a human-centred framework, called SPORE, whereby requirements as ‘solutions to identified problems’ in the application domain are developed in an open-ended and situated manner. Within this framework, people participating in the REP are able to cultivate requirements through collaborative interaction with each other in order to solve the identified problems, rather than by searching for requirements in the ‘jungle’ of users’ needs. The principles and concepts of DEM are applied to SPORE to support the collaborative interaction between participants and the situatedness of the enacted the REP. By enacting DEM using dtkeden, computer-based models can be created and used by participants to serve two functions:

- as artefacts to explore, expand and experience the solutions to the identified problems.
- as a powerful communication medium to support their collaborative interaction in order to 'grow' the solutions through incremental development in a distributed environment.

Two examples of the use of SPORE for requirements development are given in section 7.4. The first example is that of developing requirements for an interactive software system embedded in an automatic teller machine (ATM). A comparison between SPORE and a viewpoint-oriented model, called VORD [KS98], of requirements development is discussed. Another example is that of a warehouse inventory information system. A use-case driven approach to requirements development for this application is found in [JCJO92], and this is compared with SPORE.

7.1 Requirements Engineering

In his celebrated paper [Bro87], F. Brooks argued that "[t]he hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements … No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later". Indeed, a poor understanding of the needs of users has become the main cause of system failures. The detection and correction of errors arising from such poor understanding is often the most difficult and expensive task in SSD [STM95]. The worst thing is that such errors may remain undetected until system operation, thereby provoking failures that have serious consequences, especially in safety critical systems. Therefore, capturing the needs of users accurately is a vital component of successful software system.

7.1.1 An Overview of Requirements Engineering

Requirements Engineering (RE) typically refers to that part of the SSD) life-cycle in which application engineers investigate the needs of the user community and abstract from these needs to form descriptive specifications for the development of software systems. It involves intellectually challenging and creative activities, acknowledged to be the most costly and error-prone parts of SSD. A systematic process is needed for RE to derive the users' needs for the software system that is to be developed. In order to facilitate requirements acquisition, this process is conventionally divided into a set of well-defined activities characteristic of an engineering discipline. The term 'requirements' is defined in IEEE-Std.'610' as follows [IEEE90]:

1. A condition or capacity needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or processed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in 1 or 2.

In this thesis, people who are engaged with these activities for understanding and acquiring requirements are referred to as *participants*. Typically, they come from two camps: a usage camp, including end-users of the software system, managers and others affected by the system; and a development camp, including analysts and designers responsible for the system development, maintainers in charge of maintaining the developed system, and requirements engineers enacting RE. For reasons of convenience, the general terms 'users' and 'developers' will be used here to describe people from the two camps.

So far, there has been no agreement on a standard definition of RE. For example, P. Loucopoulos and V. Karakostas define RE as "the systematic process of developing requirements through an iterative co-operative process of analysing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained" [LK95]. This definition is concerned mainly with the *technical* issues of enacting RE for requirements acquisition. Non-technical issues, such as social contexts and cognitive concerns, are only peripherally implicated in this account through the references to co-operative interaction between participants and to understanding gained. By contrast, according to J. Bubenko, RE "can be said to be the area of knowledge concerned with communicating with organisational actors with respect to their visions, intentions, and activities regarding their need for computer support, and developing and maintaining an adequate requirements specification of an information system" [Bub95]. This definition suggests that RE should embrace not merely technical but also managerial, organisational, economic, social issues and problems.

Although definitions of RE vary, it is commonly agreed that RE plays a very critical role in the development of an appropriate software system with the required degree of quality assurance [Rol94, LK95]. RE is conventionally viewed as a phase in the early stage of the life cycle of SSD [LK95, KS98, SS97]. It is assumed that the gap between users and developers can be narrowed by obtaining well-structured, well-described specifications of the needs of users. Given the specifications, developers can confidently continue the remaining phases of the SSD life cycle and finally deliver the system, fully satisfying the users' needs. This linear dependency emphasises the importance of RE in paving the way for successful SSD.

In spite of its importance, the process by which requirements are apprehended and acquired is poorly understood [Fin94, Bub95]. In order to overcome this deficiency, a model of REP is needed. Many process models serving this purpose have been proposed. According to M. Dowson's classification, each of them falls into one or more of the following categories: *product-oriented*, *activity-oriented* and *decision-oriented* [Dow87]. These may be summarised as follows.

- Product-oriented

This kind of model focuses on the product of RE. It aims to help developers to construct correct descriptive documents [DBP93a, DBP93b, FHW94, KS98, LL94, LH94, Lou94]. Most such models decompose a root definition from the highest level into a number of less abstract modules in order to understand the structure and functionality of the whole software system which it is proposed to develop. The common assumption is that requirements pre-exist and are hidden, and can then be retrieved from their sources before being fixed in the form of a descriptive representation. When enacted by this kind of model, the REP can be regarded as a process of transforming informal, fuzzy individual statements of users' needs to a formal precise description of requirements that is understood by all participants. The final result of this transition takes the form of requirements specifications recording the users' needs in a well-defined and well-structured descriptive format.

Requirements specifications are typically a kind of paper-based documentation in the form of text or diagrams. Rigid formality of requirements specification is usually required in order to conform to certain characteristics such as completeness, correctness, unambiguity, understandability, modifiability and consistency, though many of these qualities are very difficult to achieve and test. In addition, from the product-oriented perspective, it is maintained that the description of requirements should not involve any design details, due to the conventional wisdom in RE that requirements are concerned with only *what* is desired without referring to *how* it is to be implemented [Dav93, HJ89, KS98, SS97].

Once the specifications are accomplished, they must be signed off by users and developers. Thus, the specifications not only become a contract between users and developers on all issues associated with the problem which need to be solved by the system, but are also used as a blueprint to enable designers to develop the system. Although, a change in the requirements can be requested after the specification is finalised, each after-the-fact change will add to the costs and extend the schedule.

- Activity-oriented

An activity-oriented model concentrates on the process of RE itself. It is concerned with finding and executing a set of activities for requirements acquisition [And94, KS98, LK95, Rei92, SDV96, Sut96]. It is believed that engaging in these prescribed activities can help developers to capture requirements from users and represent them in formal notations. These actions are sequential in nature and provide a template for the manual management of projects.

In this type of model, the REP itself is typically separated into three stages: definition, description and validation [LK95, KS98], though different models may have their own separations. The *definition* stage draws attention to the need to understand the problem domain of the system that is to be developed. It is presumed that, after this stage, the developers are well aware of the domain knowledge. The *description* stage aims to document the understanding of requirements gained from the previous stage at an appropriate level of detail. These documents, expressed in a formal manner, are the main communication medium between developers and users. Finally, at the *validation* stage, there should be a thorough certification of the documented requirements to ensure consistency and completeness. The purpose of this stage is to detect problems in requirements documents before the documents are used by designers as a blueprint for the development of the system.

Due to the differences between individual systems in respect of scope, objectives, complexities and deliverables, and the difference between individual people in respect of their knowledge and experience, it is not surprising that developers employ many different methods to serve the purpose of each stage in the REP [LK95, Gog94]. Different people usually enact a process to tackle a problem in different ways, and even one individual may not be consistent in his/her choice of problem-solving strategy. This is because the methods used for enacting the activities of a process depend largely on the specific contexts of the people involved and the environment in which the process is enacted. If the stipulated stages of the activity-oriented models are followed, it is argued that the needs of users can be correctly captured, formalised and represented in requirements specifications.

- Decision-oriented

Unlike the context-free accounts provided by the product- and process-oriented models for REP, the decision-oriented type of model centres on the contextual aspect of decision [And94, JP93, Rei92, RL93, Rol94, STM95]. In general, the context of the domain knowledge of the software system under development is not clear in advance and is very unpredictable. Therefore, the decision-oriented models argue that developers should be able to react with flexible analysis decisions to rapidly changing situations. To achieve this, the enacted model should allow developers to advance the REP by taking advantage of the domain knowledge that they have established by analogy with the previous situations in which they have been involved. In other words, this kind of model couples the context of the domain knowledge associated with a decision to the decision itself within the REP. Such models explain not only how the process is carried on but also why there is a transformation of the output[1] of the process. As a result, the risk of misunderstanding the users' needs can be significantly reduced.

Typically, a decision-oriented model[2] views the REP as a sequence of building blocks, each of which is composed of a set of interrelated concepts, such as situation, decision, action and argument, which contribute to the context definition. A *situation* is most often a part of the output under development and serves to make sense of how a decision is made. A *decision* guiding the REP reflects a choice that developers make at a particular time in the REP. An *action* performing a transformation on the output changes the context of the domain knowledge and may reveal new situations that in turn are subjects for new decisions. *Arguments* are statements that lend support to or detract from decisions within a given context. Developers make progress in the REP through dealing with a context, that is, taking the appropriate decision in the right situation on the basis of the current domain knowledge. Developers can refine the context by considering various alternative scenarios, all of which have a bearing on the decision to be reached in this context. In other words, the aim of decision-oriented models is not only to capture the activities performed during the REP but also to record why these activities are performed and when. Such an approach is intended to make it possible to determine retrospectively what decisions were taken and what were the contexts for these decisions. With these decision-making blocks, it is argued that requirements can be understood and refined.

[1] The output of a process can be in different forms, e.g. a prototype, a conceptual schema, a logical schema or the implemented software system [Rol93]. Although the conventional output of REP is requirements specifications, here "output" is used as a broad term to include diverse outputs.

[2] This paragraph is based on the reports of a famous long-term project called NATURE (Novel Approaches to Theory Underlying Requirements Engineering) [Rol93, Rol94, JP94, JP93, JPRS94]. The decision-oriented process model proposed by this project is one of the most important models in this area.

7.1.2 Difficulties Within the REP

As described above, many models have been proposed to guide the REP. However, in many cases requirements are still gathered, analysed and implemented through a great amount of informal interaction between users and developers, trial and error, and the ingenuity of a few individuals [LK95]. One of the main reasons for this is that most models of the REP offer developers well-defined guidelines for specifying requirements rather than for solving the practical problems arising from developing requirements. For example, many models suggest that developers should collect domain knowledge from existing documents, but few of them tell developers what to do when these documents are not consistent with users' practices. The documents may stipulate a detailed procedure for users to follow, but what the users actually do in practice may reflect their precious experience accumulated during many years of work. Developers are often puzzled at having to decide which view of user practices should be considered. In effect, given different contexts, developers are required to take pragmatic actions that are best suited for solving particular problems. Since it is impossible for the suggested guidelines for the REP to take all possible contexts of users into consideration, it is not surprising that these guidelines are of limited use in the real world.

In practice, there is increasing consensus that requirements are not usually pre-existent and hidden in the experts' head waiting to be dug out and put into the specification cabinet [BCDS93]. Neither can they be completely described in any form of logical algorithm. On the contrary, requirements are designed and developed through the participants' interaction [Bub95] and are always liable to change. The simple distinction between *what* and *how* (traditional in discussing specification and implementation) is inappropriate and inadequate [Dav93, SB82, SS96] because complex requirements are rarely complete and are liable to evolve faster than the REP itself proceeds [Bub95].

As K. Ryan stated in [Rya95], RE is located at the intersection of a formally based technology and an essentially informal world. By concentrating only on the technical side, most models of the REP have left developers with a number of difficulties when these models are enacted in the real world. These difficulties mainly come from two sources: the process model itself and the participants involved in the REP.

RE is more easily described by its products than its process. Current understandings of the REP are dominated by phase-based models, whether they be product, activity or decision oriented (see Subsection 7.1.1), in which a degree of rational planning through a rigid sequence of prescribed phases is assumed. The character of each phase reflects engineering practice, that is, the application of proven methods, techniques and tools in a systematic and cost-effective fashion. However, experience shows that the REP might not be as simple as these traditional models suggest, in particular for open requirements which are poorly understood and dynamic [Blu93, Gog94, HED93].

The actual situation is usually that developers, according to the different contexts involved, exploit diverse activities to understand requirements. These activities cannot easily be invoked by following a predefined order or an algorithm. Instead, the sequence may be decided by accident and varies in accordance with different situations. Also, the activity undertaken may bridge several phases rather than be confined to a single phase in the REP. This is illustrated in the case of prototyping techniques that involve both high-level conceptual design and low-level implementation. These cannot then be assigned to one particular phase, but assist the performance of tasks in several phases: analysis, design and validation [And94, LK95]. In fact, there is a very popular trend in the software community towards regarding SSD as a non-linear phase-based life cycle [Boe88, Leh97, Pre97, Rac95]. In the same manner, the REP should not be restricted to step-by-step algorithms [Gog96, Rya95, SS96].

Furthermore, since RE is regarded as an early stage of SSD, one of the aims of most traditional models for the REP is to freeze the domain knowledge. Freezing domain knowledge is essential for sensible use of orthodox techniques in the remaining stages of SSD, such as design, implementation and validation, in the specified domain; otherwise, these techniques are not applicable. To achieve this objective, many models for the REP seek to specify the domain knowledge in a formal or semi-formal description. Such specification can not only clarify the developers' understanding of the domain but also record the users' needs for the system being developed.

Unfortunately, users' needs are usually represented by fragmentary, individual, ambiguous and unorganised statements. This is partly because of industrial specialisation: individual users, limited by their own particular professional knowledge, are only familiar with individual parts of the whole system. It is also partly because of tacit knowledge: users are often able to do things without being able to describe precisely and systematically *how* they do them [Gog96]. For these reasons, it is very difficult to prevent the domain knowledge from changing. Moreover, users' environments are characterised by uncertainty. Not only the solution domain (where the real needs of users are identified and represented) but also the problem domain (the application domain where the users' needs are produced and used) is likely to change. C. Potts's field study survey of 23 software-development organisations confirmed that in users' environments, requirements change rapidly [Pot93]. In this context, it is evident that changing domain knowledge is the norm rather than the exception within the REP [HED93, RL93].

The essential contradiction between most traditional models and their practices over the status of domain knowledge leads to a major gap between research and practice. Difficulties of enacting these traditional models in practice inevitably emerge.

One of the most important reasons for the gap between research and practice is that traditional models for the REP are context-free. Most phases of a manufacturing process are formalised by applying proven methods, techniques and tools in a systematic and cost-effective fashion. The engineering discipline confers a well-defined character upon each of these phases, so that they can be repeatedly invoked without taking account of their context. However, the REP differs from a manufacturing process: most activities invoked in the REP are inherently ill-defined and ill-structured, and are hence inseparable from their contexts, especially the social context [Gog94, HORRS95]. They are often associated with the knowledge and experience of the actors who undertake the activities, and with the contexts of the organisations and environments in which the activities are invoked. Since, in general, the rigorous formalisation of activities cannot keep pace with the rate of contextual change over time, formalising these context-dependent activities within the rapidly changing real world remains a major challenge for most process models. (For example, the activity of understanding and eliciting requirements from documents and users' statements is very hard to formalise, at least with the current state-of-the-art technology.)

In order to avoid this contextual problem, the conventional approach is to view these activities at a higher-level abstraction where the change is no longer significant. By means of such abstraction, context-free models can provide developers with instruction-like abstract activities to guide the REP, such as 'definition', 'elicitation', 'understanding', 'specification', and so on. No contextual details have to be considered in these abstract activities, since they are not the concern of these models. The product-oriented and activity-oriented models mentioned above are in principle based on this context-free abstraction. Although these models have been gradually improving and are definitely helpful for developing well-defined and well-structured software systems, some researchers have confirmed the difficulty of enacting such context-free models in the real world [Bub95, Eas93, EM95, Gog97, HED93, HORRS95]. In addition, even though decision-oriented models try to take the context into account, they are still of limited use in a real world of constant change [Rya95]. These models break up the REP into many decision-making building blocks, in each of which the temporal aspect is explicitly modelled. Within each block, developers, after making a choice, take an action to transfer the old output to a new output in order to keep up with the new context of the application domain. However, the application domain often changes too fast, so that the new context emerges before the transformation of the output is finished. This makes the new output out-of-date again in the new context.

A more serious difficulty results from poor communication between users and developers [Bub95, Eas93, Pot93, Son93, STM95, VPC98, Zav95]. It is commonly recognised that user participation is helpful for requirements development, in particular for those systems whose domain knowledge is not well understood [EQM96]. However, a well-known communication problem occurs: users have domain-specific knowledge and use the vocabulary of their domain, whereas developers are familiar with information requirements methodologies and use the vocabulary of software development. On the one hand, users may not be able to express their needs in the technical terms understood by developers. On the other hand, developers may have difficulties in understanding the professional terminology of users. For example, object-oriented techniques have been widely applied to software engineering [Boo94, CY90, Jac92, JCJO92]. Developers may be keen to understand requirements in an object-oriented fashion. However, it is very difficult for users to express their needs in terms of objects and classes [BE94, McG92, OS93, Pot93a, Zuc93]. As a result, the communication obstacle between developers and users inevitably gives rise to errors in understanding the acquired requirements. These errors, embedded in the developed system, should hopefully be detected before the system is in operation. This is especially important in the case of safety critical systems. The cost of detecting and correcting these errors is inevitably very high. Previous attempts to solve the communication problem have so far resulted in little progress towards a satisfactory solution [STM95].

In addition, most approaches to REP aim to achieve an agreed set of requirement specifications in text and diagrams [Poh93]. They seek to document requirements in a detailed fashion. However, paper is passive and can only serve as a repository for collected information. It is hard for users and developers to know whether or not there are differences between their interpretations of the same text. Many users sign off requirement specifications without fully understanding the implications. It is usually difficult for users to validate the technical documentation used by developers and designers. In fact, users can often identify their true requirements only by experiencing the operation of the system [HED93, RL93, LL94]. This is because they are familiar with the operation for solving a problem in practice, but are unable to recognise its specialised, abstract description in the specifications for the system.

Most process models for the REP fail to support group work effectively [Bub95, Eas93, JP94]. The REP is dominated by participants from different backgrounds. They may be responsible for different goals and may not be aware of each other's goals. They work together to embody their individual goals into the developed requirements. By means of the successive interactions between participants, requirements are evolved and hopefully move toward a consensus. (The trend of moving towards an agreement between participants is highlighted in one of RE's three-dimensional models proposed in [Poh93].) The evolution, especially for open requirements, must be supported by collaborative group work between participants.

In practice, the method of working collaboratively in a distributed environment has been an economically necessary and efficient means of production in modern industrial societies. Any process model for the REP should be able to support, in an effective and efficient manner, collaboration among participants in a distributed environment.

However, most models for the REP are *developer-centred*, in that users are passive and need only contribute to information provision. It is the developers, whose professional experience is in different fields, such as computer science, who determine what information is needed and how to integrate and embody the required information into the intended system [You83]. They wait for users' contributions before proceeding with further activities. Process models that are centred on the developers' tasks cannot easily support group work between all participants in a collaborative fashion.

An exception to this kind of developer-centred models is provided by viewpoint-oriented models, in which requirements are developed on the basis of different perspectives or views describing parts of the intended system [KS98, NJJZH96]. These models regard the combination of a participant and his/her view as a viewpoint, and seek to provide a framework for organising and structuring viewpoints for requirements development. Though these models implement a general feature of group work, they are carried out in a centralised manner. Developers are responsible for the integration of all viewpoints. The interaction between users and developers is to a large extent similar to that in other models, except that it can conducted in a distributed fashion.

The challenge that is addressed in this chapter is that of providing a framework for the REP which recognises the difficulties identified above and provides participants with an alternative means to support their work in developing requirements. To achieve this goal, reengineering of the REP is vitally important.

7.2 Reengineering the REP

As already explained, the main cause of the difficulties described in the previous section is that most models for the REP fail to take account of the situatedness of requirements. J. Goguen argues that requirements are situated – emergent, local, contingent, embodied, open and vague – and can only be understood in relation to the concrete situation in which they occur [Gog96, Gog94]. This situatedness demands that developers should take sufficient account of context in order to satisfy the actual needs of users, so that the developed system can solve the users' problems in the real world. The context for requirements is the real-world environment in which requirements are developed and the system is used. The environment is deeply affected by its social and organisational structure and the people therein.

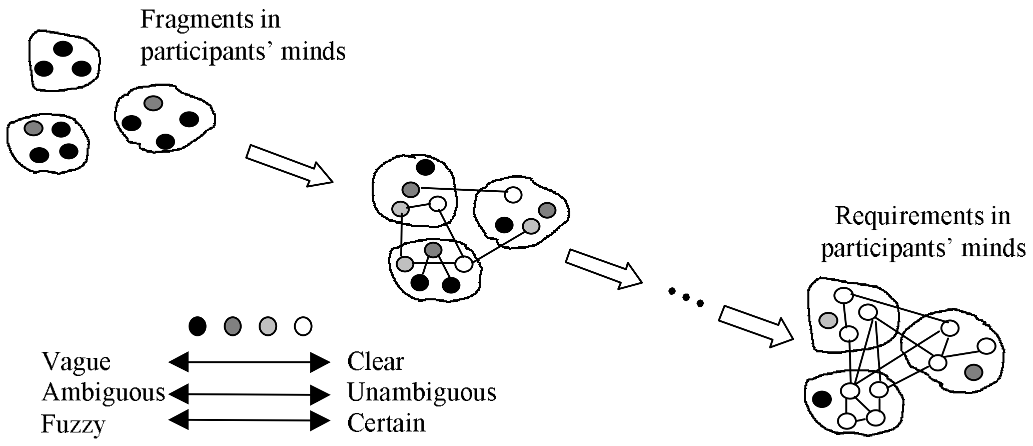
Even though the issue of context has attracted widespread attention in the RE community for many years [Bub95, HED93, HORRS95, Pot93, Sid94, SS96], most models for the REP have made little progress on this issue due to the difficulty of supporting situatedness by a step-by-step algorithm [BL98, Gog97]. To avoid being trapped in the same situation that leads to the practical difficulties of traditional process models, it is worth reengineering the REP from scratch by considering the original process of requirements development without following the algorithms of any particular model.

In this thesis, the term 'requirements' is defined as "a condition or capability that must be met or possessed by a system to satisfy the condition or capacity needed by a user to *solve a problem* or achieve an objective" (paraphrasing the definition in [IEEE90] cited in Section 7.1.1). This definition acknowledges the usefulness of descriptive documentation as a resource for RE, but does not overstate the extent to which requirements can be captured via documented representation describing the behaviour, properties and constraints of the system which is to be developed [KS98, LK95, SS97]. Traditional definitions of requirements are more concerned with the advantages of using requirements as a contract between users and developers and as a blueprint for designers. The definition adopted in this thesis, which attempts to relieve or even eliminate the difficulties of enacting the REP, is more concerned with the real-world context, since it addresses the production and use of requirements for the intended system in the real world. It also recognises the importance of reconciling social and technical issues in RE.

In keeping with this definition, the process of developing requirements amounts to the process of providing solutions to identified problems that arise in conceiving the intended application in its domain. This problem-solving process should involve all relevant participants in order to collect all necessary information. An informal account of how this process appears to operate in practice follows below. Later sections will describe the way in which the situated process of requirements engineering (SPORE), when combined with DEM, can support this process.

At the outset, some *fragments* associated with solving the identified problem emerge from the subconscious minds of individual participants in the form of concepts, ideas, intentions, expectations, experiences, and so on. Many of these fragments may be ambiguous, chaotic, vague and very difficult to articulate or record. In order to clarify these fragments, participants must undertake certain activities that involve interacting with each other and introspecting about their own mental model. Very common activities include, for example, interviewing, brainstorming, video recording [HORRS95], prototyping [And94, LR91, Luq93, RL93], goal analysis [RSB98], form analysis, scenario analysis [Hol90, WPJH98], and so on. For the sake of convenience, as in Chapter 2, the term 'interaction' is used to refer to both interaction and introspection.

When participants start to interact with each other, their individual fragments of knowledge change: some of them disappear, but some new ones also emerge. The most significant change arising from the interaction is that some of these fragments move towards being unambiguous, ordered and clear. The interaction is continued until some fragments finally become intelligible to all participants, and can be identified and represented in terms of primary elements agreed and apprehended by participants. These elements could, for example, take the form of objects and classes in an object-oriented model [Boo94], entities and relations in an entity-relation model [Che76], viewpoints in a viewpoint-oriented model [KS98], or simply statements in natural language. No matter how they are represented, the intelligible elements are not fixed but are instead liable to change.



Fragments in participants' minds
Requirements in participants' minds
Vague ↔ Clear
Ambiguous ↔ Unambiguous
Fuzzy ↔ Certain

Figure 7-1. Requirements formulation: from fragments to requirements

As the interaction continues, more and more elements are obtained and coupled with the existing ones to form a web of interconnected elements. Incrementally, this web should converge to a provisional solution to the identified problem. At that point, requirements providing a solution to the identified problem are developed. Whilst there is no such convergence, the interaction must be continued until a provisional solution emerges. Otherwise, requirements for the identified problem cannot be obtained and the success of SSD becomes problematic. Clearly, the provisional solution is not fixed. Its integrity could be undermined or its form changed at any moment as yet more intelligible elements emerge. Section 7.3 will explain how EM can support the process of composing fragments into a provisional solution illustrated in Figure 7-1 through 'structural coupling' (as described in Section 3.2).

Such a REP based on structural coupling, whereby new elements are dynamically coupled with the existing elements, is difficult to achieve by the traditional top-down or bottom-up approaches that are typically used in models for the REP. Top-down decomposition presumes that the organisation of fragments is broadly established, and is not applicable until the requirements of the developing system are sufficiently well understood [Blu93]. Bottom-up analysis generates fragments of the requirement that are exactly prescribed, and are therefore not suitable for representing vague, ambiguous or fuzzy requirements.

One of the principles of reengineering the REP is to incorporate activities that are normally undertaken by participants together with their context. It must be possible within the REP to accommodate any interaction which it is within the competence of a participant to choose as the most effective way to improve the current provisional solution. Neither specific actions nor their sequence are rigidly stipulated in advance for serving such a purpose. Instead, the development of requirements is fulfilled through what L. A. Suchman has called 'situated actions', in which performance is matched to the specific task situations existing at the time [Suc87]. The fact that the interaction between participants is appropriately situated contributes significantly to the growth of understanding and experience as the interaction continues.

Requirements cannot be isolated from the subsequent development and operation of a software system. As explained earlier, requirements are closely associated with the context of use in which the system is operated to solve users' problems. On the one hand, the system is implemented in order to provide users with the solution represented by the description of the developed requirements. On the other hand, the developed requirements are validated and clarified through the operation of the system in the real world, and the operation may in turn bring out the need for new requirements or a change in old requirements. The contextual dependence between requirements and the system forces the REP to be intertwined with the process of SSD in a symbiotic fashion as illustrated in Figure 7-2. The interdependency between SSD and the REP fits in well with the increasingly popular arguments that the REP is never complete but should be continued throughout the whole life cycle of SSD [BL98, CGC96, Gog96, JP94, Rya95]

The metaphor shift in requirements development is analogous to that in software development. In [Bro87], Brooks highlights the fact that the *building* metaphor, which

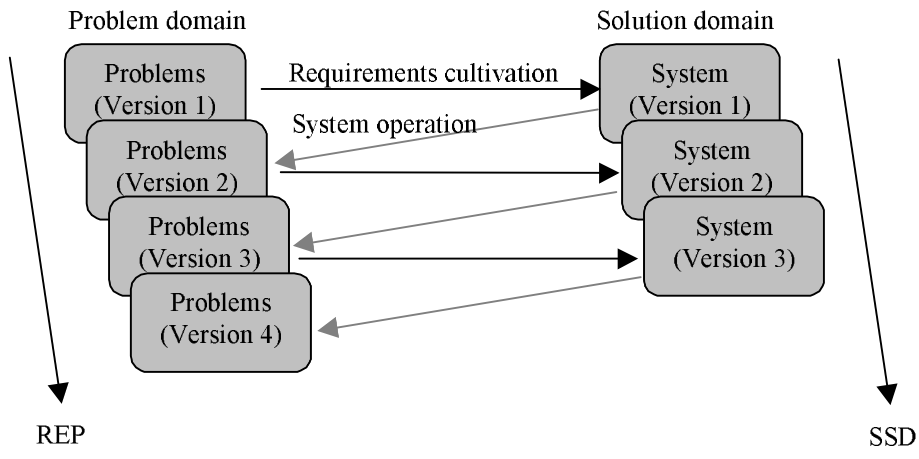


Figure 7-2 The interdependency between SSD & REP

likens the way in which software is constructed to a building process, has outlived its usefulness, since software systems have become so complex that they cannot be fully specified and designed in advance. He suggests that any software system should be grown by incremental development. In fact, the *growing* metaphor should be also applied to requirements development, due to the contextual dependence described above. It is more appropriate to think of developing incremental requirements as opposed to eliciting or acquiring ones from sources directly, which is the usual method employed by many traditional models for the REP [DBP93a, DBP93b, FHW94, Rei92, SDV96, Sut96, KS98]. The concept of incremental development is consistent with the concerns of changing requirements [Gog94, HED93].

It is evident that the interaction between participants provides the main impetus for the process of developing requirements discussed here. A well-known drawback of most RE models is that it takes the effective interaction between participants for granted. C. Potts's field study shows that interaction breakdown is the major problem in the REP [Pot93]. In fact, the information arising from the interaction between participants is the main resource for requirements development. A process model should not hinder the emergence of the essential information, but should facilitate such emergence by supporting the interaction between participants as much as possible. Recognising this need, therefore, the alternative framework proposed in Section 7-3 exploits computers as the best communication medium for achieving this purpose.

Another important principle of reengineering the REP is that the REP is guided by participants and not by a process model. Within most models, human behaviour is embedded into the mechanism of enacting the REP by assuming the invariability of human factors and the context of requirements. Although this assumption reduces the uncertainty surrounding human beings and their environment, it accordingly generates a gap between research and practice, thereby leading to the difficulties discussed in the previous section. This is because the inflexible mechanism hinders the essential ability of human beings to accommodate themselves to the rapidly changing environment. As the REP is located at the intersection of formal and informal, of objective and subjective, and of technical and non-technical approaches, it needs to rely to a large extent not only on the participation of human beings but also on their accommodating nature. Any process model should recognise the existence of uncertainty, and make due provision to autonomous interpersonal interaction to this end. For instance, it is not in general appropriate to presume that interpersonal interaction is so reliable that it can be replaced by a mechanism. It may also be necessary to allow human intervention to mimic the unreliability of mechanism. This echoes Tully's concern for enacting a software process model as a symbiosis of human agent and computer that does not hint at particular roles for either partner (see [Tul88] cited in Section 2.2.2).

In spite of the importance of individual experience and knowledge, the transition from informal, fuzzy statements to formal, unambiguous requirements usually needs to be carried out through interactions between all participants. The different relationships that can shape this interaction have provided the foundation for most models and methods in requirements engineering. There is a particularly significant distinction between coordinative and subordinative relationships. A *coordinative* relationship stresses the importance of user participation in design, and postulates responsibilities for all the participants. A *subordinative* relationship assumes that users should be responsible for providing all the knowledge required by designers because only they know what they want.

Traditional patterns of interaction favour relationships of these two kinds, since they presume a clearer separation between analysis, design and use that modern business practice and associated information technology promotes. In the development of information systems, it is standard practice for feedback from users to affect the product. This feedback operates both in validating and debugging the original design, and in its subsequent enhancement. In the concurrent engineering of other products, the use of information technology has subverted the rigid sequential stages of the traditional design process. The ease with which design representations can be visualised and modified enables wider and more opportunistic intervention from all kinds of participants. In these contexts, the interaction for developing requirements becomes exceedingly subtle [SKVS95]. In effect, the design of a software system and the shaping of the requirements satisfying the needs of users often have to be negotiated in a symbiotic fashion. The interaction amongst all participants that is appropriate in this context will be characterised as a *collaborative* relationship.

A useful analogy can be drawn between the relationships of all participants for developing requirements and the relationship between a teacher and pupils in a classroom. A subordinative relationship resembles the context of a lecture context, where the teacher imparts knowledge in the role of the expert, and there is no participation from the pupils. A coordinative relationship, in which a rigid agreement sets out the respective responsibilities of designers and users, resembles a tutorial context in which the teacher imparts knowledge through a prescribed pattern of small presentations, exercises for the pupils and evaluation of their performance. A collaborative relationship is concerned not only with responsibilities but also with expectations, beliefs and other psychological states that make developing by learning more feasible and powerful [DL91]. The appropriate context for such interaction resembles a seminar, where the precise learning goals are not set out initially, and the knowledge content is shaped dynamically by the contributions of the participants. In the same way that all three paradigms can be used in one educational context, each of the three different kinds of relationship amongst all participants can be represented in the same process of developing requirements.

Collaborative relationships are concerned with interaction that is socially distributed. They engage with issues of subjectivity and objectivity associated with distributed cognition [Hut95] and common knowledge [Cro94, Edw87]. This involves a reappraisal of distinctions that are taken for granted in other contexts. There is a potential for several kinds of conflation:

- between the roles of all participants,
- between the properties associated with individuals and with artefacts,

- between the characteristics to be attributed to the internal mind and to the external environment.

In a collaborative relationship, there is typically no possibility of relying entirely upon closed-world representations and preconceived patterns of interaction. The interaction between all participants has to be situated intelligent interaction that can only be planned in advance to a limited degree, and domain knowledge for the process of developing requirements emerges on-the-fly.

Supporting the situatedness of the REP is not a trivial task. Firstly, an environment that enables participants to interact with each other in a collaborative manner is necessary. All participants involved in the REP share the responsibility of developing the requirements that satisfy the users' need to solve problems in the real world. Not only developers but also users are responsible for the success of requirements development. The most common method, called introspection, that is embedded in most process models for developers to collect information about the users' needs and habit, cannot serve the purpose of supporting collaboration [Gog97].

Secondly, each participant must be sufficiently qualified to make his/her actions accountable to others. Considering the above example of a classroom, common knowledge will obviously not be established unless the teacher is capable of taking actions which make sense to a pupil, and *vice versa*. This may entail a coordinative relationship, or even a subordinative one in which the responsibilities of each participant are stipulated. Similarly, within the REP, if a participant is incapable of interacting with others, it is inevitable that information pertinent to that participant will be missed. Accordingly, this is likely to give rise to major problems. Fortunately, advances in end-user computing have increasingly reduced the problem of incapability [DL91].

Needless to say, the crux of supporting the situatedness of the REP resides in the construction of the provisional solution in an open-ended, interactive fashion. Obviously, if a fixed problem domain of the application is specified, it is not too difficult to find a solution by means of so many existing tools and techniques that contribute to the search process. However, in order to cope with the dynamics of context, the solution to the identified problem, and even the problem itself, must keep changing in response to new domain knowledge emerging from the interaction between participants. Therefore, the solution needs to be constructed incrementally and interactively. Few tools and models support the construction of a changing solution, or, more precisely, of changing requirements. This is partly because of the technical difficulty in coupling the old solution with a new context.

One of the main contributions of this thesis is to demonstrate that the computer-based interactive modelling technique discussed in Chapter 4 has the potential to support the situatedness of the REP in a significant way. The next section introduces a novel framework for the REP motivated by the perspective on reengineering the REP described here. The principles and concepts of DEM are applied to the framework in order to provide a human-centred, computer-based environment to support the REP.

7.3 A Situated Process of Requirements Engineering

Due to the need to model the real world in which the target systems reside, to manage many fragmentary yet interrelated requirements statements, and to cope with changing assumptions and perceptions of requirements, the REP must be situated and human-centred. This section provides a novel framework for the situated process of requirements engineering. First, the framework called SPORE is proposed. Within this framework, people participating in the REP are able to cultivate requirements through collaborative interaction with each other in order to solve the identified problems, instead of searching for requirements in the 'jungle' of users' needs. The environment supporting the framework is established by applying the principles and concepts of DEM. By means of a computer-based *interactive situation model* (ISM), participants can collaboratively interact with each other to 'grow' requirements in an incremental development fashion.

7.3.1 A Framework for the REP

According to the definition of requirements and the principles of reengineering the REP presented in the previous section, requirements may be seen to provide solutions to identified problems. The REP begins in the problem domain associated with the requirements of the developing system. This domain is generally informal, situated and open to the real world [Gog96, Blu93]; hence it cannot be specified completely in advance. Instead, the domain is represented by a situated, provisional, subjective, but computer-based, model. It is *situated* because it is represented as organically connected to its referent (the domain). Such connection is achieved by being continuously open to revision through a comparison between the experiences of interaction with the domain and those of interaction with the model. Accordingly, the model is not divorced from the domain, as required for a preconceived 'system' with boundaries made sharp by some form of idealisation or abstraction.

A human-centred framework, called SPORE, for building situated models for the process of requirements engineering, is depicted in Figure 7-3. Key problems of the domain are identified by the participants within the grey box in Figure 7-3 with reference to their concerns for the functional, non-functional and enterprise attributes of the

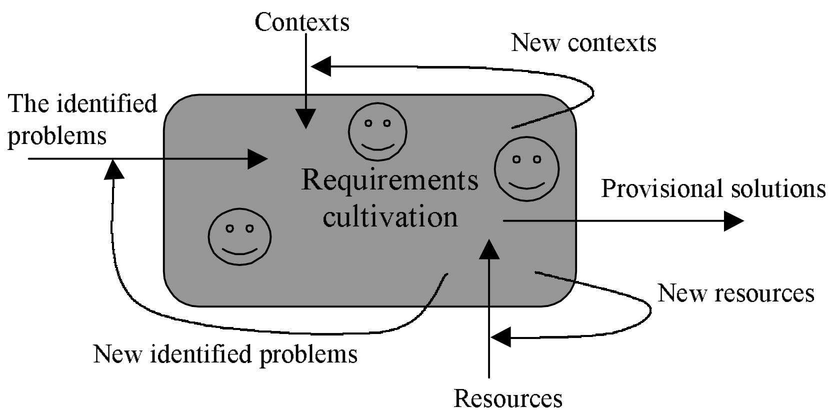


Figure 7-3. The SPORE framework

developing system. The identification of problems can occur at any time during the REP and is never regarded as being completed. Another two inputs of the SPORE model are the available resources and the current contexts. The resources, such as documents, technology and the past experiences of participants, are used by participants to facilitate the creation of the SPORE model's outputs. The contexts, such as the organisation's goals and policy, and the relationships between participants, act as motives and constraints for the participants in creating the outputs. These three kinds of input may impact on different parts of the model at different stages of its evolution. The arrows ending at the inside of the grey box in Figure 7-3 convey this idea.

A SPORE model has outputs of four kinds. The most important one consists of solutions to the identified problems. These are developed by participants on the basis of the available resources and the current contexts. Moreover, the other outputs, including new contexts, new resources and new problems, combine with their earlier versions and form new inputs for creating the next output. That is to say, all these contexts, resources and identified problems, even during the development of solutions, are still modifiable and extensible. In view of this, participants can develop requirements in a situated manner to respond to the changes in the contexts, resources and even the problems themselves. This implies that requirements are apt to change all the time and thus are never completed. In this respect, the SPORE framework is consistent with J. Goguen's concern for the situatedness of requirements [Gog94].

The SPORE framework determines neither specific activities nor their sequence. In many cases, several problems can be identified simultaneously. Some may be very difficult to solve under the current contexts and resources, but others are not. Some are interdependent and need to be solved concurrently, but some can be solved independently. Different problems are likely to need solution by different methods. No rule or algorithm can be postulated in advance to take all these factors into account. A generic strategy for taking actions is 'divide and conquer', where the highest priority is to undertake action for the easiest problem. But this is not a golden rule. Participants must still take their current context and available resources into consideration in order to cope with the diverse issues arising from the development of solutions.

The central activity in the SPORE framework is the requirements cultivation, in which participants interact with each other and with their environments to develop requirements, i.e. the emerging solutions to the identified problems. The term 'cultivation' is used to convey the idea that requirements (like plants) should grow gradually rather than be conjectured from their initially fragmentary, chaotic and rapidly changing states. It also emphasises the use of deliberate design activities by participants on the basis of their contexts and available resources in order to develop requirements in an effective and efficient fashion. Some models for the REP assume that requirements are pre-existent but hidden in some sources [LK95, SS97], just like grown plants in a huge jungle. The purpose of building these models is to search for (elicit) the right plants (requirements) in (from) the jungle (available sources, such as documentation and the expertise of users). Typically, the jungle is a mixture of numerous kinds of elements that are fluctuating in response to the changes in their environment. It is clear that searching in such a jungle for one element, which has never been seen before and might keep changing all the time, remains a very difficult challenge [LK95, Bub95].

The concept of requirements cultivation, unlike that of searching for requirements in a jungle, refers to the 'growing' of requirements for the developing system by participants themselves through their collaborative interaction. The cultivating process focuses on neither the problem domain nor the solution domain but instead on the interaction through which participants seek to solve the identified problems on the basis of their current context and available resources. For example, let us consider the development of a simplified automated teller machine (ATM) which contains an embedded software system to drive the machine hardware and to communicate with the bank's customer database. In order to acquire the requirements of the software system, a problem of accessing the service is identified. The participants relevant to the identified problem, such as customers, bank staffs, machine designers, database managers, security officers, software designers and so on, must work together to solve the problem. The solution is not located in someone's head but is socially distributed across all participants. Also, it is formed and shaped through the iterative and creative activities invoked by participants in their interaction with each other. In this sense, requirements are cultivated by participants through a variety of purposeful activities.

The work of requirements cultivation can be focused further on individual participants. Within the REP, each participant has his/her own individual insight into the identified problems and their solutions. This insight is based on the participants' various contexts and available resources. It is clear that individual insight is often of limited use and inevitably has a bias. For example, in the ATM example mentioned above, for reasons of security, bank staff may demand more rigorous security checking for access to the service provided by an ATM machine. But from the customer's viewpoint, the convenience of using the service might be the main concern.

7.3.2 Applying DEM to SPORE

There are several approaches to cultivating requirements, but one of the most efficient and cost-effective ways is by computer-supported modelling. Conventional computer-based modelling is better oriented towards assisting subordinative and coordinative, rather than collaborative, relationships. To fully support the collaborative interaction between participants discussed above, it is essential to establish an individual ISM for each participant within a distributed environment that:

- allows data about requirements to be collected in such a way that participants are engaged in activities in their customary context [Gog96, LK95];
- makes it possible to explore and experiment with individual insights for different participants;
- provides for open-ended interaction.

Given the principles and concepts of DEM described in Chapter 4, each participant can construct an ISM within an interactive, distributed environment supported by dtkeden. According to the principles of SPORE, the cultivation of requirements has to stem from a representation of those identified fragments that are pertinent to the identified problem being addressed. This representation will take the form of a *seed* ISM that incorporates matter-of-fact observations of the current context. An ISM to represent these observations will supply a visual representation for those identified fragments. Participants can thus interact with their own ISM to extend, expand and explore their individual insights through 'what if' experiments resembling interaction with a spreadsheet. The accumulated results of experiments not only change the participant's individual insight immediately but also are stored in the memory of the participant. The latter is the most important resource used by the participant for taking situated actions.

In effect, this experimental interaction, using the computer as a modelling medium, can – when integrated with other methods – provide more accurate and more powerful resources for developing solutions to the identified problems. In this sense, interacting with the computer model becomes a very critical situated action for creating a resource for further actions. Figure 7-4 illustrates how a participant can take situated actions by interacting with his/her computer model and/or external environment on the basis of various contexts and available resources to explore individual insight. The insight into the identified problems and their solutions evolves with the interaction. This will be illustrated with reference to developing the requirements for an ATM system, as described in the next section.

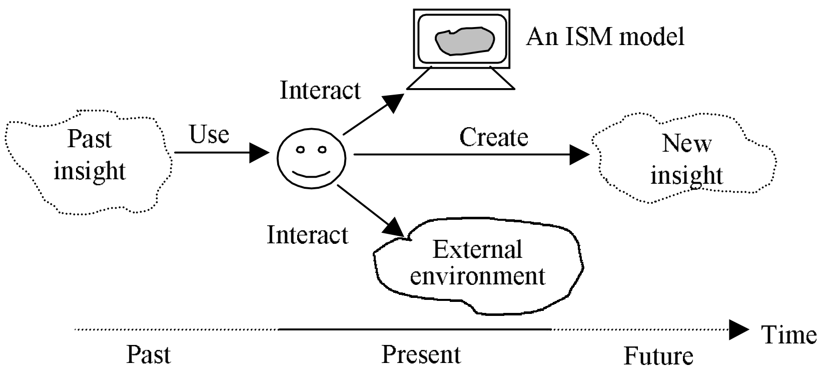


Figure 7-4. The experimental interaction of a participant

The ISM, with visualisation corresponding to the observed real world, plays an enabling role in SPORE. Given the visualised scenarios, participants can 'preview' the system in the current context to enhance and explore their understanding of the developing requirements. More importantly, *experiencing* these kinds of visualised scenarios can give users greater confidence that they understand their actions and those of the intended system. The confidence prompted by experience is very significant and useful for getting rid of user resistance in ISD. This concern is consistent with the result of an empirical study proposed in [Kuw93] in which users and developers are mostly concerned with how users themselves can recognise how the system will behave.

To some extent, an ISM is similar to a prototyping model [And94, BD93, DF98, LR91, Rei92, SAGSZ97]. Both focus on "evaluating the accuracy of problem formulation, exploring the range of possible solutions, and determining the required interaction between the proposed system and its environment" [LR91, p.77]. They are both working models, so that their users can have operational experience of what the system should do and how it should look. This experience enables more effective communication between participants to help requirements development, to reduce the risk of misunderstanding and to clarify a designed solution to an identified problem.

However, unless it supports the collaborative interaction between participants, a prototyping model is of limited use in exploring the domain knowledge of the developing system. Typically, a prototype demonstrating a part of the developing system is used in order to understand requirements for providing a solution to an identified problem. Users and developers are separately responsible for model validation and model development. Any improvement of the provisional solution must be fed back to developers so that a new prototype can be reconstructed in a traditional fashion. This 'backward' reconstruction is very different from the 'forward' reconstruction of an ISM in which any domain knowledge emerging from the REP is directly implemented into the ISM in an interactive manner. Feedback is too late and too passive. It constrains participants from exploring unknown territory.

What makes ISMs particularly powerful in this context is that they enable participants to interact with each other in an open-ended, interactive manner. Through the network communications facilities, all ISMs are connected together to create an environment that can be viewed as a radical extension and generalisation of a distributed multi-user spreadsheet. The connection makes it possible to propagate the experimental interaction of each participant with his/her ISM to those of others, so as to consequently affect their individual insights. Participants can interact with their own ISM privately by making a variety of definitions in order to explore their own insight into the identified problems and corresponding solutions. They can also interact with others and their ISMs by propagating definitions through communication networks. The propagated definitions first change the visualisation of others' ISMs (given suitable authorisation) and consequently may change their insights as well. Thus, participants can collaboratively interact with each other through their ISMs and communication networks. Figure 7-5 shows this collaborative working environment.

Within the collaborative working environment described above, a working understanding of the identified problems and their corresponding solutions, that is, of the requirements, is established. This working understanding is distributed across participants rather than in an individual mental representation. It is not expressed by a literal specification that establishes a fixed relationship between the individual ISMs and their referent, but as a commitment to constrain the interaction between participants in a way that respects their common insight, but does not prevent new distinctions from emerging.

The working understanding is then cultivated, that is, grown incrementally, through the successive interaction between participants for exploring and integrating individual insights. Generally speaking, greater consistency between the individual insights is associated with a better working understanding. For this reason, participants continually refine their interaction with a view to achieving more coherence and consistency. This process is open-ended, and consistency can only be achieved in relation to some restricted work activities and assumptions about reliability and commitment. In practice, there are likely to be singular conditions under which a higher viewpoint must be invoked to mediate or arbitrate where there is conflict or inconsistency. The 'global view' perspective depicted in Figure 7-5 represents such an overall viewpoint. It could also be the view of a requirements engineer when acting in the role of negotiator between differing or incompatible insights.

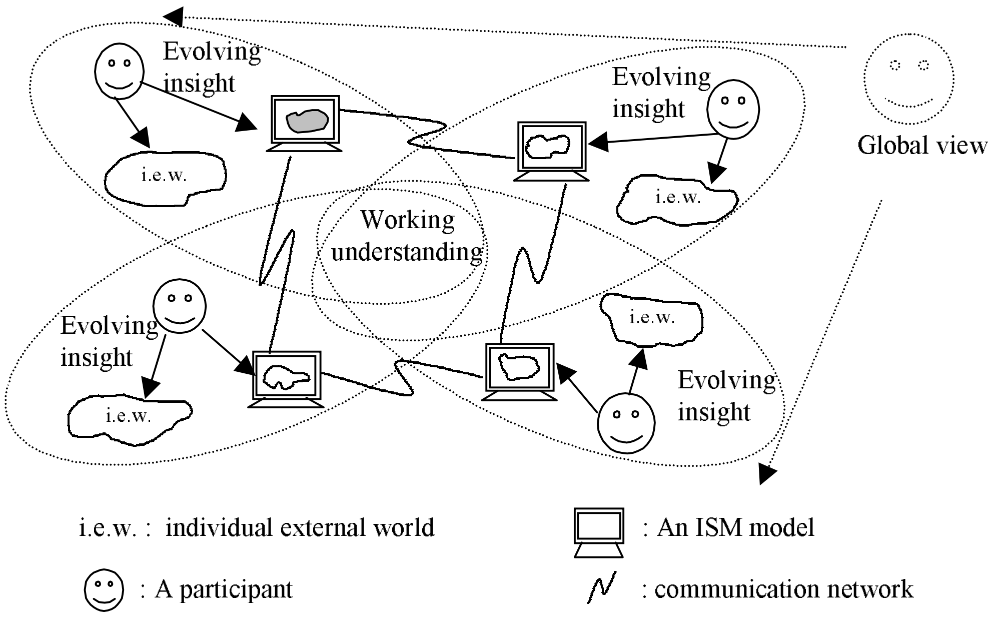


Figure 7-5. A Collaborative working environment for cultivating requirements

The most important benefit of interacting with computer models is to make individual insights and the working understanding between participants *visible* and *communicable*. Of course, most models for the REP involve the interaction between participants in order to facilitate the establishment of the working understanding, for example by requirements elicitation and validation [LK95]. However, the working understanding within these models is invisible and incommunicable. Even given a requirements specification, the visibility and communicability of the working understanding are still restricted to the boundaries of language description and comprehension. Also, paper documentation, as used in a repository or archive, fails to support the needs of its users in exploring and integrating information. In practice, it is very difficult to keep requirements specifications synchronised with the working understanding between participants [DS97, LR91, Luq93], because the latter emerges from experimental interaction and evolves much faster than the evolution of specifications.

In contrast, the experimental interaction between computer models invoked by participants immediately changes the visualisations of these models. The change leads quickly to the evolution of individual insights as well as to a working understanding. The synchronisation between the evolution of computer models and individual insights allows participants to 'see' the viewpoints of other participants and to 'communicate' with them by interacting with their own ISM. In the same manner, the working understanding is also embodied in these computer models. From the perspective of users, the visible and communicable computer models illustrating the solutions to the identified problems represent a crucial contribution to understanding that complements the passive textual descriptions of conventional specifications. In this sense, ISMs are communication media through which the commitments between participants are conveyed. This account of ISMs as 'communication media' fits in well with the view expressed by T. Winograd and F. Flores in [WF86, p.79]: "computers are not only designed in language but are themselves equipment for language".

Whilst the interaction between participants through ISMs has obvious advantages, it should not be thought that other more traditional methods of communication between participants need be foregone and replaced by the computer-mediated communication through ISMs. On the contrary, those methods, such as face-to-face communication, assume even greater importance because they are the best means for compensating for the limitations of computer-mediated communication, such as the absence of the normal social cues inherent in group work [Smi97].

7.4 Two Examples of SPORE

Two examples using the SPORE model for requirements cultivation have been studied. One is the software system embedded in the automated teller machine (ATM). This example has been studied by using the viewpoint-based requirements method (VORD) in [KS98]. A comparison between two models (i.e. SPORE and VORD) for developing requirements is also provided. In the second subsection, another example, relating to a warehouse information system, is given. The example attempts to compare the SPORE model with the use-case approach proposed in [JCJO92].

7.4.1 An ATM Software System[3]

The system embedded in an automated teller machine (ATM) has been used as an example by several researchers in the field of requirements development [KS98, RSB98, SDV96] The ATM system accepts customers requests, produces cash and account information, drives the machine hardware and communicates with the bank's customer database. Multiple participants are involved, such as bank tellers, bank managers, ATM operators, customers, hardware designers, bank database managers, bank security officers and so on. It is a good example to show the development of requirements in a distributed fashion.

The principles and concepts of SPORE have been used to cultivate requirements for the ATM system by the present author. First, participants, such as customers, bank

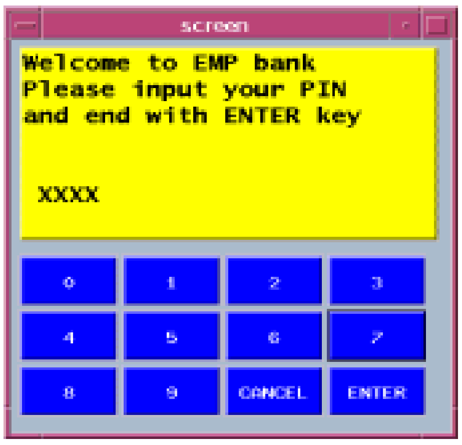
[3] Due to the limitation of the author's research time, the system is not completely accomplished (only the part discussed in this subsection has been finished).

tellers, bank managers, database managers, ATM hardware designers and software designers, are involved. Then, problems are identified by them in order to highlight their respective concerns. Table 7-1 shows some of these problems. It should be noted that they are not frozen, so new problems can be added and old problems may disappear or be changed. Also, it is unnecessary for them to be solved sequentially or independently. A typical strategy for solving problems is 'divide and conquer', where the easiest or the most important problem has the highest priority.

Participants	Identified problems for developing the system embedded in an ATM
Bank tellers	Start-up and shut-down an ATM machine Gaining access to an ATM for administrative services Available services Gaining access to customers' account details The identification of customers The notification of notes deficiency
Customers	The identification of customers Available services Acceptable response time
Hardware designers	Allowing customers to gain access to an ATM Allowing tellers to gain access to an ATM Security control Supporting available services
Software designers	Driving and communicating with hardware devices Interaction with tellers and customers Communication with bank database Supporting available services
Bank managers	Security control of the ATM The identification of customers Accuracy and performance of the requested services Reports of each transaction Reports of each administrative access Cost of each ATM
Bank database designers	Providing and updating the details of customer's account Recording each transcation Security control over database access

Table 7-1. Some problems identified by participants for an ATM system

To illustrate the collaboratively experimental interaction between participants, one of the identified problems (see. Table 7-1) is considered as an example: the identification of a customer accessing an ATM ('the ID problem'). Based on their different contexts and resources, each participant creates his/her own seed ISM and prepares for the



```
Welcome to EMP bank
Please input your PIN
and end with ENTER key

 XXXX
```

Figure 7-6. The ISM of a bank customer

experimental interaction with others. Figure 7-6 shows a snapshot of the seed ISM of a bank customer, as developed using dtkeden. A collaborative working environment for participants in this example has also been constructed (cf. Figure 7-7, for which Figure 7-5 is the archetype). Here requirements engineers interact through 'God's view' to guide the negotiation between participants and the integration of their individual insights in the solution of the ID problem. Table 7-2 illustrates some of their initial insights expressed

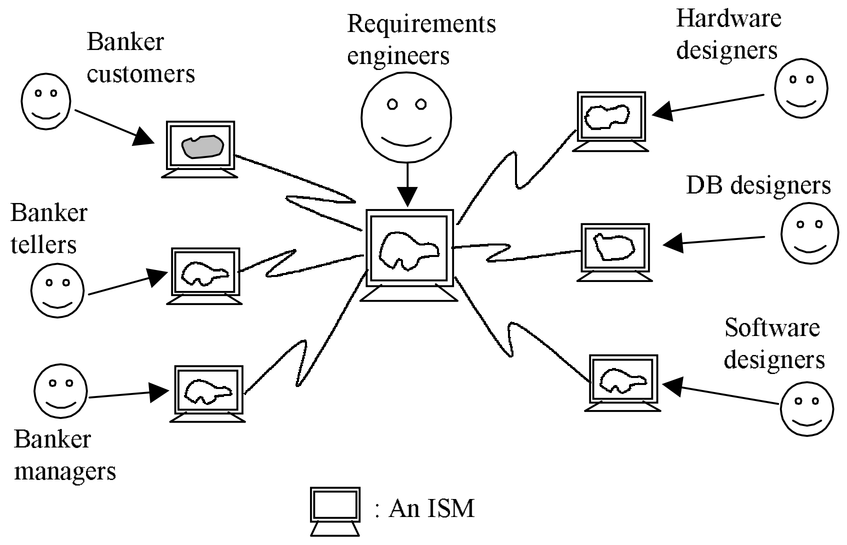


Figure 7-7. A collaborative working environment for an ATM system

using the EM definitive notations. For example, the bank manager is concerned with the safest control of access to the service. Hence, confirmation of user identity, to include the card, the card-holder and the card account, is rigidly demanded.

Participants	Individual insights

Bank tellers	customer_account_NO is card_NO;
	customer_ID is 6-char-PIN_received;
	customer_confirmed is check_customer_ID(customer_account_NO, customer_ID);
	accessing_to_service is customer_confirmed;
	current_screen is
	(accessing_to_service)? send_out_services_manu() : send_out_try_again_screen();

Customers	customer_account_NO is card_NO;
	customer_ID is 4-digit-PIN-received;
	customer_confirmed is check_customer_ID(customer_account_NO, customer_ID);
	accessing_to_service is customer_confirmed;
	current_screen is
	(accessing_to_service)? send_out_services_menu() : send_out_try_again_screen();

Hardware designers	checking_card_logo is (card_being_inserted)? (check_card_logo()) : FALSE;
	card_confirmed is card_logo_confirmed;
	card_magnetic_NO is (read_card_NO)? read_magnetic_NO() : FALSE;
	received_4_digit_PIN is (get_4_digit_PIN)?input_4_digit_PIN() : FALSE;
	received_6_char_PIN is (get_6_char_PIN)? Input_6_char_PIN() : FALSE;

Software designers	get_6_char_PIN is (card_confirmed)?send_input_6_char_PIN_screen(): send_invalid_card_screen();
	read_card_NO is card_confirmed;
	card_NO is card_magnetic_NO;
	6_char_PIN_received is (get_6_char_PIN)? Receive_6_char_PIN() : FALSE;
	get_4_digit_PIN is (card_confirmed)?send_input_4_digit_PIN_screen(): send_invalid_card_screen();
	4_digit_PIN_received is (get_4_digit_PIN)? Receive_4_digit_PIN() : FALSE;

Bank managers	*accessing_to_service is card_confirmed & customer_confirmed & account_confirmed;*
Bank database designers	account_confirmed is check_customer_account(customer_account_ID);

Table 7-2. Individual insights of different participants for an ATM system

On the basis of individual seed ISMs, participants interact with each other for cultivating requirements. For example, it is found that bank tellers, bank managers and customers have different perspectives on the ID problem (see the italicised entries in Table 7-2). For the sake of convenience, customers prefer to be identified by the account number on the inserted card together with a 4-digit personal identification number (PIN). For safety reasons, bank tellers instead suggest using a 6-char string as a PIN code for the verification, since many customers use their birthdays as PIN codes. However, bank managers have a broader insight into the identified problem and wish to take the status of the customer's account and the cost of building each ATM into consideration. In addition, since these insights of customers, bank tellers and bank managers impinge on the hardware and software systems of the ATM, the bank database, hardware and software designers and database managers are also involved in the interaction for cultivating requirements.

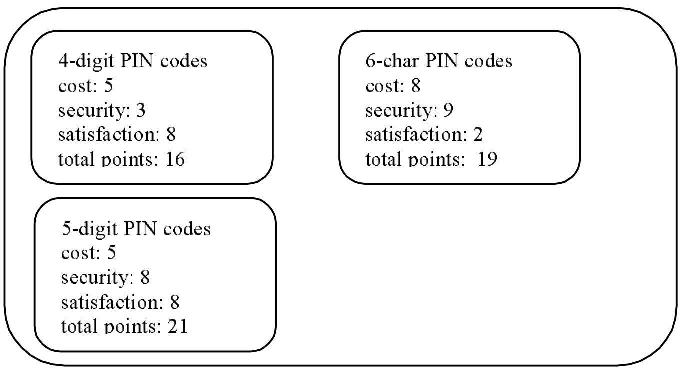


Figure 7-8. The ISM of a bank manager (snapshot)

At this stage, a new problem is identified: the data format of PIN codes ('the PIN problem'). To address this problem, the contexts and resources around participants are changed and mutually affect their ISMs in a situated manner. For example, for hardware designers, two different panels for inputting PIN codes have to be provided in order to support the conflicting ideas of bank tellers and customers. This provision can give insight into the cost of building each ATM, which is a main concern of the bank managers. Another concern of 'customer satisfaction' may also be introduced by bank managers to measure the feelings of customers about using both kinds of panels. The measures[4] of these concerns are shown in Figure 7-8. These situational changes highlight the fact that requirements are situated and depend greatly on the context and resources.

[4] The created ISM may involve the implementation of a decision support model, but the details are out of the scope of this thesis. It is assumed here that these measures can be accomplished and obtained by the bank manager through certain ways, when such measures are identified.

Now 'what if' experiments can be invoked. For example, customers can make use of their ISMs to explore different data formats of PIN codes, as when taking into account upper case and lower case characters in PIN codes.

If an agreed solution to the PIN problem cannot be obtained, it might either be left unsolved or managerial authority might be invoked to make a decision that suits the current context. In the former case, an unresolved conflict between participants occurs. Conflicts are not allowed in most traditional models, and are always viewed as errors that need to be corrected. In practice, a conflict need not always be regarded as an error. Conflicts disclose possible alternatives and are actually a very useful resource for making a decision. For the PIN problem, the conflict reveals individual concerns about PIN codes from different viewpoints. Customers focus on convenience of use, bank tellers pay attention to security control, but bank managers have an economic account of cost. In order to highlight its importance, the conflict is deliberately left unresolved here.

In a similar manner, it is clear that many problems can emerge and be either solved or unsolved. Details are beyond the scope of this thesis. It is supposed that some problems, such as the maximum number of permitted attempts to enter PIN codes, cancellation of PIN code input, validation and retention of a cash-card, and the diverse messages displayed to customers, have been identified and solved during the interaction between participants. It is not necessary for these problems and their respective solutions to be developed in a particular sequence; they can be addressed as they arise in particular contexts in a responsive manner. In this process, a provisional solution to the earliest problem – the ID problem – is obtained through collaborative interaction between participants.

At this point, with the ID problem provisionally solved, the PIN problem (still unresolved) may arise again, since a new context emerges: if the number of attempts to enter a PIN code exceeds the permitted number of attempts, the cash-card will be retained. In such a context, customers may strongly object to using 6-char PIN codes by awarding lower points of satisfaction. The change will immediately be propagated to the ISM of bank managers so that bank managers will understand the objection of customers to 6-char PIN codes. On the other hand, from the security perspective, a 4-digital PIN code is too simple to protect against fraud, especially given the evidence that customers are prone to use someone's birthday as a PIN code. Brainstorming activity supported by informal social interaction and 'what if' experiments thus commences. A new proposal that takes both viewpoints into account may be developed, for example, using a 5-digit number as a PIN code and providing customers with the facility to change their PIN codes on ATMs. This new proposal not only changes the provisional solution to the ID problem, but also has an impact upon the problem of 'available services' identified by banker tellers and customers in Table 7-1.

This special case of providing a solution to the ID problem illustrates the fact that changing requirements is the norm in developing requirements, even in such trivial a problem. Recognising this fact, SPORE deals with the rapid change of the identified problems, contexts and resources by the collaborative interaction between participants in a situated manner. Just as the knowledge of a human being grows in everyday life, requirements are grown by participants through interactive, iterative and creative activities on the basis of the principles and concepts of SPORE.

To explore the difference between SPORE and traditional process models, a viewpoint-based model called VORD is chosen. This is because some of the details of using VORD to formulate the requirements for the ATM are given by the authors of VORD and can be found in [KS98]. With these details, a fair comparison can be made, since the possibility of using VORD incorrectly can be eliminated. More importantly, VORD is a viewpoint-based model whereby requirements are principally developed in a distributed manner [FS96]. Within VORD, the information needed for developing requirements has been separated to different viewpoints from diverse perspectives. The general feature of supporting group work, which is given little or even no support in most models for the REP, is one of the main concerns in SPORE.

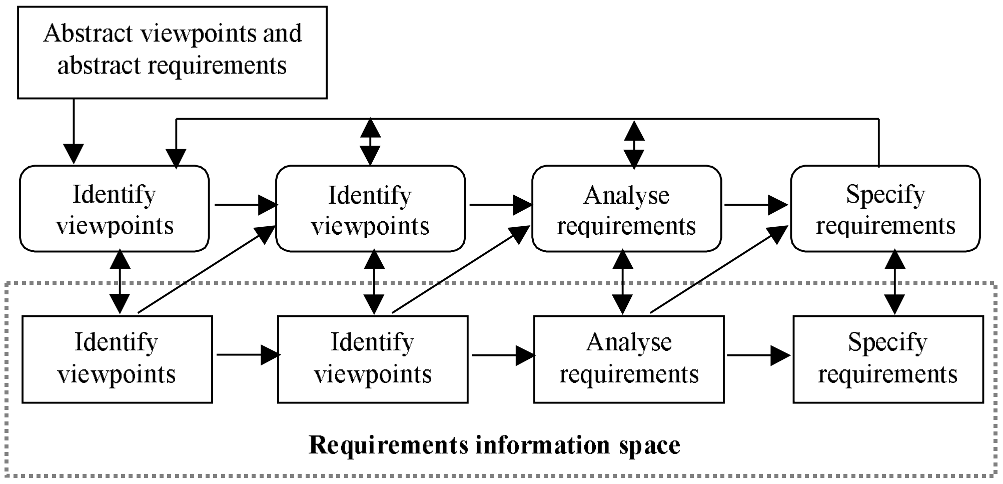


Figure 7-9 VORD process model. (from [KS98, p. 218])

VORD[5] is primarily intended for specifying an interactive system and is based on viewpoints from different perspectives. The following are its three main iterative steps:

1. viewpoint identification and structuring

2. viewpoint documentation

3. viewpoint requirements analysis and specification

Figure 7-9 shows the iterative process model of VORD. The processes are shown as round-edged boxes, and the products as square edged boxes. Each product can be viewed as the checkpoint for a review process. In the ATM example, VORD commences from the identification of abstract viewpoints by recognising what are called 'system authorities' from relevant perspectives. These abstract viewpoints can then be further

[5] The following paragraphs and some figures about VORD are mostly extracted from [KS98].

decomposed from the highest level into a number of less abstract modules in order to understand the structure and functionality of the whole software system that is to be developed. Information can be inherited by sub-class viewpoint, and so global requirements are represented in the more abstract classes and inherited by sub-classes. Figure 7-10 shows some of the ATM's viewpoints.

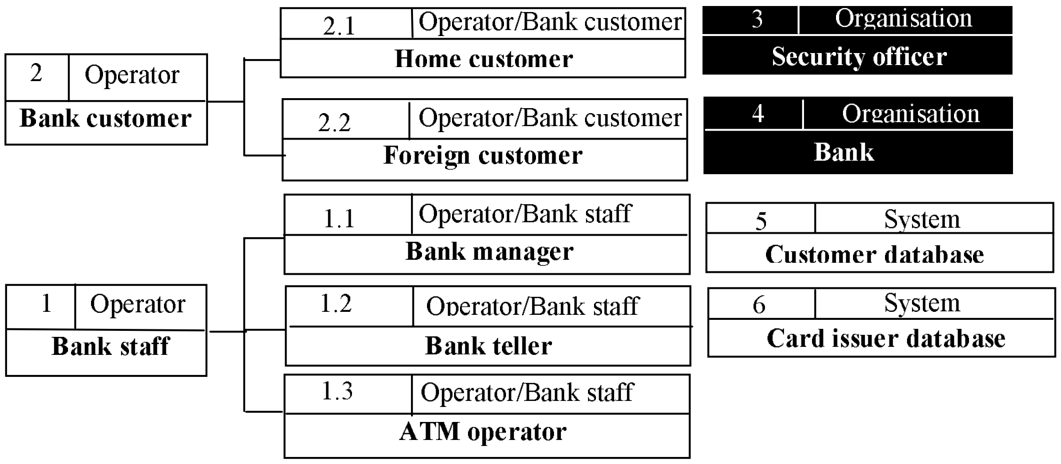


Figure 7-10. Viewpoints for an ATM system in VORD (from [KS98, p. 221])

The second step of VORD is to document the requirements of different viewpoints identified in the first step. Viewpoint requirements are made up of a set of functional, non-functional and control requirements. Control requirements describe the sequence of events involved in the interchange of information between a viewpoint and the intended system. These viewpoint requirements are documented in natural language or graphical notations. For example, Table 7-3 describes the initial requirements from the customer viewpoint. Also, Figure 7-11 illustrates an event scenario for service access. The method of using different notations to represent the same requirement in VORD is for the purpose of enhancing communication and aiding understanding between different participants.

The third step is concerned with validation by identifying errors and conflicts and resolving them. The end result is a requirements specification document.

Viewpoint		Requirements			
Identifier	Label	Description		Type	Source VP
1	Bank staff	1.1	Provide access to administrative service based on valid staff PIN and the access permission set out for the bank staff	sv	4
1.1	Bank manager	1.1.1	Provide transaction reports to bank manager	sv	1.1
		1.1.2	The bank manager requires transaction reports to be provided on a daily basis	nf	1.1
2	Bank customer	2.1	Provide access to ATM services based on valid cash-card, valid PIN and access permission set out for the bank customer	sv	4
		2.2	Provide for withdrawal of cash by bank customers	sv	4

Table 7-3. Initial requirements from some participants in VORD

VORD is a hybrid of the product- and process-oriented models discussed in the first section. It aims to specify requirements from multiple viewpoints in a distributed manner. The principle advantages offered by viewpoints [SS97] are:

- to extract more complete requirements,
- to avoid dealing with conflicts between viewpoints before they are well-informed,
- to enhance traceability.

However, VORD takes no account of context and does little to help cope with the relevant issues of changing requirements. For example, if a change (entering the cancel key) is added to Figure 7-11 as shown in Figure 7-12, other documents associated with this requirement become inconsistent with the changed documents. In addition, since VORD ends up with specification documents rather than a working model, some problems associated with system design and operation cannot be easily disclosed (a typical consequence of conventional wisdom concerning *how* and *what* in RE mentioned earlier). For example, the PIN problem discussed above could be left to designers or be neglected in VORD. When the problem is identified in the latter stage of SSD, the steps described in VORD will need to be revisited. Moreover, the understandability of these specification documents to all participants could be another problem for validation. A brief comparison of SPORE and VORD in developing requirements for the ATM system is given in Table 7-4.

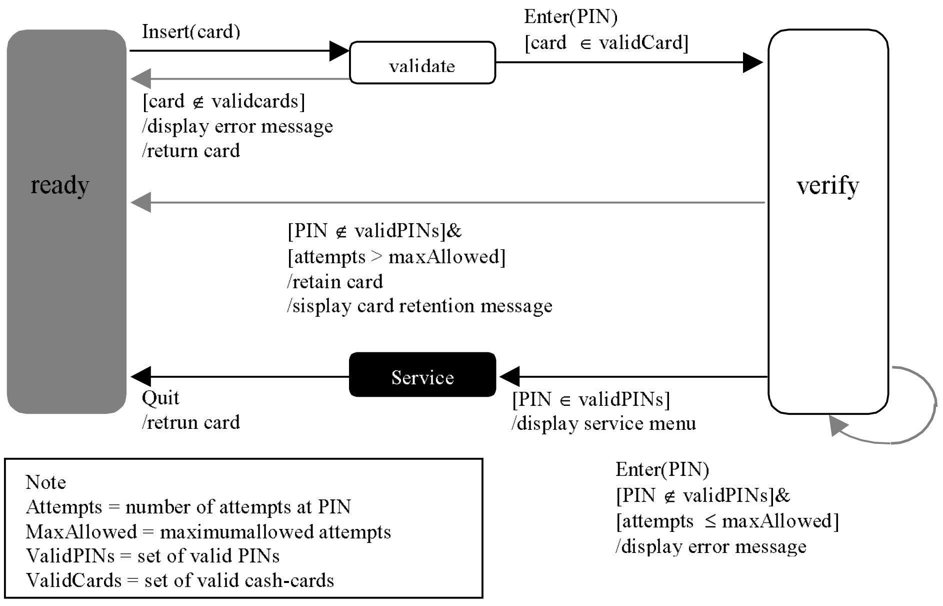


Figure 7-11. Event scenario for service access (quoted from [KS98, p.233])

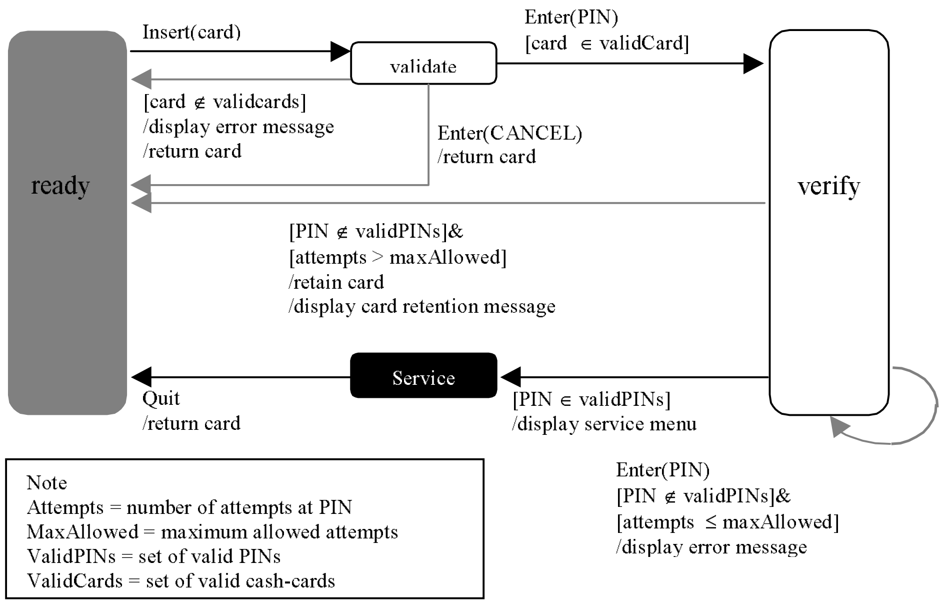


Figure 7-12. A modified event scenario for service access

	SPORE	VORD
Fundamental principle for the REP	Situatedness	Step-by-step algorithm
Main aim	Requirements development	Requirements definition
Final target	Working models	Documented specifications
Main information sources	Collaborative interaction, 'what if' experiment, and domain knowledge	Domain knowledge
Orientation	Problem-focused	Solution-driven
Participants' relationship	Collaborative	Subordinative/ Coordinative
Participation	Users and developers	Users and developers
Group work	Supported (human-centred)	Semi-supported (developers-centred)
Work style	Interactive, open-ended	Non-interactive
The relationship between the REP and design	Design has been embedded into the construction of working model	Support the transition to object-oriented design manually
The relationship between the REP and SSD	Throughout the whole life cycle of SSD	Only in the early stage
Context in the REP	Contextual dependence	Contextual independence

Table 7-4. A comparison between SPORE and VORD

7.4.2 A Warehouse Distribution System[6]

Specifying the requirements for a warehouse is taken as a case-study by Jacobson in [JCJO92]. Jacobson's concern is to identify the software requirements of a computerised system, and his approach is based on use-case analysis. For Jacobson, each use-case is associated with a particular kind of interaction between human agents and the computer system, such as might be directed towards one of the required functions of the warehouse (e.g. manual redistribution between warehouses).

[6] This case study is still proceeding. Most practical work described here has been conducted by another Ph.D. student, Y-C Chen. This subsection is closely based on Beynon's account of joint research reported in our paper: *Cultivating requirements in a situated process of requirements engineering* [SCRB99].

Within the framework of SPORE, the requirements engineering task can be seen in the broader context of developing a business process model and determining the role that computer technology can play in carrying out the characteristic transactions of the warehouse. The perspective proposed here is through-and-through agent-oriented in the sense that warehouse activity is conceived with reference to state-changing protocols for human and automated components with the system. In effect, where the action of human agents is constrained by the business process so that it follows reliable patterns, it is possible to regard their co-operative activity as a form of computation. The characteristic transactions of the warehouse are then analogous to use-cases in Jacobson's sense.

- Seed ISMs for the Warehouse State

In SPORE, the cultivation of requirements has to start from a representation of those elements of the warehouse state that are pertinent to the particular problem being addressed. This representation will take the form of a *seed* ISM that – because of the situated nature of SPORE – incorporates matter-of-fact observations of the current state of the warehouse. Typical observables that are significant in this view are the items and locations in the warehouse, and the inventory that connects items with locations. An ISM to represent these observables will supply a visual representation for items and locations, and the status of the inventory

Such a representation of the current state of the warehouse will be complemented by informal actions, for example: represent the relocation of items, look up an item in the inventory, or take receipt of a new item for storage. In some contexts, this will motivate visualisations to represent intermediate states in the operation of the warehouse that are associated with items in transit, or items located via the inventory but yet to be retrieved from the warehouse.

A model of the warehouse has to incorporate such aspects of state and state change in order to be faithful to its referent. If such aspects are neglected, there is no means to consider behaviours that, though undesirable or outside the scope of normal operation, have a profound influence on the requirement. For instance, the requirements activity has to address matters such as the loss of items or warehouse locations, the concept of items being mislaid, or the significance of perishable items.

There is no single ISM that can represent all the aspects of the warehouse state that are potentially relevant to a requirements identification. The state of the warehouse will typically be represented by different seed ISMs according to what problems are being addressed in the SPORE, and each will be introduced to mimic particular scenarios. For instance, it may be appropriate to construct seed ISMs to represent different varieties of perishable item, or to represent a very large number of items to assess the interface to an inventory database.

- The Warehouse Business Process Model (BPM)

Over and above the naive perception of states and state changes just considered, there is a business perspective on warehouse operation. This focuses on the particular agents that are intended to operate and the protocols that they follow in carrying out preconceived characteristic transactions. These define the business process model.

The observables in the BPM are different in character from items and locations. They relate to phases in preconceived transactions. The state changes are concerned with the systematic execution of protocols and the associated transition from one phase to the next. There may be no counterpart in the BPM for activities that might be possible in practice, such as the illicit retrieval of an item by its owner. An important aspect of the observables associated with the BPM is that they should not only serve to determine the current state, but must also incorporate a transaction history appropriate for auditing.

The ISM which is to be developed to represent the BPM is modelled on the practices that were used in the operation of the warehouse prior to the advent of computers. In that case, forms and paper inventories serve to record the operation of the BPM by rendering the abstract observables associated with phases and roles visible and tangible. Manual data entry, following systematic processes of form transfer, was the means to represent both the current status of all transactions (such as: which items were in transit) and the history of transactions.

To some extent, the forms and inventories can be interpreted as a paper-based ISM for the business process. In performing a particular transaction, specified procedures are to be followed in filling forms and transferring them between personnel. These manual activities effectively identify which agents have roles in the transaction, which are currently active in any phase, and how their interaction is synchronised (cf. Figures 7-13 and 7-14). The current status of any transaction is determined by what sections of forms are currently completed and who currently holds the forms.

The full details of how the BPM is construed to operate are reflected in the specific details of what each agent enters on a form. These details refer to the observational and interactive context for each agent: the observables the agent can refer to (its **oracles**), those that can conditionally change (its **handles**) and the protocol that connects these. Note that the relevant observables in this context may refer to the state of the warehouse itself (e.g. an item can be signed off only if it is presently to hand), and relate to the high-level context for interpretation (e.g. issues of legality, safety, etc.). The persistence of the record that the forms supply is also significant for auditing and traceability.

- Applications of SPORE to Warehouse Requirements

Just as paper records and protocols for interaction with them can be viewed as an ISM, so the process by which such procedures evolved can be construed as EM. The activities involved in this evolution are as described in the above discussion:

- the identification of agents: e.g. foreman, warehouse worker, driver, office clerk;
- the conception of the roles for these agents corresponding to their characteristic skills;
- the apportioning of responsibilities for particular phases within a given transaction;
- the refinement and formalisation of their precise observables and protocols.

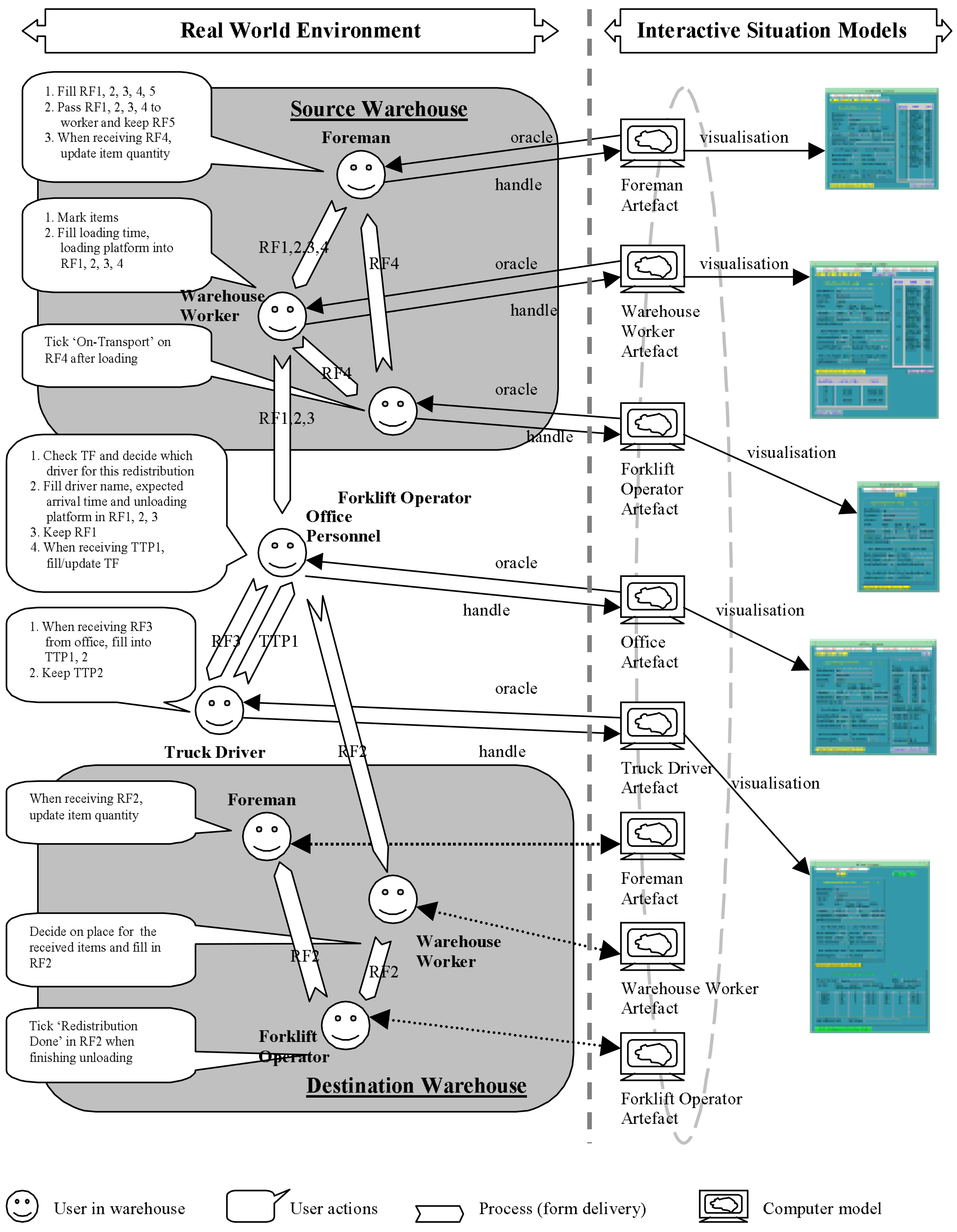
In applying SPORE to developing warehouse requirements, this general process is emulated using computer-based technology. The ISM constructed for this purpose incorporates the seed ISMs for the warehouse; the form-based abstractions that capture the state of the BPM and the activities of the agents; and additional observations such as those associated with the wider significance of the warehouse operation (e.g. those concerned with the legality and the integrity of the business process). The transformation from a paper-based to a computer-based ISM illustrates the potential of SPORE as a framework for business-process re-engineering.

The distributed nature of dtkeden makes it possible to separate the viewpoints of the agents in the model, and to complement these with an external interpretation. In the first instance, computer-based forms are used to represent the environment for each agent's interaction. The mechanisms through which a particular kind of agent, such as a warehouse worker, interacts can be subsequently elaborated through the development of special-purpose interfaces. In this way, the distributed ISM serves as a medium in which to identify and enact appropriate transactions, and to debug and refine these through collaborative interaction between the various participants.

Examples of how requirements can be addressed by SPORE in this way include:

- Through experimentation at different workstations, it is possible to identify issues that are problematic from the perspective of particular agents: for instance, "how does the office know which drivers are available?" "How does the office determine whether a transaction is completed?"
- Through the elaboration of different seed ISMs, additional issues can be addressed, such as transportation costs, perishable goods, security and trust concerns.
- Through the modification of dependencies and communication strategies, the effects of different technologies, such as mobile communications, the Internet, optical bar code readers, or electronic locking agents, are considered,
- Through collaboration and synthesis of views, it is significant to distinguish between subjective and objective perceptions of a state e.g. to contrast "I remember doing X" with "I have some record of doing X" with "There is an official record of X", or to model misconceptions on the part of an agent.

Through intervention in the role of superagent, it is possible to examine the consequences of singular conditions that arise from opportunistic interaction or Acts-of-God, and to assess activities outside the scope of normal operation such as are associated with fraud, or manual back-up to automated procedures.



Real World Environment
Interactive Situation Models
Source Warehouse
Foreman
1. Fill RF1, 2, 3, 4, 5
2. Pass RF1, 2, 3, 4 to worker and keep RF5
3. When receiving RF4, update item quantity
1. Mark items
2. Fill loading time, loading platform into RF1, 2, 3, 4
Warehouse Worker
Tick 'On-Transport' on RF4 after loading
RF1,2,3,4
RF4
RF4
RF1,2,3
oracle
handle
visualisation
Foreman Artefact
Warehouse Worker Artefact
Forklift Operator Artefact
Forklift Operator
Office Personnel
1. Check TF and decide which driver for this redistribution
2. Fill driver name, expected arrival time and unloading platform in RF1, 2, 3
3. Keep RF1
4. When receiving TTP1, fill/update TF
Office Artefact
1. When receiving RF3 from office, fill into TTP1, 2
2. Keep TTP2
RF3
TTP1
Truck Driver
RF2
Truck Driver Artefact
When receiving RF2, update item quantity
Foreman
Decide on place for the received items and fill in RF2
Warehouse Worker
Tick 'Redistribution Done' in RF2 when finishing unloading
Forklift Operator
Destination Warehouse
Foreman Artefact
Warehouse Worker Artefact
Forklift Operator Artefact
User in warehouse
User actions
Process (form delivery)
Computer model

Figure 7-13: A collaborative working environment for manual redistribution between warehouses

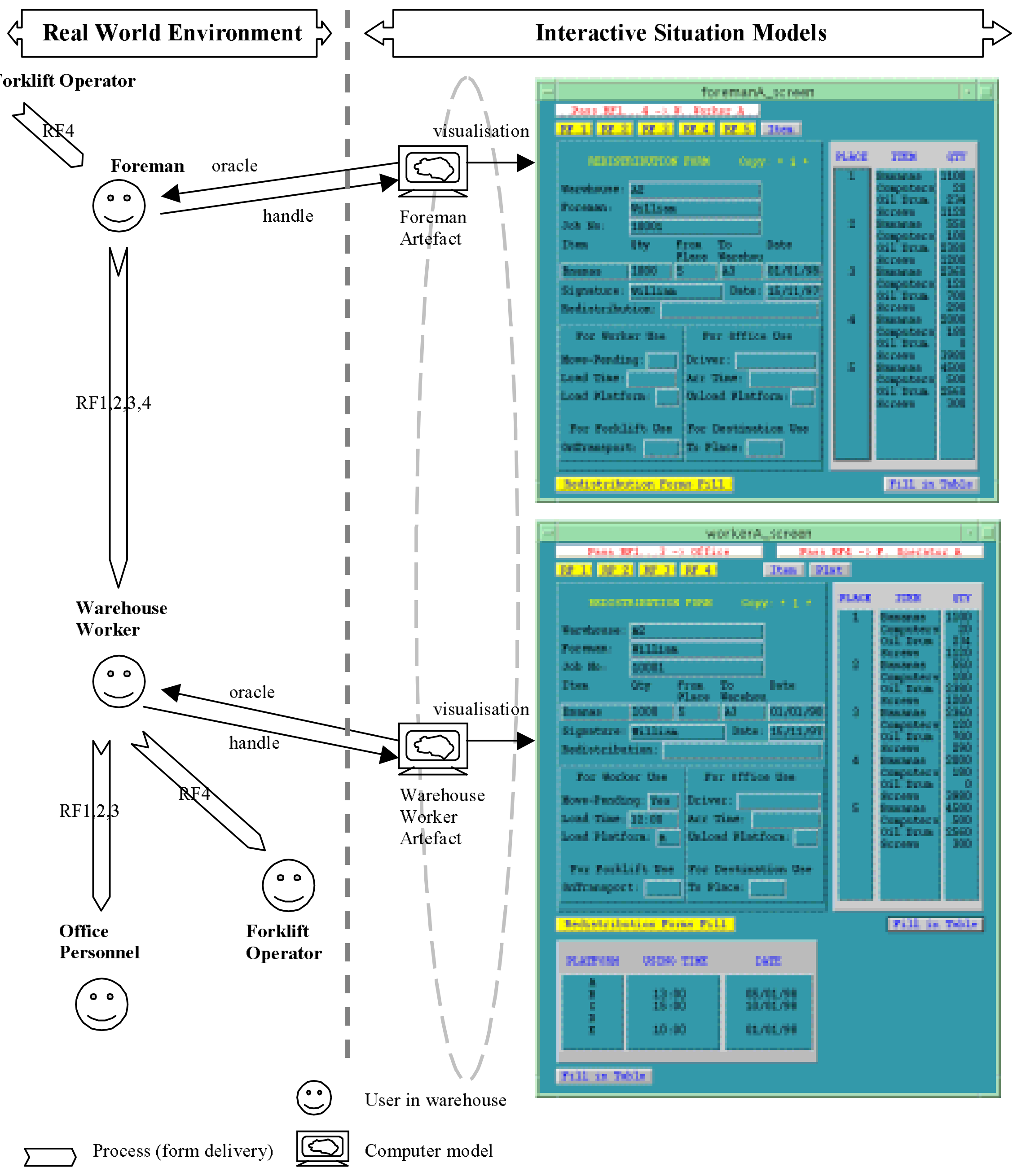
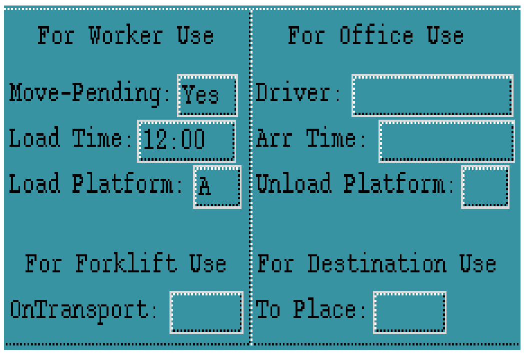


Figure 7-14a (above): Detailed view of the forms used in the warehouse artefacts

Figure 7-14b (right): Detail of panels representing observables (**handles** or **oracles**) for some warehouse agents



For Worker Use		For Office Use	
Move-Pending:	Yes	Driver:	
Load Time:	12:00	Arr Time:	
Load Platform:	A	Unload Platform:	
For Forklift Use		**For Destination Use**	
OnTransport:		To Place:	