The CSCW Paradigm for Software Development

by

ZELDA VILJOEN

submitted in part fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

in the subject

INFORMATION SYSTEMS

at the

UNIVERSITY OF SOUTH AFRICA

Supervisor: PROFESSOR A.L. STEENKAMP

JUNE 1995

INDEX

ABSTRACT

People work together to solve a wide variety of problems using different forms of cooperation for each class of problem. Modern technology is complex, and therefore it is unusual for an individual to attempt the development of a major project single-handedly. In an attempt to provide computer-based support for the problems that arise when two or more people attempt to cooperate to perform a task or solve a problem, the area of Computer Supported Cooperative Work (CSCW) becomes relevant. The software development process almost invariably involves cooperation that crosses group, professional, and subcultural boundaries. The complexity of software development demands that highly integrated groups of analysts, designers, and users are involved in the process. Many development activities may occur concurrently. The area of CSCW and advanced information technology, with its enormous capabilities for transmitting and storing information, holds considerable promise for the software development process.

KEYWORDS

Software Development, Software Engineering, Software Development Life Cycle, Cooperation, Computer-Based Support, Computer Supported Cooperative Work (CSCW), Groupware, Object-Orientation.

ACKNOWLEDGEMENTS

To my Lord and Saviour Jesus Christ - Thank You for the strength and hope given to me to complete this work.

My sincere gratitude to my husband, Merwe, for his indulgence as sacrifices were made for the sake of my studies - I love you for that!

A special word of thanks to my parents for their loyalty and support throughout the years. Thank you Mom for being my best friend.

Thank you to Professor Bornman, Head of the Department of Computer Science and Information Systems, University of South Africa, for approving my studies and sabbatical leave.

Finally, a sincere thanks to Professor Lerine Steenkamp. I gratefully acknowledge her invaluable advice, support and guidance throughout this project.

TABLE OF CONTENTS

PREFACE

This dissertation has been done in partial fulfilment of the MSc degree in Information Systems at the University of South Africa. The other half of the degree requirements were the completion of five study modules, which were:

- *Software Design:* The module covered advanced software design concepts, design methodologies, real-time software design, and object-oriented software design.

- *Expert Systems:* The module covered the concepts, characteristics and classification of expert systems. The architectures and methods of different types of expert systems were discussed. Most importantly, the manner in which expert systems are constructed was illustrated.

- *Software Engineering Environments:* The principles of software engineering formed the underlying theme of the module. The concepts, requirements, and characteristics of environments supporting software development according to software engineering principles were covered.

- *The Object-oriented Approach - A Comparative Study:* A special topic module which covered the underlying concepts of the object-oriented approach. Frameworks were established for the comparison of object-oriented approaches, and for the comparison of traditional (structured) software engineering techniques with object-oriented techniques.

- *The RASCAL Approach - A Middle Way To Software Engineering:* A special topic module (project) which described the *Rascal Approach*, which uses the strengths of both the object-oriented and structured approaches - the best of both worlds! A case study was conducted following the *Rascal*

Approach and the comparative framework of the previous special topic module was used to evaluate the approach.

The research presented in the dissertation is concerned with the area of software development, with emphasis on computer-based support for the development process. Much of today's work is not done individually, but rather in groups. Software development is also collaborative in nature, due to the complexity of the task, the severe time constraints, and the requirement for broad expertise. The idea of computer support for group-based work activities, such as software development, is a useful and challenging one, for it represents a break from approaches that focussed on centralised and bureaucratic systems of communication and control.

Computer Supported Cooperative Work (CSCW) has emerged as an identifiable research field which focusses on the role of the computer in group work. The study explores the dynamic and distributed nature of cooperative work and determines the requirements of CSCW technology in general, and in particular for support of the software development process.

Thus, the purpose of the research was to explore how CSCW could be used in support of the software development process. This is in accordance with the *Object-oriented Information Systems Engineering Environment* (referred to as OISEE) project undertaken by the Department of Computer Science and Information Systems at UNISA, which aims at establishing a foundation for an information systems engineering environment within which other research projects may be undertaken at postgraduate level. For the purpose of this research, the spiral life cycle model for object-oriented development, as specified by Du Plessis and Van Der Walt (1992), is adopted. The fundamental paradigm is *object orientation* which was found to be especially suitable for the simultaneous independent work nature of CSCW.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
CASE	Computer-aided Software Engineering
CIM	Computer-integrated Manufacturing
CSCW	Computer Supported Cooperative Work
DBMS	Database Management System
DSS	Decision Support Systems
EIS	Executive Information Systems
GDSS	Group Decision Support Systems
HCI	Human Computer Interface
HIPO	Hierarchical Input, Process, and Output
JIT	Just-in-time
KBMS	Knowledge Base Management System
LAN	Local Area Network
LCM	Life Cycle Model
MAN	Metropolitan Area Networks
MIS	Management Information Systems
NGT	Nominal Group Technique
OODBMS	Object-oriented Database Management System
ODP	Open Distributed Processing
OISEE	Object-oriented Information Systems Engineering Environment
OOA	Object-oriented Analysis
OOD	Object-oriented Design
OSI	Open Systems Interconnection
PC	Personal Computer
RVL	Relational View Language (RVL)
SADT	Software Analysis and Design Technique
SDLC	Software Development Life Cycle
SE	Software Engineering
SEE	Software Engineering Environment
SPMP	Software Project Management Plan
WAN	Wide Area Network
WYSIWYG	What You See Is What You Get

DEFINITIONS OF TERMS

CASE - The automation of the software engineering process by means of tools which can be applied to the full or partial software development life cycle.

CSCW - Work by multiple active subjects sharing a common object and supported by information technology. In more simple terms, CSCW is the attempt to provide computer-based solutions to the problems that arise when people cooperate in groups to perform a task.

Groupware - The electronic (computer-based) components supporting participants engaged in group activity.

Groupware System - A computer-based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment.

Life Cycle Models - The management of large software projects by dividing the development into cycles or phases, each with a prescribed work breakdown structure, deliverables, review points and management procedures for planning, monitoring and controlling a project.

Method - A definite, orderly way of doing something, for the purpose of this investigation applied to the specific approach of carrying out one or more of the software development activities; it is usually based on an intellectual model of how to accomplish an activity; it is implemented through the use of procedures and tools.

Methodology - A collection of methods, techniques, procedures and tools that provides the overall approach to developing and improving software; it is usually based on an underlying intellectual model (the paradigm).

Paradigm - An abstract (conceptual) or intellectual model on which something is based.

Processes - Any software related activities, normally having a time factor.

Products - Any artefacts, documents or deliverables which result from processes.

Software Development Life Cycle - A life cycle model to support the breakdown of work during development, implementation and maintenance of

a software system.

Software Engineering - A discipline that proposes that software development should be followed according to sound engineering principles which entails the production of quality software systems efficiently.

Software Engineering Environments - An environment consisting of integrated CASE tools, utilities, procedures and storage facilities to support a particular development methodology which covers the entire software development process.

Technique - An informal method.

Tool - A mechanism by which a method can be executed;

CHAPTER 1

Statement of the Area of Investigation

CONTENTS

1.1 Introduction

In recent years, literature about *work* has increasingly focussed on the importance of collective communication, tacit knowledge, and group activities (Greenbaum, 1988). The idea of establishing computer support for group-based work activities, which may be called *cooperative work*, is a useful and challenging one, for it breaks away from centralised and bureaucratic systems of communication and control. The shift from computer systems that support a single user working alone to those supporting a group of users working together has considerable impact. It leads to the consideration of the ways people work together in everyday life, and possible ways to support and extend their interactions. Perhaps more importantly, it suggests that the unique capabilities of computers should be

embedded more firmly in ordinary work practices (Gaver, 1991).

In recent years, increasing attention has been focused on the problem of *Computer Supported Cooperative Work (CSCW)*. It has gained acceptance both within the research community and in the popular press as denoting a new perpective on computer system development and use. CSCW, broadly defined, is the attempt to provide computer-based solutions to the problems that arise when people (more than a single individual) attempt to cooperate to perform a task or solve a problem (Borenstein, 1992). *CSCW* is a subject that draws on research in various disciplines, such as psychology, sociology, computer science, and artificial intelligence. Group work may be considered of special interest for all these areas, but the provision of computer-based support for groups of people working together is central to CSCW.

Software development involves substantial integrated and cooperative elements. This has particular implications for the way the software system should be developed. The software development methodology should reflect these elements of cooperation (across functional areas or within the organisational hierarchy) and demands some form of *participatory* approach (Green, 1991). This leads to the belief that CSCW should be incorporated in the software development process.

1.2 The Problem and its Relevance

Software development according to sound *software engineering (SE)* principles aims at producing quality software systems efficiently. More specifically, the software development process involves a set of activities aiming at conceptualising, specifying, and producing software systems in accordance with the requirements of the users under existing economical and technological constraints.

Many of the activities of the software development process could be performed either individually or in collaboration. As modern technology is complex, it is unusual for an individual to attempt the development of a major software project single-handedly. Integrated groups of analysts, designers and users work together to establish a clear understanding of the software system that should be developed. These software teams have the challenge of solving the difficult problems of task coordination and information integration.

Software development almost invariably involves cooperation that crosses group, professional, and subcultural boundaries (Bannon, 1991). Different groups, professions, and subcultures embody different perspectives - they communicate in different "jargon". The difficulties of working in such situations where individuals in the group have different practices, traditions, and working objectives may lead to a lot of time wasted in clearing up differences between the parties involved. Different groups reflect different ways of doing things (a different ontology[1] and epistemology[2]). These distinct groups will be referred to as semantic communities (Bannon, 1991). Efforts within the systems development process to emphasize the importance of good communicative practices between the differing semantic communities attest to the difficulties that are experienced.

Many problems are experienced in the early phases of the software development process, particularly in the analysis cycle. One of the challenges that software developers face in this cycle is the overall analysis (and subsequent detailed specification) of complex and ill-defined opportunities, opportunities surrounding the coupling of user needs with system functionality. The opportunities span a spectrum that ranges from easy to define (and specify) to continually evolving and almost impossible to specify. There are often extensive communication and

1 *Ontology* is a branch of metaphysics dealing with the nature of being (Oxford Dictionary, 1982).

2 *Epistemology* is the theory of the method or grounds of knowledge (Oxford Dictionary, 1982).

cooperation barriers between the analysts and the users. This could lead to software being developed without fully understanding (or being able to build on) what users already know about their tasks. Software developers are often forced to assess situations, determine user requirements, outline system functionality, and develop software within very short time frames and at low budgets, without compromising system functionality or putting users into compromising positions.

Groupwork during the software development process introduces problems of organisation, coordination and communication. The tasks carried out by the groups can rapidly become overwhelming because of the complexity of the interactions and the amount of information that is generated. Due to the interdependence in conducting their work, cooperating workers have to articulate (divide, allocate, coordinate, schedule, mesh, interrelate, etcetera) their respective activities. Entering into cooperative work relations, the software developers must engage in activities that are in a sense extraneous to the activities that contribute directly to fashioning the software product and meeting the requirements. A justification for incurring this overhead cost and the reason for the emergence of cooperative work formations in software development is that workers could not accomplish the task in question if they were to do it individually, at least not as well, as fast, as timely, as safely, as reliably, and as efficiently as a group could (Schmidt, 1991).

Nowadays, truly distributed applications seem natural in the face of a high ratio of computing power to available communication bandwidth (Borenstein, 1992). The users and software developers involved in a specific software development project may be widely separated geographically. For such cooperative development efforts, distributed solutions seem inevitable. Unfortunately, the practical success of such systems has been extremely limited. Three problems that are not generally faced by single-user development environments should be solved (Borenstein, 1992). Firstly, there is the problem of the remote installation, where the participants of the cooperative development effort may have differing

degrees of motivation for allowing the installation of the cooperative development software. Related to the problem of getting software installed at distributed sites is the problem of getting the participants in the software development process to use the distributed development software once it is installed. Getting software to work on a wide variety of platforms is a major task, and one that has to be carefully considered before establishing a fully distributed development environment. Complex organisations are characterised by distributed decision making, and require a sharing of perspectives among participants if they are to coordinate activities and adapt to changing circumstances.

The problems have resulted in the realisation of a need to unleash all the resources of cooperative work: horizontal coordination, local control, mutual adjustment, critique and debate, and self-organisation (Schmidt, 1991). Tools and techniques for cooperation between end users, management and professional designers should be applied in the software development process.

Advanced information technology , with its enormous capabilities for transmitting and storing information would seem to hold considerable promise for groupwork (Gorry, 1988). This is where CSCW has a role to play. CSCW is the attempt to provide computer-based solutions to the problems that arise when people cooperate in groups to perform a task. This also has relevance for the software development process.

1.3 Current Status of the Area of Investigation

A few main topics form part of the area of investigation. The research is concerned with the area of software development, with emphasis on computer-based support for the cooperative elements of the development process. The dynamic and distributed nature of cooperative work, and the requirements of CSCW technology in general, and in particular for support of the software development process is discussed.

This section concentrates on key issues relevant to the area of investigation, namely: the software development process, and CSCW.

1.3.1 The Software Development Process

The complexity of the software development process has resulted in a lot of research efforts in this area over the years. In the 70's, a lot of concern was expressed over the so-called *software crisis*. It was found that over 50% of software systems developed had not met the requirements of users, and exceeded the time and budget limits originally set. Some software projects even failed to deliver any sort of software product.

Software engineering (SE), first identified as a discipline in 1968, proposed that software development should be followed according to sound engineering principles which entailed the production of *quality* software systems *efficiently*. In the seventies, the advent of the *structured* approaches to programming, analysis and design has improved the software development process slightly. Dijkstra (Dahl et al., 1972), Boehm (1975), and Parnas (1975) have made important contributions to the acceptance of structured programming. The work of De Marco (1978), Gane and Sarson (1979), and Yourdon and Constantine (1979), has resulted in structured analysis approaches that strengthened the interface to design. The structured design methodologies may be classified according to their primary focus (Couger et al., 1982):

1. *Hierarchical Focus*
    (a) Functional Decomposition
    (b) Structured Analysis and Design Technique (SADT)
    (c) Hierarchical Input, Process, and Output (HIPO)

2. *Data Flow Focus*
    (a) The Yourdon and Constantine, and Myers Approaches

3. *Data Structure Focus*
   (a) The Warnier-Orr Approach
   (b) The Jackson Methodology .

Many of these structured design approaches consider additional structural issues on a secondary level. In addition to their methodological development, the work of Yourdon and Constantine (1979), and Myers (1978), and others is significant to the area of evaluating the "goodness" of a design.

The adoption of *life cycle models* facilitated the management of large software projects by dividing the development into phases, each with a prescribed work breakdown structure, deliverables, review points and management procedures for planning, monitoring and controlling a project (Du Plessis & Van der Walt, 1992). Various *software development methodologies* have been proposed incorporating methods and techniques in support of the development and management tasks of each of the life cycle phases. Emergent technologies, such as the microcomputer with enhanced graphic capabilities and the mouse, has created opportunities for tools to support the software development process. Firstly, *computer-aided software engineering (CASE)* tools, and later *software engineering environments (SEE)* that supported all phases of software development appeared in the market place.

Although, all these developments have greatly improved the software development process, modern technologies and the greater complexities of software systems have encouraged the need for more support of the software development process. During the last decade an increasing number of researchers have been looking at the role of *end users* in the development process in trying to find ways of improvement (Kyng, 1988). The end users are not professional designers, and, therefore, the standard set of tools and techniques for systems development does not allow them to play an active and creative role. Modern technologies, such as distributed systems and network facilities have given advent

to the possibilities of distributed software development. Related to the aspects of user involvement and distributed software development, is the aspect of group activity which forms a major part of the software development process. Computer-based tools in support of this have not, as yet, been integrated into the development process.

1.3.2 Computer Supported Cooperative Work

CSCW has now acquired considerable momentum. It has engaged the interest of researchers in both academic and commercial environments, gained the attentions of commercial think-tanks, systems houses and service providers, and raised spirits among potential users, or "victims". In this section, a brief overview of groupware tools for computer supported cooperative work will be given. In Chapter 2, specific groupware tools for use in CSCW will be discussed in greater detail.

Much of the research in cooperative work develops or examines software that is explicitly designed to support information sharing, such as electronic mail (Malone et al., 1987; Borenstein and Thyberg, 1988), group-oriented decision tools (Stefik et al., 1987), and project management tools (Sathi et al., 1986). Recently, there has been much research done in computer applications for facilitating collaboration among multiple distributed users. Multi-user applications support multiple users performing a related task in a distributed context (Graham & Urnes, 1992). Examples of multi-user applications include groupware systems which provide computer support to group activities.

Computers are now commonplace in work environments and are influencing the way people interact. Examples of computer-supported interaction mechanisms include electronic mail, newsgroups, and distributed file systems. All support non-interactive styles of communication. So-called "talk" problems are more interactive but are generally restricted to two users and allow very simple forms

of interaction, such as the exchange of messages in different windows. There has been a growing interest in providing collaboration tools to support more closely coupled interactions among a group of people, such as where more than one person are involved in interactive sessions (e.g. group editing) on computers that are connected through a network. A group editor allows several people to jointly edit a shared document in a distributed environment. Researchers have identified the potential for major improvements in organisational productivity made possible through the use of computers to link people into task-oriented teams (Bullen & Bennett, 1990).

A number of group software products, usually labelled "groupware" have appeared in the market place. Groupware tools exist for sending messages (electronic mail), indicating moods, editing collaboratively, giving slide shows, and monitoring the group. Some of the problems of the groupwork tools for computer supported cooperative work have been addressed. Many new groupware products have appeared in recent years of which only the important types are mentioned. Specific groupware products will be discussed in more detail in Chapter 2.

New technologies include multi-user drawing tools to enable informal communication between people who are geographically dispersed, similar to the use of a whiteboard (Brinck & Gomez, 1992). Much work has also been done in the development of multi-user applications to be used along with video communications systems. Computer support has been developed to enable cooperative work in three-dimensional computer-aided design (Shu, 1992).

The merging of computer and communications technology and the development of network-based windowing systems has seen the development of real-time cooperative systems. These systems provide multi-user interfaces to enable application sharing across a number of workstations. The approach followed by the majority of these systems has been to extend existing windowing technology

to replicate output on a number of displays (Bentley et al., 1992). It has been successfully applied within shared window or desktop conferencing systems. Recent real-time cooperative systems have considered the development of architectures supporting multi-user applications with more control over the user interface. Awareness and coordination facilities of shared workspaces allow users to move smoothly between close and loose collaboration, and to assign and coordinate work dynamically. Groupware environments exist which allow unsophisticated computer users to create their own cooperative work applications using a set of simple, but powerful, building blocks (Lai & Malone, 1988).

There have been efforts to design systems to facilitate the capture of early software development deliberations. Examples of other CSCW systems to support software development are:

- Office systems which include electronic mail, calendar, and scheduling functions (for project management).
- Multimedia personal communication tools for communication in the early phases of the software development process (Root, 1988).
- Drawing surfaces which allow analysts or designers to share screen images, and use drawing surfaces (Bly, 1988).

Other examples of attempts to incorporate CSCW in software development exist, but that will form a discussion point in Chapters 3 and 4 of the dissertation.

1.4 Proposed Solution

The investigation presented in the dissertation proposes that CSCW should be incorporated in the software development process to provide computer-based solutions to the problems that arise when people cooperate in groups during the software development process.

A few essential issues guided the research. The most important issue is that software development involves substantial integrated and cooperative elements. The end user should participate closely in the software development process, especially in the analysis cycle, to ensure that the end product meets the original requirements set. Tools and techniques should exist for them to play an active and creative role in the software development process.

The users and software developers involved in a specific software development project may be widely separated geographically. This has particular implications for the way the software system should be developed. The software development methodology should reflect these elements of cooperation (across functional areas or within the organisational hierarchy) and demands some form of *participatory* approach (Green et al., 1991). This leads to the belief that CSCW should be incorporated in the software development process, especially in the analysis cycle where cooperation is a vital factor.

1.4.1 Scope of the Investigation

The research is concerned with both the process of cooperation and with computer systems to support that process. The focus is on the shared intellectual activity as required in the software development process, in which groups of users and software professionals work together to build a large, complex structure of ideas which should be transformed to form the software system. It is assumed that these groups are geographically distributed, interacting with one-another using communications networks.

CSCW concepts which are concerned with support of aspects in the software development process will be addressed in the dissertation. The aim is to provide a team of analysts with a medium in which all aspects of their work may be computer mediated and supported. This includes the traditional documents such as requirements and specifications, and other important aspects, such as

interviews with users, scenarios, reviews, early notes and sketches, constraints, and development meetings. Groupware tools should also exist to support quality assurance (validation and verification), and project management (project scheduling, project monitoring, management meetings).

The scope of the study is provided with clear boundaries when considering the hypothesis, the assumptions, and the constraints of the study.

(i) ***Hypotheses***

The investigation is based on the hypothesis that it is possible to provide computer-based support for the cooperative elements of the software development process. CSCW should be incorporated into the software development process to provide computer-based solutions to the problems that arise when people cooperate in groups.

In support of the hypothesis, a conceptual model will be used throughout the dissertation to illustrate important aspects of the CSCW paradigm. Initially, a general conceptualisation of the CSCW paradigm is established, as illustrated in Figure 1.1.

The model depicted in Figure 1.1 comprises the following main components:

- *CSCW*, the computer supported cooperative work paradigm including groupware and other components that are necessary for the computer-based support provided to participants engaged in group activity.

- *Cooperative Agents*, which allow the participants in the groupware communications to organise their work, to communicate and to

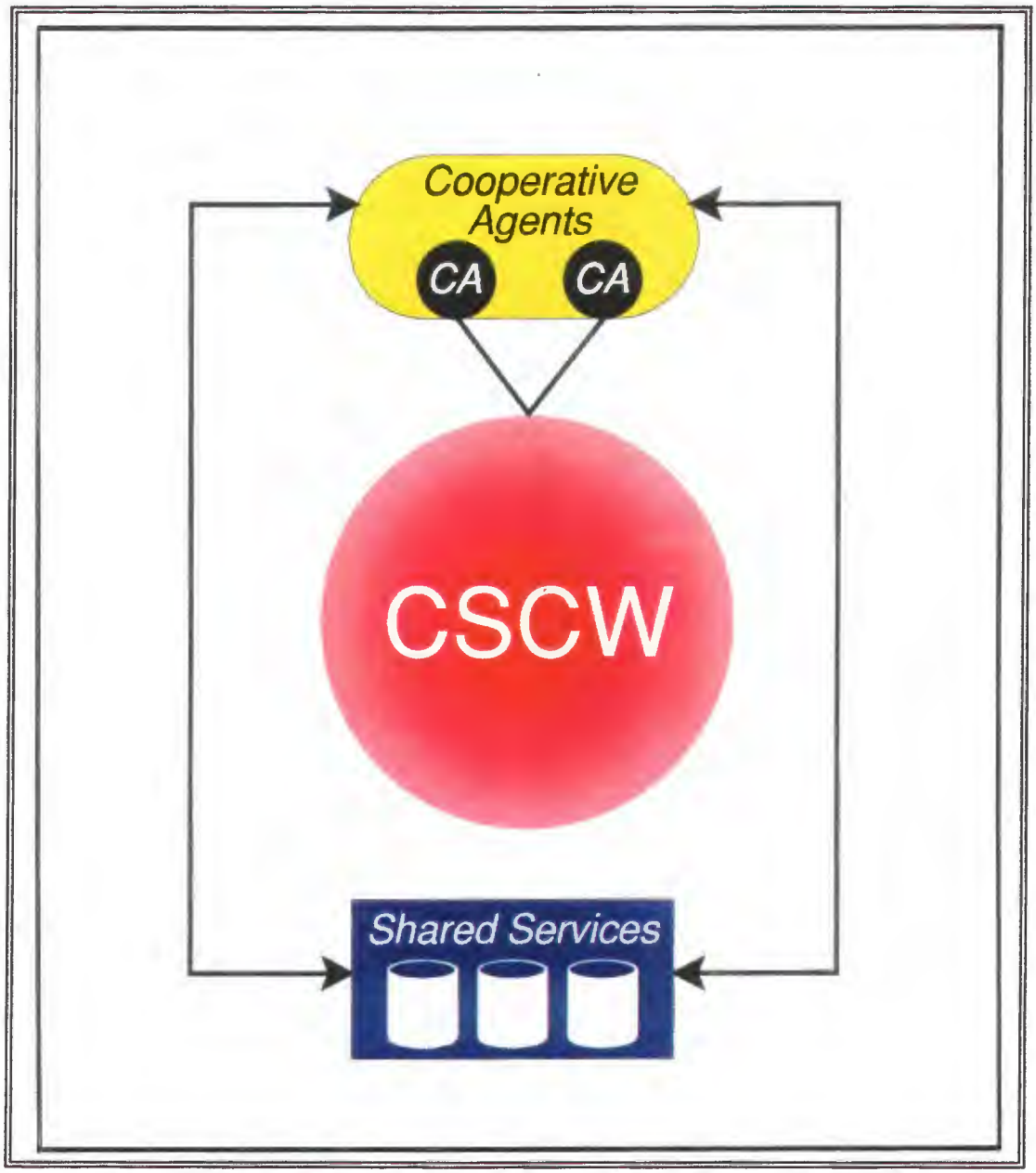

Figure 1.1 *General Conceptualisation of the CSCW paradigm*

ensure that the communications and cooperative work are performed in a coherent and consistent way.

- *Shared services*, which allow the participants to access common resources and to share common information.

Further instantiations of the conceptual model, depicting primitives of the

CSCW paradigm, will be done as more insight is gained in following chapters.

Other principles pertaining to the subject under investigation are also fundamental. These include:

- Sound software engineering principles, a methodological approach, and the application of project management techniques can guarantee the success of a software product.

- The requirements engineering performed during the analysis cycle is crucial to the success of a software system under development because all the work of the other cycles will be based on the deliverables of the analysis cycle.

(ii) Assumptions

The object-oriented paradigm is followed in describing the software development process. The software development process model for this investigation is the revised spiral model proposed by Du Plessis and Van der Walt (1992).

The revised spiral model is used because it is believed that the spiral model alleviates many of the problems of traditional software development process models, and secondly because it has been revised specifically to cater for object-oriented development.

(iii) Constraints

The constraints of the investigation include the following:

- The scope of the research has boundaries to meet the requirements for a partial dissertation.

- CSCW is interdisciplinary. CSCW can be placed on the boundaries of computer science, sociology, organisational and management studies, perhaps even anthropology, and human computer interfacing (HCI) concerns which already place it on the boundaries of psychology, linguistics and ergonomics. The dissertation will only touch the relevant aspects from these different disciplines which are necessary for the study.

- The software development cycles of the revised spiral model are studied within the parameters of the OISEE project in the Department of Computer Science and Information Systems at UNISA.

Several relevant issues are addressed but fall outside the scope of the investigation. They include:

- Project management of a software development project.
- Quality assurance and verification of the software product.
- Prototyping of a proposed software system.
- Distributed software development and networking architectures.

1.4.2 Method of Investigation

The state of the art of CSCW as it may be applied during software development was examined. This was done by means of a literature study during which issues were analytically explored and relevant issues identified. An attempt was made to conceptualise the problem, and establish a CSCW framework for software development.

(i) Literature Study

An extensive literature study was conducted, which included the use of CD-ROM technology. The references were analytically reviewed, interpreted and classified for significance in terms of the hypotheses and aims of the investigation. The following keywords were used in the search for literature on the subject of the research:

- Computer Supported Cooperative Work
- Groupwork
- Groupware
- Software Engineering
- Requirements Engineering
- Object Orientation

(ii) Conceptualisation

A synthesis is made of ideas concerning computer-based support for the groupwork during software development. A conceptual model is formulated as a proposed solution to the problem of supporting cooperative work during software development.

1.5 Structure of the Dissertation

The dissertation consists of six chapters.

Chapter 1 serves as a general introduction to the rest of the dissertation. The relevant research of the investigation is identified. A motivation for investigating the area of research is given. The particular aspects that will be considered are mentioned. A possible solution is proposed for dealing with the problems of cooperation during the software development process.

In *Chapter 2* an overview of computer supported cooperative work is given. The origins of CSCW are discussed and the important issues of work in groups are analysed. The nature of CSCW and some ongoing debates concerning it are examined. The main CSCW technologies are surveyed and classified. The requirements for CSCW are discussed in terms of general, specific, and architectural requirements. The meta primitives of the CSCW paradigm are derived from the main components and dimensions of CSCW. The conceptual model of CSCW meta primitives is constructed.

Chapter 3 discusses the software development process, the software development life cycle, software process models, methodological aspects, and tools to support the process. The information system building blocks form the basis for determining the possible role that CSCW may play during software development. Particular attention is given to topics such as the object oriented paradigm, and the roles of participants in the software development process. The issues surrounding cooperation and group activity during software development are investigated. Finally, an attempt is made to describe the use of CSCW during software development.

Chapter 4 provides an overview of CSCW technology that may be applied during software development. The discussion will focus on CSCW tools which may support specific group activities during software development.

Chapter 5 is devoted to the establishment of a CSCW conceptual model for software development. A meta model of cooperative software development consolidates important aspects of groupwork during the software development process. The CSCW conceptual model for software development is constructed and finally a framework for software development within a CSCW Environment is established.

The contribution of the research is summarised in *Chapter 6.* The original hypothesis are validated in the light of the research results. Areas for further investigation are proposed.

CHAPTER 2

Computer Supported Cooperative Work in Perspective

CONTENTS

2.1 Introduction

Workgroup computing, collaborative computing, groupware, computer-based support, coordination-technology, cooperative work support, are all terms that have stirred a wide interest in recent years. The area of Computer Supported Cooperative Work, or CSCW, appears to have established itself as a research field in its own right over the last few years, judging from the wealth of conferences and papers devoted to the topic. Despite the interest, confusion concerning the very nature of the field continues to surface (Schmidt & Bannon, 1992). Differences of opinion exist, for example, as to what is or is not a CSCW system, and the relationship between CSCW and what has been termed groupware.

What precisely is meant by the term CSCW? Does it denote a new approach to problems of harnessing information technology to human needs? Where does this new field originate from, and where is it going? In this chapter, an attempt is made to answer these questions by providing a conceptualisation of CSCW, firmly anchored in a framework and meta model that pays attention to the requirements for computer support of cooperative work. The aim is to construct a conceptual model of the field that allows those active within CSCW to have some common reference point, and some understanding of the important aspects of the field.

This chapter is organised as follows. Firstly, a brief historical account of the field is given. A number of factors that may have contributed to the emergence of CSCW are then presented. Further detail is provided regarding the numerous attempts to delineate the core concerns of this field culminating in a definition of CSCW. CSCW technology is discussed in terms of existing CSCW products and a classification of CSCW systems. The requirements of CSCW are then explored. Finally, the main components and meta primitives of CSCW are identified.

2.2 Origins of CSCW

The birth of the term *Computer Supported Cooperative Work* (now commonly abbreviated to CSCW) is attributed to Irene Greif and Paul Cashman who organised an interdisciplinary workshop on the development of computer systems that would support people in their work activities (Greif, 1988). It brought together participants from different disciplines and environments, such as office information systems, hypertext, distributed information systems, and computer-mediated communication.

The first open conference, with the title *Computer Supported Cooperative Work* was organised in 1986 in Austin, Texas. Around 300 people from a variety of backgrounds, including artificial intelligence, human-computer interaction, office information systems, computer science, psychology and anthropology attended the conference. Interest in the field has continued to grow and an even larger conference on the topic was held later in Portland, Oregon. At this conference there was an interesting shift in focus towards concern with the nature of the software development process. A number of papers, especially several Scandinavian contributions, addressed the issue of user involvement in the early phases of software development as a prerequisite for quality software development.

Despite the fact that both the boundaries of the field and its focus were still somewhat unclear, the enthusiasm for the topic has continued over the years. The first European conference on the topic was held in 1989, followed by yearly alternate conferences in North America and Europe. There has also been a number of other CSCW-related conferences and workshops on topics such as group decision support systems, multi-user systems, and collaboration technology. Many journals in the areas of human-computer interaction, decision support, office systems, and software engineering now include CSCW in their research areas. New journals with a more specific focus on CSCW have appeared. Many

edited papers and monographs on the subject were published in recent years (Bannon & Schmidt, 1992).

A broadly accepted framework for the study of CSCW systems classifies the type of work according to its temporal and geographical distribution (Ellis, Gibbs & Rein, 1991). In the two-dimensional space, four types of cooperative working can be identified as shown in Figure 2.1. These working types correspond to four different meeting types, or scenarios, in which the cooperative work is supported by computers.

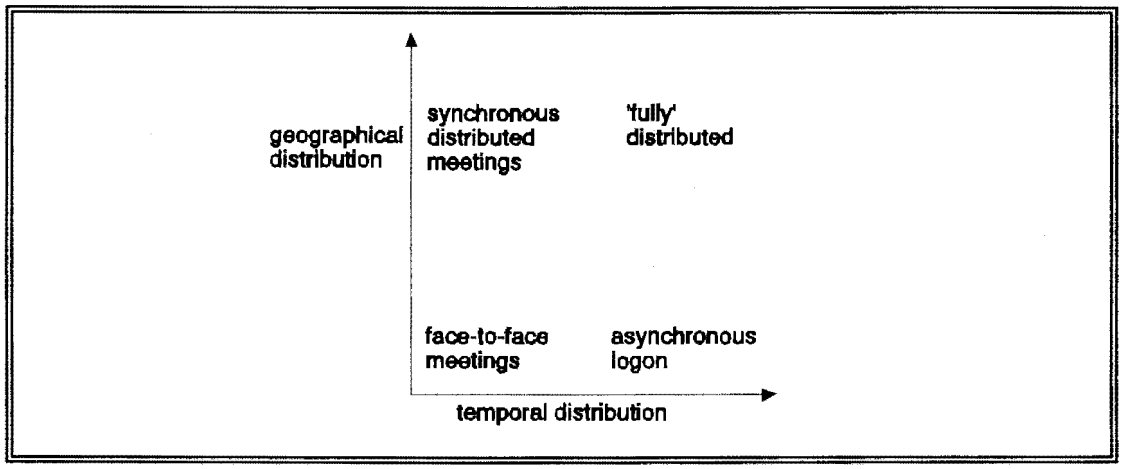

Figure 2.1 *CSCW Scenarios (Viller, 1991)*

It is clear that the area of CSCW has become accepted as a legitimate sphere of academic research and development activity, with a growing number of interested researchers and many software manufacturers. But what caused the emergence of an apparently "new" set of issues in the field of information technology, development, and use? To what extent were existing approaches unable to fulfil the needs of computer-based support for cooperative activities? Ongoing developments in technology, approaches to work, and people's needs in the work place have contributed to a concern for a technology that suited the actual work practices of people. These issues are examined in the next section.

2.3 The Nature of CSCW

Bannon (1993) believes that a number of changes which occurred in the field of information systems practice, in the environment of organisations, in technology, and in people's expectations concerning computer-based support of their work activities have contributed to the emergence of the CSCW theme. The changes can be structured under human and organisational changes creating demands for more appropriate technologies to support work practices.

2.3.1 Human and Social Factors

The organisational environment, people's expectations, and the inclination to be social are addressed.

(i) The Organisational Environment

The increasingly complex and turbulent environment in which organisations operate has stimulated a need for better ways of organising and coodinating work activities. Organisations have requirements for closer integration, up-to-date information, and access to information "anytime, anywhere". In the domain of production systems computer-integrated manufacturing (CIM), just-in-time (JIT) methods, and various other techniques to reduce stock inventory have made a contribution to productivity and profitability. Computers are connected both intra an inter-organisationally. Organisation strategists believe that organisations should make networking a reality. CSCW requires a networking environment to support group interactions over local and wide areas. CSCW encourages collaboration and coordination via flexible information systems that are accessible in the office, at home, on the factory floor, and even when travelling.

(ii) People's Expectations

More and more work is being mediated by computer systems, and workers in a group need to share and jointly manipulate information. Software and hardware vendors are expected to support greater inter-connection between computers, and applications that would allow for sharing, multi-user access to facilities, and greater integration of applications. There is a need for augmenting interaction by using the computer to help coordinate activities and support joint problem-solving. Shared workspaces, as well as shared tools may be provided where people may dynamically create objects and modify them. Computer users themselves demand more flexible and tailorable user interfaces and additional functionality that would allow them to accomplish their work more efficiently and effectively (Bannon, 1993). Although, a lot of research has been done in the area of human computer interfaces (HCI) that have contributed to better interfaces, problems remain regarding incompatibilities between systems, and the inability of systems to support groups of users. These concerns are explicitly addressed in the CSCW research community, thus attracting the attention of user representatives who wish to have more useful and effective tools for group-based work activities.

At another level, people are becoming more skilled in using modern technology, and wish to exert this influence when work practices are to be changed by new technology. A number of software developers also wish to support the software development process with better tools for group activities. They see CSCW as a possible avenue for greater cooperation among the participants, through the development of better computer-based tools for the cooperative analysis and design work process (Bodker et al., 1988).

(iii) *Social aspects*

A certain amount of dissatisfaction has set in among researchers in human computer interfacing (HCI) as a result of disappointment with the relevance of many HCI studies to actual design and work practice (Thomas & Kellogg, 1989). Anything beyond the human-computer dyad was neglected, and there was a total failure to take into account the situated nature of everyday activity and the social complexity of work (Suchman, 1989). Academic research groups who are concerned with the use of more qualitative methods and field study techniques, have shown interest in CSCW. CSCW would allow them to make HCI work more ecologically[3] valid. Sociologists and anthropologists also have been able to find CSCW valuable in their empirical studies of work settings. The focus in CSCW on the requirements of the work, and the need to study the work domain closely, have lead to the importance of field studies.

2.3.2 Technology Factors

(i) New Software Markets

Commercial software developers have shown interest in CSCW as a possible growth area for new software applications in terms of supporting group activities rather than individual activities. These software applications include groupware. Although, some early groupware products, such as shared calendaring systems have enjoyed limited success, software developers are still optimistic. It has been noted by Greif (1988) among others, that in the long run it may not make sense to differentiate a segment of the software product market in terms of groupware, as all software will be groupware. Any particular software application will have

[3] *Ecology* is the study of the interaction of people with their environment (Oxford Dictionary, 1982)

the required features to support group work when it is needed. For the moment, however, there is a marketing interest for software supporting group work in both real-time and non-real-time settings.

(ii) Technological Developments

Significant developments within the technological arena have made infrastructural computer networking possible. The isolated personal computer (PC) is becoming less popular in organisations. Various forms of local area networking (LAN) and wide area networking (WAN) allow for shared resources in organisations. Networks and technical developments in the areas of interoperability, security, transparency, and distributed systems support greater connectivity between people, locally, regionally, and globally. These elements are of importance in the CSCW arena.

2.3.3 Changing practices

Information systems researchers have accepted the need to understand more fully the practices of people at work, in order to build more appropriate supportive technology. The earlier goal of "automating the office" has been discarded on both theoretical and practical grounds. Information flow diagrams of office activities do not specify how work is actually accomplished, and they cannot serve as an adequate base for capturing or automating office activities. Instead of considering a office as a place where people perform a set of well-structured tasks according to prescribed procedures, it is viewed as a social community where work is accomplished through the locally situated activities and interactions of office members. The shift in emphasis for office information systems can be seen in the change in terminology from "automating" the office to "supporting" office workers. Computer systems are seen as support systems for the human workers, rather than as replacements for them. It is being recognised that in most

work situations the accomplishment of work involves multiple individuals, together with their computer-based tools, and that many inefficiencies in work practice stem from inadequate computer-based support for the smooth interleaving and coordination of tasks across people and machines (Bannon, 1993). Many information systems researchers see the need for further understanding of work place practices as a key aspect of improving the quality of information systems development.

The next section attempts to identify the viewpoints, characteristics and boundaries of CSCW and to find a definition for it amidst the confusion that currently exists.

2.4 Definition of CSCW

Despite interest in the new field there is a lack of consensus about the term CSCW and the corresponding research field (Wilson, 1991). This may be illustrated by considering the different viewpoints of CSCW (Bannon, 1993):

- *CSCW as simply a loose agglomeration of cooperating and at times competing communities*

  At the most simple level, it can be argued that CSCW is simply an "umbrella" term for an idea that is concerned with people, computers, and cooperation in some form. This vagueness allows people from different disciplines, with partially overlapping concerns as to the current state of technology and the understanding of use contexts, to come together and discuss issues of mutual interest. CSCW in this view is an area where different groups vie for the attention of participants, rather than a coherent focussed field.

- *CSCW as a paradigm shift*

  Hughes et al. (1991) stresses that CSCW should be considered not as a

specialised subdiscipline but as a paradigm shift in the perspective from which computer support systems are designed. The emphasis is on "the turn to the social" with the realisation that much work on people-technology systems has systematically avoided issues concerning the social organisation of work and the consequent implications for the design of appropriate support technology.

- *CSCW as software for groups*

  A view commonly found among information technology and business consultants, and even among software developers and researchers is that computer support of "groups" or teams is the hallmark of the field. This has given rise to the term *groupware* for computer products marketed in this area (Johansen, 1988). This view has been criticised, because the focus may fall on small teams or homogeneous groups with convivial work relations, thus ignoring situations in everyday organisational life where issues such as politics and power play an important role. Other criticisms stem from difficulties of enumerating properties of "groups" as found in the work place, and the relevance of many group studies undertaken in laboratory and workplace situations.

- *CSCW as technological support of cooperative work forms*

  Here the emphasis is on understanding cooperative work as a distinctive form of work (Schmidt, 1991), and on appropriate technology to support these cooperative work forms. In this view, cooperative work does not imply any notion of shared goals or conviviality, but rather people engaged in work processes related as to content. The distinction between cooperative work and individual work, and the functionalist perspective, neglecting subjective factors of cooperation, are criticised.

- *CSCW as participative design*

  The proponents or practitioners of participative design focus on

developing alternatives to traditional systems development, alternative ways of doing design, of involving users, etcetera. The Scandinavian school of systems developers in the CSCW community has influenced some people to equate CSCW with participative design. This may lead to confusion because, although various forms of user involvement are important to the development of successful CSCW systems, use of such techniques does not automatically signify any focus on cooperative work. Participative design researchers are in many cases not interested in computer support for design practices.

The variety of views on the nature of the CSCW field is the subject of debate and the definition of the field, its core concerns and its boundaries, is best viewed as contested terrain, even more so as the field struggles to find a unique identity. The concepts of *groupware* and *groupware systems* have greatly influenced the realisation of the specialised field of CSCW. Groupware can be defined as the electronic (computer-based) components supporting participants engaged in group activity. Joosten et al. (1993) considers groupware to be a collection of tools to facilitate the cooperation within an organisation, whereas a groupware system is defined as a computer-based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment. According to Hughes et al. (1991), most practitioners acknowledge three semi-articulated characteristics of CSCW. Firstly, it involves locations where two or more people interact with each other through a computer. Secondly, it falls within a particular class of system to service such settings, for example groupware. Thirdly, CSCW is interdisciplinary. As already mentioned in Chapter 1, CSCW can be placed on the boundaries of computer science, sociology, organisational and management studies, perhaps even anthropology, and human computer interfacing (HCI) concerns which also place it on the boundaries of psychology, linguistics and ergonomics. The dissertation will only touch the relevant aspects from the different disciplines which are necessary for the study.

In 1988, Irene Greif defined CSCW as an identifiable research field which focusses on the role of the computer in group work. A basic definition is "CSCW looks at how groups work and seeks to discover how technology (especially computers) can help them work." (Ellis et al., 1991). Although this may intuitively sound acceptable, it is based on a naive view of groups without any clarification, as if it has some clear, widely accepted meaning (Kuutti, 1991).

Besides these intuitive efforts there have been some other attempts, mostly based on distinguishable external features of the work to be supported, or design metaphors, such as tools, shared material, and communication medium. (Kuutti, 1991). Recent authors, e.g. Bannon & Schmidt (1991), Lyytinen (1990) and Suchman (1989) has made some attempts to overcome the vagueness of the term CSCW. Bannon & Schmidt (1991) has made a radical departure from the use of the external features of cooperative work or design metaphors as starting points for a definition. Their view is as follows: "Cooperative work is constituted of work processes that are related as to content, that is, processes pertaining to the production of a particular product or service". Lyytinen (1990) uses structuration theory to analyse work and the role of CSCW applications. Structuration theory perceives the work process as a social structure, constructed continuously by "human agents" and possessing a detailed internal structure. Lyytinen's definition is similar to that of Bannon and Schmidt and puts special emphasis on the formation of social structures in interactions and thus on the role of CSCW applications both as a medium and an outcome in the formation of the work process. Suchman (1989) emphasises fundamental aspects of work: that practice is always fundamentally social and that it is always mediated by artifacts (some computer-based).

The question is: Would it be possible to find a common denominator for the different definitions and still maintain an acceptable delineation of the research field? The definition of CSCW as "work by multiple active subjects sharing a common object and supported by information technology" obviously covers a

great part of recent research and draws clear boundaries around the object of it (Kuutti, 1991). In the definition, the *active subjects* distinguish CSCW from traditional information systems, where predetermination of work sequences by the system is the normal case. A common *object of work* is different from a shared goal (criticised as being too restrictive) or shared material (criticised as being too loose). According to Kuutti (1991), negotiators may have opposite goals, but a common object, a problem space. Database users may share material, but the objects of their work may not be similar. In conclusion, a community which shares a common object of work is always delineated in practice, whatever the contributions of the different participants may be.

Besides the definition, additional needs expressed by researchers should be fulfilled by basic CSCW research. Bannon and Schmidt (1991), Lyytinen (1990) and Suchman (1989) all agree that:

i) Work is mediated by artifacts and the basic unit should cover this aspect.
ii) The basic unit should allow considerations of social and cultural aspects of a work situation.
iii) Work and the means for it are continuously reconstructed, therefore transformation and development aspects should be considered.
iv) The basic unit of research should have a detailed internal structure (Lyyttinen, 1991).
v) The topics of control and conflicts should be considered in the basic unit of research (Kling, 1991).

Now that CSCW is defined it is apt to investigate the main CSCW technologies.

2.5 CSCW Technology

CSCW product development is still in its very early stages. Few products built on pure CSCW principles currently exist, though there are many other products which may support the group working process in one way or another. While there is considerable overlap between CSCW and existing groupware products, CSCW has an altogether new kind of software and a new kind of user-to-user relationship (Nolle, 1993). For example, E-mail is analogous to an interoffice mail exchange, while CSCW will resemble face-to-face or phone conversations. The "shared workspace" concept of groupware remains, but in CSCW applications, the computer workspace (the files and applications being shared) is maintained in real-time as much as possible. In this section CSCW product categories will be discussed and the important CSCW systems classified.

2.5.1 CSCW Products

A number of CSCW products which support the group working process in one way or another have appeared. Product categories include (Wilson, 1991):

- Message systems
- Computer conferencing systems
- Procedure processing systems
- Calendar systems
- Shared filing systems
- Co-authoring systems
- Screen sharing systems
- Group decision support systems (GDSS)
- Advanced meeting rooms
- Team development and management tools

Message systems include electronic mail systems, some of which provide facilities

to label messages. The labelling may include the classifications such as question, answer, and promise, and enables the system to track conversations and to prompt users to provide appropriate responses. As an example the abstract architecture of *Active Mail* (Goldberg et al., 1992) which underlies the electronic mail idea is illustrated. The *Active Mail* configuration consists of *agents*, interconnected via two-ported bidirectional *communication channels*. There are two types of agents: *users* and *applications*. A configuration of *Active Mail* is given in Figure 2.2.

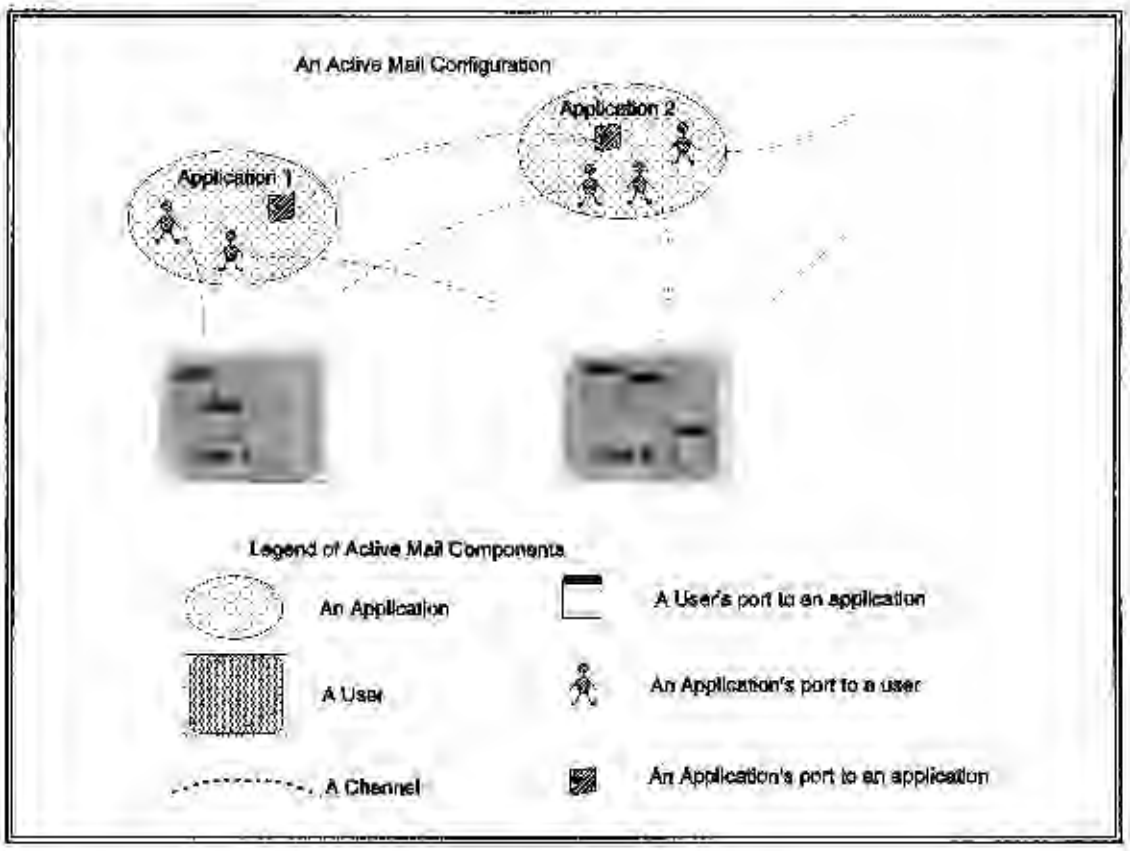

Figure 2.2 *An Active Mail Configuration (Goldberg et al., 1992)*

The *conversation agent* depicted in Figure 2.3, supports an ongoing asynchronous conversation among a dynamically changing group of users. The conversation agent maintains a conversation log and when the agent receives a contribution from a participant, it appends the data to the log and sends a message containing the contribution and the name of the contributor to all participants. When new

participants join the conversation a copy of the current conversation log is made available to them.

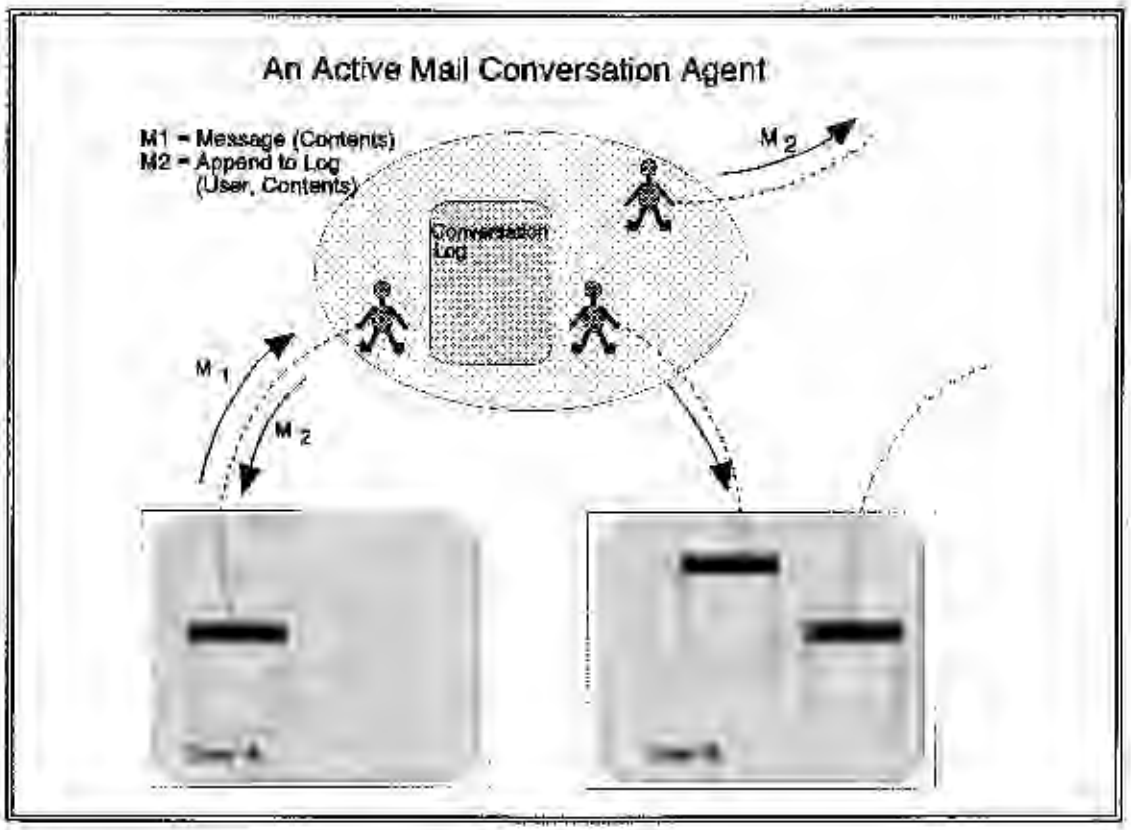

Figure 2.3 *Active Mail Conversation (Goldberg 1992)*

Computer conferencing systems are for example bulletin board systems in which messages are not sent to other users but to a "conference" which has members. In starting a conference, members are shown all the new material that has been produced since their last meeting. Figure 2.4 illustrates the basic process structure in a conference. The diagram shows how the *conference managers* handle communication among machines. Matching processes within a tree represent *application peers*, and matching trees are *conversations*. The following terms are important (Crowley et al., 1990):

- A *conference* is two or more identical collections of application processes which are all in synchronised execution states. Each collection belongs to one specific conferee.

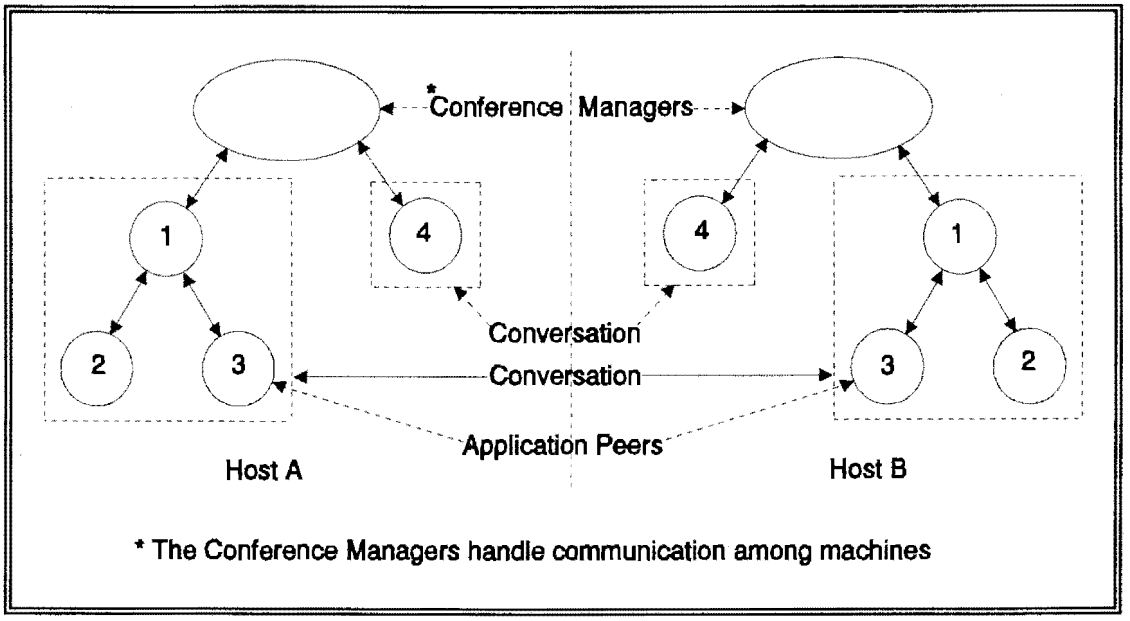

Figure 2.4 *The Process Structure in a Conference (Crowley et al., 1990)*

- The *conference manager* is a special application process that comprises the conference management and communications functions. A single conference manager represents each conferee.

- *Application peers* are all the copies of a specific application process.

- A *conversation* is a single top-level set of application peers initiated by the conference manager, as well as all sets of application peers the top-level set creates.

- The *floor* identifies the active user in a conversation, the conferee currently providing events to the conference. Every conversation has exactly one floor. *Floor control* is a mechanism and policy under which users exchange possession of the floor.

- Software is *conference-aware* if it takes special action when it detects that it is running in a conference.

Procedure processing systems are systems which support the routing, input requirements and status reporting of electronic form-based processes.

Calendar systems are systems which allow individuals to share their diaries, and provide automated support for the arrangement of meetings. Figure 2.5 illustrates the *Active Mail* meeting scheduler.

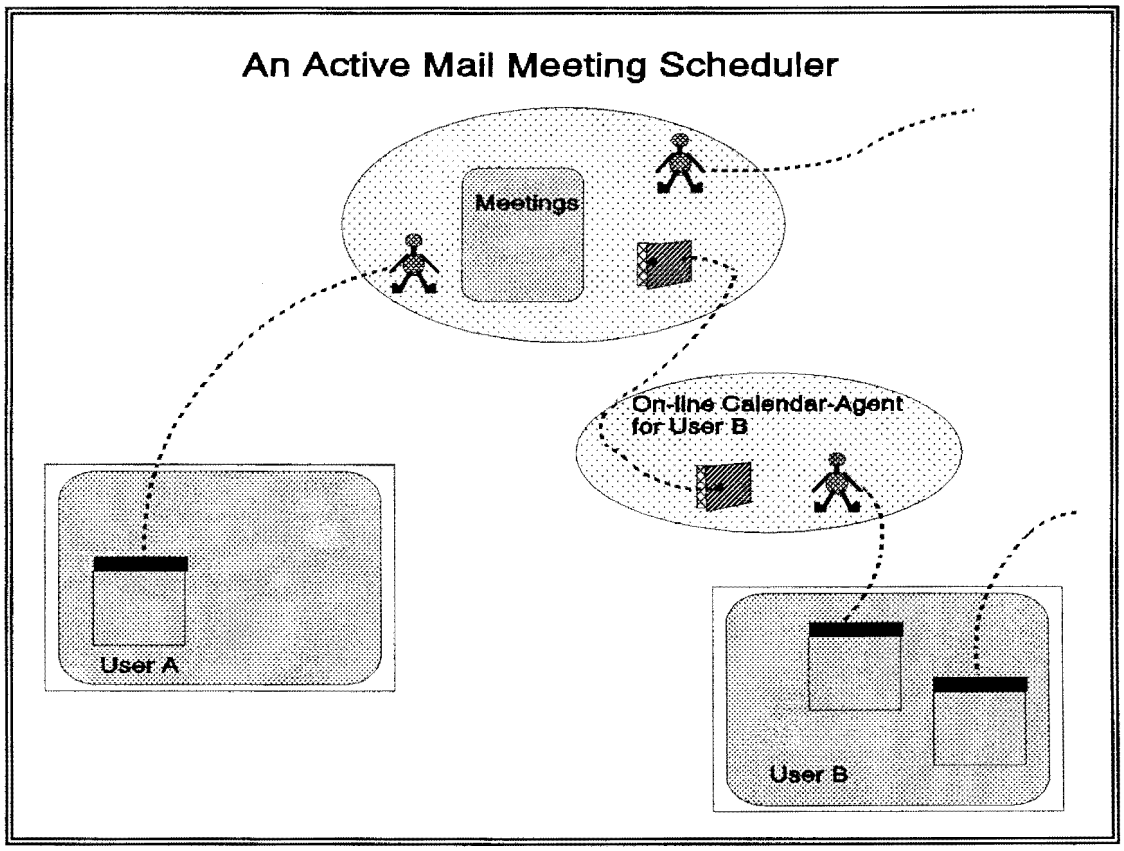

Figure 2.5 *A Meeting Scheduler (Goldberg et al., 1992)*

Shared filing systems are systems which enable individuals to create and access information in a shared database. Some systems enable links to be made between information held in the file (often referred to as hypertext systems).

Co-authoring systems are systems which support some or all aspects of joint creation of documents. Examples are idea generation, outlining, peer review and version control.

Group decision support systems (GDSS) are systems which help groups to solve unstructured problems. Some GDSS systems are designed to be used in face-to-face meetings while others for use when participants are in different locations.

Screen sharing systems are systems which enable people using their own workstations to see the contents of other peoples' screens. Some products provide facilities to view and control other screens and to transfer files from one screen to another. More sophisticated systems provide integrated voice communication and additional facilities for specific activities such as voting or co-authoring. As an example three typical display arrangements of media spaces are illustrated in Figure 2.6 (Ishii et al., 1992). Media spaces represent computer-controlled video environments.

In (a), a display providing a live video image of the partner's face is alongside a display for shared work. The *ARKola* simulation (Gaver et al., 1991), in the *IIIF* environment, and some modes of *CAVECAT* (Mantei et al., 1991) adopted this arrangement.

In (b) the displays are repositioned to resemble the situation of interacting across a table. *VideoDraw* (Tang & Minneman, 1991) and *Commune* (Bly & Minneman, 1990) experiments adopted this arrangement.

In (c), the live video images and the shared workspaces are incorporated into different windows of a single screen. *TeamWorkStation* (Ishii, 1990), *PMTC* (Tanigawa et al., 1991), *MERMAID* (Watabe, 1990) and some *CAVECAT* nodes employ this desktop-video technology.

Team development and management tools provide facilities such as the analysis of group inter-activity based on individual responses to questionaires. Management tools specifically include project planning and project monitoring facilities.

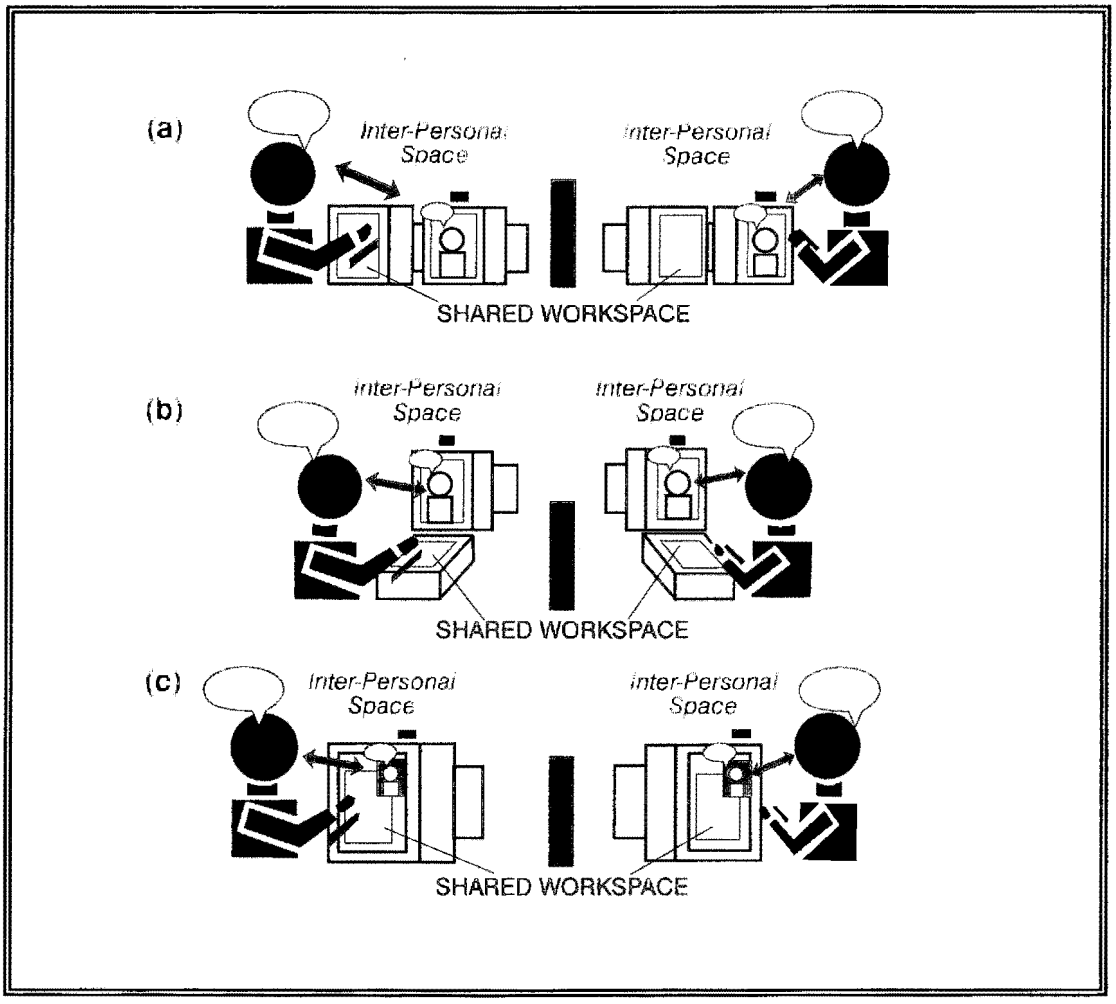

Figure 2.6 *Typical Screen Arrangements in Media Space (Ishii et al., 1992)*

Advanced meeting rooms are systems which incorporate a range of technology to improve the effectiveness of face-to-face meetings. Of particular significance is the ability to project a computer screen onto the meeting room wall screen. In this way information, graphics, and presentation material are displayed to meeting participants. Work accomplished during meetings is displayed as it is input. More sophisticated systems provide a computer for each participant and a range of meeting support software. Figure 2.7 gives an example of the possible monitor arrangements that could be utilised in a meeting.

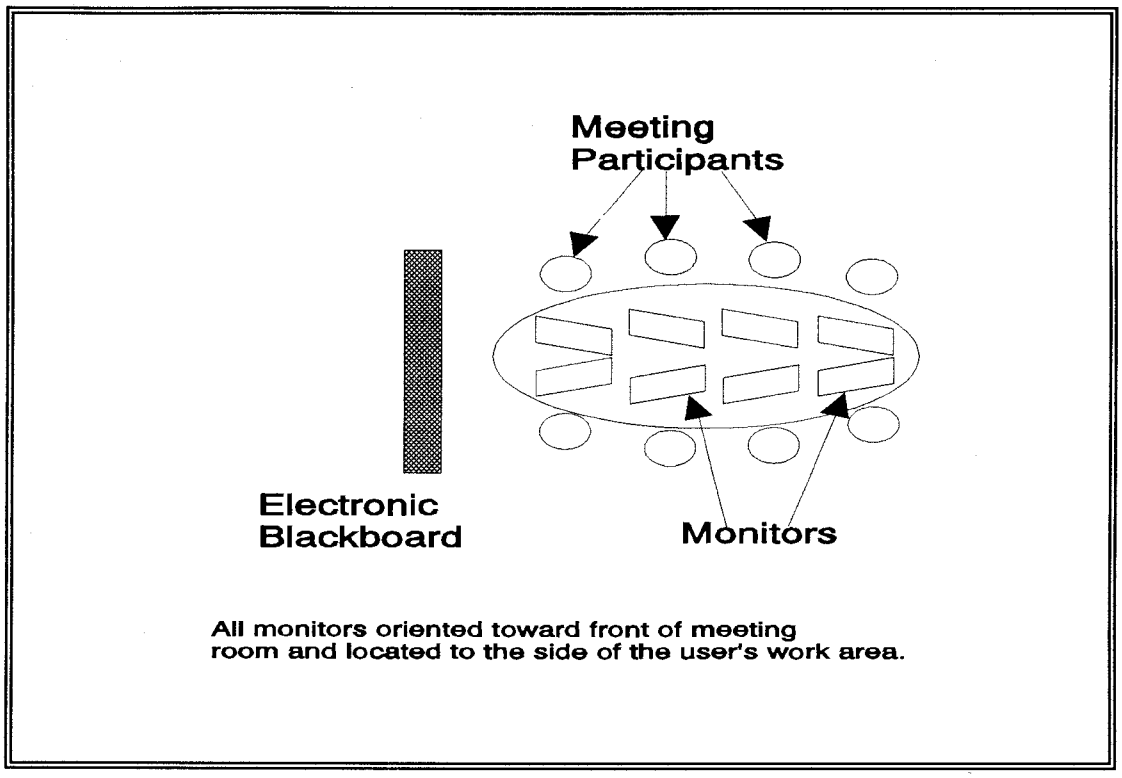

Figure 2.7 *Monitor Orientations and Placements possible in the Layout of an Oval Conference Table as in Capture Lab (Mantei, 1988)*

2.5.2 Classification of CSCW systems

CSCW systems are primarily concerned with supporting a number of cooperating users. The nature of this cooperation can be distinguished by the way in which group members interact. Interaction or cooperation may be either *synchronous* or *asynchronous* (Rodden & Blair, 1991). Synchronous interaction requires the presence of all cooperating users while asynchronous cooperation occurs over a longer time period and does not require the simultaneous interaction of all users. Figure 2.8 shows how a number of classes of CSCW systems fit into this division.

Also, cooperative systems can be considered as being either *remote* or *co-located* (Rodden & Blair, 1991). In this classification the division between remote and co-located is as much a logical as a physical one and is concerned with the accessibility of users to each other rather than their physical proximity.

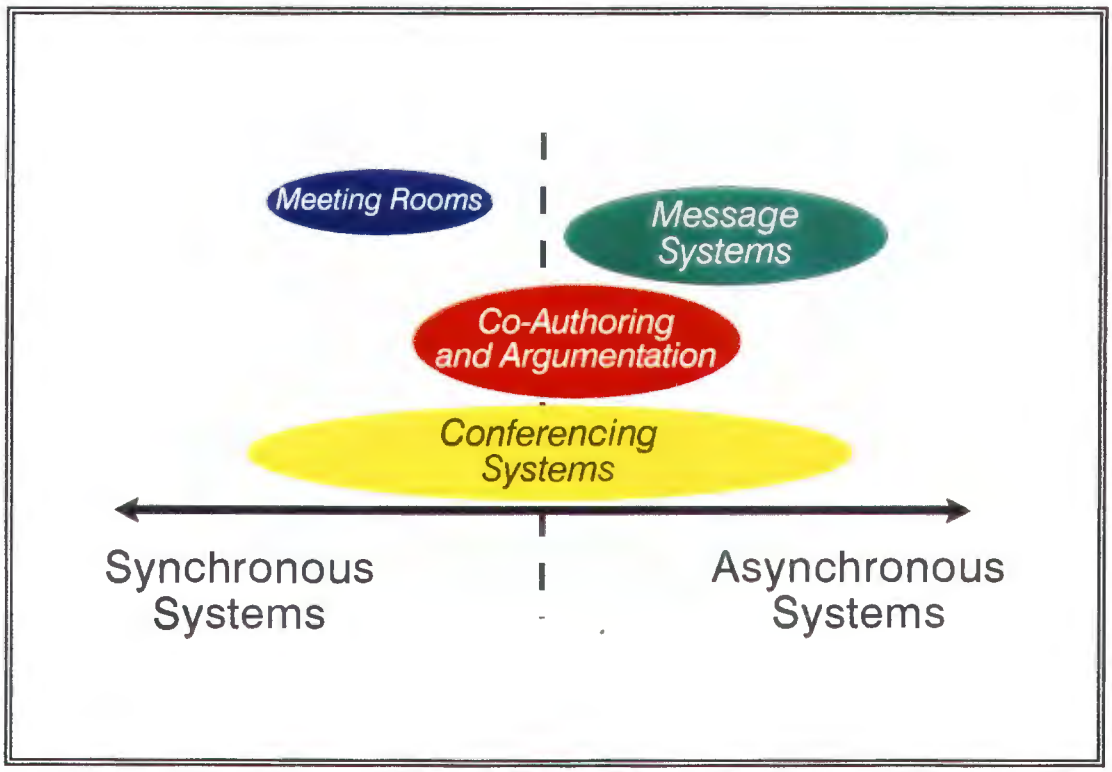

Figure 2.8 *Forms of Cooperation in CSCW Systems (Rodden & Blair, 1991)*

Figure 2.9 shows a range of CSCW systems in terms of their geographical nature. The four geographical divisions are:

i) *Co-located*

Purely co-located systems require the local presence (in one room) of all users. This class of system normally takes the form of a meeting room with a large projected computer screen and a number of personal computers linked by a local area network (Kraemer, 1988).

ii) *Virtually co-located*

These systems are similar to co-located systems but do not have the requirement that participants need to be in one room. This is often achieved by the use of multimedia technology in which real-time audio and video links are maintained. Systems in this class include real-time multimedia conferencing systems such as those developed for *MMConf*

(Crowley, 1990) and *Mermaid* (Watabe, 1990).

iii) *Locally remote*

These are systems which provide high-bandwidth (for remote purposes) real-time accessibility between users often using shared screen techniques. Examples of such systems are argumentation and co-authoring systems such as *CoAuthor* (Hahn, 1991) and *gIBIS* (Conklin, 1987) and real-time conferencing facilities such as *RTCAL* (Sarin, 1985).

iv) *Remote*

These systems are those that assume the existence of only minimal accessibility between users. They include message systems which assume only the simplest of communication systems and computer conferencing systems which assume only rudimentary "dial-in" mechanisms.

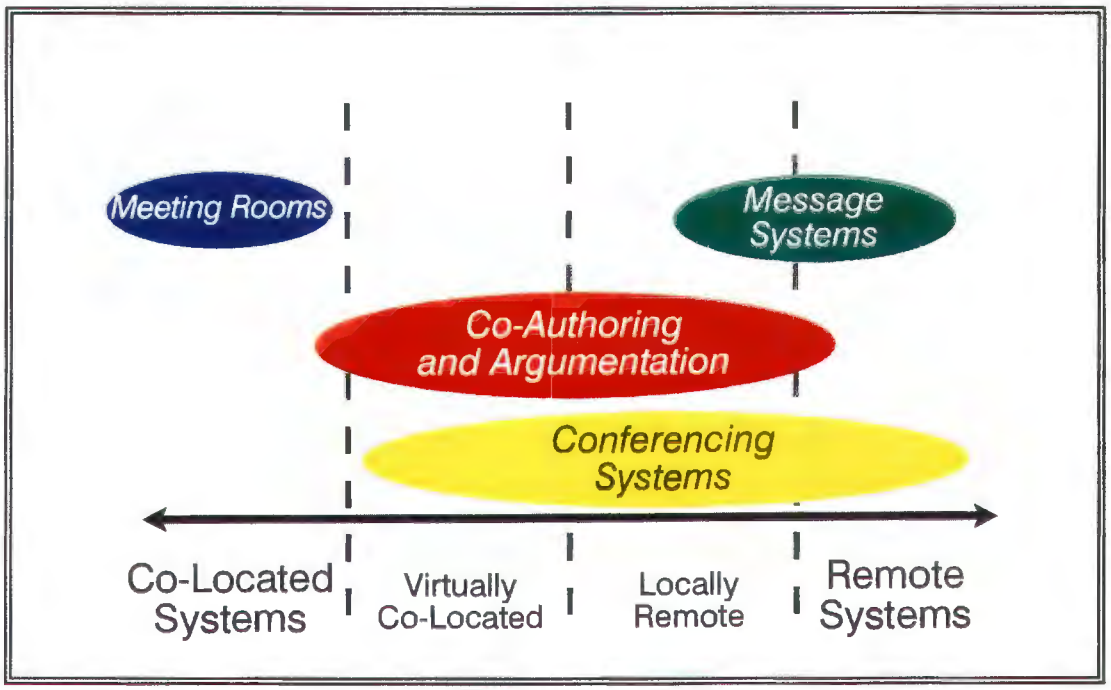

Figure 2.9 *Geographical Nature in CSCW Systems (Rodden & Blair, 1991)*

Synchronous systems require highly cooperative distributed systems while asynchronous systems tend to be more autonomous in nature. Asynchronous

cooperative systems need only the facilities provided by electronic mail systems while synchronous systems test the facilities provided by the most cooperative of distributed operating systems. Most distributed systems are found at the lower end of the spectrum of synchronous systems and this may account for the lack of highly interactive synchronous cooperative systems. Figure 2.10 illustrates the two views of cooperative systems.

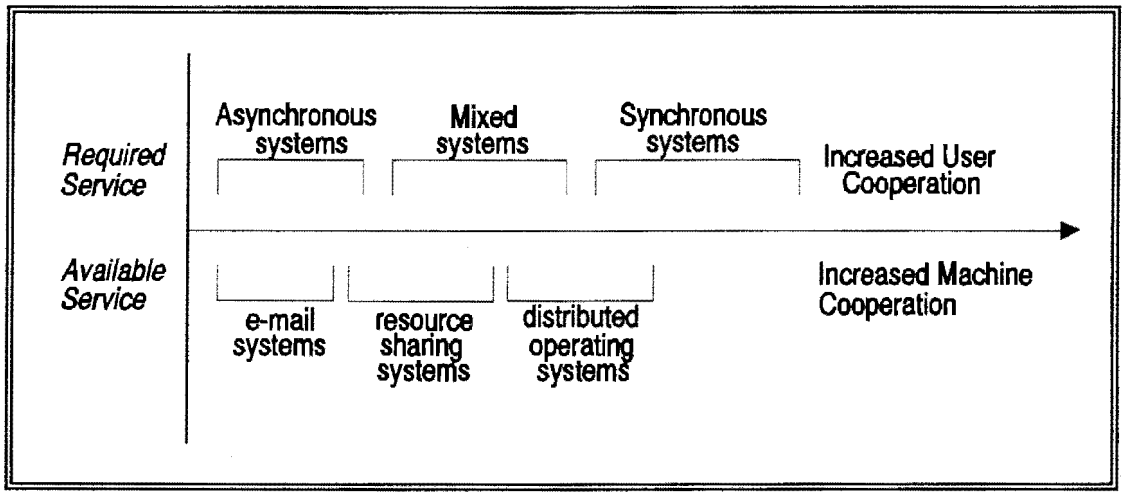

Figure 2.10 *Comparison of Models of Cooperation (Rodden & Blair, 1991)*

It has been recognised that CSCW systems either fall within systems based on the principle of information exchange, or within systems based on information sharing. Systems based on the principle of information exchange are often called structured or active message systems and assume an asynchronous and remote mode of cooperation. The assumption underlying these systems is that members of a group cooperate primarily by exchanging messages. Systems based on the principle of information sharing consider how users share information and develop mechanisms to support sharing. In this form of cooperative system, users interact through a shared information space, as shown in Figure 2.11. This model of interaction is often augmented by the use of direct user-to-user communication, normally provided by either electronic message systems or by a video or audio link.

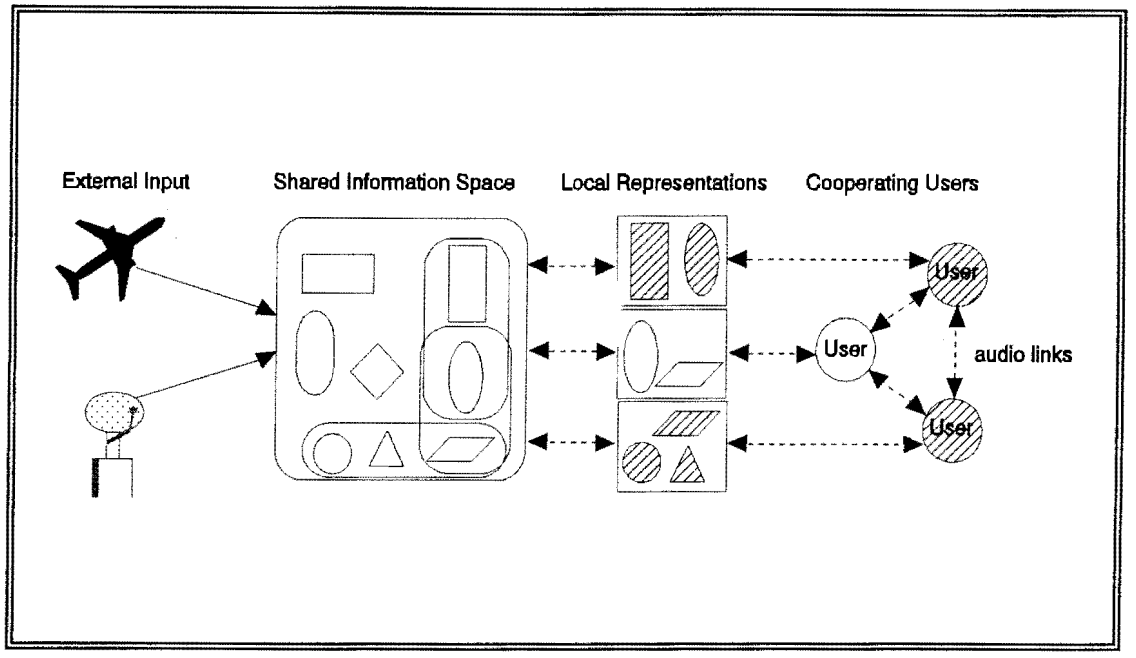

Figure 2.11 *Interaction in Shared Information Systems (Bentley et al., 1992)*

A simple classification of the *representation and control of cooperation in message based cooperative systems* yields three major classes of system:

i) *Speech act or conversation based systems*

Speech act systems are based on speech act theory, and apply a linguistic approach to computer-supported cooperation. It has been primarily applied by Winograd and Flores (1986) to form the basis of several computer systems including the *Coordinator* system (Winograd, 1987) and the *CHAOS* project (De Cindio, 1986). In this class of system cooperation is represented and controlled using some form of network structure detailing the patterns of message exchange.

ii) *Office procedure systems*

These systems attempt to describe tasks performed within an office in terms of the combined effect of a number of subtasks or procedures (ranging from well defined to less well defined tasks). Research has concentrated on finding a unifying language which allows the specification

of office procedures and a description of their interaction. Artificial intelligence (AI) technology and AI based planners have been used in systems such as *POLYMER* (Croft, 1988) to execute actions and coordinate interaction often handling uncertainty and exceptions. Systems which also fall within this class are *AMIGO* (Danielson, 1986) and *COSMOS* (Wilbur, 1988). In this class of system procedural languages are used to describe and control cooperation by defining roles and activities.

iii) Semi-formal active message systems

These systems follow a philosophy of semi-automation, automating parts of the system which are suitable for automation while leaving other parts of the system manual in nature. Examples include *Object Lens* (Malone, 1988) and the *Strudel* project (Sheperd, 1990). In this class of systems mechanisms are provided for automatic message handling and the concepts of roles and autonomous agents are supported.

Table 2.1 lists a range of research projects using message based techniques and the form of control adopted by these systems.

Message based systems have used the available message handling standards. For example, the work on *COSMOS* and *AMIGO* has taken place within the framework of OSI standards such as X-400 (CCITT, 1987) and X-500 (Prinz, 1991). The message standards have provided a technical infrastructure which enabled the exchange of structured information. Figure 2.12 shows the representation and control of information within the system as a layer on top of the message standard being exploited.

Table 2.1 *Message Based Systems and their Form of Control (De Meer et al., 1992)*

Research Project	Control Model	Representation
Coordinator	Formal - Speech Act	Network
Object Lens	Semi-Formal	Production Rules
Chaos	Formal - Speech Act	Network
Domino	Formal - Procedural System	Script Based
Cosmos	Formal - Augmented System	Script Based
Amigo	Formal - Augmented System	Script Based
Strudel	Semi-Formal	Production Rules

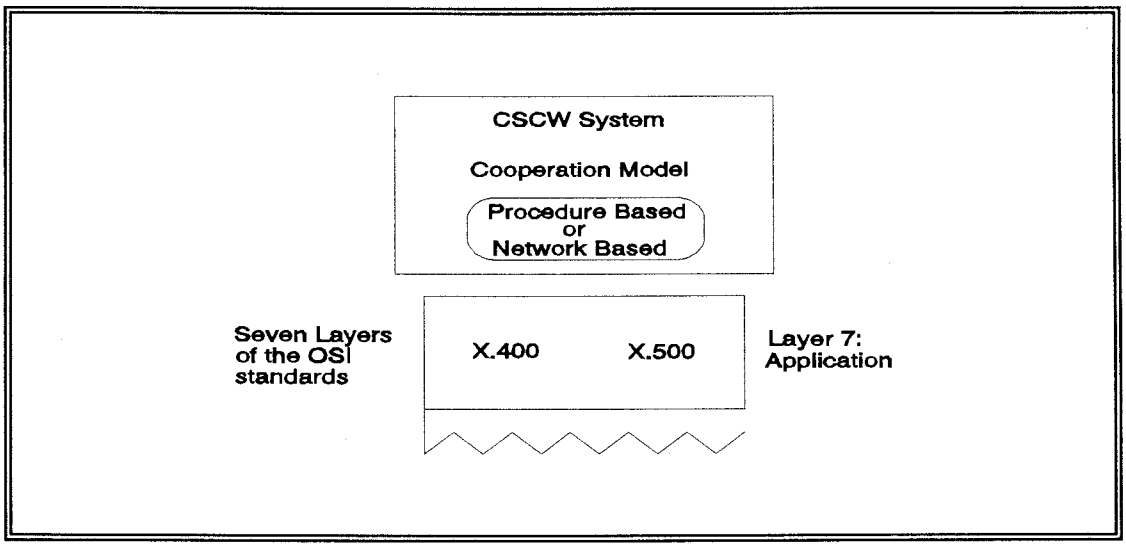

Figure 2.12 *The Role of Standards in Message Based Systems (De Meer et al., 1992)*

Systems based on *information sharing* have emerged depending on two principal characteristics of the shared information space. These characteristics are:

i) *The form of interaction with the shared information space*

The user may interact with the shared space either asynchronously over a long time period or synchronously in a real-time manner.

ii) *The type of information represented in the information space*

Many forms of information may be represented in the shared information space. Usually, this information has been textual, but with the advent of modern workstation capabilities a variety of media representations are possible.

Systems based on information sharing has led to the emergence of a number of distinct forms of system, as shown in Table 2.2.

	The form of interaction	
Type of information	ASYNCHRONOUS	SYNCHRONOUS
TEXTUAL	Traditional Conferencing Systems	Real-time Conferencing Systems
MULTIMEDIA	Multimedia Information Systems	Multimedia Desktop Conferencing Systems

Table 2.2 *The Classes of Conferencing Systems (De Meer et al., 1992)*

One of the most important aspects of CSCW, is to control interaction. Two major forms of control are prevalent in CSCW systems, namely *explicit* and *implicit control.* In systems which provide explicit control, users may both view and tailor group interaction and cooperation. Systems exhibiting implicit control provide no

techniques for representing or coordinating group interaction. These systems dictate cooperation by the styles of interaction they allow. Control mechanisms for message based systems exploit explicit control, while cooperative systems based on shared surfaces mainly exhibit implicit control over cooperation. In shared space systems there are three major forms of *implicit control*:

i) *Traditional conferencing systems*

These systems provide basic control mechanisms which are minimal and fixed within applications. Information is grouped into defined conference topics and conference moderators control the addition of information to these topics.

ii) *Floor control systems*

In modern real-time conferencing systems, the control is based on the floor control mechanism embedded in the conferencing application. Normally, a floor control token is passed beteen users to distinguish who has write access to the shared conference space at any given time.

iii) *Peer-group meeting or control free systems*

Peer meeting systems such as the *Colab* system (Stefik, 1987) do not provide any control mechanisms and rely on the meeting participants to formulate their own meeting protocols. All the users have equal priorities and status, and may amend and use the shared space freely. The system does not keep track of the nature and form of group work being undertaken and provide limited support for these work processes.

There are four basic approaches followed for real-time shared workspace as illustrated in Figure 2.13. Unlike their message based counterparts, systems based on sharing do not comply to definite communication and distribution standards as yet. Recently, a CCITT standard for traditional computer conferencing has been investigated.

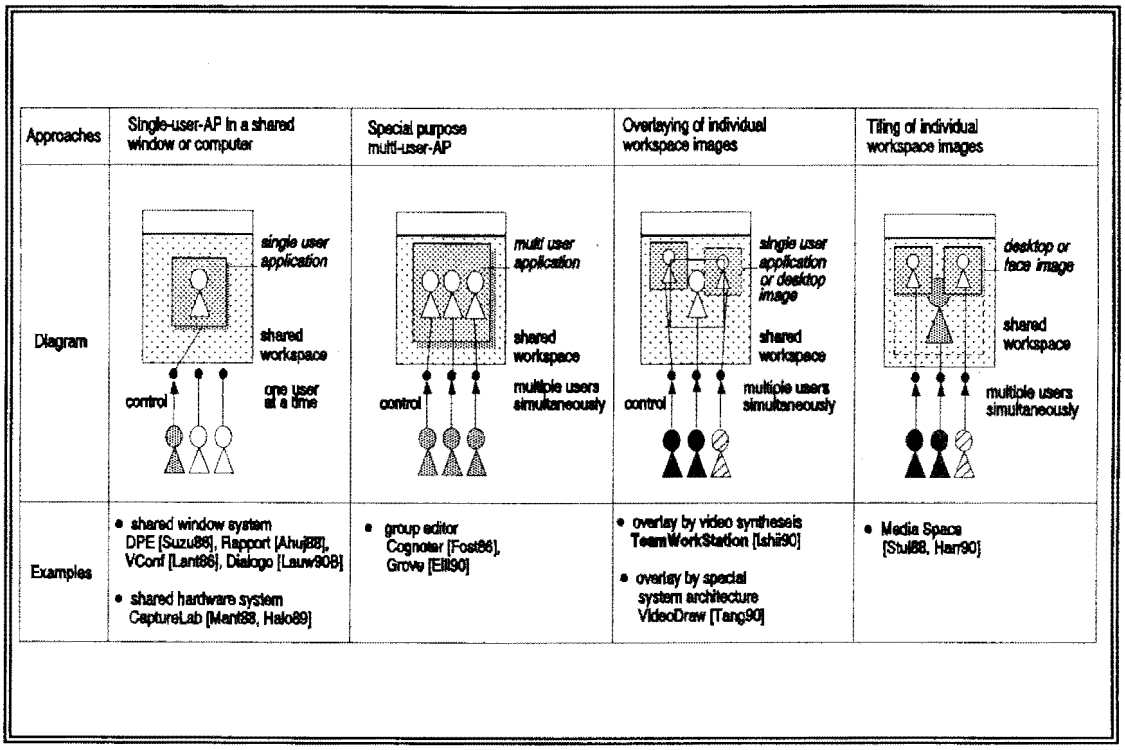

Figure 2.13 *Four Approaches for Real-time Shared Workspace Design (Ishii, 1990)*

2.6 Requirements for CSCW

In addition to the requirements of single-user applications, multi-user applications as found in the CSCW domain make several new demands. The most obvious is a need to control the degree of sharing and nonsharing among users. Since multi-user applications are generally network-based, an architecture must help manage this network connectivity.

The identification of requirements for a CSCW application environment provides the starting point for the development of open CSCW systems (Navarro et al., 1993). The requirements represented here are by no means complete but are intended to illustrate requirements which are characteristic and distinctive of CSCW systems and will directly impact future distributed systems. A distinction is made between general requirements and specific CSCW requirements. The general requirements are also applicable to open distributed systems, but are of

particular importance as basic requirements of CSCW.

2.6.1 General Requirements

CSCW systems share similar demands to existing applications which have already influenced the development of open systems. However, the manner in which these needs are met, is of importance to CSCW. Coborra (1988) has summarised *general requirements* of work group support system, which include:

- To standardise tasks, thus reducing task uncertainty.
- To standardise interfaces between execution of subtasks, thus streamlining coordination.
- To facilitate reporting and monitoring of performance.
- To encourage communication through creation of new channels or improvement of existing ones, thus reducing hierarchical barriers and allowing new ideas to flow more easily.

The requirements that should be addressed by multi-user applications may also be considered as general requirements and should be adapted to suit CSCW. Patterson et al. (1990) describe the following multi-user application requirements:

- Provide flexible control over the dimensions of sharing

  The essence of multi-user applications is sharing and the control of sharing. Concerning the dimensions of sharing, a distinction should be made between data objects and view objects. Data objects are *underlying objects* and correspond to the abstract information comprising the application, but without an indication of how this information should be displayed. View objects are *interaction objects* and contain the information about how to display and interact with an underlying object.

The requirement can now be discussed in terms of the three dimensions of sharing. The first is the sharing of the underlying objects. There should be ways to control how *interaction objects* become attached to or detached from underlying objects, which in turn controls whether the underlying objects are shared or not.

The second dimension is the sharing of the presentation or view provided by the interaction objects. The application should have the ability to handle situations where the underlying objects are the same, but the view on these objects are individualised (different views to different users). Alternatively, in some situations the underlying information, as well as the view are shared.

The final dimension of sharing is the sharing of the input authorisations or access provided by the interaction objects. In some situations a user might have a literal copy of another user's view, but should be prevented from interacting with the information in any way. Alternatively, equal access should be possible, but in such a case control measures should be enforced in some or other way.

- Provide robust session management

In order to better understand what is needed, a few definitions are necessary. A *session* is the duration of an application process from its first invocation to its termination. *Joining* a session is the act of connecting a user's display resources to the session. *Dropping* a session is the act of disconnecting a user's display resources from the session.

The typical mechanisms of starting and stopping single-user applications are inadequate for multi-user applications. Whereas single-user applications join a user as soon as they start and drop the users as soon

as they end, multi-user applications must support a richer regime. A session may be started by someone on behalf of others without the participation of that individual in the actual session. Users should be able to join as they can and drop as necessary without halting the session. Additionally, sessions may or may not be terminated by the departure of the last user. Multi-user applications should know from inherent communication software how to communicate with its users. These aspects of session management are necessary for multi-user applications to be readily accessible to their users.

- Maximise the joint probability of successful execution

  The joint probability that several users will be able to join the application should be maximised. This may be difficult when the community of users is open or very heterogeneous. Operating systems may differ; terminals may vary: and communication protocols may not be standardised. An architecture for multi-user applications should attempt to minimise the restrictions that it places on a user's environment. At a minimum it is necessary to adopt a common protocol for establishing communication paths and a common protocol for controlling user terminals, but these common protocols should be as widely available as possible.

The general requirements of information sharing and communication are discussed in more detail as they are essential for cooperative working in an open distributed environment.

i) Support for information sharing

Sharing of information is essential in cooperative working. Patterns of sharing should be adopted within the environment for effective cooperation to take place. The environment needs to provide a set of services which would encourage the

cooperative sharing of information (Navarro et al, 1993). These *services* include:

- Services for access and exchange of information between the different applications running on different platforms and used by different users.
- Maintaining a knowledge of people, resources and ongoing activities, for example by the integration and use of standard information repositories such as the X.500 directory service.
- Mechanisms for modelling the organisational context in which the cooperation takes place and which are based on the appropriate access control mechanisms.
- Services that provide awareness of the ongoing actions in a shared information space. The user should be better informed about the current status of the cooperative work.

According to Navarro et al. (1993), a number of standard initiatives have examined the role of information within group work. Researchers have examined the relevance of the X.500 directory service to CSCW systems. More recently, working groups have investigated standards to allow the sharing of conference structures across groups. The issues of access control, locking and transactions are central to the needs of CSCW applications and yet are missing within current standards initiatives.

ii) Support for communication

Communication plays a vital role in cooperation, and it is important that this role is reflected within a CSCW environment (Navarro et al, 1993). A CSCW environment will need to provide a range of *communication services* which should include:

- Support for a wide range of media, including telefax and where applicable paper communication.

- The provision of many different forms of communication, including both real-time and asynchronous communication.
- Support for interchange of cooperation across communication media.

Specific requirements in terms of *communication in a cooperative environment* are mentioned in Reder and Schwab (1990). A CSCW environment should:

- *Support remote access to communication and information technologies*

  Extensive travel, complex schedules and tendencies to work from home which some workgroups exhibit, raise a need for facile remote access to office-based communication and information technologies to support work continuity across environments.

- *Support channel switching*

  The frequent switching over time among channels as individuals work together on a given task has serious implications for the design of supportive technologies. In addition to the synchronous integration of multimedia for advanced workstations, capabilities should be developed for transposing communication from one medium to another and/or integrating multimedia events across time (i.e. asynchronous integration).

- *Support individual multitasking*

  Existing calendar or scheduling tools assist individuals to manage their time, but they do not assist workgroups to manage members' time. New constructs and constraints need to be represented in order for software to be useful for such purposes.

Traditionally, communication support for CSCW has been provided by

asynchronous OSI communication standards such as X.400, but they do not allow a sufficiently diverse range of communication styles for many CSCW systems (Navarro et al., 1993). Accordingly, most CSCW systems adopt and augment the basic services. Communication standards have merely provided a technical infrastructure for the exchange of structured information. The representation and control of cooperation within the system has generally been a layer on top of the message standard being exploited, often within a closed application.

2.6.2 Specific CSCW Requirements

CSCW as an application domain needs a unique set of requirements for the developers of open systems support. The requirements emphasise the need to support a wide range of work practices. CSCW systems should incorporate some concept of *activity* (Navarro et al, 1993). Here, an activity is interpreted as a cooperative process of interactions between people. CSCW systems should provide means to describe activity-specific dependencies and interrelationships applied to various applications in a flexible manner to allow for the evolution of new working styles. New concepts which arise within CSCW makes *tailorability* essential and extend the set of *transparencies* (easily understood, obvious pretext) of importance to open systems. These requirements are now reviewed.

i) Support for activities

CSCW systems have a strong relationship with the various organisational activities which they support. A CSCW environment should provide mechanisms for representing the various relationships between these activities. Additionally, the environment needs to provide a set of services to allow the management of the different activities within the environment. The *services* include:

- Managing the membership of activities.
- Sharing resources between activities.

- Scheduling activities and monitoring the progress of activities.
- Mechanisms for negotiating the responsibility for activities.
- Mechanisms for negotiating the division of competence within activities.
- Coordination of activities.

These services should be provided in a neutral way to allow a wide range of CSCW applications to be supported by the environment.

ii) Support for tailorability

Cooperative working is essentially a dynamic activity, and thus CSCW applications need to be tailorable. This has two important consequences. Firstly, the environment needs to provide a set of services similar to a developers toolkit to enable this tailorability. Secondly, and more importantly, is that the traditional division between users and developers becomes less clear, with users attaining similar powers and status as systems developers. The limits and bounds of tailorability and possible notations, languages or services to support this tailorability will be an important research area for future CSCW developers.

iii) Support for transparency

Cooperative activities are generally carried out by a distributed group of people who are located at possibly different places, employed at different organisations, working at different times, using different user-group interfaces, having slightly different goals, having different understandings of the activities, language, competence and culture. The CSCW environment should provide some *degree of transparency* to facilitate the cooperation of people from different coordinates. Complexity should be reduced by hiding some dimensions that are unnecessary for the cooperative activity. According to Navarro et al. (1993), significant dimensions include: organisational (*organisational awareness*), temporal (*integration of synchronous and asynchronous*), linguistic, cultural and physical

units (*distributed working*).

A number of different *forms of transparency* are important to CSCW systems.

- *Transparency of organisation* means that activities need not be dealing with the complexity of the possibly different organisations involved. Interorganisational connections should hide the complexity of different organisational and inter-organisational (free market or other) policies. Widely distributed CSCW systems should provide means for the establishment of virtual organisations which exist only by the definition of shared electronic workspaces and high-speed communication links.

- *Transparency of time* deals with the mode of work, namely synchronous or asynchronous. Transparency here implies that interaction will be independent of the mode we are using.

- *Transparency of view* means that the functioning of applications should not be influenced by the way users view data. This transparency is not applicable in WYSIWYG activities.

- *Transparency of activity* means that a set of objects cooperating in one activity should not necessarily be aware of other unrelated objects present in the distributed environment. The unrelated objects may be located in a different location or in the same location but participating in other activities. In this way activities are not disturbed by other unrelated activities.

iv) Support for technology transformations

Currently, there is a tremendous technology transformation in the key domains of information processing and computer communications (Karmouch, 1993).

These *technology transformations* include:

- The emergence of high-speed networks with powerful workstations and new storage technology (repositories) imposes a new way of information processing, and provides opportunities for applications that were not possible with previous technologies.

- Distributed system configurations where several powerful workstations share resources at several sites by communicating through LANs are now common practice. Other configurations may cover wider areas such as LAN interconnection through MANs and even WANs, allowing communications between geographically dispersed users.

- Multimedia has enabled the direct manipulation of new types of information such as video, voice and image, all integrated in a single entity. It requires faster networks to be transmitted, high performance processing, and multimedia storage systems or repositories.

- Optical fibre technology is perceived to overcome the limitations of current networking technology in providing high-speed networks, thus permitting the transfer of large amounts of multimedia information over a single channel in an integrated and synchronised manner.

CSCW should make valuable use of new technology directions to satisfy the requirements of diverse cooperating groups.

v) Support for advanced networking facilities

The aim of CSCW is to provide integrated support to groups in terms of three key variable conditions of work: face-to-face group activity, activity at different times (caused by both time zone differences and the passage of time), and

activity at different geographical locations. Networks are the key to this ambition, but if they are to provide this support effectively they must match the design standards of CSCW systems in general. CSCW has contributed a broad set of requirements in the network design area (Wilson, 1991). These networking requirements include:

- The use of voice, image and video in documents, databases and electronic mail systems.
- Multi-party desktop videoconferencing linked to studio based videoconferencing systems.
- Intelligent agents which roam the networks on behalf of their user(s) searching for information and carrying out tasks.
- Network resources which enable groups to establish themselves and work within the network environment. These resources should support aspects of group work, provide shared working space and storage, and support user interfaces which represent the shared group environment.

These and other advanced network facilities should be considered in conjunction with the changing and growing usage of networks. People will be expected to work in teams in dual, but integrated worlds: the real physical world, and an electronic world underpinned by networks.

vi) Support for different media

New technologies are being used for developing integrated networks capable of providing the level of service required by different media (Karmouch, 1993). At this moment, the challenge is to define a model and communications architecture for *multimedia* cooperative applications. Applications operating in a distributed cooperative environment greatly complicate the system's design and architecture when compared to single-user applications, where the system is designed to serve one user at a time (even if it operates in a distributed environment). Multimedia

cooperative applications have several related factors: group cooperation strategies and paradigms, computer communications, a multi-user interface, and shared services such as multimedia databases. The requirements for *multimedia cooperative applications* may be categorised into four classes: storage and processing, common functionality, cooperative aspects and communications (Karmouch, 1993). The four classes of requirements are summarised as follows:

- *Storage and processing*: multimedia information is a composition of different kinds of complex objects such as text, graphics, image, voice, audio and video. Each type of information requires appropriate tools for acquisition, processing, transfer and storage. The different types of information may have semantic and temporal links between them and require a uniform and homogeneous representation architecture within one generic object called a *multimedia document*. Efficient support of multimedia documents is a primary requirement, since they play a major role as an information vehicle.

- *Common functionality*: In a cooperative environment the functions that are common to all applications should be extracted from a class of applications. The remaining functions (local to specific applications) should be left to the applications themselves. Thus, common functions should be provided by a unique entity shared by the class of applications.

- *Cooperative aspects*: Firstly, cooperative rules (group organisation, time and space considerations) must be defined between the users to achieve a common task or goal. Secondly, facilities and protocols need to be defined between the applications (e.g. propagation of changes, floor negotiation, role attributions, synchronisation). Thirdly, interfaces to a shared environment (shared space) are needed to provide the same (visual) view to the users. Fourth, access to shared databases (repositories) must be provided to the group for decision support.

- *Communications*: group communications by nature take place among geographically distributed users. Thus, communications over networks are crucial, and the requirements for handling the transfer of multimedia documents and data control in a cooperative environment are complex and diverse. Among these are the requirements for multicasting (multiple media transmission), appropriate bit rates for each type of medium, synchronisation of single- and multimedia and time constraints (real-time communications, acceptable user feedback). These are requirements beyond the basic requirements for much faster networks capable of transporting data in the hundreds of Mbit/s range, and the use of protocols providing a variety of functions to meet expected multimedia services such as error-free transmission, and time transparency.

2.6.3 Architecture Requirements

The requirements identified in the previous section are used to guide the design of a generic distributed multimedia CSCW environment. Navarro et al. (1993), in their MOCCA project, have identified a number of perspectives from which to view the environment. Each perspective offers an abstract view of specific environment functionality. Formally, the perspectives can be refined to a set of corresponding models which collectively specify the environment. Adopting an object-oriented paradigm, the environment consists of a set of objects representing people, applications, organisations, resources and relationships between these objects. Each object may appear in several models, viewed differently in each. Applications and their resources (documents, messages) are integrated into the environment by so called object adaptors. The different MOCCA viewpoints are illustrated in Figure 2.14.

The *distributed architecture* considers how the models can be used to specify components of an open distributed platform. The *organisation viewpoint* considers

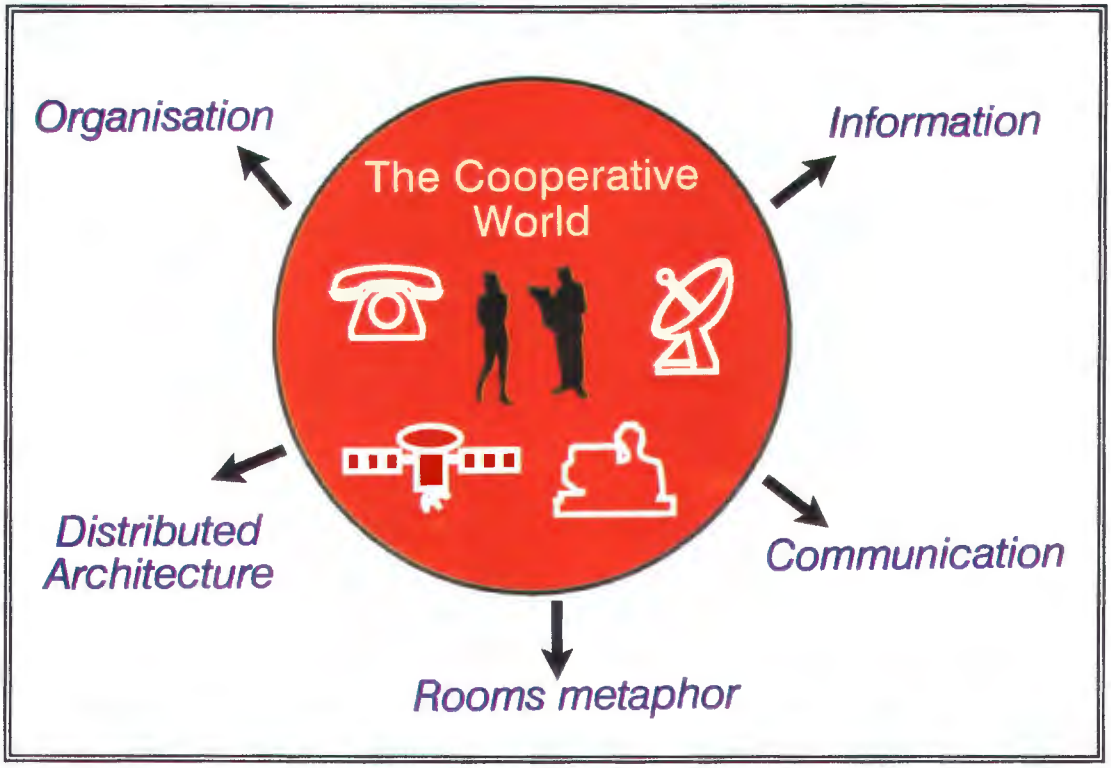

Figure 2.14 *MOCCA Viewpoints on CSCW (Navarro et al., 1993)*

the organisational context required for CSCW in terms of a set of resources. The *information viewpoint* examines the definition of shared information, relationships between information objects, and the global naming of information. The *communication viewpoint* groups together mechanisms of interaction dealing with the information resources. The *rooms metaphor* provides the user's conceptual map of the global environment and addresses the issues of location, navigation and social interaction. Starting from each viewpoint, a model that describes the objects as they appear in that viewpoint can be defined. The emphasis now is on the *architecture viewpoint*.

In the general architecture a set of cooperative support functions are decomposed into a set of loosely connected managers which provide appropriate portions of cooperative support. Active support within the environment is provided by the *domain manager*, the *activity manager*, the *security manager, the information manager*, and the *communications manager*. The general architecture

is presented in Figure 2.15. The managers are able to abstract over the services provided by existing OSI and open distributed processing (ODP) platforms. Two of the managers are aimed at supporting the manipulation and storage of information. Most cooperation involves the sharing of information. Subsequently, a central repository, an *information store*, for shared information plays a significant role in a cooperative environment. This is augmented (within a cooperative environment) by the provision of an *organisational database* which allows organisational information to be shared.

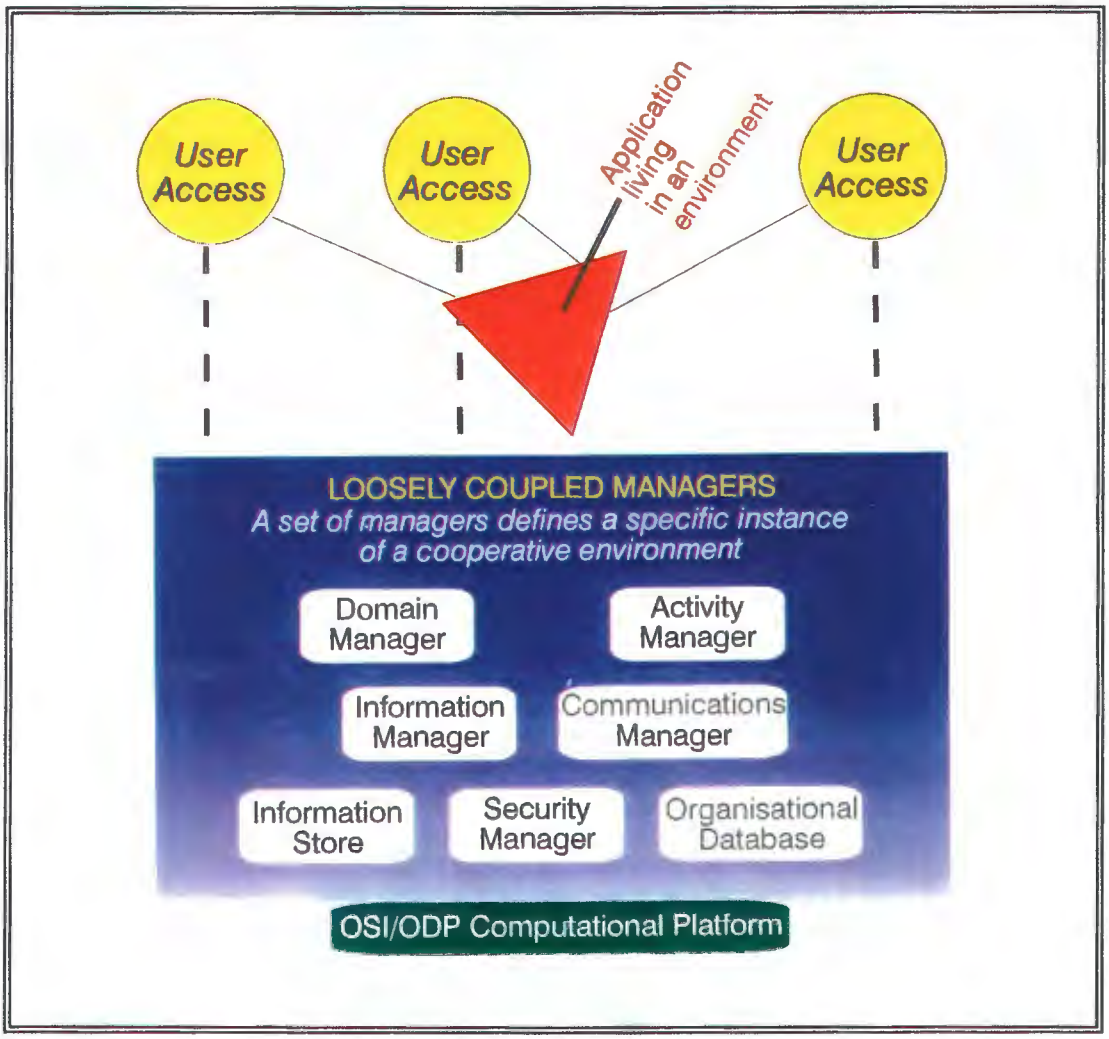

Figure 2.15 *Basic CSCW Architecture (Navarro et al.,1993)*

The *domain manager* provides the function of visibility. Objects are classified into domains to reduce the number of visible objects to those of interest. Domains are

also a unit of policy: different policies will be associated with different domains. Domains are also a unit of cooperation: cooperative activities take place in a domain (or a set of domains). Thus, domains reflect how the cooperative space is structured. Network support would, for example be included in the domain manager.

The *activity manager* allows activities generated or supported within an application to be registered outside the application in the CSCW environment. Other applications interested in a specific activity can also register their interest in the status of the activity with the activity manager. For example, an application may wish to announce that it is involved in the development of a production plan for a bridge. To do so an application would make an activity entry in the activity manager. This entry may describe the purpose of the activity, the state of the activity, the participants, and the resources allocated.

The *information store* provides common storage services to the objects in the CSCW environment. Information elements, structures, rules and complex objects should be stored. Requirements for concurrent access and quality (accuracy, reliability, consistency, availability, persistence) will be diverse, ranging from critical long-term data to short-term state information data. A key function of the information store is to enable inter-operability among different objects by sharing a common service access point and information representation. The service will be used by the managers mentioned as well as the application related objects.

The *organisational database* contains information about the organisation in which the environment resides. It represents an organisation in terms of the *resources* used within it, the *employees* working for it, the *roles* they play, *organisational relationships*, and the *organisational procedures and policy* exploited within the organisation. The information will often map onto underlying structures and services such as those provided by X.500. Finally, the organisational database allows organisational roles and policy to be stored and accessed by a number of

different cooperative systems.

An important concern in inter-organisational distributed cooperative environments is security. The *security manager* groups all functions related to security. Security is concerned with ensuring that the resources of the system are available to those who are allowed to use and/or access them, and are not available to others. Organisations are responsible for describing a security policy which allows measures to be taken against possible threats to the security of the distributed system (unauthorised disclosure, contamination, misuse of resources, denial of service).

An *information manager* controls the acquisition, processing, transfer and storage of multimedia objects such as text, graphics, image, voice, audio, and video.

The *communications manager* controls the group communications which by nature takes place among geographically distributed users. Communications over the networks are crucial, and the transfer of multimedia documents and data control are important issues. The functions of the communications manager include multicasting, appropriate bit rates for each type of media, synchronisation and time control, and the use of protocols providing a variety of features to meet expected multimedia services.

2.7 Main Components of Computer Supported Cooperative Work

CSCW research and reference material contain a bewildering array of diverse technologies, applications and concerns. According to Wilson (1991), there are two major concerns: the support of human groups, and the technology which can be used for that purpose. Of particular importance, is the dimensions of the CSCW field (Olson et al., 1993).

2.7.1 The Support of Groups

Major topics associated with the support of groups (Wilson, 1991) include:

- *Individual human characteristics* such as conversation patterns and the making of commitments.
- *Organisational aspects* such as the culture and structure of organisations.
- *Group work design issues* such as user involvement in the work design process, rapid prototyping and usability testing.
- *Group work dynamics aspects* such as group decision making and the cooperative or collaboration process.

2.7.2 Categories of CSCW Technology

Classes of CSCW systems were previously mentioned in Section 2.5.1., but the intention here is to identify the main categories of technology for supporting group work (Wilson, 1991), which include:

- *Communication mechanisms* enabling people at different locations to see, hear, and send messages to each other, for example electronic mail and video conferencing.
- *Shared work space facilities* enabling people to view and work on the same electronic space at the same time. An example is remote screen sharing.
- *Shared information facilities* enabling people to view and work on a shared set of information, for example multi-user databases.
- *Group activity support facilities* to augment specific group work processes, for example the co-authoring of documents, and idea generation.

2.7.3 Dimensions of the CSCW Field

The dimensions of the CSCW field should be considered to obtain the total

picture of what CSCW entails. This would be vital in working towards identifying the meta primitives of CSCW. The group work situation, the players and their tasks, and the technology and media used for supporting them should be characterised (Olson et al., 1993).

(i) The global characterisation of the group work situation

CSCW covers a wide spectrum of situations in which groups perform their work. Johansen (1988) characterised the different sets of situations by separating the dimensions of time and location of work, noting whether they are the same or different, as shown in Figure 2.16 (an instantiation of Figure 2.1).

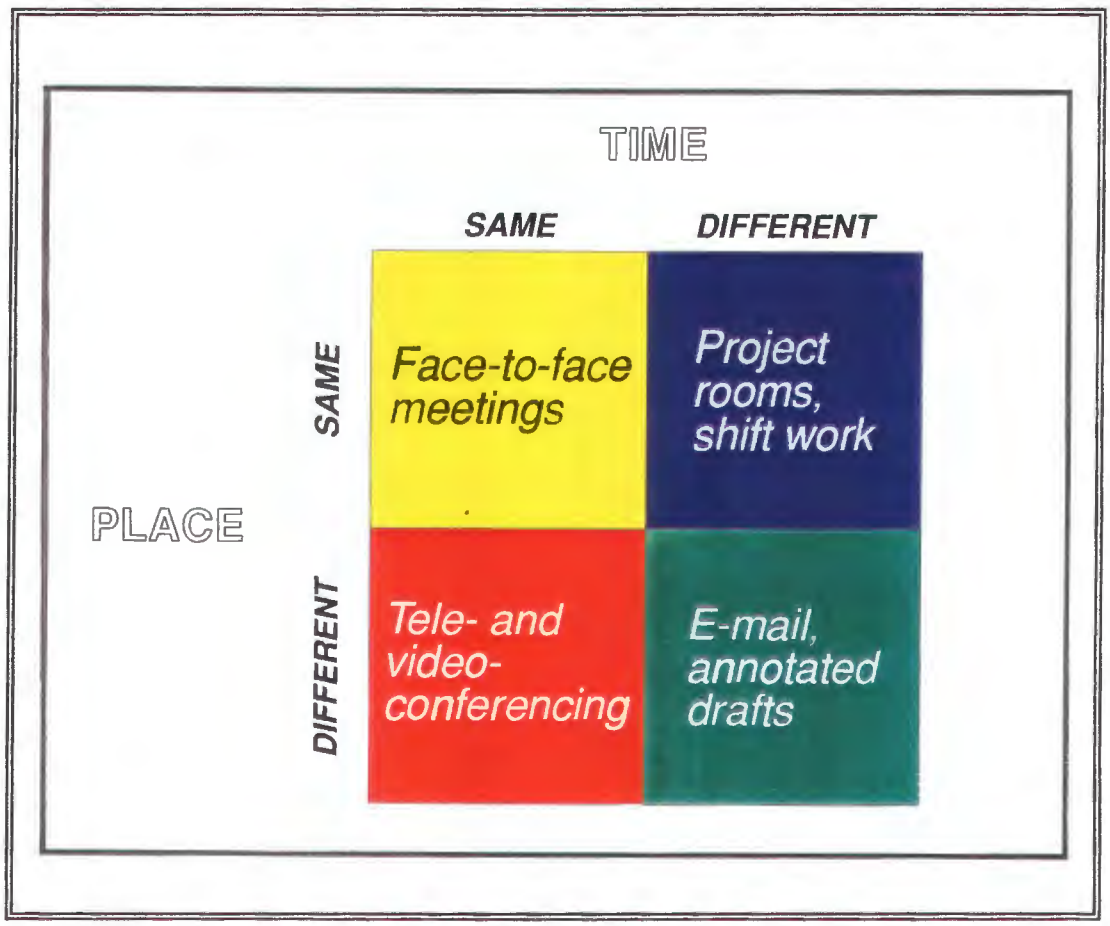

Figure 2.16 *Time and Location of Work (Olson, 1993)*

In the case of face-to-face meetings and informal project work, work may be done in the *same place at the same time*. Electronic meeting rooms such as *Colab* are intended to support this type of work (Olson et al., 1993). Group work also occurs in *different places but at the same time*. Systems in this category focus on remote work and attempt to help individuals communicate effectively with a different set of channels and tools than those used in traditional face-to-face work. Video and audio teleconferencing systems fall in this category. The third common situation is work that is *neither in same place nor same time*. The situation is supported through email, conferencing, and group authoring tools. In addition to the transfer of work objects and comments, this kind of work requires the overhead of coordinating people. The fourth situation, work that takes place in the *same place but at different times*, is less common. It is seen in shift work in hospitals and factories, and in project rooms, places where all the material for a project resides, but individuals in the team come and go.

This characterisation helps separate the major modes of group work. With in-room technology, which supports each person's ability to jot down ideas as they occur and can display one or more person's work, two changes are apparent. Work gets done in the meeting. There also is a smooth swing from silent, parallel thought and development of ideas, to a focussed, one-at-a-time viewing of each person's ideas which may be shared. According to Olson et al (1993), technology thus has the power to blend synchronous and asynchronous work in new ways.

(ii) The tasks of groups

Social psychologists have investigated aspects concerning the nature of group work for a number of years, and a taxonomy of kinds of group work has emerged. McGrath (1984) has categorised eight types of work, as shown in Table 2.3. In the original formulation, the list was drawn in a circle with adjacent type sharing some features in common. The result is referred to as the task circumplex.

Table 2.3 *Task types (McGrath, 1984)*

Planning tasks	(problem solving, generating plans)
Creative tasks	(generating ideas)
Intellectual tasks	(solving problems with a correct answer)
Decision-making tasks	(solving for preference)
Cognitive conflict tasks	(conflict of view)
Mixed-motive tasks	(conflicts of motive/interest)
Contests/battles	(conflicts of power)
Performances	(psychomotor tasks)

According to Olson et al. (1993), the taxomony specifies whether the work involves cooperation or conflict, whether it involves conceptual work or motoric actions (as in sport), and which phase of development the work is in, whether the work involves generating, choosing, negotiating, or executing. This is not the only task typology possible. Steiner (1976) has examined in more detail how tasks require information from group members to be combined in order to identify how technology might support activity. However, neither of the typologies consider the kinds of tasks groups engage in when they are trying to learn, to coordinate, or to get to know each other better.

It is important for researchers in CSCW to adopt some categorisation. In understanding the details of the tasks people are engaged in, the difficulties and obstacles they encounter in achieving their goals, and the techniques that they successfully use, better systems may be developed to support the work.

(iii) The technology and media

A variety of media have been used to support group work, and each medium presents different aspects of interaction. The primary differences have to do with whether the technology supports communication about the work, or whether it represents the work itself. Video connectivity supports conversation, including gestures and, sometimes eye contact, as well as artifacts (e.g., a model of a new

landscape). On the other hand, a real-time shared editor supports the work itself, providing to all participants text, outlines, and diagrams for both viewing and changing. Bulletin boards and email blend these roles, supporting both the work and the conversation about the work. This may cause problems in understanding what a particular message means.

It is difficult to dimensionalise the space of CSCW technologies, and attempts are just beginning to appear (Ellis et al., 1991; Malone and Crowston, 1990; Olson et al., 1990). These efforts are still preliminary, and there is no widely accepted framework. It is, however, a very promising area of work, moving beyond the early point systems of technology towards general theories of cooperation and coordination that are needed for understanding how technology may fit into human social, organisational, and cultural practices. This is the key to developing effective tools in the future. Some researchers have characterised the aspects of media that affect communication. Firstly, the features of communication that help people to understand each other, called "grounding", are identified. These features are co-presence, visibility, audibility, cotemporality (receipt as it is produced), simultaneity (ability to send and receive simultaneously), sequentiality, reviewability, and revisability. In Table 2.4 various technology media are evaluated for support of these features. The features may assist or disrupt a person's ability to understand others and to move work forward. The analysis is the foundation for evaluating the effects of degradation that technology sometimes causes on the quality of work that the group delivers.

(iv) The group members

Besides the considerations associated with the support of groups mentioned in Section 2.7.1., groups differ in both the characteristics of the participating individuals and how they interact with each other, and in their style and pattern of management. How can groups be characterised in order to both discover appropriate technology support and generalise the findings of the success of one

Table 2.4 *Seven media and their associated constraints*

MEDIUM	CONSTRAINTS
Face-to-face	Copresence, visibility, audibility, cotemporality, simultaneity, sequentiality
Telephone	Audibility, cotemporality, simultaneity, sequentiality
Video teleconference	Visibility, audibility, cotemporality, simultaneity, sequentiality
Terminal teleconference	Cotemporality, sequentiality, reviewability
Answering machines	Audibility, reviewability
Electronic mail	Reviewability, revisability
Letters	Reviewability, revisability

kind of technology from one group to the next?

The individuals who comprise the group have various expertise and talents, attitudes, and personality characteristics. Equally important are features of the individuals' interactions with each other: how long they have known and worked with each other and what process they have adopted to manage themselves. These two factors are known as cohesiveness and structure, and both of these vary as a function of the size of the group.

Old studies on group interaction and the proper division and exercise of leadership and participation roles, which were done before the availability of computer-mediated communication should be renewed and exploited as sources and suggestions for identifying needed functionality for CSCW (Olson et al., 1993). It is important to consider how the characteristics of individuals and group structure relate to the embedded structure in technology. A group with a pattern of democratic, cohesive, but free-for-all behaviour in unsupported work settings

might react poorly to a technology that has embedded in it an autocratic method.

(v) Integrating the dimensions

The above considerations are just a first attempt at understanding the dimensions on which computer supported cooperative work varies. Determining what type of group technology will be successful depends on specifying the four dimensions. The relationships that hold among them are the core of future research issues. The components and dimensions introduced up to now will be used in the next section to specify the CSCW meta primitives.

2.8 CSCW Meta Primitives

The purpose of this section is to identify the CSCW meta primitives and to construct a meta model based on these primitives. The word *meta* refers to a very high level of abstraction, and in terms of perspective it refers to a global or general perspective. A model may help to establish a common frame of reference showing explicit structure, and may serve as a communication means when dealing with complex problems. *Meta models* are models of modelling approaches (Joosten et al., 1993; Du Plessis, 1992).

Before constructing an object meta model of the CSCW meta primitives, two important examples of meta models that have relevance in CSCW will be illustrated. Firstly, the group process will be modelled in Figure 2.17. The second example models the concept of activity in CSCW as shown in Figure 2.20 and Figure 2.21.

According to Joosten et al. (1993), one of the reasons that many CSCW projects fail is that technology is developed with little consideration for the problems in the group process that are to be solved. In the literature it is hard to find models of working groups that are based on solid research. One validated framework

proposed by Gladstein (1984) is given in Figure 2.17, and may be considered to be a meta model for the group process. The purpose of the framework is to determine which variables are most predictive of group effectiveness. Each box in Figure 2.17 represents a set of variables that have been correlated to find predictive relations. The discussion of the full detail of Gladstein's framework is beyond the scope of this dissertation. However, it is assumed that the seven sets of variables depicted in the figure clearly describe an intended contribution to group work.

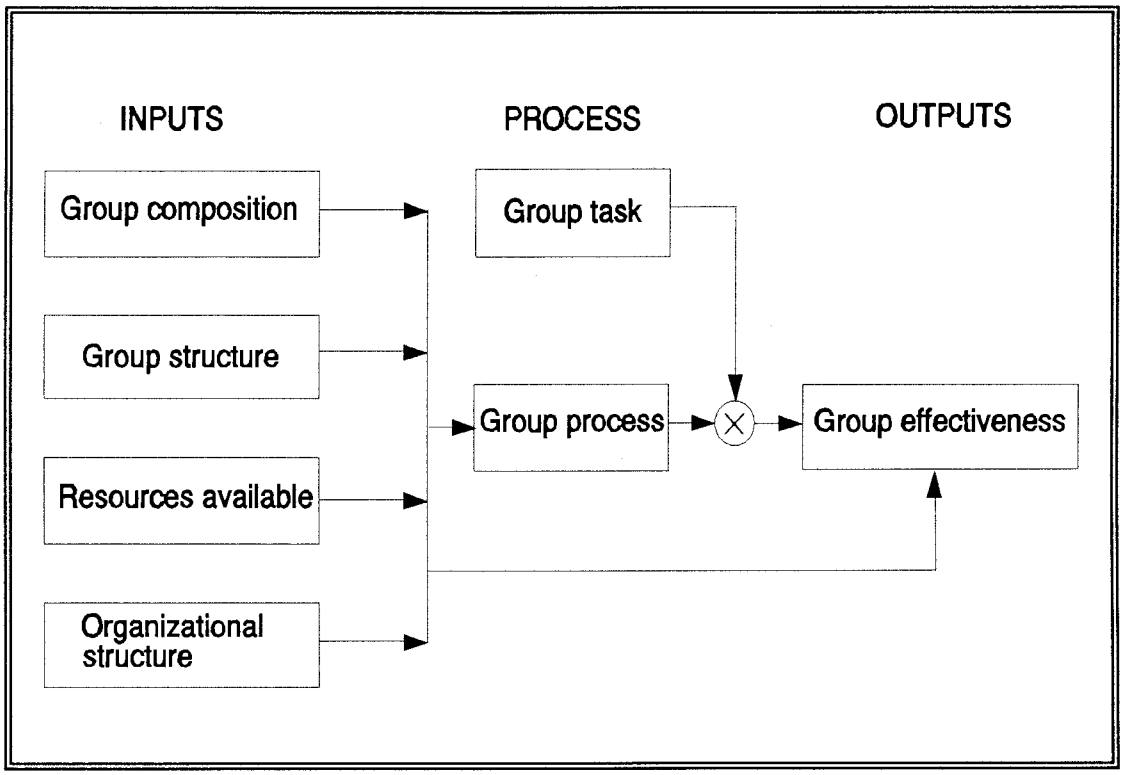

Figure 2.17 *Gladstein's framework for group performance (Gladstein, 1984)*

In Gladstein's framework, an organigram describes organisational structure. Organigrams are modelling techniques to illustrate the hierarchical structure of an organisation. Figure 2.18 is an example organigram. There are three graphical notation elements in an organigram: a rectangle, a diamond, and forks that connect them. Rectangles and diamonds describe groups of people in an organisation. Rectangles represent line-departments (involved in primary process

of an organisation), while diamonds represent staff-departments (secondary support for business processes in line-departments). The tree structure, as illustrated in Figure 2.18 describes the syntax of an organisation, while the semantics are described in the form of a tuple (L,S). L and S are finite disjoint sets of line-departments and staff-departments.

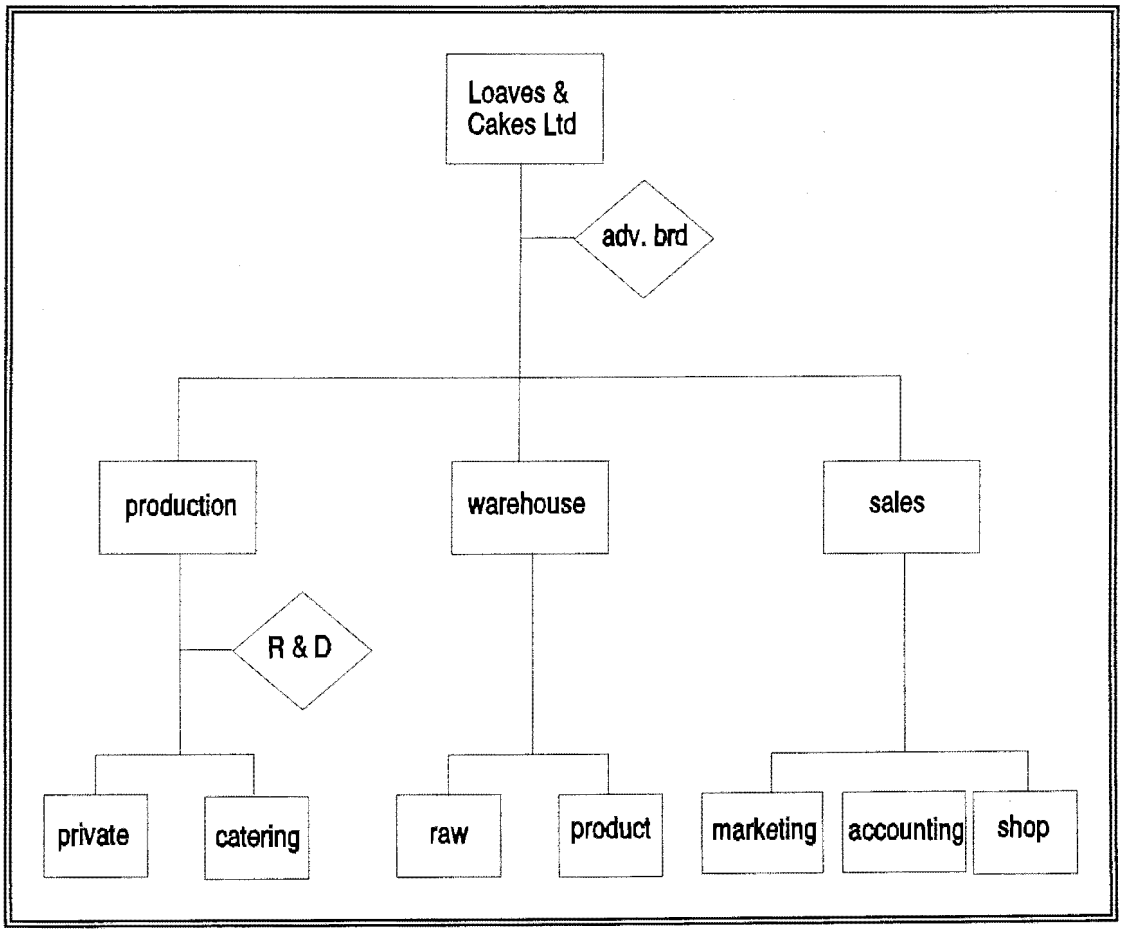

Figure 2.18 *Example of an Organigram (Gladstein, 1984)*

In the framework of Gladstein, process interaction diagrams (PID) describe the group process. Process interaction diagrams are another example of modelling techniques for group processes and are useful in describing workflow processes. Figure 2.19 is an illustrative example of a process interaction diagram.

Two graphical notations are used: circles and arrows. A circle represents a

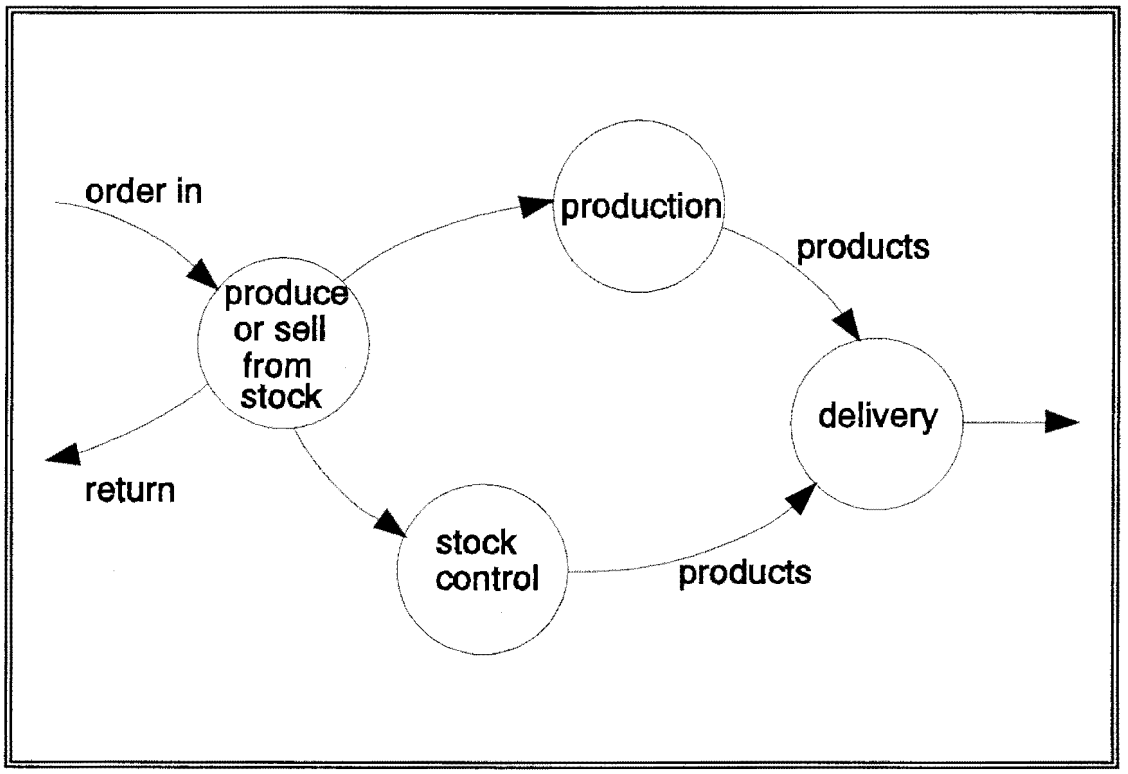

Figure 2.19 *Example of a Process Interaction Diagram (Joosten, 1993)*

process and an arrow represents an interaction (a stream of events). The meaning of an interaction should be seen in the context of time, and the processes are considered to work in parallel. The syntax and semantics of the PID model can be described in the form of mathematical set theory.

Other models, such as the logistic model and Petri nets (Joosten, 1993) are also useful for illustrating CSCW concepts.

A second model suggests that the concept of *activity* from Activity Theory might be useful for defining the basic research unit in the CSCW area. Broadly defined, activity theory is a philosophical framework for studying different forms of human praxis as development processes, with both individual and social levels interlinked (Kuutti, 1991). The solution offered by activity theory is that there is a need for a concept to be a minimal meaningful context for individual actions. The activities in which humans participate are the basic units of development and

human life, and thus form a foundation for the study of all contextuality. Activities have the following properties (Kuutti, 1991):

- An activity has a *material object.* Activities can be distinguished according to their objects. The transformation of an object towards some desired state or in some direction motivates the existence of an activity.
- An activity is a *collective phenomenon*, a collection of individual actions.
- An activity has an active *subject*, who understands the motive of the activity. The subject may be individual or collective.
- An activity exists in a *material environment* and transforms it.
- An activity is an *historically developing* phenomenon.
- *Contradictions* are the force behind the development of an activity.
- An activity is recognised through conscious and purposeful *actions* by participants.
- The relationships within an activity are *culturally mediated.*

Engeström (1987) has made an attempt to establish a simple structural model of the concept of activity and culturally mediated relationships within it, as illustrated in Figure 2.20.

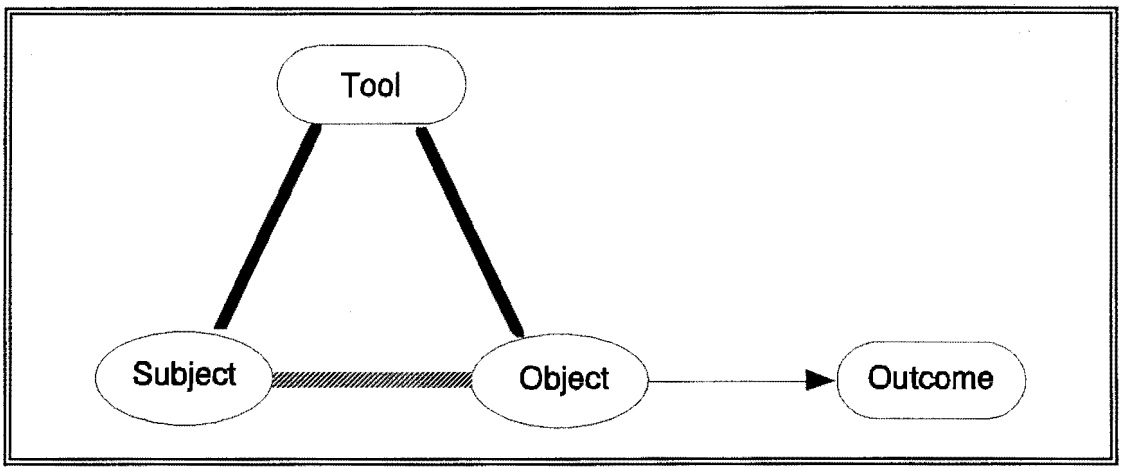

Figure 2.20 *Structure of an individual, mediated Action in an Activity (Engeström, 1991)*

Cultural mediation is dealt with by replacing binary relationships with mediated relationships. The model illustrates that the central relationship between the *subject* and the *object* of an activity is mediated by a *tool* into which the historical development of the relationship between the subject and object is condensed. However, the simple structure is not adequate for consideration of the systemic relations between an individual and his environment. Thus, a third main component, namely *community* (those who share the same object or activity) is added. Two new relationships are formed: subject-community and community-object. Figure 2.21 represents the basic structure of an activity and may be considered as an activity meta model. In this model, the systemic model contains three mutual relationships between subject, object and community. The relationship between subject and object is mediated by *tools*, that between subject and community is mediated by *rules*, and that between object and community is mediated by the *division of labour.*

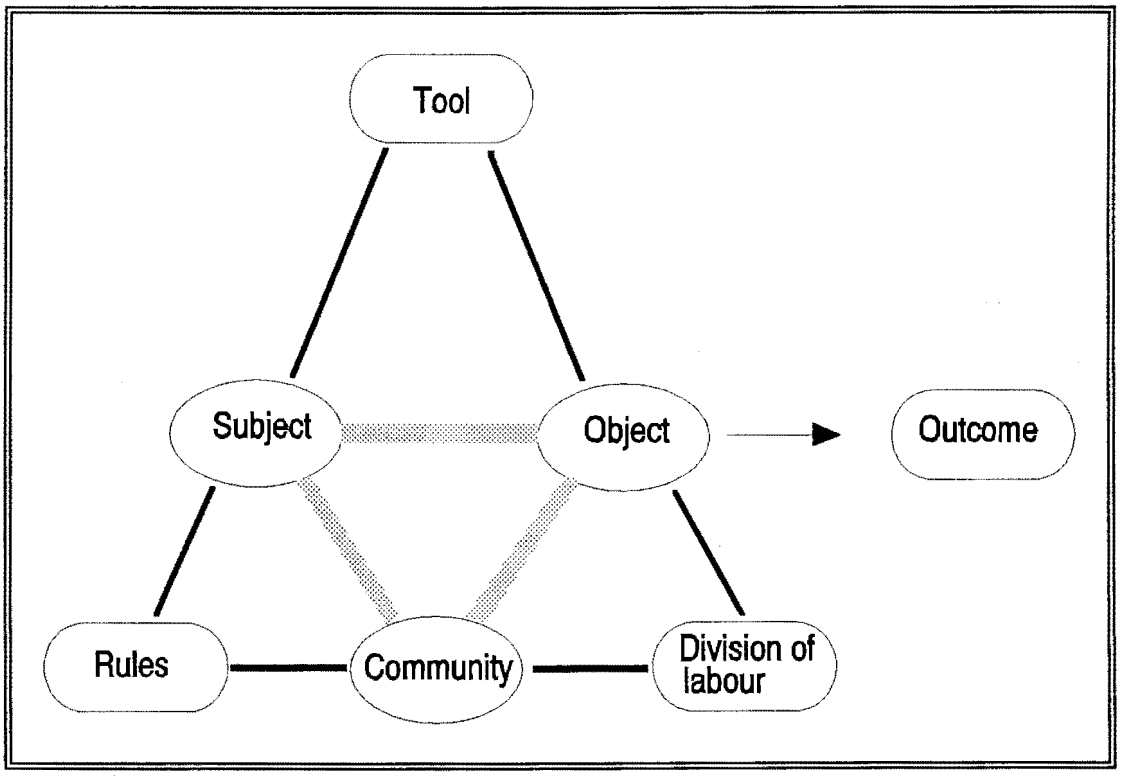

Figure 2.21 *Basic Structure of an Activity (Kuuttii, 1991)*

How well does the concept of work activity suit the needs of CSCW research in analysing work settings? The concept of work activity fits exactly with the definition of the work to be supported and may span from the individual to organisation-wide level, and even broader. It is possible to study formal organisational units as activities to the extent that a community of active objects sharing the same object can be found. In most traditional hierarchical organisations only the managers of the organisational units are active subjects, thus the activities found at the traditional organisation level are mostly managerial ones. However, the concept of activity makes it easy to cross any departmental, organisational or geographical border, but only inclusion of active subjects sharing an object is relevant.

The concept of work activity also suits the additional needs which should be fulfilled by the basic unit of research as mentioned in Section 2.4, the definition of CSCW:

- Mediation of work by artifacts is a fundamental feature of work activities. The concept of a mediating artifact, such as a tool or instrument, is rich and also includes signs, symbols, models and theories.

- The need for the existence of socially constructed meanings and cultural aspects is supported by the concept of work activity. Mechanisms allow cultural features to be brought into every activity by the corresponding artifacts. Apart from the tool artifact immediately used in transforming the work object, there are two other groups of socially constructed artifacts, namely rules and division of labour.

- Work reconstruction and the associated transformation and development issues are supported, because the concept of an activity has been applied to the study of developmental processes. The reconstruction of various artifacts is a basic feature of activities.

- The concept of work activity has a rich internal structure to help structuring the work settings to be studied.

- The concept of activity considers the issues of control and conflict. The concept of activity contains two different channels of control: hierarchical power structures embedded in the division of labour, and control through norms and values embedded in rules. There is also a mechanism to deal with conflicts. The conflicts are regarded as surface symptoms of contradictions. Conflicts influence the development dynamics of activities.

The meta primitives of the CSCW paradigm are now derived from the definition of CSCW, the main components and dimensions of CSCW, and the models illustrated in this section. The primitives constitute the basic modelling elements of a specific paradigm.

By virtue of the first part of its name, the "CS" part of the name CSCW, the professed objective of CSCW is to *support via computers* a specific category of work, namely cooperative work. The term *computer support* conveys a commitment to focus on the actual needs and requirements of people engaged in cooperative work. In analysing the word *computer support*, the attention is immediately focussed on support in the form of *hardware* and *software*. In the context of CSCW, the computer support is achieved by means of hardware which typically consists of *basic hardware* components, as well as *groupware technology*. Basic hardware components include network facilities, processors, peripherals such as printers and terminals, and other hardware components which may be necessary to provide basic computer support to anyone who uses a computer for work purposes. Architectural components which form part of the basic hardware include the information store, organisational database, and the information and communication servers. Groupware technology, on the other hand, includes hardware technology that should be added to the basic hardware components for the support of people who work in groups to attain specific goals in their work.

Groupware technology includes video conferencing capabilities (e.g. video and audio equipment), multimedia components, electronic whiteboard components, and other CSCW technology as discussed in Section 2.5.1. The architectural components which may be added are the CSCW managers (information, domain, activity, security, and multimedia managers).

Software may be classified as *system software*, *application software*, *middleware*, and *groupware* in the CSCW realm. The system software includes the operating system software, utility software, and other system software that are necessary for a computer to function. Customised software systems and software packages which are used for specific applications are referred to as application software. The general functions performed in the CSCW environment are referred to as services. The principle is to provide as many such services as possible via sharable servers. Collectively, the systems providing these services are called *middleware*. The most important services include (Brodie and Ceri, 1992):

- Data/object/knowledge/information managers (DBMS, OODBMS, KBMS, file systems, distributed object management).
- Presentation services/user environment (windows, forms).
- Communication infrastructures (peer-to-peer messaging, queued messaging, X-400, mail).
- Security services (authentication, encryption, access control).
- Reliability services (transaction manager, recovery manager, log manager).
- Advanced/distributed operating system services (resource allocation).
- Naming services (global name directory and management).
- Library services.
- Control services (job and request scheduling, brokering).
- Distributed computing/programming services.
- Interoperability services (information and language translation, data interchange, information/object migration, copy management, transparency services).

- Network services.
- General services (sorting, mathematics, data conversion).
- Repository services.

Groupware software refers to software that is applied for the support of group activity. The groupware provides various *shared services* for fulfilling the needs of people working in groups. As already mentioned in Section 2.7.2, the shared services are categorised as:

- Communication mechanisms.
- Shared workspace facilities.
- Shared information facilities.
- Group activity support facilities.

The meta primitives for the computer support aspect may now be derived. The previous discussion highlighted the important elements that are related to the computer support aspect, and it is therefore, logically assumed that they should form meta primitives. The computer support meta primitives are:

A. Software
    A.1 System software
    A.2 Application software
    A.3 Middleware
    A.4 Groupware
        A.4.1 Shared services
B. Hardware
    B.1 Basic hardware
    B.2 Groupware technology

Turning now to the second pair of characters in CSCW, "CW" or cooperative work, a specific category or aspect of human work with certain fundamental

characteristics common to all cooperative work arrangements is covered. There are many forms of cooperative work, and other terms used are collaborative work, collective work, and group work. Work is, of course, always *social* in the sense that the object and the subject, the ends and the means, the motives and the needs, the implements and the competencies, are socially mediated (Schmidt & Bannon, 1992). However, people engage in *cooperative work* when they are *mutually dependent* in their work and therefore are required to cooperate in order to get the work done. The term *cooperative work* should be taken as the general and neutral designation of multiple persons working together to produce a product or service. In terms of cooperative work, the *application domain* and the *participants* involved in group activity are important elements. The participants also fulfil different *roles* (e.g. manager, secretary, and chairman) in the group. For example, in the group situation when developing software a specific person may play the role of either domain specialist, project leader, analyst, or designer. The *level of work* element refers to the organisational level in which a participant performs cooperative work. The organisational level and job particulars are attributes of the level of work primitive. The highest level is the *universal level*, then the *worldly level*, and then the *atomic level*, the lowest level. The application domain element refers to the specific application supported by CSCW. The variety of application types include manufacturing applications, production control applications, broadcasting applications, and health care applications. The *nature of cooperative work* taking place within the application domain is also of importance. These include projects, meetings, committees, and task forces. Different *types of work* refers to either managerial, creative, physical or technical work. The specific work performed, the location of the work, and the temporal characteristics of the work are attributes of the type of work.

The meta primitives for the cooperative work aspect may now be derived. The previous discussion highlighted the important elements that are related to cooperative work and as before they will form the meta primitives. The cooperative work meta primitives are:

C. Participants
  C.1 Roles
  C.2 Level of work
D. Application domain
  D.1 Nature of cooperative work
    D.1.1 Types of work

An *object model* illustrating these meta primitives and their relationships is represented in Figure 2.22. An *object* is defined as a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand. An *object model* captures the static structure of a system by showing the objects in the system and their relationhips. Most object models, including those proposed by Rumbaugh et al. (1991), Booch (1994), Coad & Yourdon (1991), and Jacobson (1992), capture the attributes and operations that characterise each class of objects. For the purpose of this discussion, the attributes and operations are not important and are not incorporated in the CSCW object model.

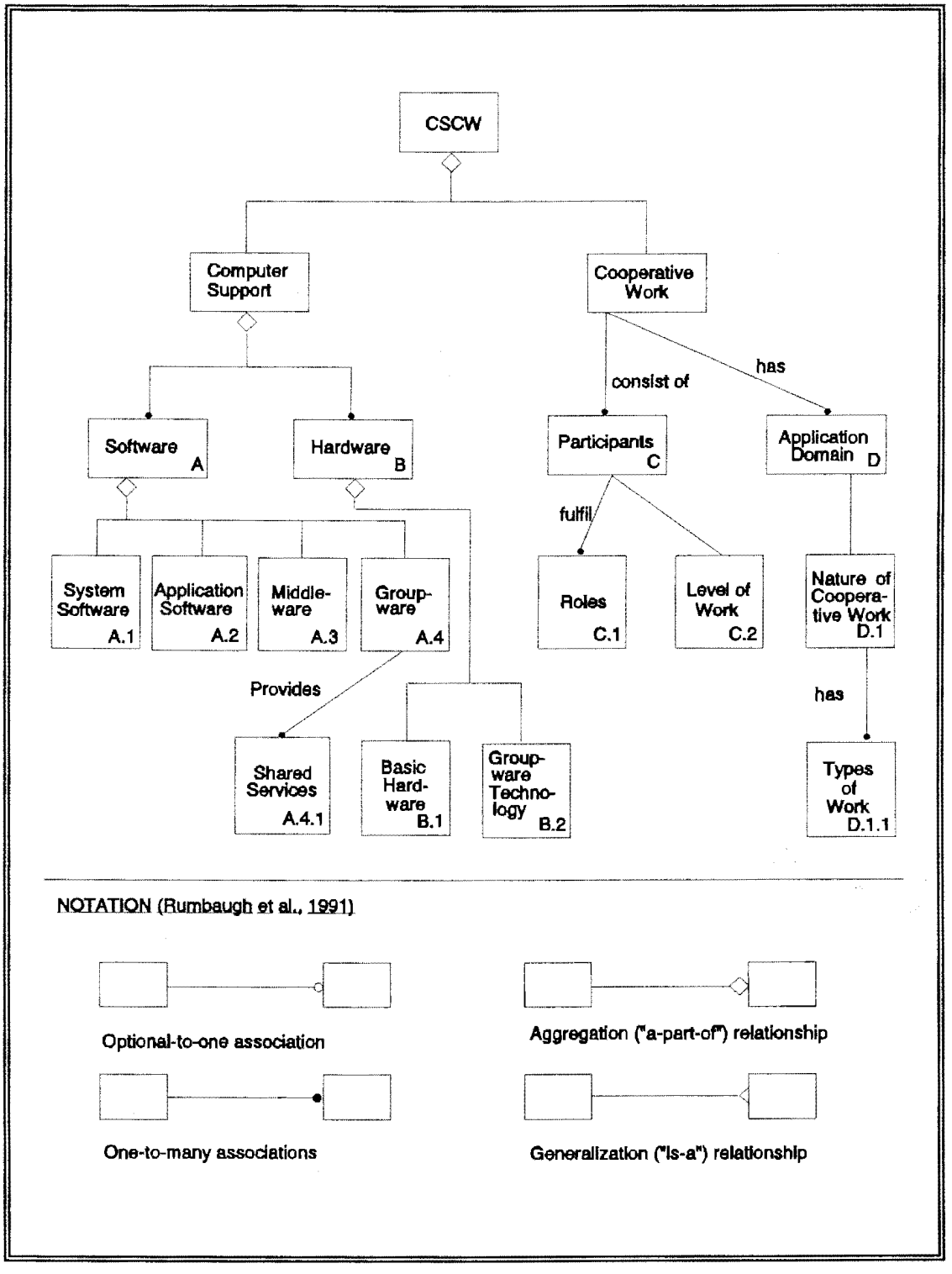

Figure 2.22 *The CSCW Object Model*

2.9 Summary and Conclusions

This chapter has outlined a number of important issues in the field of CSCW which come to the fore as a result of taking seriously the concept of cooperative work and its computer support. Initially, the nature of CSCW and some of the ongoing debates concerning it were examined. The main CSCW technologies were surveyed. At the very least, the field of CSCW has assisted in the process of re-examining a number of "fictions" concerning how people use tools and perform their work. It has, amongst other things, focused attention on how people work together in different settings, and the need for better integration across applications. Requirements that future systems for cooperative working will have to meet were examined. The meta primitives of the CSCW paradigm were derived from the main components and dimensions of CSCW. The general conceptual model as abstracted in Figure 1.1 of Chapter 1, can now be instantiated with the meta primitives of CSCW. This first instantiation of the conceptual model is illustrated in Figure 2.23.

While debate about exactly what is "new" in the field continue, there is no doubt that the area of CSCW has succeeded in attracting and holding an interesting interdisciplinary community over the last few years, from both technical and social disciplines. The task that lies ahead is whether these different interests and research traditions can be moulded together in order to produce software systems that truly support cooperative work. Having studied the nature of CSCW, the CSCW technologies, and determining the meta primitives of CSCW, one may conclude that the CSCW paradigm has potential to motivate the development of groupware systems that support the cooperative nature of software development.

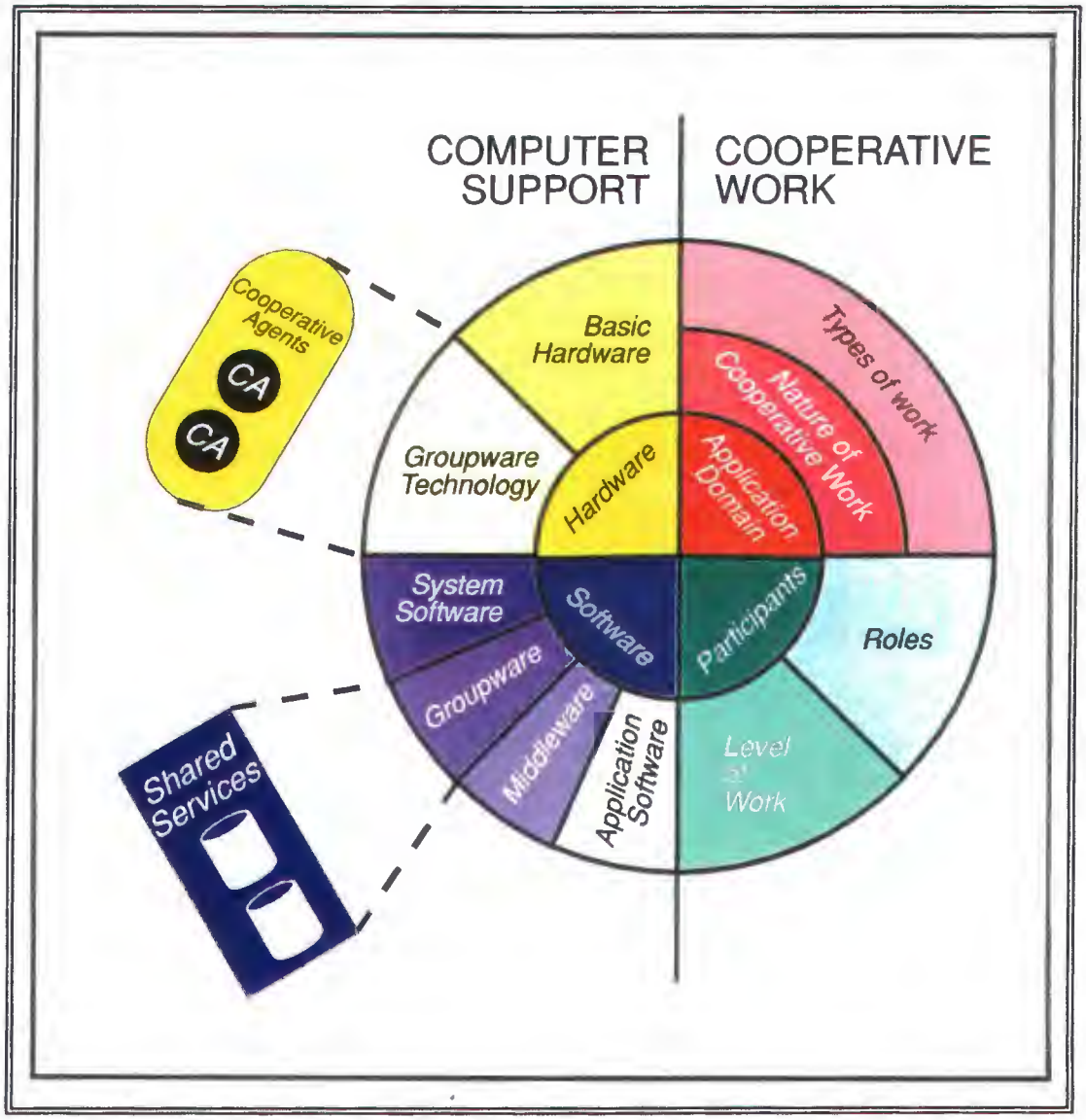

Figure 2.23 *Conceptual Model of CSCW Meta Primitives*

CHAPTER 3

Cooperative Aspects of the Software Development Process

CONTENTS

3.1. Introduction

Organisations around the world depend more and more on software in the very basics of their operations. Moreover, the quality of the software product and the quality of business service are increasingly linked. Software quality is dependent on the way the software development process is conducted. Software development, being notoriously difficult to manage, should be an engineering discipline. One of the characteristics of established engineering disciplines is that they embody a structured, methodological approach to developing and maintaining artifacts (McDermid & Rook, 1991). McDermid (1991) gives the following definition which may not cover all facets of software engineering, but captures the essential spirit and breadth of the notion of software engineering:

> *"Software engineering is the science and art of specifying, designing, implementing and evolving - with economy, timeliness and elegance - programs, documentation and operating procedures whereby computers can be made useful to man."*

The goal of software engineering is to produce a high quality software product and to follow a high quality development process (Conger, 1994). In addition to a quality product, quality of process is desirable. Closely related to software engineering principles, are the concepts of abstraction and structure for mastering the complexities of large-scale software systems.

In addition to the aim of developing software according to software engineering principles, other factors should also be taken into account when developing large-scale software systems. An information system is developed by means of an arrangement of people, activities, data, networks, and technology which form the building blocks of information systems. The building blocks are arranged and integrated to accomplish the purpose of supporting the day-to-day operations in a business, as well as the problem-solving and decision-making needs of business

managers. Software development is a people-intensive process during which developers and users collaborate to create a new information system. The crux for management is to deal with software development by a team. For technical activities, automated support has been provided in the form of tools which facilitate activities undertaken by a single designer or programmer. Integrated project support environments are particularly relevant from a managerial point of view as they provide a framework for dealing with teams, rather than just providing the functionality required by individuals. CSCW seems to have the right qualities for satisfying the need of management to deal with the group activities during software development.

The aim of this chapter is to address the cooperative aspects of the software development process in terms of group activity during the process, the roles of people in these groups, and existing computer-based tools to support the process. Different perspectives of the software development process are considered. These perspectives are of importance for investigating the possibility of CSCW during software development.

3.2 Abstraction and Structure in Software Development

Two key intellectual weapons for mastering the difficulties of large-scale software systems are *structuring* and *abstraction* (McDermid, 1991). Structuring enables the decomposition of systems into components, or views, and to understand the system a little at a time, yet still understand how the components or views relate one to another. Abstraction provides a way of drawing away from the details of the system and, again this aids comprehension. The major advance in software engineering over the last two decades is in automation - the provision of computer-based tools to assist the development of large-scale software systems. However, the tools are not effective if they are not based on appropriate abstractions and ways of structuring systems and their development process. Structuring and abstraction are key intellectual weapons providing ways of

handling complexity, and are relevant in the technical and managerial domains. This also has implications for CSCW systems. Abstraction and structuring are important in the computer-based support that should be provided to people involved in group activity. The CSCW systems should be structured to provide the relevant views of the system to specific group members, preventing confusion in the use of the system. The functions provided by the CSCW system should be abstracted to be similar to the groupwork activity that is performed without computer-based support. This is important, because in a survey performed by Bullen and Bennett (1990) it was found that only CSCW tools that corresponded to the non-electronic activities were used by people involved in group activity.

Management in software development primarily involves planning, decision making including allocation of resources, progress monitoring and estimation. All of these areas depend on having an appropriate model of the software development process and this, in turn, depends on having appropriate structuring and abstractions. The ability to provide appropriate abstractions and structuring for the software process is almost synonymous with the establishment of life cycle models (McDermid, 1991). By common usage, *life cycle models* (LCMs) are abstract descriptions of the structured development and modification process, typically showing the main stages in producing and maintaining executable software. The earliest *software development life cycle* (SDLC) model is the so-called Waterfall model which gave a simple abstraction of the software. The Waterfall model has a number of severe limitations, for example it does not adequately show iteration. There have been many attempts to enhance the Waterfall model, but in recent years the trend has been towards the development of much more sophisticated models. The main advance has been the realisation that it has been appropriate to structure, or model the software process differently from a managerial as opposed to a technical point of view. This resulted in separate, but related, abstractions dealing with the technical development activities including iteration, and the management processes which seek to control the technical development activities. Most notably, Boehm's

(1988) spiral model identifies different management activities such as planning, risk assessment and control. The ability to structure the software development process into phases and to structure the work through a breakdown structure, provides the basis for good project control. Software development methodologies complement the software development life cycle. They provide comprehensive and detailed support for the entire software development life cycle by:

- incorporating step-by-step tasks for each phase or cycle,
- specifying individual and *group* roles to be played in each task,
- prescribing quality standards and required deliverables for each task,
- providing development techniques to be used for each task, and
- technology in the form of tools to support the development process.

In the context of software cost estimation, the monitoring of progress in software projects is important, and in risk management appropriate abstractions should be used. While the notions of abstraction and structuring are important they do not address all management issues (McDermid, 1991). It is of paramount importance to have good communications within the project team and between the team members and the project manager. Similarly, managers should have good interpersonal skills. These do not really derive from any notion of abstraction and structuring. Communication and *cooperation* form the basis of a controllable project and the ability to control projects will advance by finding better forms of structuring for projects and better abstractions from the details of projects. This, of course, has implications for CSCW systems that are used to support the software development process. A strong management perspective should form part of these systems.

In the **technical** domain, most of the improvements in technology for developing large-scale software systems have depended on finding improved abstractions or improved structuring techniques for writing or specifying programs. Also of major significance here is *software development methodologies*. The so-called

structured techniques have evolved from their early, rather simple, beginnings into a large array of techniques and methods. These methods cover the design of databases (including the normalisation of data structures), the structured development of programs from high-level designs, and real-time systems development using models based on the notion of state machines. Abstraction is enhanced by the *object-oriented* approach. Hierarchical composition or decomposition enables users of the object-oriented approach to control the level of abstraction at which they model the system under development. Within the context of system and software specification the mathematically-based *formal methods* which, while being abstract, enable more facets of the software to be specified than is possible with structured techniques. The object-oriented approach has also greatly improved software specification. Two more trends worth noting are the trend to focus on the *earlier stages* of software development, e.g. requirements analysis and design, and the general belief that *tools* play an important role in software engineering. Computer-based tools have emerged which support teams working on a project rather than individuals (McDermid, 1991). *Groupware tools* and *CSCW systems*, such as integrated project management tools and message systems, to support certain group activities during software development have emerged, but a need exists for more sophisticated CSCW systems to support the software development process.

Models of the software development process on which this investigation is based, are discussed in the next section.

3.3 Software Process Models

A software process model can be defined as a descriptive representation of the software process. The software process is conducted according to a development framework or software process architecture, and includes the set of technical and management activities that are applied during the production of software (Du Plessis & Van der Walt, 1992). The software process model should represent

attributes of a range of particular software processes and be sufficiently specific to allow reasoning about them (Dowson & Wileden, 1985). The main function of a software process model, from a project management point of view, is to establish the order in which a project performs its main stages and to establish the transition criteria for proceeding from one main stage to another.

The *Object-oriented Information Systems Engineering Environment (OISEE)* project, within which this study is conducted, is formulated in terms of a general framework of reference models that structures the technological foundation of information systems engineering into separate concerns. The following reference models were proposed for the project (Du Plessis, 1992):

- The Development Process Reference Model
- The Quality Assurance Reference Model
- The Technology Reference Model
- The Target System Reference Model.

This study is mainly concerned with the *Development Process Reference Model*, but the possible incorporation of CSCW into the software development process may also concern the *Technology Reference Model.* This will only be determined in Chapter 4 of this study. The *Development Process Reference Model* addresses the information system development life cycle, according to the following aspects:

- The Management Aspect
- The Life Cycle Aspect
- The Methods Aspect.

Although these aspects were formulated in the original documentation about the OISEE project, the feasibility of incorporating a *cooperative aspect* will be investigated. Humphrey's (1989) three-level software process model is adopted for the *management aspect.* The model consists of three levels of abstraction, a

universal level, a worldly level, and an atomic level. The *universal level* presents a global view of a software project for senior management. Structure is given to this view by means of a software development life cycle (SDLC) framework which guides the project. On the *worldly level*, the sequence of development tasks and work steps of the cycles of the SDLC is prescribed. On the *atomic level*, the orderly and prescriptive manner of performing the tasks of the wordly level is defined in detail for junior management. The *DesignNet Model* is the representation scheme for presenting tasks or activities, deliverables, and status reporting on all levels of the three-level model (Du Plessis & Van der Walt, 1992). Information is conveyed regarding the schedule, the work-breakdown structure, manpower allocation, costing and current status of the project. All participants involved in the management of a project share information and may communicate across project levels. This, of course, has implications for CSCW systems supporting the management aspect of software development. Figure 3.1 illustrates the universal level of the development process model in DesignNet notation depicting the cycles of software development and the important deliverables. (The representation scheme is derived from AND/OR graphs and Petri nets).

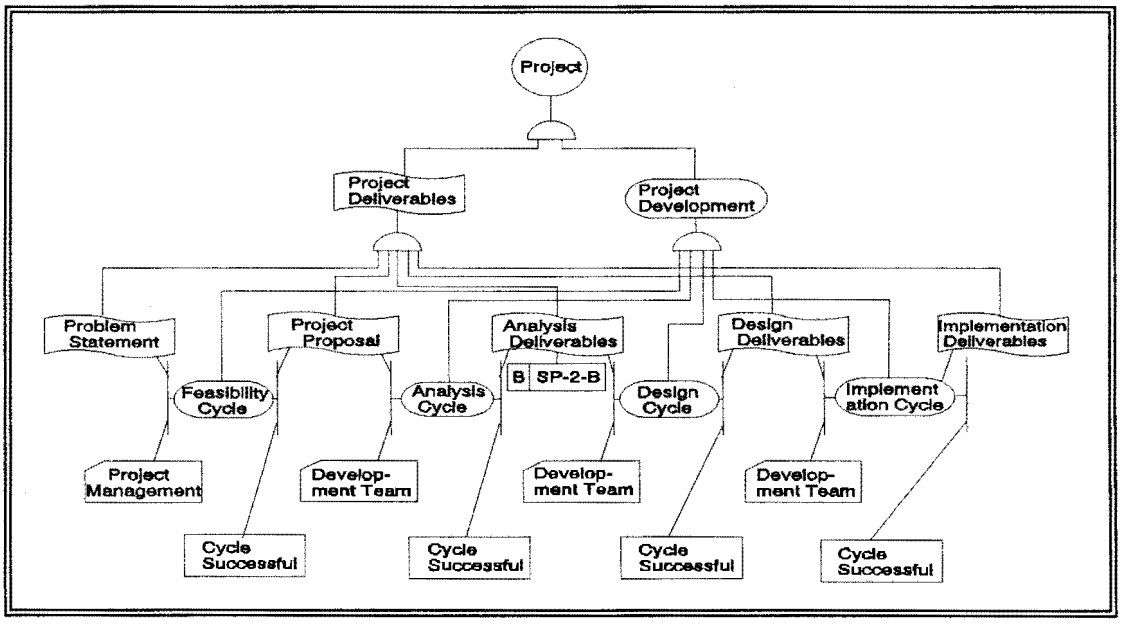

Figure 3.1 *Universal Level of the Development Process Model (Du Plessis & Van der Walt, 1992)*

Considering the *life-cycle aspect*, the spiral model (Boehm, 1986) was adopted for the OISEE project. This cyclic model is essentially a meta-model of the software development process and is used as a framework for Humprey's three-level model as illustrated in Figure 3.2.

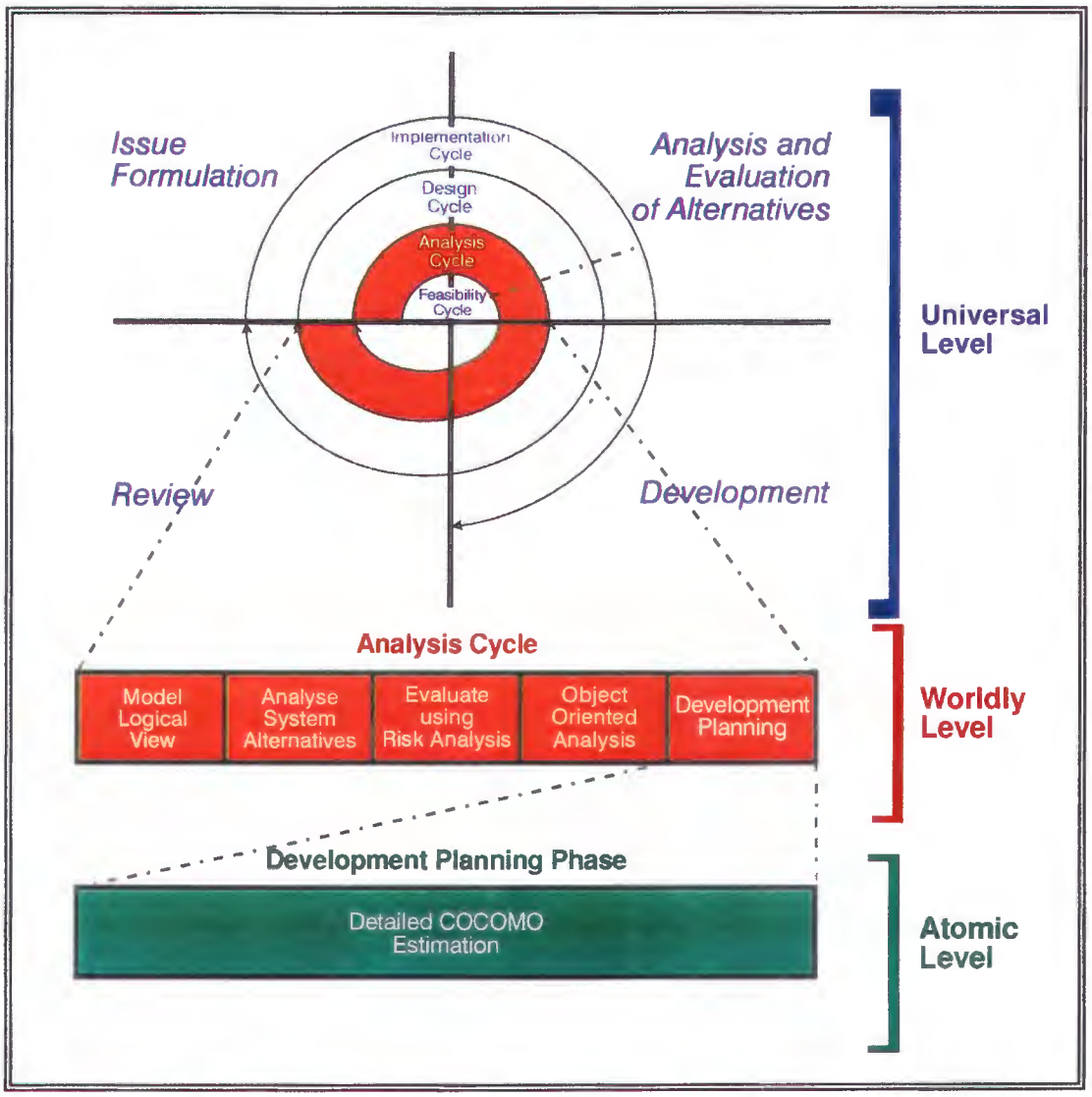

Figure 3.2 *The three levels of software development modeling (Du Plessis & Van der Walt, 1992)*

The radial dimension in Figure 3.2 represents the cumulative cost incurred in accomplishing the steps to date. The angular dimension represents the progress made in completing each cycle of the spiral. Following the commitment to go ahead, each cycle of the spiral begins in the top left-hand quadrant. The major

review to complete each full cycle is an important feature of the spiral model. It is shown as the left-hand axis in Figure 3.2, and involves all participants concerned with the product. The review covers all of the products developed during the previous cycle and the resources required to carry them out. The objective is to ensure that all concerned parties are mutually committed to the plan for the next cycle which may include a partition of the product into increments for successive development, or components to be developed by separate organisations, teams or individuals. Thus, the review and commitment step may range from an individual walkthrough of the design of a single programmer's component, to a major requirements review involving developer, user, customer and maintenance organisations. The most significant emphasis of the diagram of the spiral model is on decision making to ensure management of all aspects of risk (McDermid, 1991). Planning, decision making, and determining, evaluating and resolving alternatives for detailed plans are not a simple linear progression but must be iterated for risk management planning in each cycle as necessary. If the project gets the go-ahead after a risk evaluation, the next cycle of the spiral model is started. Otherwise, the development stops and an evaluation of the entire project is done.

The spiral model may be useful in the choice of any appropriate mixture of specification-oriented, (automatic) transformation-oriented, simulation-oriented, prototype-oriented, incremental, or other approach to software development (McDermid, 1991). The appropriate strategy is chosen by considering the relative magnitude of the program risks, and the relative effectiveness of the various techniques in resolving the risks. For the purpose of this investigation, *object-oriented development* is incorporated into the model, and therefore a revised spiral model as proposed by Du Plessis and Van der Walt (1992) is adopted. For the purposes of this investigation, the revised spiral model is extended to include group or cooperative activity, as shown in Figure 3.3. This is an initial step for exploring cooperative activities during the software development process.

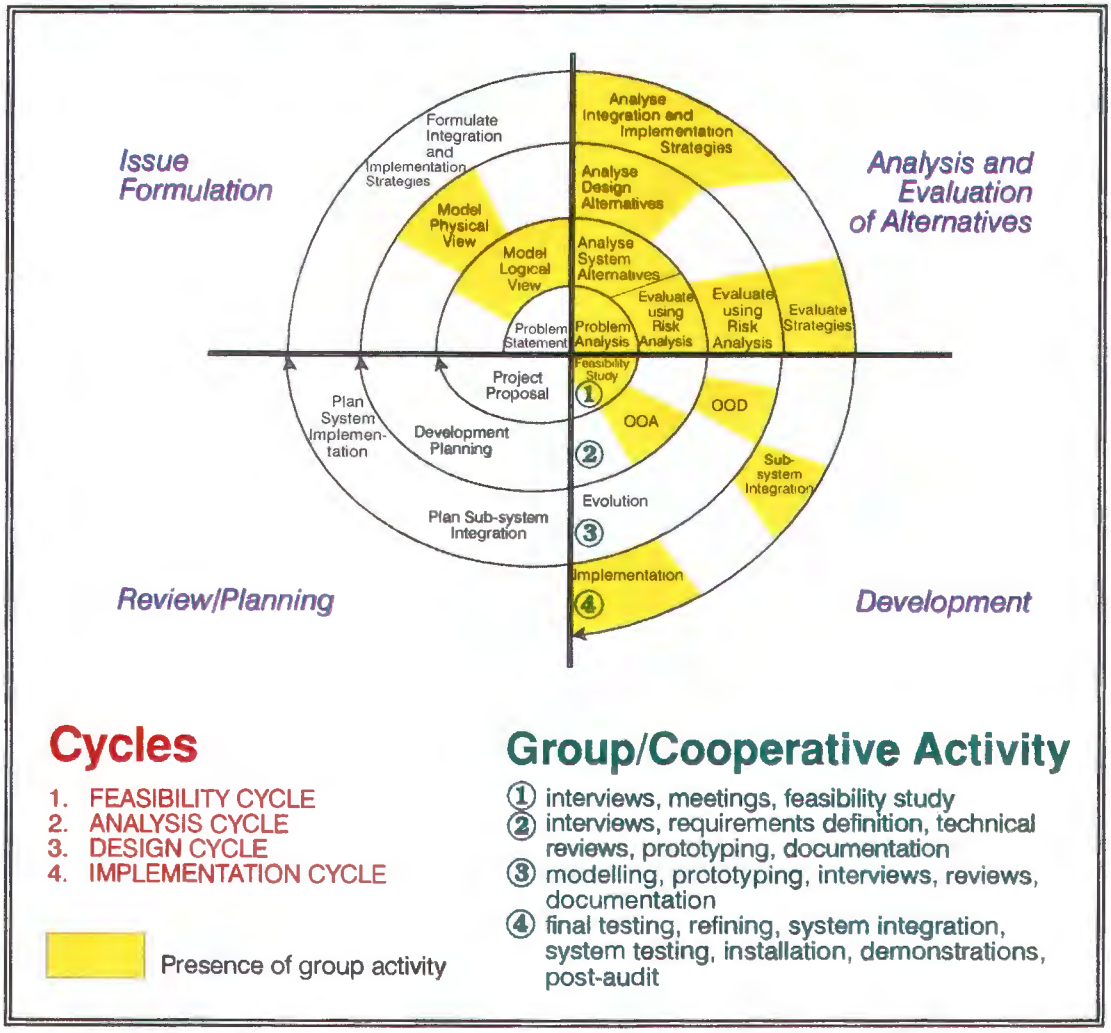

Figure 3.3 *Revised Spiral Model for Cooperative Object-oriented Development*

In the revised spiral model, the four cycles of software development are:

- *Cycle 1* - Feasibility.
- *Cycle 2* - Analysis.
- *Cycle 3* - Design.
- *Cycle 4* - Implementation.

As shown in the model in Figure 3.3., each cycle is characterised by four quadrants:

- Issue Formulation (determine objectives, alternatives, constraints).
- Analysis and Evaluation of Alternatives (identify, resolve risks).
- Development (develop, verify next-level product).
- Review / Planning (plan next phases).

The *methods aspect* is addressed choosing a set of object-oriented methods to guide the technical development process and the corresponding management tasks.

Considering the incorporation of the *cooperative aspect*, it is important to realise that software development is a group effort, involving participants with different responsibilities and skills. As already mentioned, the management, as well as the technical domains of software development need computer-based support for specific group activities. A CSCW system with the purpose of supporting group activity during software development in all its dimensions seems to be the solution. A framework proposing groupware technology for different group activities during software development will be proposed in Chapter 5.

The basic information systems building blocks will be discussed in the next section in order to establish a rationale for the roles that people play during software development and the technology support during the process.

3.4 Information System Building Blocks

A five-component model of an information system is presented, which is the basis of the systems development approach followed in this discussion. With this emphasis it is necessary to understand information systems in terms of five "building blocks" (Jordan & Machesky, 1990). The purpose of the discussion of

the building blocks, is to give greater insight into the nature of the software development process, and to motivate that the presence of these building blocks in software development as well as in CSCW, makes the use of CSCW systems during software development feasible.

The notion of building blocks have been emphasised by various authors, such as Jordan & Machesky (1990) and Whitten et al. (1994). The five building blocks are *people*, *data*, *activities*, *networks*, and *technology* (Whitten et al., 1994). The five component-model of an information system, adopted from a model devised by Jordan and Machesky (1990), is represented graphically in Figure 3.4. The models in following figures, which each represent a specific building block were derived from the five component-model for the purposes of this study. The colour-coding of the building blocks enhances the composition of each model. The components in Figure 3.4 are arranged symmetrically to denote their equal importance in the software development process. In the figure, it is clear that data forms the centre of the system and the left-hand components concern people and the two right-hand components concern computers. At this stage, it is interesting to note that people and computers are two important meta primitives of CSCW as already discussed in Chapter 2.

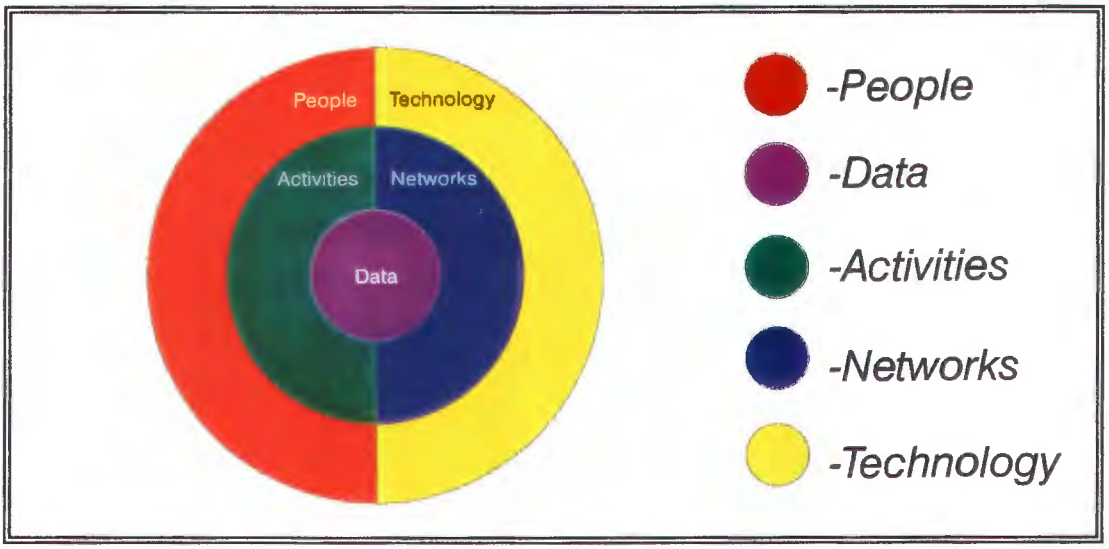

Figure 3.4 *The Five-Component Model of an Information System*

3.4.1 Building Block - PEOPLE

The first and most vital component of the system model is people. The overriding philosophy in the development of software should be that the system is for people by people. The people involved in an information system assume many different roles, but all the players in the information systems game share one thing in common, they are classified as information workers by the U.S. Department of Labor (Whitten et al., 1994). The term *information worker* is used for people whose jobs involve the creation, collection, processing, distribution, and use of information. The roles that people play can be classified into four categories: system owners, system users, system designers or developers, and system builders. The roles will be discussed in the context of the model in Figure 3.5 which is colour-coded according to the colours specified in Figure 3.4.

The *system owners* are also known as the information system's sponsors and chief advocates responsible for budgeting the money and time to develop and support the information system. They also make the final decision concerning the acceptance of the information system. From the model in Figure 3.5 it is clear

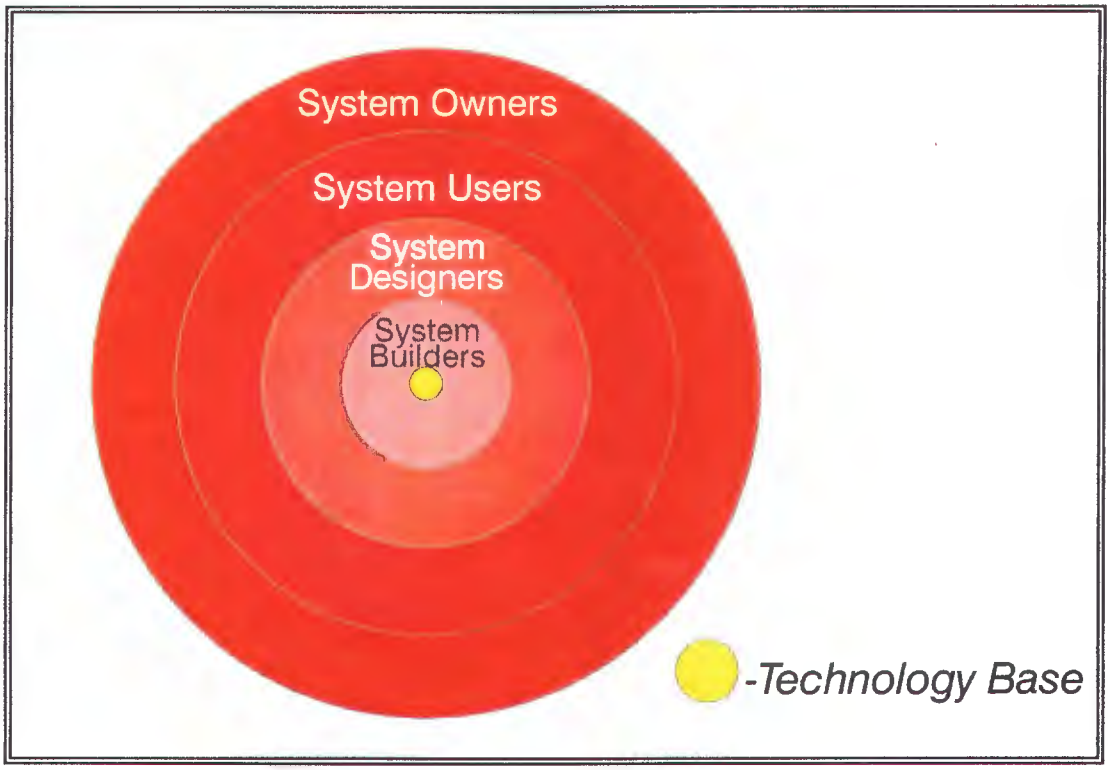

Figure 3.5 *The People Building Block of Information Systems*

that the system owners are the furthest removed from the technology base being least interested in the technical details of the system. They are, however, aware of the costs of technology which should be offset by equivalent benefits to the business.

The *system users* are the people who use and directly benefit from the information system. Their tasks include the capturing, entering, validating, responding to, and storing of data and information on a regular basis. At the start of a software development project it is the system users' responsibility to define:

- the problems to be solved
- the opportunities to be exploited
- the requirements to be fulfilled
- the constraints to be imposed by or for the information system.

The users play an important role in the design of the user interface, the display screens and the reports of the system. They are less concerned with costs and benefits than system owners and more concerned with the business requirements of the information system. From the model in Figure 3.5 it is clear that system users are still relatively removed from the technology base. It is, therefore, important that system developers (e.g. analysts) keep discussions with users at the business detail level as opposed to the technical detail level.

The next category in the role-oriented model in Figure 3.5 is *system designers*. They are alternatively called *systems analysts*. They translate the user's business requirements and constraints into technical (computer-based) solutions. This includes the design of inputs, outputs, screens, networks, programs, computer files, and databases. From the model it is clear that system designers move closer to the technology base because they should consider technology alternatives and design systems within the constraints of those choices. According to Whitten et al. (1994), the systems analyst is the principle system designer of most multi-user information systems. Designers also include other hardware and software specialists, such as database analysts, network analysts, microcomputer specialists who provide additional expertise to the software development process.

The *system builders* are responsible for the eventual construction of a multi-user information system which meets the requirements as set out in the design specifications. Although, the applications programmer is a classic example, other technical specialists such as database programmers, network administrators, and system programmers may be involved. From the model in Figure 3.5 it is clear that the system builders are located nearest to the technology base.

The roles and responsibilities of people involved in systems development, as perceived by Jordan and Machesky (1990) are summarised in Table 3.1.

In many software development efforts, the user is a member of the development

Table 3.1 *Information Systems Role Summary (Jordan & Machesky, 1990)*

ROLE	RESPONSIBILITY
User	Contributes to requirements determination, alternatives evaluation, and project management. Also may document procedures and train other users.
Programmer	Codes, tests, documents, and maintains software.
Analyst	Elicits help from users in order to determine requirements and evaluate alternatives; specifies design and oversees implementation.
End-user consultant	Advises users about any aspect of the development process and managing the process so that end users can successfully develop their own systems.
Project manager	Plans, monitors, coordinates, and manages the development process.
Trainer	Trains information systems staff or users on any aspect of the development process and managing the process; most frequent training is on development tools.
Database analyst	Advises users and information systems staff about technical and detailed data storage design issues, including maintaining standards for integrated systems. (Also called database administrator or database specialist.)
Fourth generation language consultant	Trains and advises users and information systems staff about fourth generation tools.
Communication specialist	Develops, coordinates, and maintains telecommunication systems accessed by many users.
Office automation specialist	Develops, coordinates, and maintains office automation systems; important responsibilities include evaluating alternatives and end user consulting.

team. This team approach to software development gains the user's commitment to the system. Placing the user on the development team to share development responsibilities with the analyst may speed up delivery of a system. Early user involvement improves the system's chances of success. The team composition

varies with the size and nature of the system.

3.4.2 Building Block - DATA

Data is the central component of an information system. Although, the terms *data* and *information* are used interchangeably, there is a distinction (Jordan & Machesky, 1990). Data is a collection of raw facts in isolation, while information is data that has been manipulated to be useful. The different people in the model of Figure 3.6 view data differently.

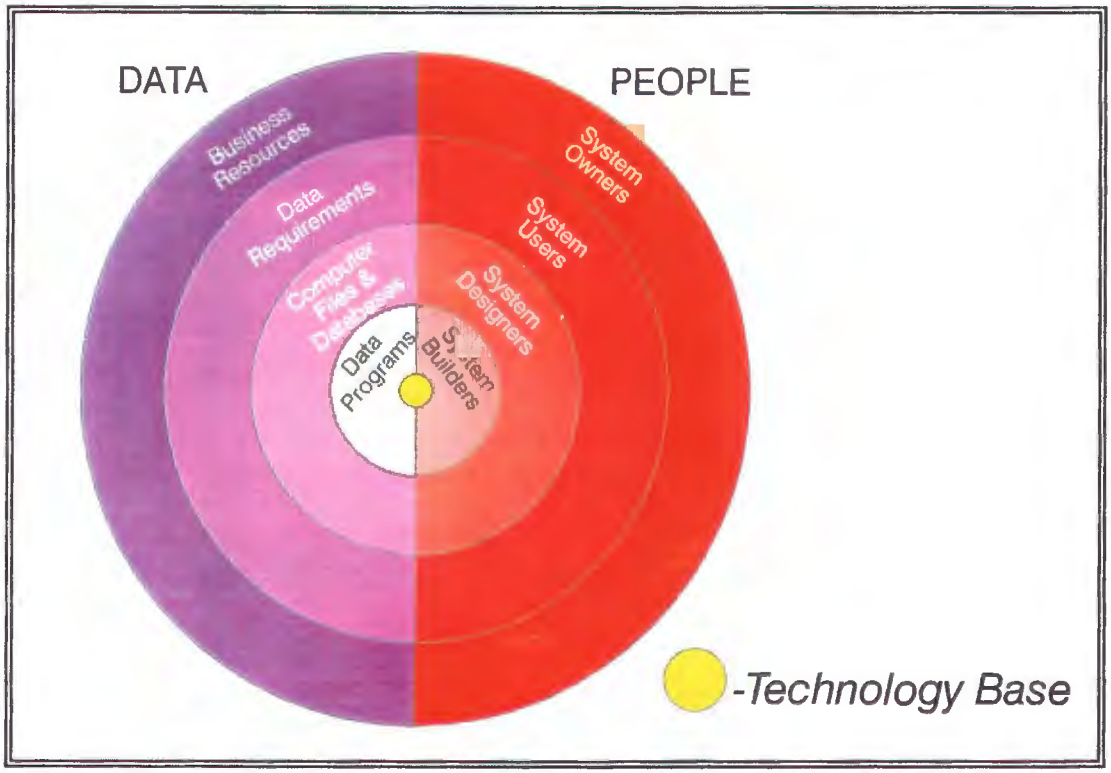

Figure 3.6 *The Data Buiding Block of Information Systems*

The system owner views data as business resources. These are things that are essential to the system's purpose or goal, or things that must be controlled in order to achieve business objectives. Business resources that are of importance to system owners are tangible things (such as vehicles and products), roles (such as customers and employees), events (such as orders and sales), and places (such

as sales offices).

The system user's view of data is in terms of data requirements, which are a representation of user's data. The data requirements, namely entities, attributes, and rules are expressed in a form that is independent of the physical implementation of it.

The system designer's view of data consists of data structures, database schemas, file organisations, record layouts, index files and other technical details. The system builder's view of data is implementation oriented, with the responsibility to write data programs and implement computer files and databases.

3.4.3 Building Block - ACTIVITIES

Activities are the work performed by people and machines for the business. They define the functionality of an information system and two main types of activities are identified (Whitten et al., 1994). *Business activities* are the day-to-day processes that support the purpose, mission, goals and objectives of the business. *Information system activities* are processes that support business activities by providing information processing, and improving upon and enhancing the business activities. Many of the activities involve *cooperation* between different people who view activities differently, as shown in Figure 3.7.

System owners are usually interested in the big picture and view information systems in terms of groups of activities called functions. Functions in this context are ongoing activities that support the business. *Business system functions* include sales, service, manufacturing, shipping, and accounting. *Information system functions* such as data processing, decision support, and office automation support these ongoing business functions. Information systems provide different levels of support for different business functions and different users, as illustrated in Figure 3.8.

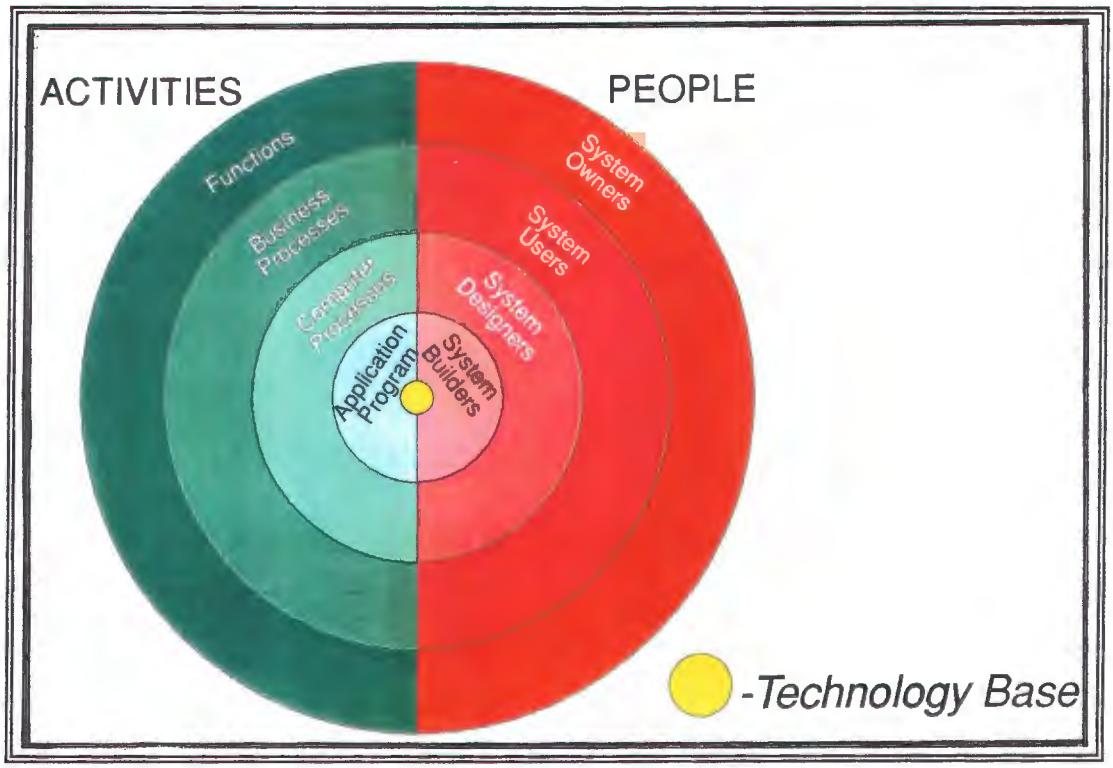

Figure 3.7 *The Activity Building Block of Information Systems*

The system users view activities in terms of distinct *business processes*. Business processes are distinct activities that have inputs, outputs, and specific methods and step-by-step procedures that underlie these procedures. These processes can be implemented by people, machines, or computers.

Similarly to the data building block, the system designer's view of activities is constrained by the specific technology used. The designer's view of activities are more technical in terms of *computer processes*. Computer processes are business processes that should be implemented on the computer. They may automate the business process or simply support it. Computer program specifications such as program structure charts, data flow diagrams, and record layout charts are used to describe the designer's view of the computer processes.

System builders view activities in terms of *application programs* using precise

computer programming languages that describe inputs, outputs, and logic. Application programs are language-based, machine readable instruction of how a computer process is supposed to accomplish its task.

<table>
<tr><td rowspan="2">DSS

Expert Systems</td><td>EIS</td><td>Strategic Level

Senior Management</td></tr>
<tr><td>MIS</td><td>Tactical Level

Middle Management</td></tr>
<tr><td colspan="2">Transaction
Processing
Systems</td><td>Operational Level

Professional, Technical
Office Workers</td></tr>
<tr><td colspan="3">DSS - Decision Support Systems

MIS - Management Information Systems

EIS - Executive Information Systems</td></tr>
</table>

Figure 3.8 *Information System Types in support of Business Functions*

3.4.4 Building Block - NETWORKS

Networks are the distribution structure of people, data, activities, and technology (the other building blocks) to business locations, and the movement of data between those locations. The intent of networking is to provide *cooperative working* between systems, computers, and people. Although, the term network has a technical meaning to most computer professionals, different respresentations are suited to different people. Different people's views of information systems networks are summarised in Figure 3.9.

To the system owner, networking is a *geographical*, and not a technical issue. The

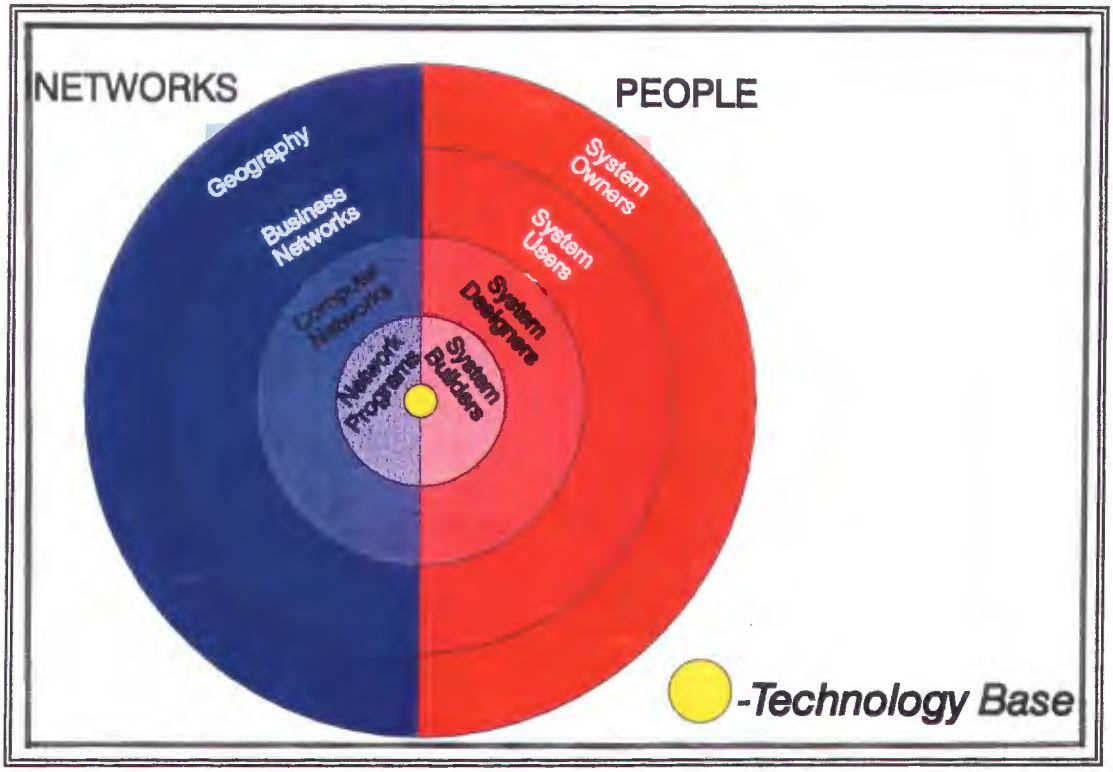

Figure 3.9 *The Networks Building Block of Information Systems*

geography of a system in this sense is the geographic locations in which the business chooses to operate. A modern trend in many organisations is to strategically seek to integrate their information systems into the businesses of their suppliers and customers at geographically dispersed locations. Some executive managers have also tested another geographic location, that of the cottage office in which employees are allowed to work out of a home office with access to organisational databases and communication tools.

System users view networks in terms of *business networks*. A business network defines detailed working locations, particular resources available at each location, and business communications requirements between the locations. The location specific resources are equipped in terms of the building blocks people, data, and activities.

Again, the system designer's view of networks is constrained by the limitations of specific technology. The emphasis is on a *computer network* that can support the business network. A computer network is a technical structure that interconnects computers and peripherals in order to share technical resources and exchange data. The system designer's view of networks is technical and computer networks should be created to implement distributed information systems in which data and processes have been divided into subsets that are technically distributed to, or duplicated in multiple locations. The system designer's view is expressed in terms of technology locations and the people (technical), data (the design of data files and databases), and activities (software) and computer-based technology needed at those locations. The system designer should also determine the type of physical connection between locations, for example terminals connected to a mainframe using modems, and minicomputers connected to a mainframe via a network operating system.

System builders use telecommunication languages and standards to write *network programs*. Network programs are machine-readable specifications of computer communications parameters which include node adresses, protocols, line speeds, flow controls, and other complex technical parameters. Examples include CICS, and Novell Netware/386.

3.4.5 Building Block - TECHNOLOGY

The technology building block forms the centre of the model in Figure 3.5. Collectively these technologies are called *information technology* which refers to the merger of computer technology with telecommunications technology. Information technology includes computers, peripherals, networks, fax machines, intelligent printers, telephones and other technology that supports either business communications or information processing.

The *data building block* of information systems is implemented in terms of *data*

technology which includes all hardware and software for the capturing, storing, and managing of the data resource (repositories, databases). The data technology includes the software for data storage, manipulation, and management, which in turn includes file management systems such as ISAM and VSAM, database management systems such as DB2 and dBASE, and spreadsheets such as Excel and Lotus 1-2-3.

The *activities building block* of information systems is implemented in terms of *processing technology* which includes all hardware and software for the support of business and information system activities. The processing technology includes computers, their input/output peripherals (printer, mice, scanners, and optical mark readers), and application programs. Additionally, processing technology includes operating systems and other systems software, and any machine or device that contains a computer processor.

The *networks building block* of information systems is implemented in terms of *communications technology*, also called networking or telecommunications technology. It includes the hardware and software necessary for interconnecting data and process technology at different locations. Examples of the networking technology are:

- Technology to connect office terminals at a single site to a host computer at the same site.
- Local and wide area networks of personal computers.
- Telecommunications networks that link large computers and possibly networks at one site to large computers and networks at another site.

Technical specialists should support system owners, users, designers and builders in the maintenance of technology. These technical specialists sell, configure, repair, and maintain information technology.

Given this fundamental understanding of the building blocks of information systems and its relation to the meta primitives of CSCW, as discussed in Chapter 2, it is clear that CSCW has a role to play in software development. The building blocks of information systems already form a platform from which CSCW systems may be built to support software development. In the next section, the process of building information systems are briefly examined with the emphasis on the roles of people, group activity and technology support.

3.5 Building Process of Information Systems

A disciplined approach must be followed in the building process of an information system. The software development approach followed in this section is based upon the revised spiral model for cooperative object-oriented development, represented in Figure 3.3. Each of the cycles is briefly discussed with the emphasis on group activity and existing technology support. The detailed nature of the technical tasks of object-oriented development are not addressed here. The work in each quadrant of the specific cycle follows the generic format of the spiral model.

3.5.1 Cycle 1 - Feasibility

The main purpose of the feasibility cycle is to formulate the problem statement and to determine the feasibility of developing a software system to solve the problem. The cycle is started as a result of a need for a software system, or an enhancement of an existing system.

i) Generic Tasks

Issue Formulation - a problem statement is formulated, which includes the scope of the system, the objectives, and the system constraints.

Problem Analysis and Evaluation - alternative solutions are proposed and the different system configurations are evaluated. A risk analysis is performed to ensure that on account of the available information on the software project and resources it is still worthwhile to continue with the study.

Development - a feasibility study is conducted involving a cost benefit analysis.

Review / Planning - a technical feasibility evaluation and an analysis of the legal implications of the project.

The deliverable is a project proposal which should be accepted and authorised by management before the Analysis Cycle is started.

ii) Group Activity

Group activity takes place during the Feasibility cycle. The cycle is driven by the cooperation of the system owners. It, therefore, addresses PEOPLE, DATA, ACTIVITIES, and NETWORKS from the system owners' perspectives. The system owners initiate the development process by expressing a need for a software system. Facts and opinions of the system owners and system users are used to formulate the problem statement. With the help of systems management and/or consultants the scope of the system is determined, a cost-benefit analysis performed and the system proposal compiled. The group activity is mainly conducted in the form of meetings and interviews with participants as discussed above. The feasibility study is mainly a group effort.

iii) Technology Support

Computer-based support is provided for the cost-benefit analysis and planning could be performed with the aid of project scheduling tools. Group activity in the form of meetings should be supported by meeting support tools and possibly

video conferencing tools which will be discussed in the next chapter.

3.5.2 Cycle 2 - Analysis

The main purpose of the analysis cycle is to study the current business and information system and to define the user requirements and priorities for a new information system. Following an object-oriented approach the cycle begins with an identification of the main abstractions of the problem space taking the user requirements into consideration.

i) Generic Tasks

Issue Formulation - object diagrams representing class hierarchies provide an initial logical view of the proposed software system. The dynamic and functional views of the software system, and ease of accommodating change are also considered.

Problem Analysis and Evaluation - alternative logical architectures are proposed in collaboration with the user by means of rapid prototyping techniques. From a project management perspective the risks involved in proceeding with a chosen alternative are evaluated. This determines the development strategy for the rest of the life cycle.

Development - Object Oriented Analysis (OOA) is performed according to a specific method. The user requirements are specified formally in a requirements specification which is the deliverable of this cycle and which forms the basis of the contract between the client and the software developer.

Review / Planning - a review is conducted to evaluate deliverables and to determine the status of the software project. A decision is made whether to continue with the project. If the decision is to go ahead a Software Project

Management Plan (SPMP) is drawn up which contains the managerial and technical functions, activities and tasks to be performed, resource and budget requirements estimations, and a development schedule.

ii) Group Activity

Group activity is eminent during the Analysis cycle. The cycle is driven by the cooperation of the system users. It, therefore, addresses PEOPLE, DATA, ACTIVITIES, and NETWORKS from the system users' perspectives. Facts by the system users are used to study and analyse the current system. With the help of systems management and/or consultants, the requirements and priorities are defined . The group activity is mainly conducted in the form of meetings between the system owners, system users and systems management, system analysts or consultants to finalise the requirements. The system analysts play the key role during this cycle. They work together in modelling the system by presenting diagrams which describe data requirements, process requirements, and geographic requirements. Another approach to translating and validating requirements is prototyping in which a small-scale, representative or work model of the users' requirements is built. The analyst team work together to construct such a prototype. The users may then react to the prototype to help the analysts refine or add to the requirements. The preparation of the Requirements Specification document is also a group effort, involving editing and reviewing activities. A list of the group activities during this phase includes:

- Requirements definition
- Interviews and meetings
- Technical reviews
- Verification of requirements
- Prototyping
- Preparation of the specification document.

iii) Technology Support

Computer-based support is provided in the form of upper-case tools to assist systems analysts in the modelling process. Project scheduling tools would again be used to review the planning of the software project. Group activity in the form of meetings should be supported by meeting support tools and possibly video conferencing tools which will be discussed in the next chapter. Analysts use powerful prototyping tools such as 4GLs to quickly build computer-based prototypes. The requirements specification is added to the repository within which the project information is stored.

3.5.3 Cycle 3 - Design

The main purpose of the design cycle is to evaluate design alternatives and to specify a detailed computer-based solution.

i) Generic Tasks

Issue Formulation - the physical views of the system are modelled and system analysts begin addressing technology issues and details. Candidate design solutions are identified.

Problem Analysis and Evaluation - the design alternatives are evaluated according to technical, operational, economic and schedule feasibility criteria. The risks of each alternative are determined using risk analysis techniques. A specific solution strategy which satisfies the user constraints and the optimisation criteria is documented in a system design document or system proposal.

Development - Object Oriented Design (OOD) is performed according to a specific method. The final deliverable is a technical design statement which is

generally divided into two parts: general design and detailed design. The general design serves as an outline of the overall design while the detailed design focusses on the detailed specifications for components in the outline. In the object-oriented approach these components include the classes of objects, the data attributes inherent in each class, the methods of each object and functions which may be called by methods. Most detailed designs employ some combination of systems modelling and prototyping. Design-by-prototyping allows system designers to quickly obtain scaled-down but working versions of a system or subsystem. Incremetal prototyping may be used as part of the iterative and evolutionary strategy of OOD.

Review / Planning - The prototypes will typically go through a series of iterations and user reviews until they evolve into an acceptable design. The middle management monitors the progress on the development of the classes of the logical system by means of formal and informal design reviews with the development team. More specifically, the progress is measured by logging the classes in the logical design and the modules in the physical design that are completed and functioning correctly. The stability of the main interfaces indicates the progress towards the final product. Middle and junior management plans the subsystem integration and system implementation as the modules and classes are completed. There may be a need to re-iterate the Analysis cycle resulting in a revision of the development strategy. A software project is rarely cancelled in the design phase, unless it is hopelessly over budget or behind schedule.

ii) Group Activity

Group activity is prominent during the Design cycle. The cycle is driven by the cooperation of various system designers, including the systems analyst. Therefore, PEOPLE, DATA, ACTIVITIES, and NETWORKS are involved from the system designers' perspectives. Candidate design solutions are proposed as design ideas and opinions by various sources such as systems analysts and designers, other

information systems managers and staff, technical consultants, system users or system vendors. System managers may, however, limit choices by approving technology architectures. The group activity is mainly conducted in the form of meetings between the system analysts, system designers, system users and systems management to evaluate candidate design alternatives. The system analysts and designers play the key roles during this cycle. They work together in physically modelling the system or prototyping the system. The user reviews are group activities involving the systems analysts, designers and the users. The final design review will also involve management. A list of the group activities during this phase includes:

- Modelling the system with the help of design methods (Functional designs, Class and method designs (in OO), Human computer interface design, design of database)
- Interviews and meetings
- Technical reviews
- Verification of designs
- Prototyping
- Preparation of the Design document.

iii) Technology Support

Computer-based support is provided in the form of upper-case tools to assist systems analysts and designers in the physical design of the software system. Project scheduling tools would again be used to review the planning of the software project. Group activity in the form of meetings should be supported by meeting support tools and possibly video conferencing tools which will be discussed in the next chapter. System designers use powerful prototyping tools such as 4GLs to quickly build computer-based prototypes. The ideal is to support the review group activity with computer-based tools. The design specifications are added to the repository within which the project information is stored.

3.5.4 Cycle 4 - Implementation

The main purpose of the implementation cycle is the construction of the new system and the delivery of that system into production. The Implementation cycle is usually performed only once.

i) Generic Tasks

Issue Formulation - Various strategies for subsystem integration and integration testing are established. Installation plans for the hardware of the final system are determined. A training programme for new users of the system is established.

Problem Analysis and Evaluation - the risks involved in the different subsystem strategies are analysed. The integration tests are also evaluated to ensure that they are adequate. The software packaging is reviewed and the hardware installation layout analysed. Evaluation of the contents of the training programme is reviewed to ensure that it covers all aspects of the system.

Development - if the new application calls for new or modified networks or databases, they must normally be implemented prior to writing the computer programs. If prototypes were developed in the design phases, they might be expanded into the final working system. The building and testing of programs is frequently the most time-consuming and tedious phase of the life cycle. Thereafter, subsystem integration is performed and tested. Finally, the new system is installed and tested and delivered into operation.

Review / Planning - System tests ensure that the application programs written in isolation work properly when they are integrated into the total system. If programs do not work properly when combined with other related programs, the programmer must often return to the build/test phase. The subsystem integration may also reveal incompatibilities between subsystem interfaces requiring that the

Design cycle be revisited. Some time after the new system is placed into production, a post-audit may be conducted to evaluate the new information system.

ii) Group Activity

Group activity is prominent during the Implementation cycle. The cycle is driven by the cooperation of various system developers (builders), including the programmers. The DATA, ACTIVITIES, and NETWORKS building blocks of the system are addressed. During the optional phase of building and testing networks and databases the key facilitators are various system designers, not programmers. Computer networks are usually implemented by the same networking designers who designed them, while databases are usually implemented by the database specialists who designed them. The complexity of these activities make it suitable for group activity. The structures of databases as well as networks are usually finalised in design meetings. The application programmers build and test the programs. Individuals, usually build specific programs and initially test them, but final testing, refining, system integration and system testing are group activities involving system designers, system builders, and sometimes the system users. Systems analysts must be available to clarify requirements and design specifications. The systems analysts facilitate the delivery of the new system into operation involving other system professionals, system users, and system owners. These group activities may be conducted in the form of demonstrations. The post-audit is also a group activity and may take place in the form of a meeting.

iii) Technology Support

Group activity in the form of meetings should be supported by meeting support tools and possibly video conferencing tools which will be discussed in the next chapter. System builders may use powerful tools such as 4GLs (like Focus, SAS,

and CSP) to improve productivity in building computer-based application programs. Also, design specifications could be input to an automatic code generator. Programming environments, such as those found in modern packages (like TurboPascal) consist of tools to quickly construct programs, and tools to perform debugging and testing of programs. Also worth mentioning, is distributed software development where development may take place in environments where developers of the system are in geographically dispersed locations and where groups may all partake in program reviews, editing and testing activities in this distributed environment.

The building process of information systems demonstrates the importance of good teamwork in group activity and the need for improved support for group activity, specifically during the software development process. *Computer supported cooperative work* (CSCW) seems to have the right qualities for satisfying the need of both management and the project team to deal with the group activities during software development. This aspect will be investigated in the following sections.

3.6 CSCW in Software Development

In the previous section the nature of software development was investigated and in conclusion it was found that CSCW may play an important role in the software development process. Large-scale development of software systems requires the cooperation of many developers and frequently development activities may occur concurrently. The goal of cooperation is to enhance, not restrict, developer productivity, while ensuring that concurrent development activities do not clash with one another (Harrison et al., 1990). A key aspect of such cooperation is ensuring that the developing artifact remains consistent in the face of concurrent modifications.

The increasing size of and limited time for the development of software is the basis for growing importance of distributed development teams (Hahn et al.,

1990). A major challenge is the support of (possibly large) teams of software developers who may be geographically dispersed. The support of this type of group activity may increase synergy and paralellism making the software development process more productive. The issues of group activity, concurrent development, distributed development, and technological support for group activities motivate the use of CSCW in software development.

Within the area of cooperation the work of software development teams constitutes a natural and important application. The shift in emphasis that seems to be implied by the emerging paradigm of computer supported cooperative work for this application is twofold (Hahn, 1990):

- Software development by teams, though being constrained by technical requirements, is recognised as a *social* process comprising the formal or informal interactions of the members of the team (*group work*) within an organisational setting.

- Social processes (*work procedures*) in task-oriented groups underlie particular conditions for *negotiations*, *commitments*, and *responsibilities* that are essential for smoothly accommodating dynamic changes of the project environment.

In discussing teamwork support, it is useful to distinguish between different levels of support and different scales of cooperation (Jarke et al., 1992). In terms of support levels, *connectivity* (the ability to technically exchange data), *interoperability* (the ability to exchange semantically meaningful information), and *cooperation* (the ability to enhance individual work by contributing towards a common goal) are relevant. In terms of cooperation, it is appropriate to distinguish between *collaboration*, *communication* and *coordination*. According to Ellis et al. (1991), collaboration refers to joint work on a common object, whereas communication refers to the exchange of pieces of information

(messages or development objects). Collaboration and communication tools augment human ability for cooperation in small groups by breaking barriers that have existed in traditional real-time and asynchronous group work (Jarke et al., 1992). On the other hand, coordination structures (and sometimes limits) the flow of communication and the way of collaboration in a typically larger group, to overcome overloading.

As already mentioned, software development is a complex process and, therefore, any computer support has to be as flexible as the development process itself. In establishing computer support for software development, some important characteristics of the process should be considered (Marmolin et al., 1991). Firstly, software development is an iterative process in which each activity is characterised by a mixture of analytic, structured, linear, creative, chaotic and nonlinear behaviour. The behaviour depends on the development phase, the state of the problem, and the size and skills of the development team. The bottom-up approach seems to be dominant in small research oriented development tasks, when the problem is ill-defined, the design teams are small and prototyping is used (Marmolin et al, 1991). Although support for processes is important, both analytic and creative software development activities should be supported.

Secondly, the earlier stages of software development are characterised by intuitive information gathering processes rather than by formal analytic processes. Concrete representations play an important role in understanding and evaluating development ideas. Thus, the support given has to focus on informal cooperation. According to Marmolin et al. (1991), tools for idea generation (story board facilities) and facilities for observing other systems, and tools for visualising and describing ideas are often more valuable than analytic tools.

Thirdly, the view held by Curtis et al. (1988) is adopted in which good development is characterised by the ability to integrate knowledge into a unified view and transform it into computational structures. Support is necessary for

integration of knowledge by learning and development from a common frame of reference.

Finally, software development is regarded as cooperative work. Thus, support for collaboration, coordination and communication are necessary. Curtis et al. (1988) refer to the need for informal and formal cooperative tools for recordkeeping of ideas and development concepts, change facilitation, information sharing, and project management. Support for cooperation, which is essential in software development, has to be very effective and so easy to use that it does not interfere with the development activities themselves.

The cooperative tasks in a distributed software development environment can be classified as either conference tasks, co-working tasks, information exchange tasks, or management tasks (Marmolin et al., 1991). A *conference task* is a discussion exchange of experience and knowledge between two or more team members. These discussions may take the form of negotiations, idea generation, problem solving, and briefings. The task could be asynchronous or synchronous, formal or informal and have social and communicative characteristics. It could be supported by electronic conference systems and mail systems. A *co-working task* is any activity concerned with the cooperative production of a document or other kind of product in a synchronous or asynchronous way. This task could be supported by distributed applications including co-editors, co-authoring and annotating systems. An *information exchange task* is an activity concerned with the exchange of documents and other information between two or more team members. It could be supported by shared databases, hypertext libraries, record keeping tools and other forms of group memories. The *management task* is an activity for coordinating and supervising of cooperation within a team. It includes planning and scheduling tools which may be supported by PERT and GANTT tools.

The classification of the generic cooperative tasks can be used as a basis for the

establishment of support for the cooperative development environment. Next, the requirements of CSCW in software development projects will be examined.

3.7 Requirements of CSCW in Software Projects

Based on the examination of the software development process and cooperative or group work, the following set of general functional requirements of CSCW in a distributed software development environment are relevant (Marmolin et al., 1991):

- *Support informal cooperation.* This is a most important requirement of a distributed software development environment that is also hard to fulfil. This requirement implies that the environment should support communication of social behaviour patterns, establishment and development of personal relations, and drop in meetings.

  Moreover, interactions of groups require support beyond the formal level of technical communication lines such as e-mail and electronic conferencing systems (Hahn et al., 1990). The social protocols that underlie group communication have to be accounted for in terms of human strategies and policies for argument exchange, contract assignment and decision making.

  Support is needed beyond basic *multi-user* facilities which are used to partition teams with standard schemes of concurrency control. The support of group interactions should account for human cooperative techniques such as negotiations, and commitments.

- *Support sharing and record keeping of software development information.* Research has shown the need for supporting sharing and record keeping of important development information, especially in larger teams or

groups. Thus the environment should support record keeping and sharing of requirement, design and implementation information, development deliverables, development ideas, commitments and work plans.

- *Support sharing of background knowledge.* This requirement refers to important preconditions for cooperation. These include the need for a common frame of reference, for sharing application domain knowledge and for exchanging knowledge about similar systems and other solutions to the development problem.

  Tools need content-oriented specification of knowledge beyond language facilities. According to Hahn et al. (1990), proper tool support and properly controlled tool integration should consider domain knowledge of the underlying project, working procedures and languages used for support specification, design and implementation.

- *Support presentations of ideas.* Concrete representations are very important in software development. Therefore the distributed software development environment should support different ways of presenting and visualising ideas for other team or group members. Examples are visualisation tools and story board or white board facilities.

- *Support strategies reducing the need for co-working.* Efficient tools for reducing the "collaborative load" may be more important than tools to support co-working. Division and integration of work, encapsulation and sequential processing should be supported.

- *Support co-working.* There will always be a need for co-working. Asynchronous co-working in particular should be supported by annotating and reviewing systems. The support of synchronous co-working are not as important, except for support of interface design together with users at

other locations.

- *Support management activities.* Management should be supported in terms of planning, monitoring, reviewing and control of software development projects. An example of a management tool is electronic calendars which have been proved not to be very useful (Bullen & Bennett, 1990).

Although the main concern is with distributed software development environments, it is not assumed that groupware should be a substitute for all face-to-face meetings. Generally, complex cooperative work involves a continuing need for face-to-face meetings. This is especially applicable in initiating and planning of the cooperative work. It is believed that well designed distributed software development environments may both reduce the need for face-to-face meetings and provide new and more effective ways of cooperation.

3.8 Summary and Conclusions

This chapter has investigated how the CSCW paradigm could be incorporated into the software development process. The nature of software development was analysed in terms of the building blocks and the building process of information systems. This investigation was motivated by the fact that software development is almost always carried out by groups and that technology support is essential for a quality, on-time and within budget software project. Finally, specific requirements for computer-based support of the cooperative work during software development were examined.

CHAPTER 4

Software Development within a CSCW Environment

CONTENTS

4.1. Introduction

There are numerous examples of both commercial products and research prototypes for most of the major categories of CSCW technologies. *Groupware*, is the generally accepted term for the technology that is used to support groupwork. However, some of the technology marketed as groupware is meant for small configurations, and other technology not generally classified as groupware is available and capable of supporting large infrastructures. Darnton (1995) proposes the use of the term *Co-op Ware Technology* to mean the study and application of technology in support of cooperative or collaborative working, and the term *technology-supported collaborative work* to mean the study and use of artefacts to support collaborative work.

Many isolated tools are already available to support software development processes. These include tools for modelling during analysis and design, resource management, project control, or hypertext facilities for project documentation. However, these tools tend to have limitations in terms of a sound coverage of the *knowledge* of the software domain, and a centralised view of project planning. They provide island solutions incapable of being integrated. Recently, tools have appeared that take a less centralised viewpoint of software projects and at the same time account for the human factor inherent in work procedures. Initially, software development environments incorporating CSCW tools were perceived as a forum of communication. Notions from speech act theory and other, more ad-hoc conversational models were considered to be the basis of tools such as typed messaging or conferencing systems (Hahn et al., 1990). However, a pure conversation perspective is inadequate, neglecting technical aspects of software development.

With so many CSCW technologies available, there is now a trend in CSCW research towards integration along numerous dimensions (Sohlenkamp & Chwelos, 1994). What is needed is an *integrated project support environment* (IPSE) consisting of multi-user tools suitable for computer-aided group work in software projects. The requirements of CSCW in software projects that were determined in the previous chapter are used to establish an environment for cooperative distributed software development which may form the foundation for integrated project support environments.

This chapter will first explore CSCW technology for software development. The discussion will focus on CSCW tools which may support specific group activities during software development. Some of the CSCW tools are multi-purpose and may support various group activities for different applications. However, the discussion also includes CSCW tools that were specifically developed for the support of group activities in the application domain of software development. The generic facilities and architectural components of CSCW software

development environments are then discussed.

4.2 CSCW Technology for Software Development

It was stated in Chapter 3 that the study is conducted within the OISEE project, and that the study is mainly concerned with the *Development Process Reference Model* of the project. The *Development Process Reference Model* addresses the information system development life cycle according to the *management aspect,* the *life cycle aspect*, and the *methods aspect*. Therefore, both managerial and technical elements should form part of an integrated software development environment. In the investigation of the software development process in Chapter 3, it was found that groupwork formed an integral part of project management, as well as the technical software development activities. CSCW technology for software development should include tools supporting the group activities of project management and collaborative software development. In this section a survey of CSCW technology within different categories will be provided. In this way the *Technology Reference Model* of the OISEE project is also addressed.

4.2.1 CSCW Technology for Project Management

Project management is an essential part of software development. Generally, it sets clear objectives, provides adequate resources for the project, and controls the project team and work done during the course of the project. More specifically, project management is concerned with planning, monitoring and controlling a project. It includes two important tasks. One is the breakdown of the whole project into a number of well-defined management entities to which resources should be allocated. The entities should be organised in the most suitable manner and monitored to ensure that the proposed goal is being achieved. Project management must also coordinate all project tasks and must be closely integrated with the problem-solving process. Methods are applied to plan, control and monitor each software development phase or cycle.

Starting from the application of project management models originally developed for economic domains, software engineers soon recognised the need for automated tool support in managing the software development process. The original project management tools did not consider the social dimension of software group work. *IStar* (M. Dowson, 1987) addressed this problem by allowing formal task fulfilment relations to be created among a contractor and a client. The contracts are kept as formal objects in a contract database and the implementation (programming code) is evaluated by semi-automated devices. The various roles people play in project management and their relationships in terms of formal communication protocols have been investigated in *MONSTR* that later became the *XCP* protocol for general cooperative procedures (Sluizer & Cashman, 1984). However, these approaches provide only syntactic rules and associated mechanisms to manage team work. This has led to the incorporation of conceptual modelling languages. Although, being more explicit about basic semantic relationships of software project management, these approaches tend to concentrate on traditional notions of project management (such as deliverables and milestones) neglecting social factors such as conflict negotiation and work plan revision.

The importance of social criteria for the communication processes in project teams has been considered in systems which integrate formal speech act notions into the protocol mechanisms of a project management support system (Kedzierski, 1984). The *CHAOS* system (De Cindio et al., 1986) additionally incorporates technical aspects of project management. The *Coordinator* system (Winograd et al., 1988) includes group-specific social factors (contracting, argument exchange, multi-agent problem solving) in a comprehensive formal model that is based on the semantics of its application domain (software development and maintenance). The exchange of arguments within an information system has first been considered in *SYNVIEW* (Lowe, 1985) and was then adopted in *ArgNoter* (Stefik et al, 1987). These approaches are completely informal, but the *gIBIS* protocol (Conklin & Begeman, 1988) allows limited

formal control through a state transition network which specifies legal patterns of argument exchange.

This section will further explore examples of useful CSCW tools for project management.

(i) Tools supporting Reviews and Monitoring Task Achievement

Monitoring task achievement relates to the quality evaluation of the product (modules, programs, subsystems, or whole systems) delivered. Obviously the resulting product should meet the initial requirements. An example of a CSCW tool fulfilling this purpose is the review of a contract deliverable based on semantic and contractual analysis. This tool forms part of the *Conex* system (Hahn et al., 1990) which was developed for the purpose of project management. A standardised test bed is used for program evaluation and depending on the result of this step the deliverable is either declared completed or rejected.

(ii) Tools supporting Variant and Version Management

In terms of project management, the current state of the project is always important. The current state of the project is *monitored* in order to determine its status and to make important decisions. Variant and version management CSCW tools usually provide an explicit representation of the transformation steps among versions on the design and implementation level in terms of decisions made. This enables management, and members of the project team to perceive the logical dependencies that underlie changes to specific modules or a group of modules. The CoNex prototype (Hahn et al., 1990) provides an example of a variant and version management tool in Figure 4.1.

The alternative version of the design object *Emp PerPro_Des* results from different requirements mapping decisions. The change in the design may cause

a different implementation of *Emp PerPro_Imp* and require additional efforts from the project team.

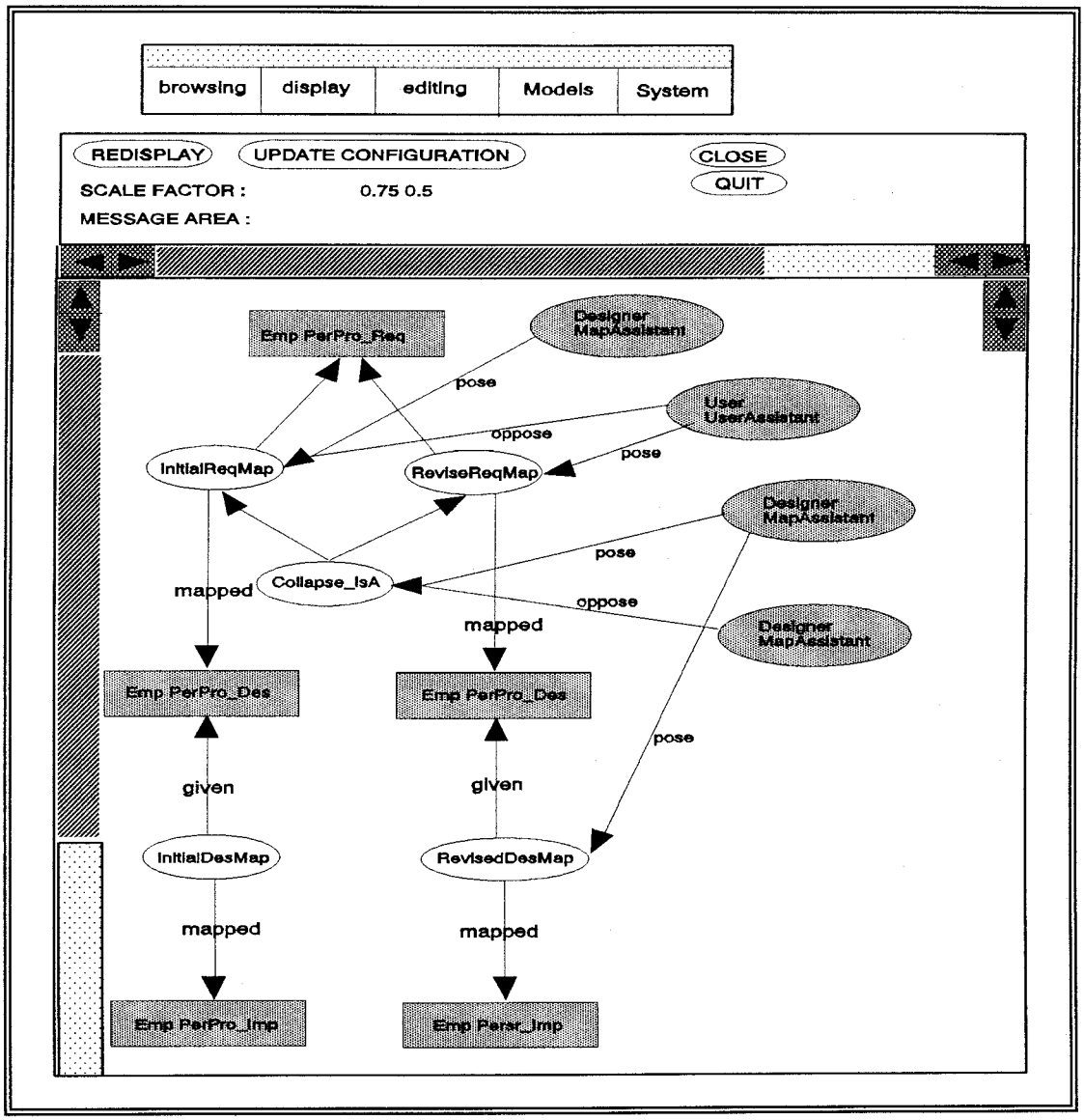

Figure 4.1 *Two Versions of a System and the Argumentation involved (Hahn et al., 1990)*

This specific tool may also be used during project review meetings. The argumentation is shown in the form of screen task debates. In the figure, the shaded oval blobs point to modules that have undergone change and shows the associated argumentation. The argumentation indicates that the user objected (opposed) to the original design, whereas the designer, while satisfied on the

conceptual level, opposes the argument on account of *workoverhead.* The argumentation that documents the change in design may be reviewed to make a decision concerning the correct way of handling a modification to the system.

(iii) Automated Decision Support Tools

The variant and version management tool introduced one particular application of the groupwork model of CoNex. It illustrated negotiation in terms of argument exchange among multiple agents. The functioning of the tool was then extended to include the integration of automatic decision support techniques into the interactive design of a project work plan. Given the data by the decision aid tool the analyst and the project team decides on the implementation of the revision.

One could also conceive more sophisticated tools such as optimisation procedures and simulation programs to be plugged into this stage. Generally, the results generated by these decision devices lead to task debates. The tool may also include the facility to do task contracting, i.e. the assignment of tasks to people. The contract dialogue is connected to the formal specifications of the task to be done. It is recommended that a strict formal protocol (conversations that may be adapted dynamically, but controlled and documented formally) be followed when members of the project team are engaged in argumentation or contract dialogues. Both kinds of dialogues are multi-agent interactions where several members of the project team may get involved.

Much effort in computer supported cooperative work is oriented towards the phenomenon of a group decision. Complex organisations are characterised by distributed decision-making of interdependent work groups, such as in software development teams,in manufacturing, and in marketing which all have unique decision domains. These organisations require a sharing of perspectives among distributed decision-makers if they are to coordinate activity and adapt to changing circumstances. SPIDER is a software environment for enriching

communication among managers by improving their ability to represent and exchange understandings of the situations and decisions they face (Boland et al., 1992).

(iv) Toolkits for building Groupware

It is all too well to propose that software development should take place within a CSCW environment, but how should such an environment be established? It is the responsibility of an organisation, more specifically the project manager, to establish the infrastructure, and to provide the necessary facilities (hardware and software components) for a CSCW software development environment. Yet construction of CSCW environments is fraught with difficulties. The groupware developer is concerned with technical issues such as synchronisation, concurrency, communication, and registration. The fundamental CSCW human factors for effective groupwork should also be incorporated.

Groupware toolkits are now emerging that address some of these issues. The toolkits may reduce the development effort, enable rapid prototyping, and increase the product quality of multi-user applications. An example toolkit is *GROUPKIT* (Roseman & Greenberg, 1992) which can be used for the development of real-time computer conferencing applications for geographically distributed or face-to-face meetings. The technical infrastructure of *GROUPKIT* is based upon a replicated architecture and communications support, as illustrated in Figure 4.2. Here, the objects owned by one user (rightmost *Registrar Client, Coordinator*, and *Conference*) interact with objects owned by another user, as well as the central *Registrar*. The small font text indicates the message passing protocol.

Another example of a groupware toolkit for creating multi-user applications is *Weasel* (Graham & Urnes, 1992). *Weasel* is based on the *relational view model* in which user interfaces are specified as relations between program data structures

and views on a display. The programmer provides a central application program, written in a traditional programming language, and a set of view specifications, written in the declarative language, *relational view language* (RVL). From these specifications the *Weasel* system creates a distributed implementation in which network communication, concurrency, synchronisation, and customisation are handled automatically.

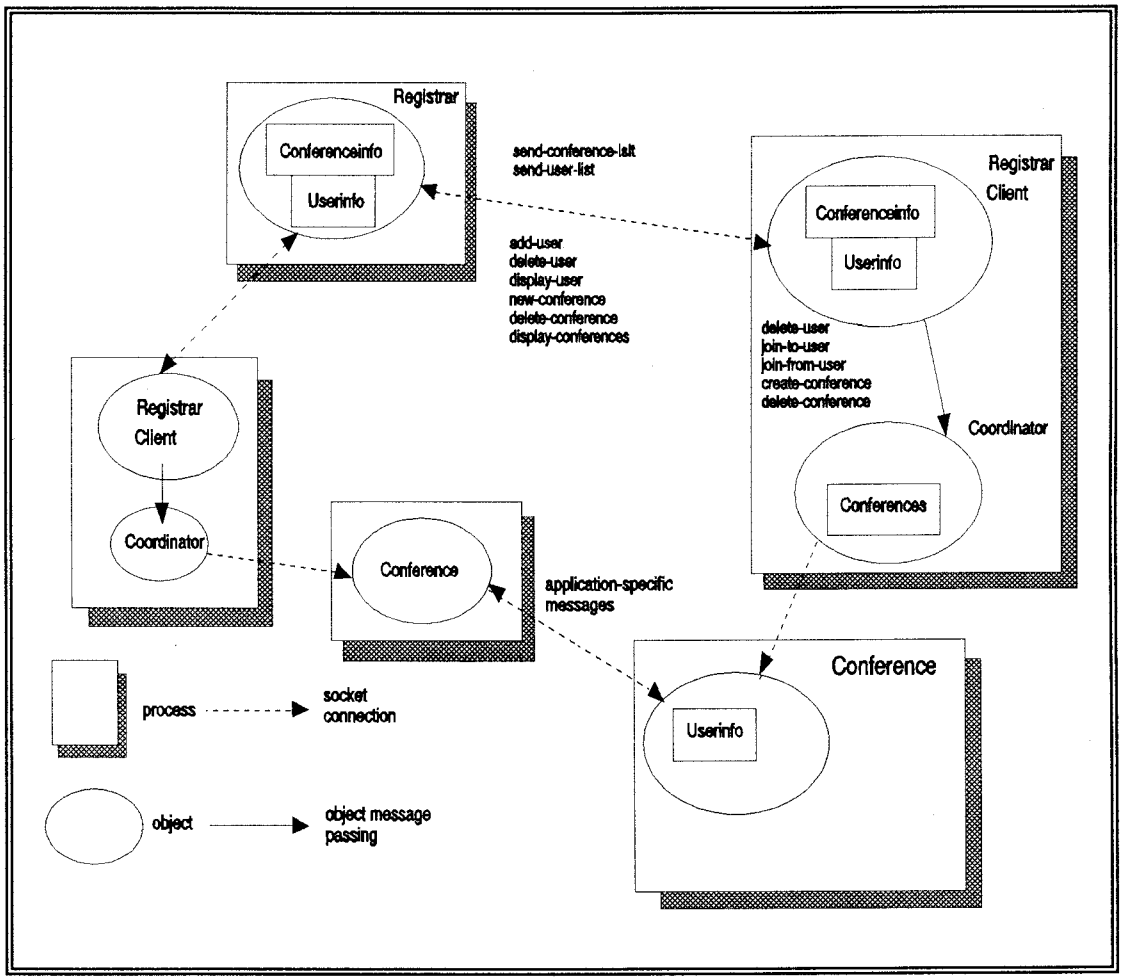

Figure 4.2 *Communications infrastructure of GROUPKIT (Roseman & Greenberg, 1992)*

ConversationBuilder (Kaplan et al., 1992), *Coordinator* (Winograd & Flores, 1987), *Chaos* (DeCindio et al, 1988), and *gIBIS* (Conklin & Begeman, 1988) are examples of support tools that are intended to provide flexible active support for collaborative work activities. These active support tools have open architectures

in which the kernel for the specification of activities and interconnections among activities, and tools (groupware) can be integrated. New tools may be added, and the flexible work support tools may change when the tasks that are supported change.

(v) Message and Meeting Scheduler Systems

A vital part of project management is to effectively communicate with project team members and to schedule meetings. A CSCW tool supporting these aspects of project management is *Active Mail* (Goldberg et al., 1992), a framework for implementing groupware designed to support more effective computer-mediated interaction. *Active Mail* piggybacks on ordinary electronic mail retaining all the features that have made it so successful. The messages in *Active Mail* are used to establish persistent interactive connections among a group of users. Receivers of *Active Mail* are able to interact with the sender, with future recipients, and with remote, distributed multi-user applications. Groupware applications realised within the *Active Mail* framework include a text conversation tool, a collaborative writing facility, revision control management, an interactive meeting scheduler, and some distributed multi-user interactive games.

According to Ephrati et al. (1994) two paradigms of meeting scheduling scenarios may be distinguished: *Open Scheduling Systems* and *Closed Scheduling Systems*. In open scheduling systems, one or more independent autonomous individuals try to schedule a meeting. The individuals are completely in control of their own time resources and have no obligation to meet one another. In closed scheduling systems, such as in organisations, the meeting mechanism is imposed on members of a group. The participants of any potential meeting have some obligation to take part in a meeting. The constraints of the meeting are determined by the system. A closed scheduling system should therefore maintain a consistent and complete calendar of the organisation's members. Closed scheduling systems are preferred and have three alternative approaches to the scheduling process,

namely *calendar oriented scheduling* (preferences specified over available time slots), *meetings oriented scheduling* (preferences expressed per alternative specific schedule), and *schedule oriented scheduling* (preferences specified over entire schedules).

(vi) Tools supporting Awareness and Coordination in Shared Workspaces

Awareness of individual and group activities is critical to successful collaboration, and more specifically to project management. The control, planning, coordination, and monitoring of components of project management are closely linked to the way CSCW is applied during software development. Awareness is an understanding of the activities of others. This provides the context which ensures that individual contributions are relevant to the group's activity as a whole, and to evaluate individual actions with respect to group goals and progress. Awareness is always required to coordinate group activities, whatever the task domain. Awareness is commonly supported in CSCW systems by active, information generation mechanisms separate from the shared workspace. Dourish and Bellotti (1992) discuss the use of a shared editor which suggests that awareness information provided and exploited passively through the shared workspace, allows users to move smoothly between close and loose collaboration, and to assign and coordinate work dynamically.

Existing CSCW systems vary in the mechanisms they provide to support awareness. One mechanism, referred to as *informational*, provides explicit facilities through which collaborators inform each other of their activities. For example, software control systems such as *RCS* (Tichy, 1992) expect users to provide text for an "edit log" which describes the nature of changes. Another example is the integration of electronic mail with an authoring system as a channel for sharing this information, as in *Quilt* (Fish et al., 1992). A second mechanism, which is referred to as *role restrictive*, arises from the explicit support for roles in collaborative systems. A role describes an individual's relationship to

shared work objects and to other participants, and is typically linked to a set of operations which can be performed. Explicit role support reduces uncertainty about the actions an individual might take, but only information about the character of the activity, and not the content is provided. A third approach is to provide shared feedback and results from individuals to the group at large through a shared workspace (Dourish & Bellotti, 1992). The emphases of this approach are on low overheads for the providers and recipients of awareness information, the availability of information as and when needed as a context for individual activities, and the avoidance of restrictive pre-structuring of group activity.

The *Milano* project (De Michelis & Grasso, 1994) aims to integrate the main components (namely a conversation handler, a workflow management system, and an organisational handbook) in a system that supports its users to situate themselves, at any time, within the work process in which they are active, with a high degree of awareness. Workflow software provides the infrastructure to design, execute and manage business processes in a network.

The *DesignNet Model* (represented in Figure 3.1) addresses the *management aspect* of the OISEE project and may also be used as a tool for the purpose of establishing awareness and coordination in software projects. In fact, the DesignNet Model may form the starting point of all project management activities taking place within a CSCW context.

(vii) Tools for Automatic Transcription of Formal Meetings

Given the importance of project meetings, it is essential to capture the spoken contents of formal group meetings. Some research has been done in terms of electronic meeting systems. There exist several CSCW tools in the form of ubiquitous audio to support informal meetings, personal notes, and telephone conversations. Many workstations are now equipped with a speaker and

microphone. An example CSCW tool is *Xcapture* (Hindus & Schmandt, 1992).

(viii) Tools for Issue Discussions, Event Calendars, Task Tracking

Telephones are well-networked and useful terminals. This makes telephones an attractive platform for cooperative work applications such as event calendars, issue discussions, question and answer gathering applications, and task tracking. An example is *HyperVoice*, an application generator for telephone bulletin-board applications (Resnick, 1992).

An approach supporting spatial workspace collaboration via a video-mediated communication system may also be used for informal discussion. Spatial workspace collaboration may be defined as collaboration in a three-dimensional environment. An example of a video communication system is *GestureCam* (Kuzuoka et al., 1994) in which a camera is mounted on an actuator with three degrees of freedom. For the interface, the master-slave and touch-sensitive CRT methods are used.

(ix) Videoconferencing Tools to support multi-party distributed Meetings

A meeting is traditionally a process in which people gather at the same time and the same place to exchange views or information to solve problems. Software project meetings take place regularly, and it is important that management, possibly the users, and members who are required at the meeting, and management attend these meetings. The following factors are important for meeting support:

- Place - "face-to-face"
- Distance
- Shared space.

The ideal meeting is for all participants to sit around a table and discuss topics

in a meeting room in which they can see each other and in which the distance between them is not very big. Eye contact and gestures are important in meetings. Okada et al. (1994) proposes a multi-party videoconferencing system, MAJIC, that projects life-size video images of participants onto a large curved screen as if users in various locations are attending a meeting together and sitting around a table. MAJIC supports multiple eye contact among the participants and awareness of the participants' gaze, hence users can have discussions comparable to face-to-face meetings.

4.2.2 CSCW Technology for Collaborative Software Development

Various tools supporting the methods and implementation aspects of software development already exist, such as CASE tools, programming environments, fourth generation languages, program generators, and software engineering workbenches. However, the incorporation of CSCW in the software development process require additional groupware or CSCW tools.

Ethnography has gained considerable prominence as a technique for incorporating the nature of work in the development of CSCW systems. The social context of work will also play an important role when CSCW becomes part of the software development process. Two trends have strongly motivated the rationale for ethnography in CSCW systems development (Hughes et al., CSCW 1994):

- The belief that the reason why many systems fail is due to the fact that their design pays insufficient attention to the social context of work.

- A realisation that the emergence of low-cost technology and the ubiquitous nature of networked and distributed computing pose new problems for software development. New methods which analyse the collaborative, hence social, character of work and its activities are

required.

Ethnography and its implications for CSCW is a study on its own, and is not pursued further for the purposes of this study. This section will explore CSCW tools which may support the cooperative work of software teams during software development.

4.2.2.1 CSCW Tools for the Analysis and Design Cycles

(i) Message Systems

Crucial to any collaborative work is the ability to communicate between participants. Voice communication is probably the most efficient medium, but has two drawbacks: it is intrusive (expected to react instantly to a vocal query), and it is hard to store and manipulate. Mail programs (such as Unix mail) are often used to communicate even in the same room. This is non-intrusive (an answer can be delayed) and mail can be stored for later reference. Some mail systems are very primitive as only two persons may be involved in communicating, and mail is neither interactive nor reliable in terms of when the mail arrived and when it has been read.

Standard communication tools like e-mail or e-talk lack a number of important features such as supporting any number of users, working in real-time, and supporting both text and graphics in messages. The *Andrew* message system (Borenstein & Thyberg, 1988) has multi-media facilities and includes some collaborative facilities, but it applies to a large area network and thus lacks close interaction between users. Similarly, *Information Lens* (Malone et al., 1987) supports asynchronous communication, but does not support the real-time conferencing aspect of a talk system. The need is for powerful message systems supporting real-time conversations.

A support system for software development teams should provide embedded communication facilities that are flexible enough to allow discussion at various levels, from specific designs to more global issues. In recognition of the crucial role discussions and rationale play in the development of complex systems, computer systems have been built that attempt to capture design rationale. Artifact-centered information spaces support communication during collaborative work as well as individual work. Design decisions are supported within an evolving information space centered around the artifact being designed. An example is *XNETWORK* (Reeves & Shipman, 1992), a knowledge-based design environment for computer network design that incorporates this artifact-based communication as a method for easy addition of network designers' understanding about the design task.

(ii) Shared Drawing Tools

The generation of design ideas in group discussion is a complex and dynamic process. Usually, sketches are important for coordination during requirements engineering and the initial stage of the design process. The group communication for this purpose can be facilitated by computerised drawing tools which permit simultaneous sketching by team members, sometimes in different locations. Computerised drawing tools support the idea generation process in terms of the fluent expression of ideas, and the ability to interact and build on existing representations. These tools need to aid not only the drawing process but also the management of design ideas during group interaction. There are five critical factors that affect a group-based analysis and design process and that should be considered in establishing requirements for shared drawing tools. These factors are *work allocation, design integration, design ownership, design recall,* and *space sharing* (Lu & Mantei, 1991). The shared drawing tool features based on the requirements supporting each of these factors are summarised in Table 4.1.

Examples of shared drawing tools are *CaveDraw* (Lu & Mantei, 1991), *Commune*

(Minneman & Bly, 1991), *VideoWhiteboard* (Tang & Minneman, 1991), and *Teamworkstation* (Ishii, 1990). Most of these tools support the features mentioned in the table above. *VideoWhiteboard*, for example, provides a whiteboard sized shared videospace allowing multiple designs to be viewed at the same time, but the designs are immovable. *CaveDraw* includes line, rectangle, oval, polygon, text, and freehand (pencil & marker) drawing tools. Users may select and erase drawing segments and use different coloured markers to identify their own work.

Table 4.1 *Shared Drawing Tool Features*

CRITICAL FACTOR	REQUIREMENTS	SHARED DRAWING TOOL FEATURES
Work Allocation	Participants can select individual segments of the analysis & design to work on simultaneously.	Participants draw their analysis & design ideas on shared transparent layers. Drawings on the topmost layers may appear in brighter colours to distinguish between layers.
	Participants are aware of the analysis & design activities of others while working on their own segment.	The layers are superimposed on each other in order for a participant to see other drawing activities taking place and who is viewing or working on each layer.
	Participants are able to select and modify all previous analysis & design ideas.	Participants can select, create and hide the display of any layer on their screen.

Design Integration	Participants are able to compare and consolidate modifications to different portions of the original analysis or design and still throw out undesirable changes.	Participants are allowed to draw alternate analysis & design ideas on different layers and superimpose the layers or subsets of the layers in any order selected by the participants. The saving of any of these combinations is also allowed.
	Participants are able to compare different modifications to an analysis or design idea at the same time without disturbing the original idea or having to view multiple displays.	Same approach as above is followed. In addition, participants may work on their own layer while other participants are performing a comparison.
	Participants are able to view both the overall analysis or design and its subunits in addition to the subunit they are working on.	Each participant is allowed to bring up a sublayer showing the connection of all subunits while working on one of the subunits in the previous layer.
Design Ownership	Participants are able to declare any portion of a sketch as private and not subject to deletion by others.	Participants are allowed to declare work public or personal by selecting public or personal tools respectively. Work drawn with a personal set of tools cannot be erased by others. Participants are allowed to convert their work from private to public and vice versa through available editing functions.
	Participants can identify with no additional interaction sequences who is working on any specific analysis or design sketch.	Each participant's work or ownership of an analysis or design is identifed by a specific colour.
Design Recall	Participants are able to review prior analysis or design ideas with minimum effort.	Participants are allowed to directly select the viewed layers that capture prior analysis or design ideas with one mouse click on the dimmed layers. Layers may be selected through using a pulldown menu.

Space Sharing	Participants are able to work in their own space yet virtually share space with other participants.	Participants are allowed to generate as many layers of drawing surface as they need. They are also allowed to select a personal set of layers to work with while retaining elements common with others.

Cavedraw differs from other shared drawing tools in its support of "transparent layers". Participants may cut or copy any portion of a sketch on one layer to a desired location on another layer. Figure 4.3 presents an example of the overlapping layered approach in *CaveDraw*.

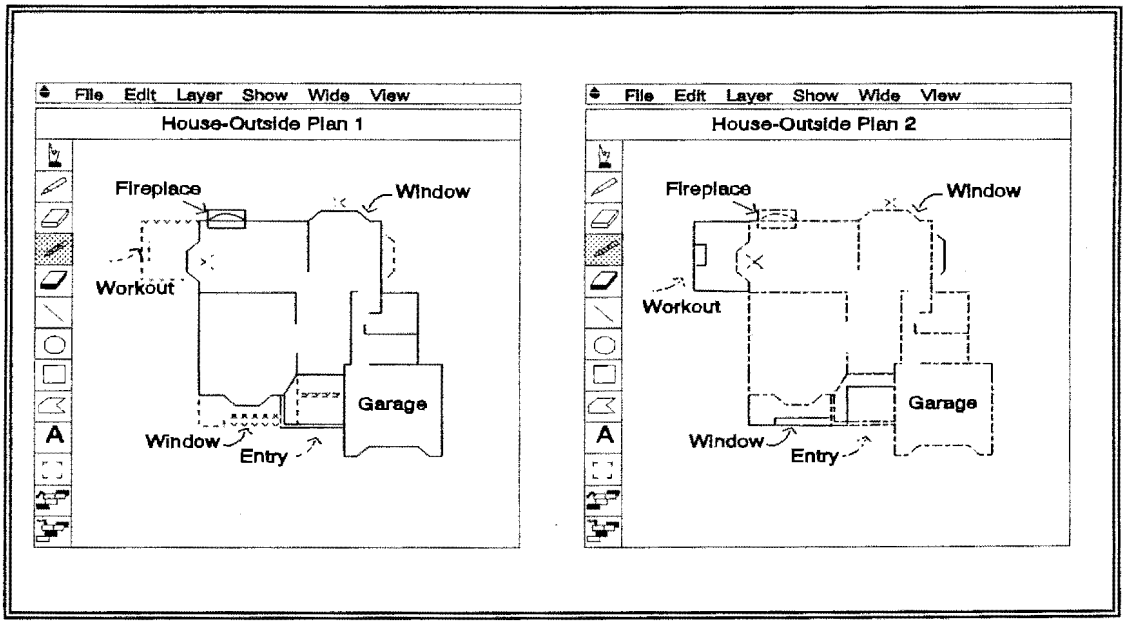

Figure 4.3 *The transparent layers of the CaveDraw Tool (Lu & Mantei, 1991)*

Another example of a shared drawing tool is *Conversation Board* (Brinck, CSCW 1992), a prototype multi-user drawing application built to be used by people who are conversing over a distance using a phone or video phone system. The *Conversation Board* is a structured graphics editor which includes drawing features such as markers, lines, circles, rectangles, text, connectors, and images. Users may draw simultaneously and telepointers make their gestures visible to

each other. *ClearBoard* (Ishii et al., CSCW 1992) provides a shared drawing medium that permits co-workers in two different locations to draw with colour markers or with electronic pens and software tools while maintaining direct eye contact and the ability to employ natural gestures. Figure 4.4 illustrates the system architecture of *ClearBoard*. The architecture includes *TeamPaint*, a multi-user computer-based paint editor running on networked computers, and digitiser pens. A CRT-based rear projection display with a transparent digitiser sheet is used to improve the screen clarity. The digitiser is mounted to the surface of a flat panel display and the screen size is 80 cm x 60 cm.

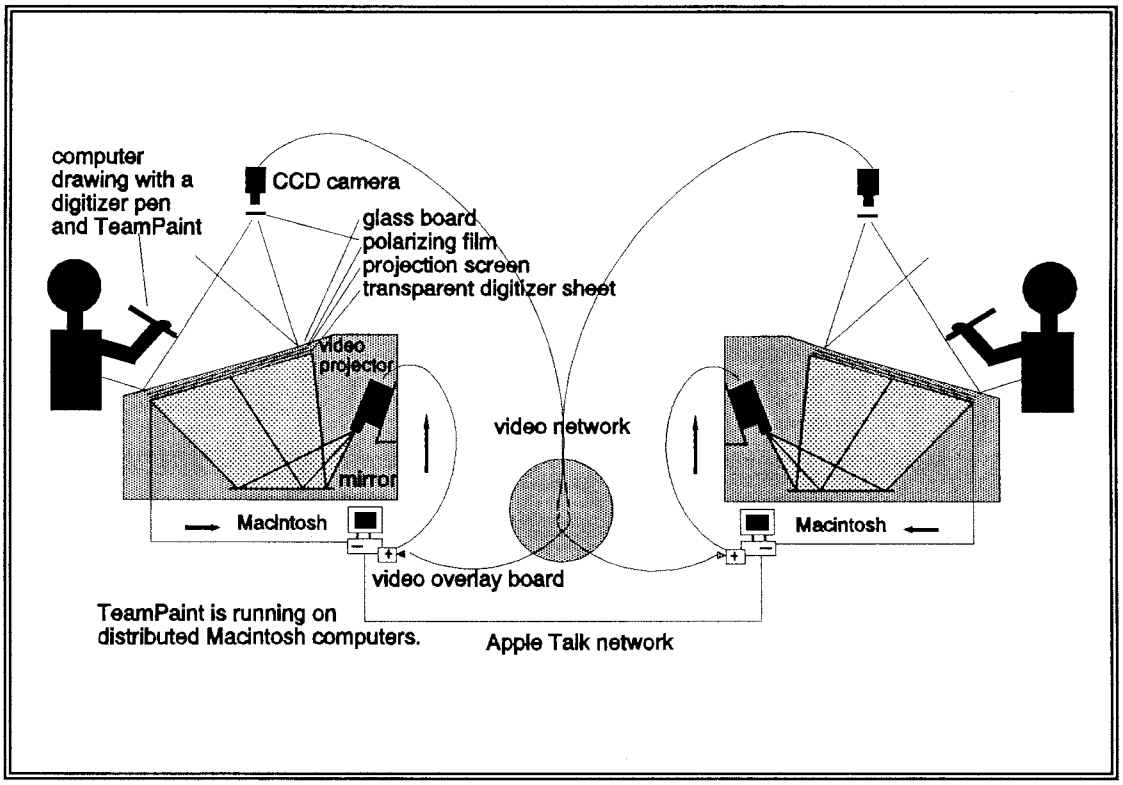

Figure 4.4 *System Architecture of ClearBoard (Ishii et al, 1992)*

Other examples of shared drawing tools are *VideoDraw* (Tang & Minneman, 1990) and *TeamWorkStation* (Ishii, 1990) in which hand gestures in a shared workspace can be supported by shared video drawing media. *Groupsketch* (Greenberg & Bohnet, 1992) allows small geographically-distributed groups to

list, draw and gesture simultaneously in a communal work surface, supporting interactions similar to those occurring in the face-to-face drawing process. *Conversation Board* (Brinck & Gomez, 1992), a shared "whiteboard" allows people to share artifacts like drawings, graphs, or photographs which are often part of face-to-face meetings. Figure 4.5 illustrates a view of the *Conversation Board* for one user and a miniature of the view of a second user. This example shows how the system is used to brainstorm a description of the *Rendezvous* system. The main window consists of a control region at the top and a canvas where all the drawing occurs. There are several tool palettes available and the main one is shown at the top right.

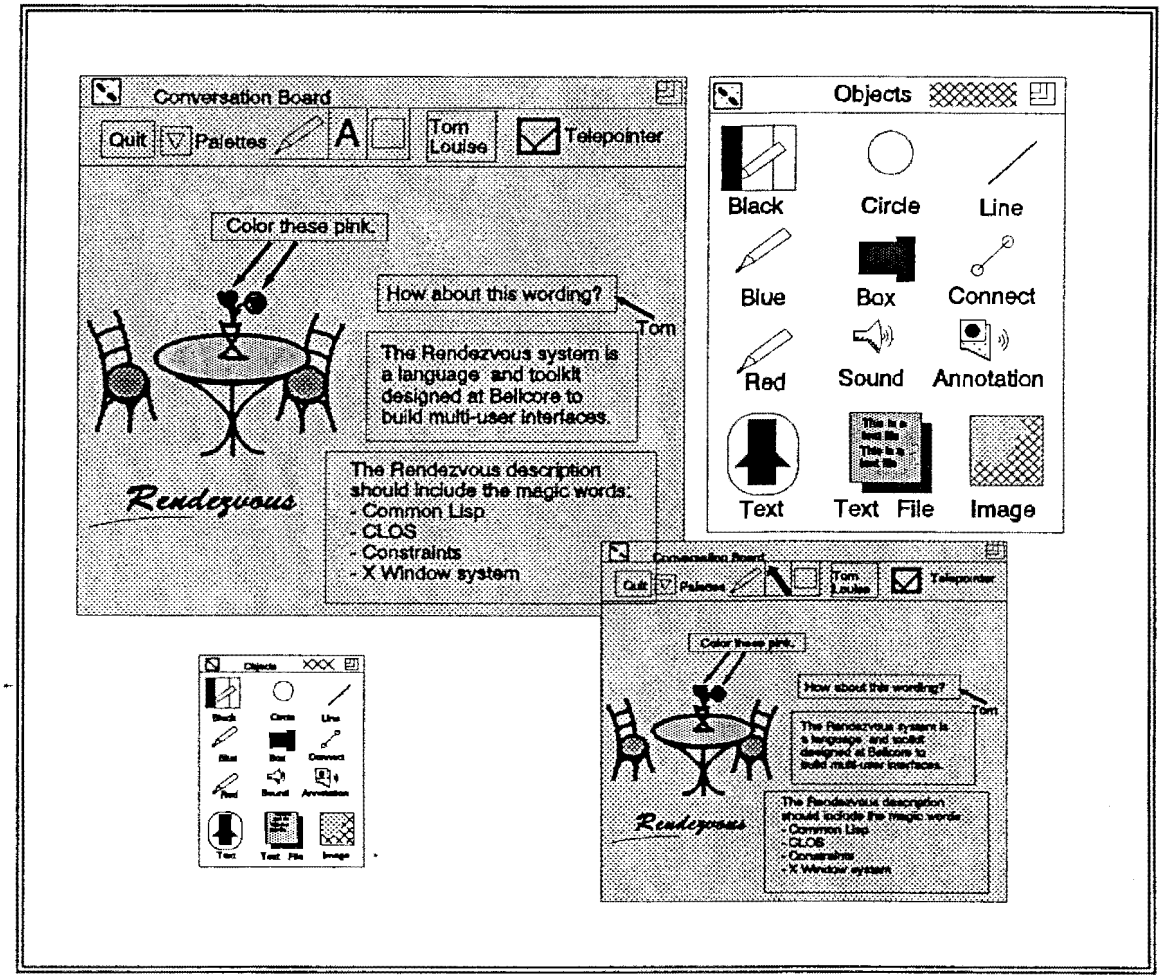

Figure 4.5 *A view of the screens of Conversation Board (Brinck & Gomez, 1992)*

(iii) Documentation Tools

Documentation forms an intergral part of the software development process and is time-consuming. Documentation is hardly ever the responsibility of one individual member of the project team. Normally, more than one member of the software team works on the software project documentation and it may thus be ideal as a group activity. Factors such as consistency, version management, change control, cross referencing and graphical capabilities are important in the documentation process.

Documents are created through a number of activities including brainstorming, outlining, writing, editing, and reviewing. *SASSE* (Nastos & Baecker, 1992) supports the different requirements of each of these activities, and smooth transitions among the activities. In the case of *SASSE*, synchronous work is aided by colour-coding of participants and their selections. The synchronous work is also facilitated by views that enhance collaboration awareness and the use of sound. On the other hand, asynchronous work is aided by automatically-generated change descriptions and by writer-inserted annotations. Another example of a documentation tool is *Aspects* (Naef et al., 1992), which allows two to sixteen users on a network or modem link to work together on a document at the same time and see each other's changes as they happen. The document contents may include text, drawing, and bit-mapped painting.

DUPLEX (Pacull et al., 1994) is a groupware application designed to fulfil the need of collaborative editing in a large-scale distributed environment. Collaborative editing involves both writing activities and communication between co-authors, and an environment or tools supporting it should consist of three separate facilities:

- a *collaborative editor* enabling users to share and maintain the state of a document written by several co-authors.

- a *direct communication facility* helping co-authors exchange information about the collaboration. The recipients of a message are selected by the sender.

- a *subject-based communication facility* enabling users to distribute data on specific subjects, such as modifications in a document.

These facilities express the means of interaction (synchronous or asynchronous) between co-authors. Several systems, such as *GROVE* (Ellis et al., 1991) and *DistEdit* (Knister & Prakash, 1990) incorporate a synchronous collaborative editor. The *DUPLEX* environment incorporates a collaborative editor with asynchronous interaction between users through a shared kernel which maintains the document context. Document decomposition is used together with a consistency criterion adapted to collaborative editing in order to limit conflicts between authors and to enable recovery using subject-based or direct communication. The decomposition enables users to specify the segments of the document they are interested in and to apply concurrency control to their segments.

(iv) Tools to support Problem Solving

Many software development collaborative tasks involve problem solving. A number of CSCW technologies have emerged to support the problem solving activities of groups. Problem solving has many stages ranging from problem identification, idea generation, idea structuring, idea selection or decision making, and implementation. One approach to supporting the many stages is to have a suite of tools, each tailored to a specific stage. Examples are *Arizona's Plexus system* (Nunamaker et al., 1992) and the *SAMM system* (Gallupe et al., 1992). Idea generation is a critical component of many problem solving tasks. One way is to allow brainstorming groups to use a computer-based tool that permits simultaneous entry of ideas, yet still allows sharing of ideas among the group

members (electronic brainstorming). Much research in this area has been done at the University of Arizona. Their decision support system incorporates a brainstorming tool that is based on *Nominal Group Technique* (NGT), a formal method for giving interacting groups the advantages of a nominal group. The NGT allows people to generate their ideas in parallel on paper, and then exchange them with each other as a way of simulating the flow of ideas. McGuffin and Olson (1992) proposed that *ShrEdit*, a shared text editor, be used as a brainstorming tool where the computer displays of the group are embedded in a special table and the group are facing each other in a reasonably normal arrangement with normal sight lines for groupwork. The group are allowed to talk and interact in any way they wanted.

Takemura & Kishino (1992) proposed a cooperative work environment using a virtual workspace (created by virtual reality technology) where more than two people can solve problems cooperatively, including design strategies and implementation issues.

(v) Computer-aided Design Tools

Shu and Flowers (1992) proposed a system, *Teledesign*, that allows people to simultaneously modify a common design in a graphically rich environment with the purpose of examining groupware interface issues unique to three-dimensional computer-aided design. Experiments with this system confirmed that a simultaneous mode of edit access is preferred over a turn-taking mode for two-person interactions. Also, independent points of view between designers optimised parallel activity.

(vi) Pen-based Meeting Support Tools

We-Met (Window Environment-Meeting Enhancement Tool) (Wolf & Rhyne, 1992) is a prototype pen-based tool designed to support both the

communication and information retrieval needs of small group meetings. *We-Met* provides a shared drawing area in which several users can work at the same time.

4.2.2.2 CSCW Tools for the Implementation Cycle

(i) Editing Tools

Usually, each member of a project is responsible for one or several modules, split among a number of files. At some point during development, one member may need to use another member's module, for instance to test a program or system. This often results in a need to access each other's files for modification. Some modifications are only for the sake of testing (like adding trace instructions) and need not be known by the file's owner. Other modifications change the behaviour of a program (such as bug fixes, or small changes in functionality) and need to be integrated by the file's owner, provided he/she agrees on the changes. The first type of modification is temporary, while the latter is structural (Beaudouin-Lafon, 1990).

With traditional tools, these modifications are usually handled by making a copy for local modifications or by asking the file's owner to do it. The first solution leads to many modified copies that may become hard to integrate and the second solution may lead to small conflicts that impair the efficiency of the group. What really is needed is software support in the development environment for the shared editing of locally modified versions of files.

A number of systems exist that support multi-user editing of documents. Some of them are real editors, while a number of others are based on hypertext concepts. *CES* (Greif et al., 1986) and *Grove* (Ellis et al., 1988) are editors that provide different levels of concurrency control, and both edit documents with a simple structure. In *CES* the nodes of this structure correspond to the level of concurrency in terms of the fact that a writer has a lock on a node. In *Grove* the

concurrency is at keystroke level as the changes are immediately visible to all users. *Mercury* (Kaiser et al., 1987) does not allow concurrent access as each user has write access to a given set of modules. When a module is changed, the users who import it (*Mercury* knows the dependencies between the modules) are notified of the changes so that they can update their modules accordingly. All three systems lack a good version control system. According to Beaudouin-Lafon (1990), hypertext systems have too flat a structure to efficiently support program editing. Hypertext systems are not truly collaborative in the sense that one does not see the changes that are being made to a hypertext frame by another member, and changes may not be recorded when somebody else has changed the frame in the meanwhile. Moreover, documents produced with such a system are not structured, and it would therefore be difficult to recreate a program source code from a set of frames. According to Beaudouin-Lafon (1990), version management systems are not suitable for multi-user editing either, because integration may be difficult.

(ii) Debugging

Although most of the editing, debugging and testing activities are individual, locating an error (or bug) often involves several members of the project team. Several people may be needed to identify a problem, or to uncover a bug in a team member's module that he/she is unable to solve. Several people debugging together usually sit down around the same workstation, struggling for the keyboard and mouse to run the program. This process would be much improved if each member could interact with the same program through his own workstation.

This capability may be used as well to communicate a running program to a team member once a bug in the module has been uncovered. Some error situations are difficult or impossible to reproduce so the only solution often is to demonstrate it at one specific workstation. It would be ideal if the program instead of the

programmer is moved. To some extent, shared windows may comply to the needs of shared debugging and communication of a running program. Today's network transparent window systems like the X Window System[1] are a sound basis for shared window capabilities. Another possibility is the building in of the shared capabilities inside the application itself. Many constraints may be imposed by window sharing like a fixed size for the same window on different screens. This is because the semantic models of window systems are not adapted to intelligent window sharing. For instance, a window does not know its contents because window systems use direct drawing. On the other hand, requiring applications to support multi-user input and multi-view windows can be essential, unless a good graphical library hides these aspects from the application.

(iii) Program Development and Testing

In modern software development projects, software developers may be widely separated geographically while having significant amounts of computing power on their individual desks. Geographically dispersed software developers which form part of the same project team working on some system or subsystem, may find it necessary to share programs (for reuse) or test programs over a network.

Computational email (the embedding of programs within electronic mail messages) promises to alleviate the problem of remote installation at separately administered sites, the problem of getting users to "buy in" to new applications, and the problem of extemely heterogeneous user interaction environments (Borenstein, 1992). The basic idea of computational email is very simple. Instead of sending plain text, or even multimedia information in a mail message, a computer program in a well-defined language is sent instead. Mail-reading software at the recipient's end recognises (via a well-defined mechanism) that the mail message is software which may be executed. Usually, when the recipient

[1] The X Window System is a trademark of MIT (Massachussetts Institute of Technology)

reads the mail message, the message may show some introductory text. This tool may also be very useful in setting up meetings. The software executed may engage the recipient in a question/answer dialogue and may request him to select the dates and times most suitable to him for a meeting. The program might then mail the answers back to a network server that collects similar answers from participants, in order to find a meeting time suitable to all of them.

(iv) End user Programming

In the participatory development process, not only are the users involved in the analysis and design cycles, but they become designers by giving them end-user programming tools. Malone, Lai, and Fry (1992) describe a "radically tailorable" tool which allow end users to create a wide range of different applications. Users create applications by combining four kinds of building blocks: *objects, views, agents,* and *links.*

(v) Building of Collaborative Applications

Application developers may employ the *DistView* toolkit (Prakash & Shim, 1994) to convert existing object-oriented applications to collaboration-aware applications with minimal effort or to create new collaboration-aware applications from scratch. The building of multi-window applications are supported in which users can share some of their application windows with other users while still keeping other application windows private. *DistView* uses a replicated object approach that ensures good interactive response times and keeps network bandwidth requirements low.

(vi) Merging of Documents and Version Control

The need to merge different versions of an object is common in collaborative computing. In the course of collaboratively producing a document or some other

artifact, people often find that they have created different versions. The objective is to have one correct version. The set of revisions should then be taken from one version and re-applied to the other version of the object. There is a need for a tool that could possibly point out the differences between the versions, that could perform the merge automatically, and respond appropriately to conflicting changes. Existing merge tools are either limited being based on plain text files, or are not adaptable to particular collaboration contexts. Munson and Dewan (1994) have developed a flexible object merging framework that allows definition of the merge policy based on the particular application and the context of the collaborative activity. It supports automatic, semi-automatic, interactive merges, semantics-determined merges, operates on objects with arbitrary structure and semantics, and allows fine-grained specification of merge policies. Tools for merging plain-text files include the UNIX *diff3* tool and the *GINA* collaborative application framework (Berlage & Genau, 1993) which allows users to merge revised versions by merging command histories.

4.3 A Software Development Environment supporting Cooperative Work

The idea of a Software Development Environment (SDE) as a comprehensive, integrated set of tools supporting the complete software development process has been the topic of research over the last number of years. Supporting the software development process means to support the modelling techniques of the process, and the development and maintenance of all kinds of documents including requirements specifications, software design specifications, code listings, technical documents, and manuals. The ultimate goal of a SDE is to improve the quality of the final product, to support reuse in and across software projects and to free developers from routine work (Peushel et al., 1991).

A major challenge for future SDEs is the support of (possibly large) teams of software developers who may even be geographically distributed. Such distributed software environments exist, but the available machine support is usually

restricted to local or wide area networks and corresponding low-level protocols that only enable simple file transfer, rudimentary configuration management support, and a mail system (Peushel et al, 1991). The need is for coordinating access to shared information on different levels of granularities (e.g. from complete systems of modules or documents down to procedure definitions in a single module). There is also a need for dedicated message servers for conveying information about project states, critical tasks to do, and getting feedback. Examples of research projects which tackled the problem of team support are MARVEL (Kaiser and Feiler, 1987), ARCADIA (Belz et al., 1988), ALF (Benali et al., 1990), and MERLIN (Peushel, 1991). A further achievement of such an environment is the computer-supported integration of development and management activities. It is envisaged that the petri net-based DesignNet Model of the OISEE project which presents the the development process model at three levels, as previously illustrated in Figure 3.1, should be computer-supported for the integration of development and management activities.

4.3.1 Generic Facilities

This section outlines the facilities that a software development environment must provide in order to meet the goals of flexibility and active support. These generic facilities were compiled by Kaplan et al. (1992) in a paper which reports on the development of an open, flexible and active support environment for software development based on the *ConversationBuilder*. The first set of activities is concerned with users being able to work on and relate among arbitrary sets of activities. These facilities are necessary to:

- Provide the ability to specify new kinds of activities to the system. The specification of such situations are called protocols.

- Allow the user to have as many activities running simultaneously as is useful.

- Allow the user to relate activities to one another as is suitable, in order to make an activity subordinate to another, or group of other activities.

- Be able to impose obligations on other users, or other contexts, from any context, and be able to manipulate one's own obligations. This entails handing the obligations over to others, changing their context, declaring them complete and refusing offered obligations.

- Allow the user to switch among activities at will and as effortlessly as possible.

The second set of facilities is concerned with helping the user to determine his current position in the system, and the options available to him. These facilities are to:

- Help users determine how they entered a particular context.

- Indicate the legal actions which may be performed in a given context.

- Perform any of the legal actions.

- Indicate to the user the obligations which must be met before a task is complete.

- Enable the user to understand the relationships (if any) that exist among the data that are present in the system.

- Enable the users to understand the relationships among all the contexts in which they are involved.

- Allow members of a group to be aware of what other group members are

doing.

The third set of facilities is miscellaneous, but necessary. These facilities are to:

- Allow the construction of arbitrary networks (hypertext systems) of data, in which both nodes and links can be typed, so that both shared and private data of various types can be modelled in the system.

- Continue to provide traditional support facilities, such as compilation, version control, editing and mailing.

- Allow the incorporation of new tools as required by users.

4.3.2 Cooperative Models for Software Development

Repositories are the central information servers for software development environments. Early repositories generally provided the service of storing evolving objects. However, there is evidence especially, in the software engineering domain, that repository technology will soon be used to integrate whole environments, even organisations (Jarke et al., 1992). Figure 4.6 illustrates these environments which incorporate human agents, their local workplaces and tools, and structured communication either directly by message-passing over the network or indirectly by shared information in the repository. In such cases, the repository matures to an active center of communication for complex cooperative development processes.

With development in the area of computer communications, information represented in different formats, such as voice, graphics, images and text can be processed, stored, retrieved and distributed. In the area of communications, new technologies are available for developing integrated networks capable of providing the level of service required by different media.

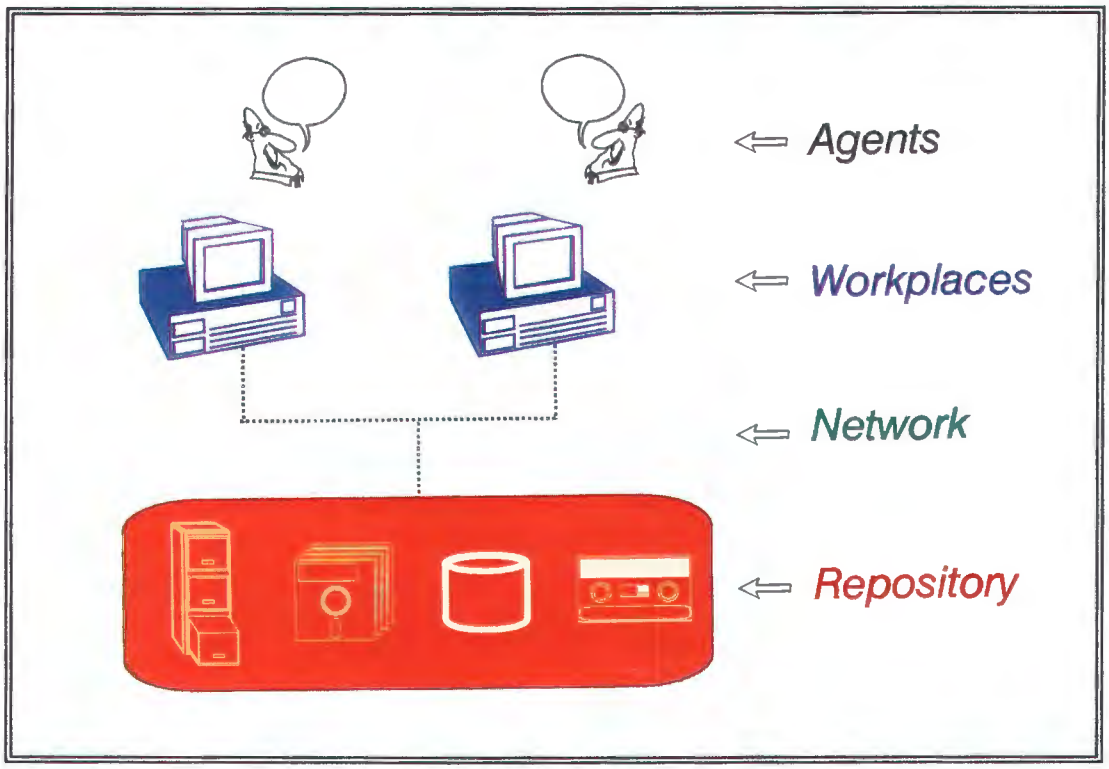

Figure 4.6 *A typical Software Development Environment (Jarke et al., 1992)*

The appearance of new technologies in the area of communication is motivated by technology transformations in the following domains (Karmouch, 1993):

- The emergence of high-speed networks with powerful workstations and new storage technology imposes a new way of processing information.

- Distributed system configurations are possible where several powerful workstations share resources at different sites by communicating through local area networks (LANs). Other configurations may cover wider areas such as LAN interconnection through MANs (metropolitan area networks) and wide area networks (WANs).

- Faster networks, high performance processing and storage systems directly manipulate new types of information such as video, voice and image, all

integrated in a single entity (the so-called multimedia document).

- Optical fibre technology has overcome the limitations of current networking technology in providing high speed networks, permitting the transfer of large amounts of multimedia information over a single channel in an integrated and synchronised manner.

The architecture that is proposed for computer-supported cooperative work in software development should be able to fulfil the future needs of multimedia distributed cooperative work. The architecture is adapted from an architecture established by Brodie and Ceri (1992) for the new generation information systems called Intelligent and Cooperative Information Systems (ICISs). These systems provide forms of cooperation and intelligence. Intelligent and Cooperative Information Systems will involve large numbers of heterogeneous, intelligent agents distributed over large communication networks. The agents may be humans, humans interacting with computers, humans working with computer support, and computer systems performing tasks without human intervention. Core technology (previously described as services provided by middleware in Chapter 2) required to support the advanced features of ICISs include (Brodie & Ceri, 1992):

- Development environment tools (UpperCASE, MiddleCASE, LowerCASE, data management and access tools)

- Repository tools (DBMS, OODBMS, KBMS, file systems, distributed object management, libraries).

- Presentation services/user environment (windows, forms).

- Communication infrastructures (RPC, peer-to-peer messaging, queued messaging, X-400, mail).

- Environment management (security, control, resources, scheduling).

- Distributed Computing Environment Tools (network services, communication services)

Ideally, these services should be transparently available to any information system or IS development environment in the distributed computing environment. Collectively, the systems providing these services are called middleware and the principle is that as many services as possible should be available via sharable servers. Figure 4.7 gives an illustration of the layers of the corresponding distributed architecture.

A multimedia configuration, as established by Karmouch (1993) for cooperative applications is depicted in Figure 4.8. Three separate networks are used - Ethernet for data, PBX for voice, and Broadband for video. When high speed networks such as FDDI, DQDB and B-ISDN become commercially available, all of the media may possibly be integrated into one network. An extension of the workstation level is a separate TV monitor for displaying video. A videowindow may also be integrated in the graphical screen, depending upon the image quality required for an application. Servers may be classified to fall within two categories: information servers and communication servers. Information servers can be further classified into a database server and various storage servers.

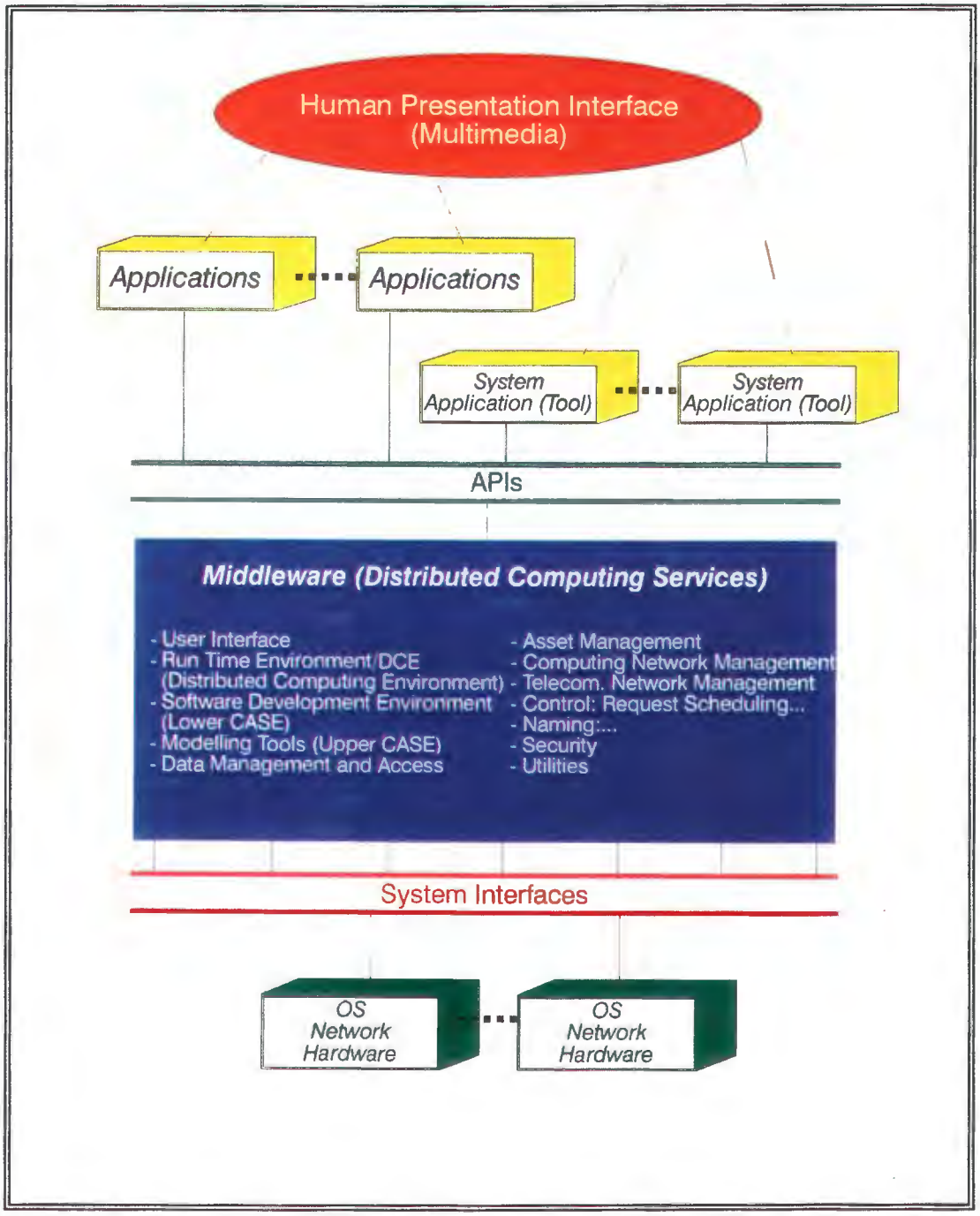

Figure 4.7 *Distributed Computing Architecture with Middleware (Brodie & Ceri, 1992)*

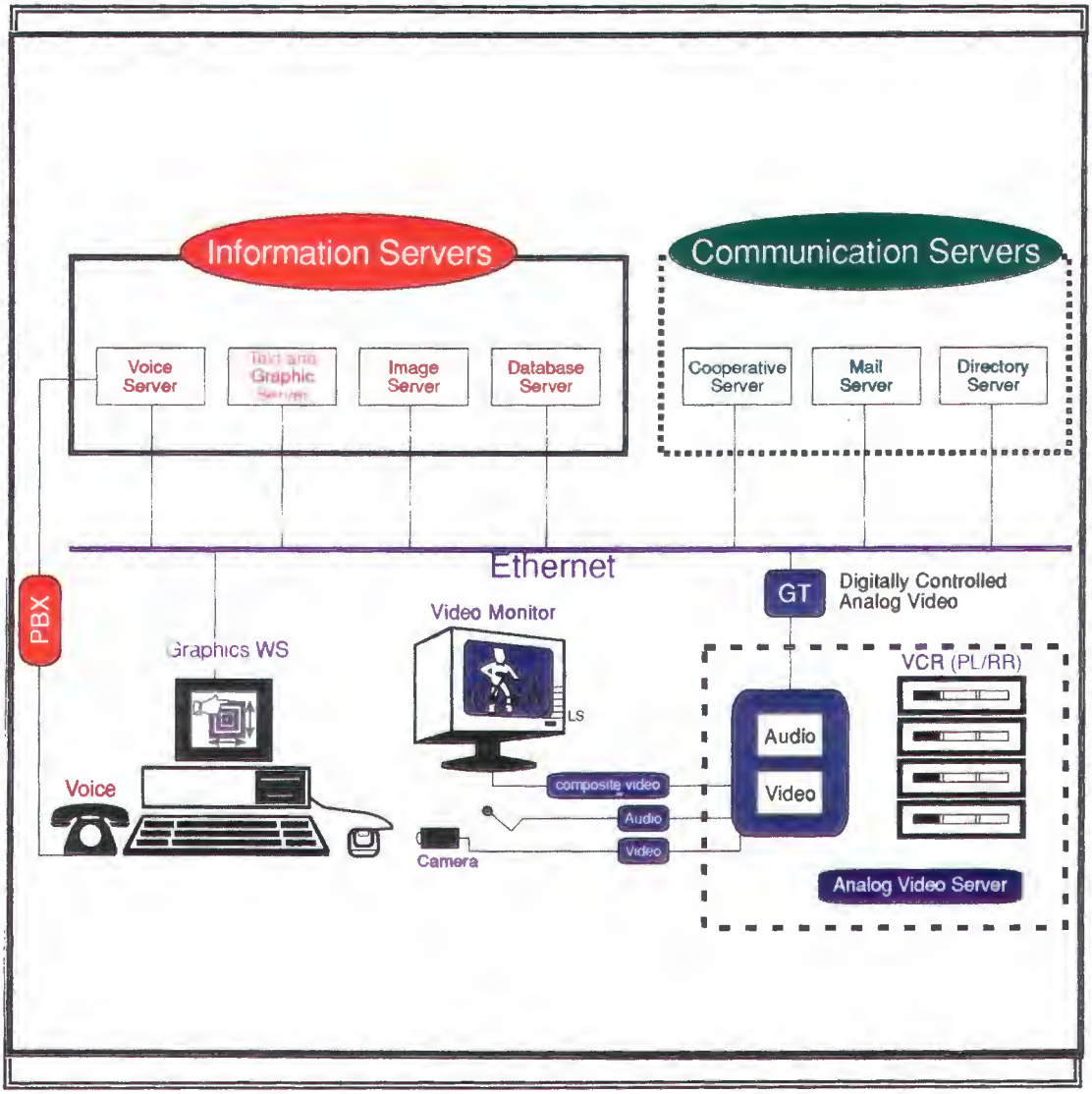

Figure 4.8 *Multimedia Platform for Cooperative Applications (Karmouch, 1993)*

4.4 Summary

Now, that a clear understanding of the CSCW technology suited to the software development process has been obtained, the CSCW technology meta primitive for software development that previously could not be specified in full may now be incorporated in the CSCW software development conceptual model. The CSCW conceptual model for software development will be the subject of Chapter 5.

CHAPTER 5

A CSCW Conceptual Model for Software Development

CONTENTS

5.1. Introduction

Within the context of cooperative support systems the software development process constitutes a natural and important application. The paradigm of CSCW for this particular application domain as it was perceived during the investigation can only be defined when all the conceptual primitives comprising it are given theoretical foundations. In the previous chapters, the two main conceptual primitive classes, namely *cooperative work* and *computer support* were determined for the software development process. The conceptual model representing the CSCW paradigm for software development can now be completed.

This chapter will first consider the proper integration of conceptual components in terms of an integrated model of groupwork in software projects. Having established the integrated meta model for cooperative software development, the formally related conceptual model may be constructed.

5.2 A Meta Model of Cooperative Software Development

A comprehensive model of the software development process should contain:

- *Knowledge* about the software engineering domain in the form of a semantic specification.
- A *group* model representing the interactions within a task-oriented *problem solving* team.
- A model of *social* activities that groups perform, representing interactions in the form of argumentation debates, meetings and agreement on contracts, and their monitoring.

The CoNex project has developed an approach to integrate different tasks encountered in software projects using a conceptual modelling strategy (Hahn et al.,1990). This strategy will be used to construct the meta model of cooperative software development and consists of submodels, including:

- A conceptual model of *software development processes* representing explicit logical dependencies between requirements, design and implementation decisions.
- A conceptual model of *groupwork* suited to the needs and particulars of software project work.
- A conceptual model of *task-oriented debates* with *multiple agents*. This model makes explicit the reasons behind logical dependencies related to requirements, designs, and implementation decisions, versions and configurations.

- A conceptual model of *task contracting* and contract supervision representing long lasting transactions in the project team by a semantic model of project coordination.

The submodels are united in a meta model of the cooperative software development process.

This section is organised as follows. The modelling of groupwork structures and conversations, and software engineering is done. Then, the integration of these submodels is presented. The object-oriented data model notation of Hahn et al. (1990) will be used for the meta level of cooperation support.

5.2.1 A Groupwork Model for Task Cooperation

Task-oriented, multi-agent collaboration is considered when identifying the requirements for a groupwork model for task cooperation:

- The *dynamic* assembly of teams of experts depends on the type of problem (software development in general) to be solved and the required skills involved, and the actual availability of individuals or material resources.

- *One* individual may be associated with *several* teams simultaneously according to different issues, obligations and levels of competence.

- Solutions to (sub)problems are devised and delivered in *parallel.*

- The rationale of a task to be solved consists of problem and task definitions, feasibility constraints and approvals; nevertheless, these are all subject to *social* negotiations.

- The definition and execution of tasks are *plan* guided.

- Facilities have to be devised to control *incomplete* and *conflicting* problem solving solutions coming from individual team members.

Given these requirements the groupwork model is composed of the conceptual entities and relations in Figure 5.1. The groupwork model is derived from two perspectives, namely the organisational perspective and the project perspective.

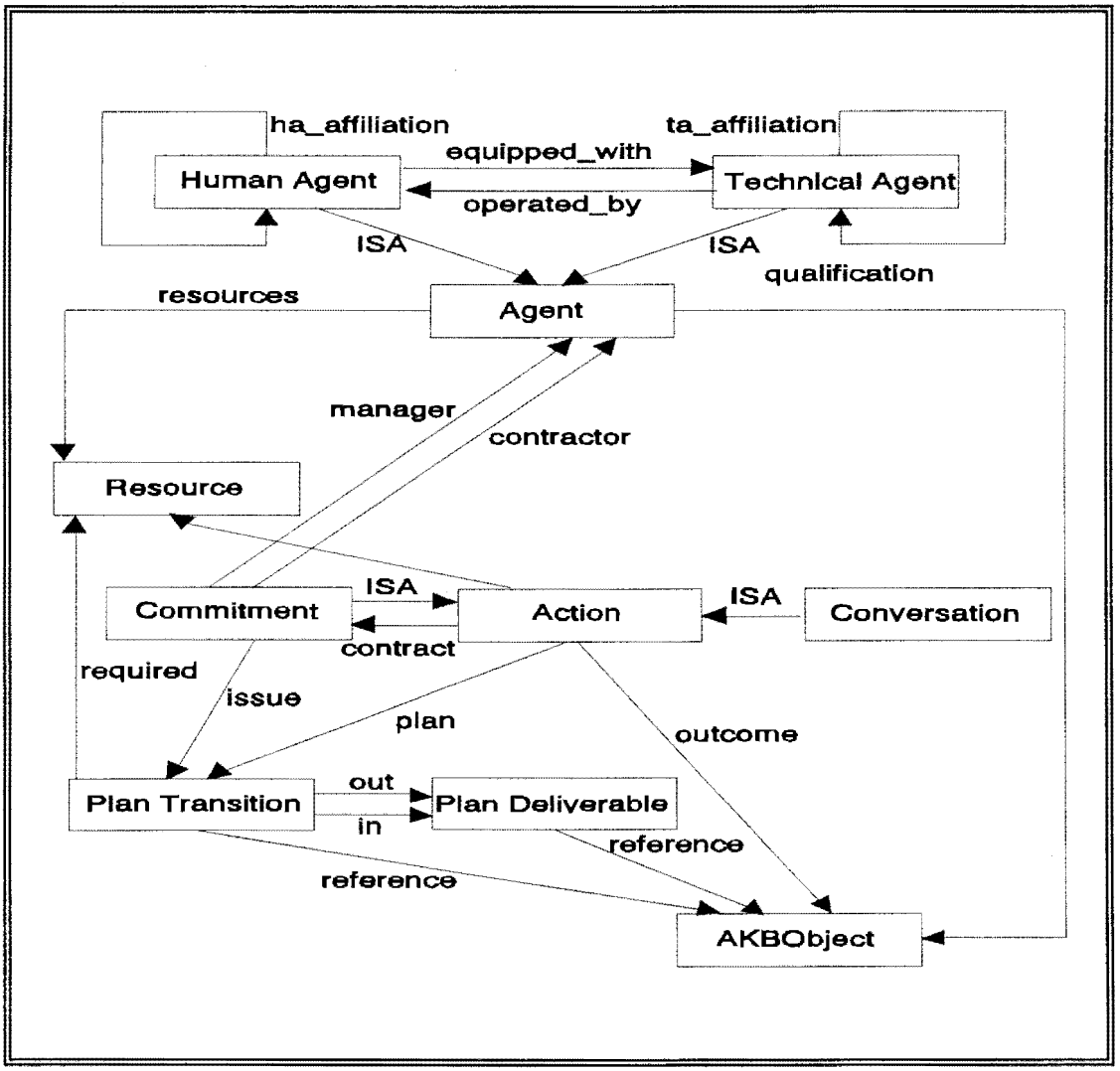

Figure 5.1 *Groupwork Model: Organisational and Project Perspective (Hahn et al., 1990)*

Organisational Perspective: Groups consist of a number of *Agents*. From a problem solving viewpoint an individual *Agent* of a group is characterised by a competence level (*qualification: AKBObject*), which are available in the Application Knowledge Base. *Resources* (in the form of technical equipment, money and time) are assigned to an *Agent*. Two basic types of *Agents* are distinguished, namely *HumanAgents* and *TechnicalAgents*. Each of them may be considered either individually or as a group. Single persons dynamically form a human group and various tools can be configured to a compound technical group. The modelling of the agents includes relevant dependencies between the subclasses. A *TechnicalAgent* can only be handled (*operated_by)* by a *HumanAgent* with appropriate qualification. The availability (*equipped_with*) of a *TechnicalAgent* for a *HumanAgent* should also be shown.

Project Perspective: Various *Resources* are necessary in order to execute a project task. Besides common economic resources, such as time, money, and technical equipment, *Agents* should also be considered as resources to be managed. Projects are performed through mutually reconciled *Actions* which should be planned. The groupwork model contains various levels of granularity to specify, to accept, and realise plans by means of activities. On the technical level, plans are represented by *PlanTransitions* and the results manifested in the form of *PlanDeliverables*. Plans are created and modified in the course of negotiations (*conversation*). The focus is not on planning as such, but on various social policies to *agree* upon plans (by contracting), to *modify* already settled plans as a result of new evidence (by debating and argument exchange) and to *monitor* plan execution. The *actions* within a plan and the execution of an action according to a plan is based upon a social *contract*. The agents (*manager, contractor*) involved agree upon the *issue* of negotiation in terms of what has to be done, how it should be done, and when it has to be done. These task oriented negotiations are subject to *Conversation* about actions. Specific applications of the group model, for example software project management, require the domain knowledge to be plugged in at the level of the application specific knowledge base.

5.2.2 A Multi-Agent Conversation Model for Task-Oriented Negotiations

The coordination of plans and actions is based on communication among agents. Two qualitative techniques are used to control the interactions that task-oriented groups often use to achieve or modify agreements:

- *Conversation for action* aims at direction of people. Messages are used to assign plans to people, to make binding commitments for the achievement of a project goal, to implement a plan in terms of the required activies, and guarantee proper termination of task oriented activities.

- *Conversation for negotiation* aims at negotiation among people. This communication mode entails opinions to be exchanged in terms of debates for the determination and coordination of goals, and to agree upon an activity to be done through argument exchange and final decision making.

The model is illustrated in Figure 5.2. An *Action_conversation* is characterised by the exchange of *messages* among several *participants* on particular topics inherited from the *Conversation* object.

A *sender* and *receiver* component form part of *messages* stating the immediate *Agents* involved in the conversation, and a characterisation of the issues (*concerns*) dealt with. *Action_conversations* are special types of *Actions* and have a particular specification for admitted sequences of conversation steps. Therefore, in addition to covering the traditional view of realising work plans by task oriented activities, plans also incorporate the discourse level for plan modification and task assignment as an integral part of project activities. Several primitives of the basic *Message* type exist within such a conversation plan including *request, counter,* and *reply*. Loosely structured argumentation-based debates (*Possibility_Conversations*) is a second major conversation type. In this case *messages* are replaced by *arguments*. *Possibility_Conversation* is a special kind of

action. An argumentation plan, thus, consists of a sequence of *argument* primitives each one characterised by its *contributor.*

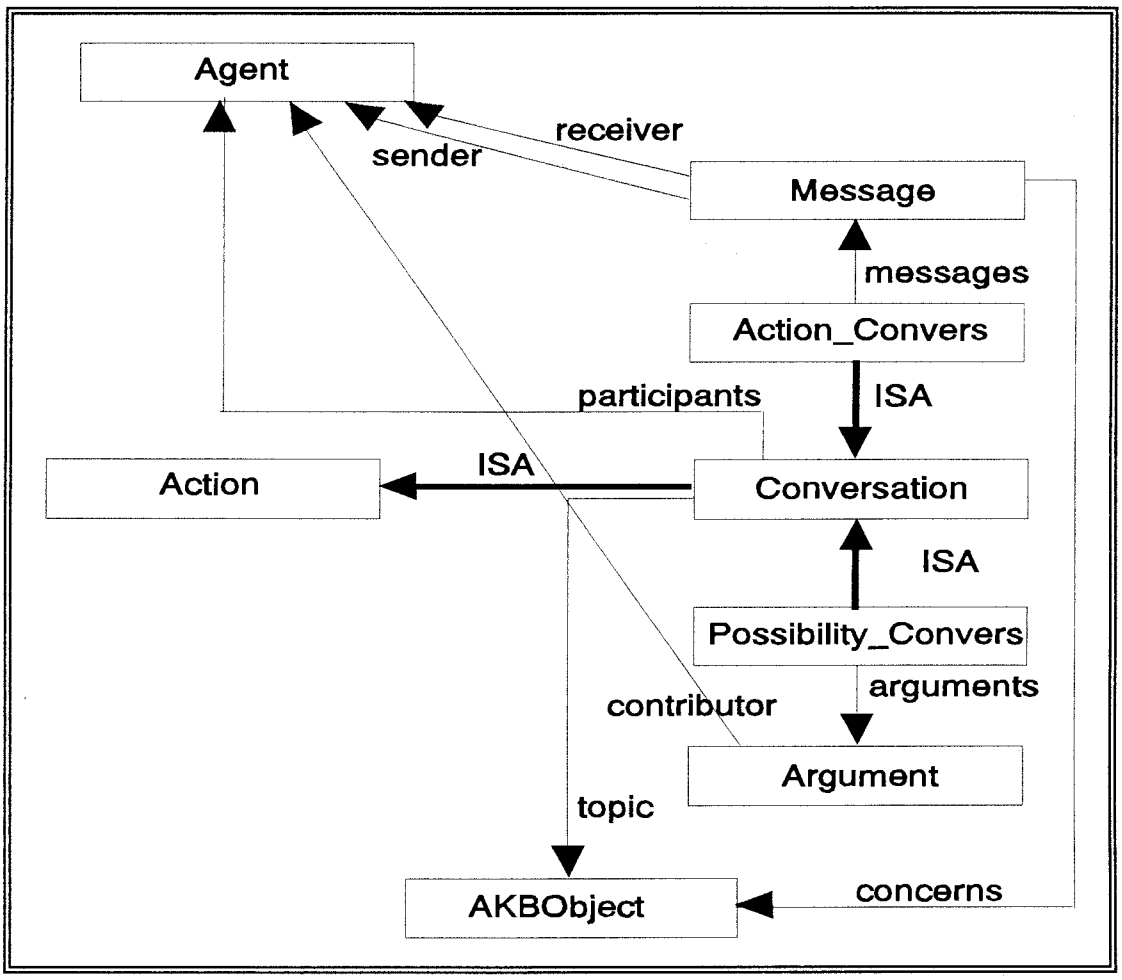

Figure 5.2 *Multi-agent Conversation model (Hahn et al., 1990)*

5.2.3 Software Process Data Model

The software process data model introduces the software engineering domain. According to the substantial literature in the field (e.g. Du Plessis, 1992), the following requirements should be satisfied by such a model:

- Recording of *administrative* aspects of software objects, in performing

requirements analysis, design, implementation, and documentation. It also includes the recording of design decisions, including source management.

- Recording of *semantic* aspects of software objects and design decisions, including the semantic *dependencies* created by design decisions. These include refinement decisions, mapping decisions, versioning decisions, and configuration decisions for reuse in maintenance.
- Integrity *control* and partial rule-based *automation* of administrative and content-oriented actions.
- Integration (i.e. administrative, semantic, and technical) of externally developed *design tools*.
- The definition of supported languages, design methodologies, and tools should be extensible.

There are also other requirements of less interest to this study. Figure 5.3 illustrates the basic idea of a software process data model. Software development and maintenance is viewed as a process of *tool*-aided *decisions* that transform software *objects* into other objects. The derived objects are then considered to be justified by the underlying design decision.

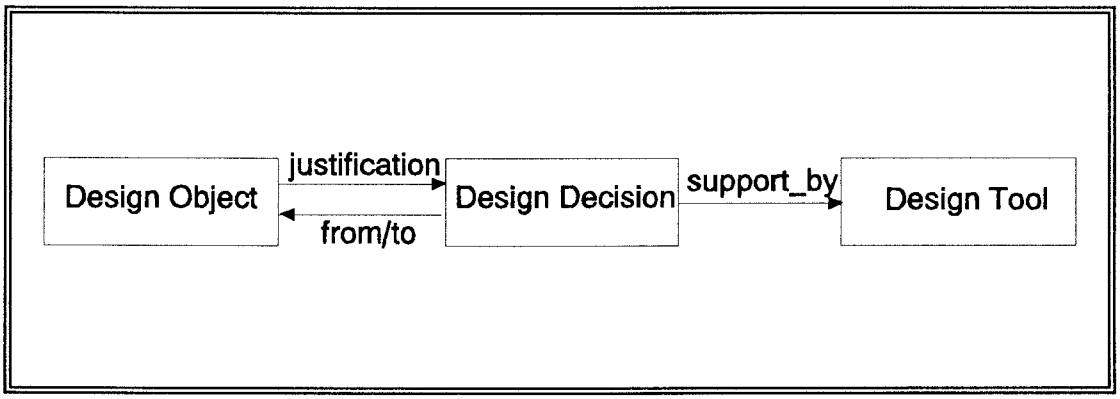

Figure 5.3 *Software Process Data Model (Hahn et al., 1990)*

Design objects and design decisions have a *reference* to an uninterpreted information container which can be worked on by certain tools. The design

objects and design decisions also come with a *semantic description* of the content of this container. Objects can be associated with *triggers* that activate directly the tools that support some decision applicable to the object. An instance of the model would define a certain software environment with its methodologies (decision classes), languages (object classes), and tools. The execution of an actual software project would be modelled as an instantiation of this software environment model. A limitation of the model is that it does not cover the *group work* aspects of software engineering although the concept of design decision provides a good handle. The integration is achieved through design decisions in the form of argumentations and contracts.

5.2.4 Model Integration

Figure 5.4 illustrates the conceptual integration of the various submodels. The group model and the conversation models intersect in the *Agent*, the *Action*, and the *AKBObject* areas. In a cooperative model multiple *Agents* converse. The linking of *Conversations* to *Actions* provides a general protocol scheme for *Actions*. The integration does not provide any contents in terms of what people converse about and act upon. The *AKBObject* is open with respect to a particular interpretation of its domain. This changes when the model must serve a specific application domain.

The application domain addressed by this study is the software development process. Requirements, designs, and implementations are formal objects of a *software development knowledge base*. The transformations from requirements to designs and from designs to implementations are formally controlled by mapping decisions specified by a *conceptual software development model.*

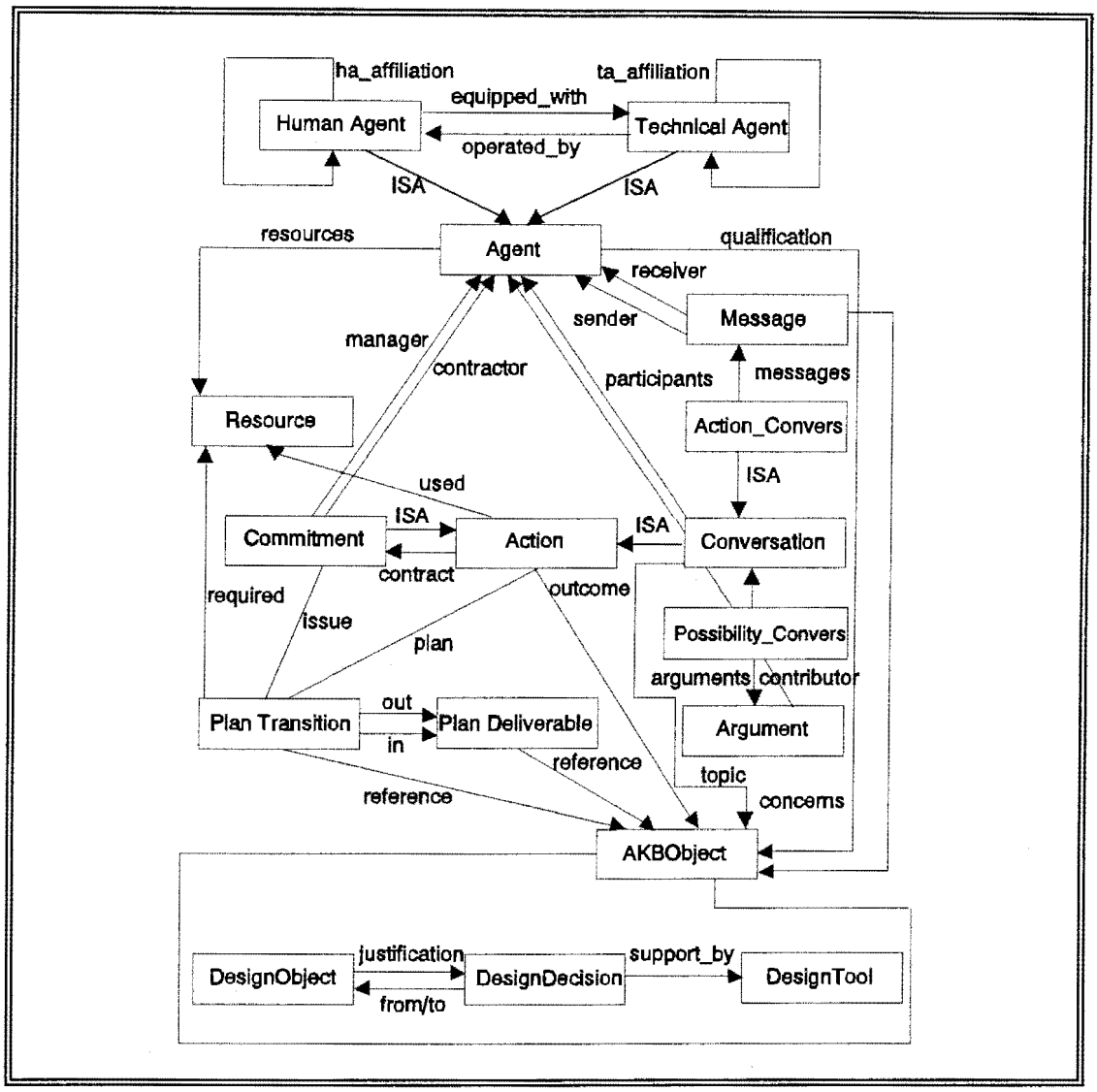

Figure 5.4 *Model Integration - Groupwork in Software Projects (Hahn et al., 1990)*

5.3 CSCW Conceptual model for Software Development

The conceptual model on which this study is based appeared in its original form in Figure 1.1 of Chapter 1. The conceptual model was then instantiated with the meta primitives of the CSCW paradigm derived from the main components and dimensions of CSCW in Figure 2.23 of Chapter 2. In Chapter 3 most of the software development primitives for the second instantiation of the conceptual model have been determined, except for the *groupware technology* primitive. The

conceptual model could therefore not be completed. Chapter 4 had to fill the gap by an extensive discussion of CSCW technology. Now, that a clear understanding of the CSCW technology suited to the software development process has been obtained, the groupware technology meta primitive for software development may be incorporated in the CSCW software development conceptual model.

5.3.1 Computer Support Primitives for Software Development

The two main primitives of computer support for software development are *software development hardware* and *software development software*. In the CSCW realm, the software development hardware typically consists of *basic hardware* components which are required for development work, as well as *groupware technology* to support software development. The basic hardware includes hardware required for a *software development environment*, and *data processing hardware*. Data processing hardware refers to basic hardware components which would provide computer support to a data processing department, including:

- *Workstations in the form of micro computers or terminals*
- *Network facilities, including network servers*
- *Mainframes*
- *Processors*
- *Peripherals, including printers.*

Architectural components which form part of the basic hardware include the information store, organisational database or repository, and the information and communication servers. Software development environments generally require additional hardware components, such as:

- *Graphic screens, for the creation of graphical models*
- *A mouse peripheral, for graphic capabilities*
- *A colour graphics printer, for professional documentation.*

The groupware technology supporting cooperative work during software development includes hardware technology that should be added to the basic hardware components for the support of the group activities of software teams. In the software development application domain two main categories of groupware technology are distinguished, namely *groupware support for project management* and *groupware support for collaborative software development.* Groupware support for project management includes hardware components, such as:

- *Video equipment*
- *Audio equipment*
- *Telephones*
- *Telefaxes*
- *Speakers*
- *Microphones*
- *Electronic whiteboards*

Section 4.2.1 of Chapter 4 provides examples of the use of the above groupware technology. Turning now to groupware support for collaborative software development, the following hardware components can be mentioned:

- *Electronic whiteboards*
- *Videophones*
- *Computer-based paint editor*
- *Digitiser screens and digitiser pens*
- *Shared video drawing media*
- *Shared workspaces, such as editors, windows*

Sections 4.2.2.1 and 4.2.2.2 of Chapter 4 provides examples of the use of the above groupware technology. The architectural components which may be added are the CSCW managers (information, domain, activity, security, and multimedia managers) as shown in Figure 2.15 of Chapter 2.

Software development software may be classified as *system software, application software, middleware,* and *groupware* in the CSCW realm. The system software typically consists of *operating system software, utility software,* and *network software.* The application software refers to the *software system in the process of development* and can be either customised software systems or software packages. Middleware consists of general functions or *services provided by sharable servers.* The services include *repository services, presentation services, security services, library services*, and other services mentioned in section 2.8 and shown in Figure 4.7. Middleware also include software development environment tools including *UpperCASE, MiddleCASE,* and *LowerCASE.* Other software development tools include *program generators, programming environments,* and *fourth generation languages.*

Groupware software for the application domain of software development include tools from the general categories:

- *Communication mechanisms*
- *Shared workspace facilities*
- *Shared information facilities*
- *Group activity support facilities.*

In Chapter 4 groupware technology for software development was discussed in terms of three categories:

- *CSCW tools for project management (section 4.2.1)*
- *CSCW tools for the analysis and design cycles (section 4.2.2.1)*
- *CSCW tools for the implementation cycle (section 4.2.2.2).*

Many interesting applications of groupware software which may be suitable for the support of group activities during software development were discussed in the above sections.

5.3.2 Cooperative Work Primitives for Software Development

The CSCW application domain is *software development* and *software development participants* perform cooperative work in this domain. Within the software development domain, the types of cooperative work may be either *managerial*, *creative*, or *physical and technical*. The nature of the cooperative work refers to the groupwork performed during *project management*, *software analysis and design*, and *implementation*. Table 5.1 summarises the specific work as an attribute of the type of work. The attributes are derived from the relevant parts in section 3.5 of Chapter 3 and section 4.2.2.

MANAGERIAL WORK	CREATIVE WORK	PHYSICAL / TECHNICAL WORK
PROJECT MANAGEMENT COOPERATIVE WORK	SOFTWARE ANALYSIS & DESIGN COOPERATIVE WORK	IMPLEMENTATION COOPERATIVE WORK
Management meetings Scheduling Planning Project & post audit reviews Project demonstrations Documentation	Software team meetings Interviews - users Documentation - shared edit Modelling - shared workspace Prototyping	Demonstrations Prototyping - shared workspace Programming - shared workspace Shared testing Debugging

Table 5.1 *The Type and Nature of Cooperative Work*

It has been mentioned in Chapter 2 that a broadly accepted framework for the study of CSCW systems classifies the type of work according to the dimensions of *time* and *location of work*. Cooperative work in software development may for example take place at the same time and the same place as in face-to-face project meetings, or at different locations at the same time as in video conferencing or shared workspace applications, or at different places at different times as in electronic mail applications. The dimension of time is classified as *synchronous* or *asynchronous*, while the location is classified as *remote* or *co-located*.

The *software development participants* fulfil different *roles* within the context of

cooperative work. In Chapter 3, the people building block of information systems indicate four main categories of participants in Figure 3.5, namely *system owners, system users, system designers,* and *system builders*. Table 3.1 provides a summary of the roles and responsibilities of participants in the software development process.

The software development participants perform cooperative work on a specific *level*, either on a *management* or *technical* level. Management may take place on the *universal level*, the *worldly level*, and the *atomic level*. In the software development application domain, the universal level refers to the upper management of the organisation in which or for which the software is developed, thus the system owner. The worldly level refers to middle management which includes the head of the DP department, and possibly the system users management. The project leader and some system users will manage the software project on the atomic level. The software development team consisting of system analysts, system designers, programmers, database administrators and possibly domain specialists performs operational or technical work.

All the relevant primitives of the CSCW paradigm for software development have been derived. The completed second instantiation of the conceptual model is represented in Figure 5.5.

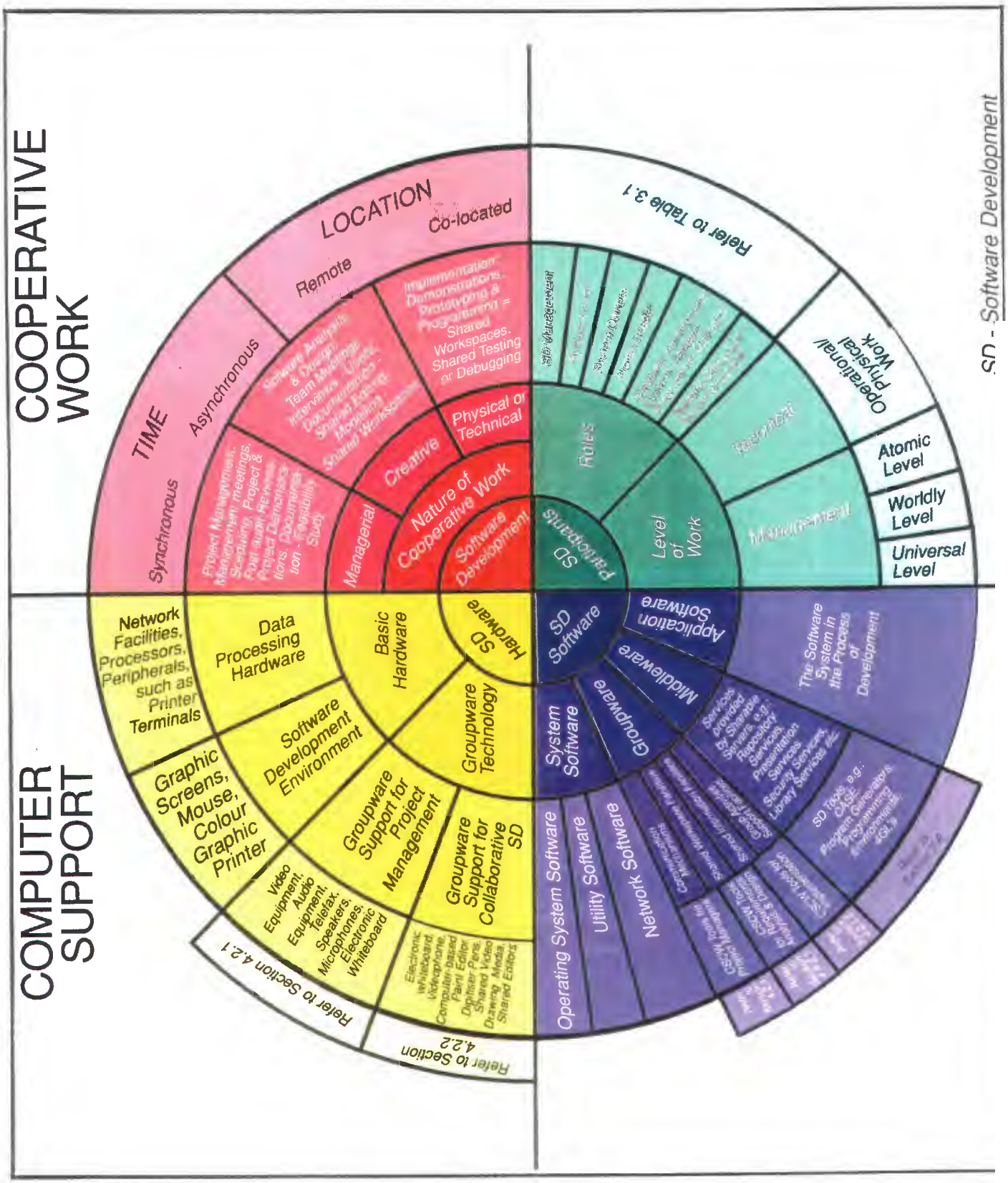

Figure 5.5 *CSCW Conceptual Model for Software Development*

5.4 A Framework for Software Development within a CSCW Environment

A summary of the research results is provided in the form of tables representing a framework for software development within a CSCW environment. Communication facilities (networks, e.g. LANs, WANs, MANs) are used by most of the CSCW tools. However, network facilities were classified as basic hardware facilities in the conceptual model, and therefore they will not be shown in the tables of the framework. The framework can be handy for future references on CSCW tools suited to the different cycles or stages of software development. The framework can also be extended as new CSCW tools for the application domain become available.

The framework consists of three tables which address different group activities (nature of cooperative work) within project management or a specific development cycle. The roles of participants involved in a specific group activity, the CSCW tools (software tool) suitable for the group activity, and the corresponding groupware technology (hardware facilities) are described.

Nature of Cooperative Work	Roles of Participants	CSCW Tools (software)	Groupware Technology
• Project & post-audit reviews • Monitoring status of project	Software managers Project leader Software team (optional) Users	• Semantic and contractual analysis tool • Variant and version management tool	• Graphical capabilities • Shared workspaces
• Argumentation & decision-making	Software managers Project leader Systems analyst	• Variant and version management tool • Decision aid tool • Optimisation tools • Simulation program	• Graphical capabilities • Shared workspaces
• Construction of CSCW environments	Software managers Project leader Systems analyst	• Groupware toolkits	• Graphical capabilities • Shared workspaces • Open architecture • Various groupware
• Communication • Scheduling	Software managers Project leader	• Message systems • Meeting scheduler systems • Electronic mail • Calendar systems • Collaborative writing tool	• Video equipment • Audio equipment • Telephone • Video phone • Telefax
• Awareness & coordination	Software managers Project leader	• Active information generation tools • Shared editors • Electronic mail • Authoring system • Edit log tools • Conversation tools • Workflow management • DesignNet model	• Graphic capabilities • Shared workspaces • Video phone • Telephone
• Group meetings	Software managers Project leader Software team (optional) Users	• Transcription tools • Electronic meeting systems • Video Conferencing tool	• Video equipment • Ubiquitous audio • Telephone • Speaker • Whiteboard • Microphone
• Discussions • Event calendars • Task tracking	Software managers Project leader	• Telephone bulletin-board • Video-mediated communication	• Telephone • Video equipment • Camera • Actuator

Table 5.2 *CSCW support for Project Management*

Nature of Cooperative Work	Roles of Participants	CSCW Tools	Groupware Technology
• Communication	Project leader Software team	• Message systems • Electronic mail • Mail programs • Artifact-centered information spaces • Knowledge-based design environment	• Multimedia • Audio equipment • Telephone • Video phone • Graphic capabilities • Shared workspaces
• Modelling	Project leader Systems analysts	• Shared drawing tools	• Graphic capabilities • Shared workspaces • Electronic pens • Digitiser pens • CRT-based rear projection display • Transparent digitiser sheet • Electronic whiteboard
• Documentation	Project leader Systems analysts	• Documentation tool • Collaborative writing tool • Collaborative editing tool	• Shared workspaces • Graphics capabilities
• Problem solving	Project leader Software team	• Electronic Brainstorming • Shared text editor • Virtual workspace	• Electronic whiteboard • Shared workspaces • Graphics capabilities • Video equipment
• Collaborative Design	Systems analysts Software team	• Computer-aided design tools • Shared drawing tools	• Electronic whiteboard • Shared workspaces • Graphics capabilities
• Group meetings	Users Software managers Project leader Software analysts Software team	• Electronic meeting systems • Video conferencing • Video-mediated communication tool • Shared drawing tools • Pen-based meeting support	• Video equipment • Ubiquitous audio • Telephone • Speaker • Microphone • Camera • Actuator • Electronic whiteboard • Electronic pen

Table 5.3 *CSCW support for the Analysis & Design Cycles*

Nature of Cooperative Work	Roles of Participants	CSCW Tools	Groupware Technology
• Editing	Programmers Systems analysts	• Shared editor tool • Hypertext systems	• Video equipment • Video phone • Graphic capabilities • Shared workspaces
• Debugging	Project leader Systems analysts Programmers	• Shared windows • Network transparent window systems	• Graphic capabilities • Shared workspaces
• Program development & testing	Project leader Systems analyst Programmer	• Computational email • Shared windows • Network transparent window systems • Shared editing	• Graphic capabilities • Shared workspaces
• End user programming	End-users Systems analyst	• End-user programming tools	• Graphic capabilities • Shared workspaces
• Construction of collaborative applications	Project leader Systems analysts Programmers	• Shared window systems • Replicated object tools	• Shared workspaces • Graphic capabilities
• Merging of documents and version control	Systems analysts Programmers	• Merge tools • Object merging framework	• Shared workspaces • Graphic capabilities

Table 5.4 *CSCW support for the Implementation Cycle*

At the 1994 CSCW Conference it was reported that there are over 300 CSCW products available on the market. The above tables represent only examples of CSCW tools available to support the cooperative activities of software development. Some promising tools are still in a prototyping phase showing the potential of the rapid-growing field of CSCW.

5.4 Summary & Conclusions

The conceptual model has been kept in its basic form, but went through an evolution process through the instantiation of the original primitives. The second instantiation illustrates the validity of applying CSCW in the software development process. The framework for software development within a CSCW environment can be useful for future references on CSCW tools suited to the different cycles or activities of software development. The framework can also be extended as new CSCW tools for the application domain become available.

CHAPTER 6

Summary, Evaluation and Conclusions

CONTENTS

6.1. Introduction

This chapter looks back on the investigation and relates the work done in each chapter for achieving the objectives of the study. A summary of the research results of the investigation is presented. The original hypothesis and assumptions are validated in the light of these results. Conclusions are drawn and the chapter concludes with proposed areas for further investigation.

6.2 Summary of Investigation

The investigation is based on the hypothesis that it is possible to provide computer-based support for the cooperative elements of the software development process. In support of the hypothesis, a conceptual model illustrating important aspects of the CSCW paradigm was constructed. The conceptual model formed an integral part of the study as it was used throughout the dissertation to illustrate research results concerning the main issues of the

investigation. The relevant issues which have bearing on the investigation were identified and as a result the subject of each chapter was determined. A motivation for the area of investigation was constructed and a method of investigation was established to guide the investigation. The assumptions for the investigation were based on the suitability of the use of the object-oriented paradigm for software development and cooperative work, and the adoption of the revised spiral model proposed by Du Plessis and Van der Walt (1992). Keeping the constraints of the investigation in mind, a possible solution was proposed for dealing with the problems of supporting the cooperation and group activity of software teams during the software development process.

A thorough investigation of the main study area, namely computer supported cooperative work, was done. The origins of CSCW were discussed and the important issues of groupwork analysed. The nature of CSCW and some ongoing debates concerning the research area were examined. The main CSCW technologies were surveyed and the requirements and main architectural components of a CSCW environment described. The meta primitives of the CSCW paradigm were derived and resulted in the first instantiation of the conceptual model.

Another important aspect of the research was the roles of participants and the issues surrounding cooperation and group activity during software development. The possibility of incorporating CSCW explicitly into the software development process is postulated and eventually proven to be feasible.

The computer-based support of the group activities during software development was explored to determine the CSCW technologies suited to different cycles and activities of the software development process. The generic facilities and architectural components of CSCW software development environments were then investigated.

The crux of the investigation was to construct a conceptual model representing the CSCW paradigm for software development. The two main conceptual primitive classes, namely *cooperative work* and *computer support* were determined for the software development process. This, and the integrated meta model for cooperative software development have resulted in the establishment of the conceptual model representing the CSCW paradigm for software development. Finally, a framework for software development within a CSCW environment, based on the important conceptual primitives, was established. The framework could be useful for future reference regarding CSCW tools suited to the different cycles and activities of software development, and could also be extended as new CSCW tools for the application domain become available.

6.3 Evaluation and Conclusions

In evaluating the work done for this dissertation, the meaningfulness and contribution of the research results, especially for the software development process as a whole should be determined. As modern technology and applications are complex, it is unusual for an individual to attempt the development of a major project single-handedly. The software development process almost invariably involves cooperation that crosses group, professional, and subcultural boundaries. The complexity of the software development process itself demands that highly integrated groups of analysts, designers, and users perform their tasks in their respective roles. Nowadays, truly distributed applications seem natural in the face of a high ratio of computing power to available communication bandwidth. This also has implications for software development. The users and software developers involved in a specific software project may be widely separated geographically. The challenge is to deliver distributed solutions by means of cooperative development efforts. Therefore, the area of CSCW and advanced information technology, with its enormous capabilities for transmitting and storing information, would seem to hold considerable promise for the software development process.

The main purpose of the investigation was to determine if CSCW could be incorporated into the software development process. For this purpose the CSCW conceptual model, which went through an evolution process with each instantiation of the primitives of the paradigm, may serve as a validation model. After the analysis of the software development process in Chapter 3, it was confirmed that cooperation formed an integral part of the process, and that because of the complexities caused by the intracacies of group activities, computer-based support is necessary to support groupwork by software teams. The building blocks of information systems and their relation to the meta primitives of CSCW motivate the role CSCW has to play in software development. The second instantiation of the CSCW conceptual model, in which all the CSCW primitives were detailed with software development attributes, illustrates the validity of applying CSCW in the software development process.

6.4 Areas for further Investigation

Concerning the study area of CSCW itself, there are many options for further investigation. The multi-disciplinary nature of CSCW makes it possible for researchers to pursue one of the following directions:

- *Group theory*, in which the group process is studied (Gladstein, 1984)

- *Activity theory*, a philosophical framework for studying different forms of human praxis as development processes, with both individual and social levels interlinked (Kuutti, 1991).

- *Ethnography*, which is concerned with the "workaday" character of work in all its richness and variety (Darnton, 1995).

- *Managerial Cybernetics*, which takes a broad perspective on CSCW when it emphasises collaborative work in an organisational context (Darnton,

1995).

- *Office systems*, where the group activities of an office environment are supported by relevant CSCW technologies.

- *Workflow management*, which provides the infrastructure to design, execute, and manage business processes on a network (Abbott & Sarin, 1994).

Concerning further branches of the investigation presented in this dissertation, the following may be worthwhile mentioning:

- Empirical projects on the incorporation of CSCW into the cycles of software development should be conducted. The requirements engineering process in the analysis cycle would be a good candidate project.

- Emerging groupware technologies hold considerable promise for the utilisation of CSCW in many different application domains. Suitable application domains should be explored.

- The role of CSCW in distributed software development environments, particularly in respect of distributed applications, may be further investigated.

Anybody with an interest in the complexities of the way in which people work and the variety of technologies to support groupwork will find interesting topics for further investigation.

REFERENCES

Books:

Booch, G. 1994. *Object-oriented Analysis and Design. With Applications.* Second Edition. Benjamin/Cummings Publishing Company, Inc.

Coad, P., and Yourdon, E. 1991. *Object-oriented Analysis.* Second Edition. Yourdon Press.

Conger, S. 1994. *The New Software Engineering.* Wadsworth Publishing Company.

Couger, J.D., Colter, M.A., Knapp, R.W. 1982. *Advanced System Development / Feasibility Techniques.* John Wiley & Sons, Inc.

Dahl, O.J., Dijkstra, E.W., Hoare, C. A. R. 1972. *Structured Programming.* New York:Academic.

De Marco, T. 1978. *Structured Analysis and System Specification.* New York:Yourdon Press.

Forsyth, D.R. 1990. *Group Dynamics.* 2nd edn. Brooks/Cole Publishing Company, Pacifc Grive, CA.

Gane, C. and Sarson, T. 1979. *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, NJ:Prentice-Hall, Inc.

Greif, I. 1988. *Computer-Supported Cooperative Work: A Book of Readings.* San Mateo, California: Morgan Kaufmann Publishers.

Humphrey, W. S. 1989. *Managing the Software Process.* Addison-Wesley.

Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G. 1992. *Object-Oriented Software Engineering.* Addison-Wesley

Johansen, R. 1988. Groupware: *Computer Support for Business Teams.* (The Free Press, New York).

Jordan, E.W., Machesky, J.J., Matkowski, J.B. 1990. *Systems Development. Requirements, Evaluation, Design and Implementation.* PWS-KENT. Boston.

McDermid, J. 1991. *Software Engineer's Reference Book.* Butterworth-Heinemann.

McGrath, J.E. 1984. *Groups: Interaction and Performance.* Prentice-Hall, Inc., englewood Cliffs, NJ.

Myers, G.J. 1978. *Composite/Structured Design.* New York, NY:Van Nostrand Reinhold.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. 1991. *Object-Oriented Modeling and Design.* Prentice-Hall International, Inc.

Steiner, I.D., 1972. *Group Process and Productivity.* Academic Press. New York.

Suchman, L. A. 1987. *Plans and situated Actions.* Cambridge University Press.

Whitten, J.L. & Bentley, L.D. & Barlow, V.M. 1993. *Systems Analysis and Design Methods*. Derde uitgawe. Irwin

Yourdon, E. and Constantine, L. 1979. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Englewood Cliffs, NJ:Prentice-Hall, Inc.

Contributions in Books:

Bannon, L., & Schmidt, K. 1991. CSCW: Four characters in search of a context in: J.M. Bowers & S.D. Benford (eds). *Studies in Computer Supported Cooperative Work.* North-Holland/Elsevier, Amsterdam.

Steiner, I.D. 1976. Task performing groups, in *Contemporary Topics in Psychology.*, edited by J.W. Thibaut, J.T. Spence, and R.C. Carson. General Learning Press, Morristown, NJ.

Journal Articles:

Abbott, K. R., & Sarin, S. K. 1994. Experiences with workflow management: Issues for the next generation. *Proceedings of the Conference on Computer Supported Cooperative Work.* October, 22-26, 1994. Chapel Hill. North Carolina. USA.

Bannon, L. J. 1993. CSCW: An initial exploration. *Scandinavian Journal of Information Systems*. 5:3-24.

Bannon, L., & Schmidt, K. 1992. Taking CSCW seriously: Supporting articulation work. *Computer-Supported-Cooperative Work (CSCW).* Kluwer Academic Publishers, The Netherlands, 1:7-40. 4, 67-83.

Belz, F.C., Clarke, L.A., Osterweil, L., Selby, R.W., Taylor, R.N., Wileden, J.C., Wolf, A.L., Young, M. 1988. Foundations in the ARCADIA environment architecture, in *[SDE88]*, 1-13.

Benali, K., Boudjlida, N., Charoy, F., Derniame, F.-C., Godart, C., Griffiths, P., Gruhn, V., Jamart, P., Legait, A., Oldfield, D.E., Oquendo, F. 1990. The presentation of the ALF-Project, in eds. Madhavji, N., Schäfer, W. and Weber, H., *Proceedings of the First Conference on System Development Environments and Factories I*, 75-90.

Bentley, R, Rodden, T, Sawyer, P., Sommerville, I. 1992. An architecture for tailoring cooperative multi-user displays. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Toronto, Canada, Oct, Nov 1992, 187-194.

Bly, S. A. 1988. A use of drawing surfaces in different collaborative settings. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Toronto, Canada, Oct, Nov 1992, 250-256.

Bly, S.A., Minneman, S.L. Commune: A Shared Drawing Surface. *Proceedings of COIS '90*, ACM, New York, 1990, pp. 184-192.

Bodker, S., Ehn, P., Knudsen, J., Kyng, M., Madsen, K. 1988. Computer support for cooperative design. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Portland, Oregon, September 26-28,

377-394.

Boehm, B. W. et al. June 1975. Structured Programming: A quantitative assessment. *Computer*, pp. 38-54.

Boehm, B. W. 1988. A spiral model of software development and enhancement. *IEEE Tutorial on Software Engineering Project Management.*

Borenstein, N. S. 1992. Computational Mail as network infrastructure for Computer-Supported Cooperative Work. *Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work.* Toronto, Canada, Oct 31-Nov 4, 67-83.

Borenstein, N. S., Thyberg, C.A. 1988. Power, ease of use, and cooperative work in a practical multimedia system. *International Journal of Man-Machine Studies.*

Brodie, M.L., & Ceri, S. 1992. On intelligent and cooperative information systems: A workshop summary. *International Journal of Intelligent and Cooperative Information Systems.* 1(2):249-289.

Brinck, T. & Gomez, L.M. 1992. A collaborative medium for the support of conversational props. *Proceedings of the Conference on Computer-Supported Cooperative Work.* Toronto, Canada, Oct, Nov 1992, 171-178.

Bullen, C.V., & Bennett, J.L. 1990. Learning from experience with groupware. *Proceedings of the Conference on Computer-Supported Cooperative Work.* Los Angeles October 7-10, 291-302.

Buxton, W. & Moran, T. EuroPARC's Integrated Interactive Intermedia Facility (IIIF): Early experiences. *Proceedings of the IFIP WG8.4 Conference on Multi-User Interfaces and Applications.* North-Holland, Amsterdam, 1990, pp. 11-34.

Crowley, T, Milazzo, P., Baker, E., Forsdick, H., Tomlinson, R. 1990. MMConf: An infrastructure for building shared multimedia applications. *Proceedings of the Conference on Computer-Supported Cooperative Work.* Los Angeles October 7-10, 329-241.

Darnton, G. 1995. Working together: a management summary of CSCW. *Computing & Control Journal.* February 1995, 37-40.

Dowson, M, Wileden, J. 1995. ISTAR and the contractual approach. *Proceedings of the 9th International Conference on Software Engineering*, IEEE Computer Society Press.

Du Plessis, A.L. 1992. CAISE: The opportunity and the challenge. Inaugural lecture. *University of South Africa.*

Du Plessis, A.L. & Van der Walt, E., Modeling the software development process, *ISCO2 Conference, Egypt*, 1992.

Ellis, C.A., Gibbs, S.J., & Rein, G.L. 1991. Groupware: Some issues and experiences. *Communications of the ACM.* 34(1):38-58.

Engeström, Y. 1990. Activity theory and individual and social transformation. *2nd International Congress for Research on Activity Theory.* Lahti, Finland, May 21-25.

Gaver, W.W. 1991. Sound support for collaboration. *Proceedings of the Second European Conference on Computer-Supported Cooperative Work.* Amsterdam, The Netherlands, September 25-27, 293-308.

Gladstein, D.L. 1984. Groups in context: A model of task group effectiveness. *Administrative Science Quarterly.* 29:499-517.

Goldberg, Y., Safran, M., Shapiro, E. 1992. Active Mail - A framework for implementing groupware. *Proceedings of the Conference on Computer-Supported Cooperative Work.* Portland, Oregon,September 26-28, 75-83.

Gorry, G.A. 1988. Computer Support for Biomedical Work Groups. *Proceedings of the Conference on Computer-Supported Cooperative Work.* Portland, Oregon,September 26-28, 39-51.

Graham, T. C. N., Urnes, T. 1992. Relational views as a model for automatic distributed implementation of multi-user applications. *Proceedings of the ACM 1992 Conference on Computer Supported Cooperatuve Work.* Toronto, Canada, November 1992, 59-66.

Green E., Owen, J. Pain, D. 1991. Office systems development and gender: Implications for Computer-Supported Cooperative Work. *Proceedings of the Second European Conference on Computer-Supported Cooperative Work.* Amsterdam, The Netherlands, September 25-27, 33-48.

Greenbaum, J. 1988. In search of cooperation: An historical analysis of work organization and management strategies. *Proceedings of the Conference on Computer-Supported Cooperative Work.* Portland, Oregon,September 26-28, 102-114.

Hahn, U., Jarke, M., Rose, T. 1990. Group work in software products. *Multi-User Interfaces and Applications.* Elsevier Science Publishers, p 83-101.

Hughes, J., Randall, D., Shapiro, D. 1991. CSCW: DIscipline or paradigm? A sociological perspective. *Proceedings of the Second European Conference on CSCW*, 1-16.

Ishii, H., Kobayashi, M, Grudin, J. 1992. Clearboard: A Seamless Medium for Shred Drawing and Conversation with eye contact. *Proceedings of CHI '92, ACM, New York, 1992, pp. 525-532.*

Ishii, H. TeamWorkStation: Towards a seamless shared workspace. *Proceedings of CSCW '90.* ACM, New York, 1990, pp. 13-26.

Jacobson, I. 1987. Object-oriented development in an industrial environment. *Proceedings of OOPSLA '87.* SIGPLAN Notices, 22(12), p 183-91.

Jarke, M., Maltzahn, C., Rose, T. 1992. Sharing processes: Team coordination in design repositories. *International Journal of Intelligent and Cooperative Information Systems.* 1(1):145-167.

Joosten, S., and Brinkkemper, S. 1993. Modelling of working groups in Computer Supported Cooperative Work. *Proceedings of the 18th International Conference on Information Technologies and Programming.*

Kaiser, G.E., and Feiler, P.H. 1987. An architecture for intelligent assistance in software development. *Procedure of the 9th International Conference on Software Engineering.* Monterey, California, p 180-188.

Kaplan, S.M., Tolone, W.J. Carroll, A.M., Bogia, D.P., Bignoli, C. 1992. Supporting collaborative software development with ConversationBuilder. *Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work.* Toronto, Canada, Oct 31-Nov 4, 11-20.

Karmouch, A. 1993. Multimedia distributed cooperative system. Computer Communications, 568-580.

Kling, R. 1991. *Cooperation and Control in Computer Supported Work.* Dept. Information and Computer Science. Univ. California, Irvine.

Kuutti, K. 1991. The concept of activity as a basic unit of analysis for CSCW research. *Proceedings of the Second European Conference on Computer-Supported Cooperative Work.* September 25-27, Amsterdam, The Netherlands, 249-264.

Kyng, M.. 1988. Designing for a dollar a day. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Portland, Oregon,September 26-28, 178-188.

Lai, K-Y, Malone, T.W. 1988. Object Lens: A spreadsheet for cooperative work. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Portland, Oregon,September 26-28, 115-124.

Lyytinen, K. 1990. *Computer Supported Cooperative Work - issues and challenges. A structurational analysis.* Manuscript. Univ. of Jyväskylä, Dept. Computer Science.

Malone, T.W., Benjamin, R.I., Yates, J.. 1987. Electronic markets and electronic hierarchies. *Communication of the ACM*, 30:357-370.

Malone, T.W., and Crowston, K. 1990. What is coordination theory and how can it help design cooperative work systems. *Proceedings of the Conference on Computer Supported Cooperative Work*. Los Angels, October 7-10, 357-370.

Mantei, M., Baeker, R., Sellen, A. Buxton, W., Mulligan, T. 1991. Experiences in the use of a media space. *Proceedings of the CHI '91*, ACM, New York, 1991. 203-208.

Marmolin, H., Sundblad, Y., Pehrson, B. 1991. An analysis of design and collaboration in a distributed environment, 147-162.

Navarro, L., Prinz, W., Rodden, T. 1993. CSCW requires open systems. *Computer Communications.* May 1993. 16(5):288-297.

Nolle, T. 1993. Groupware: The next generation. *Business Communications Review,* 23(8):54-58.

Olson, J.S., Card, S.K, Landauer, T.K., Olson, G.M., Malone, T, Leggett, J. 1993. Computer Supported Cooperative Work: Research issues for the 90s. *Behaviour & Information Technology*. 12(2):115-129.

Parnas, D.L. April 1975. *The inluence of software structure on reliability* in Proceedings 1975 Int, Conf. Reliable Software. pp. 358-362.

Peuschel, B., Schäfer, W., Wolf, S. 1991. A knowledge-based software development environment supporting cooperative work. *International Journal of Software Engineering . 2(1):79-106.*

Rodden, T & Blair, G. 1991. CSCW and distributed systems: The problem of control. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Portland, Oregon,September 26-28, 25-29.

Root, R.W. 1988. Design of a Multi-media vehicle for social browsing. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Portland, Oregon,September 26-28, 25-29.

Sathi, A., Morton, T. E., Roth, S. F. 1986. Callisto: An intelligent project manegement system, *AI Magazine*. 7, 5, 34-52.

Schmidt, K. 1991. Riding a tiger, or Computer Supported Work. *Proceedings of the Second European Conference on CSCW*, 1-16.

Schmidt, K., and Bannon, L. 1992. Taking CSCW Seriously: Supporting articulation work. *Computer-Supported Cooperative Work (CSCW)*. 1:7-40.

Shu, L. & Flowers, W. 1992. Groupware experiences in three-dimensional computer-aided design. *Proceedings of the Conference on Computer-Supported Cooperative Work*. Toronto, Canada, Oct, Nov 1992, 179-186.

Stefik, M., Foster, G., Lanning, S., Bobrow, D., Kahn, K., Suchman, L. 1987. Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications of the ACM*, 30:32-47.

Suchman, L. 1989. *Notes on computer support for cooperative work.* Working paper WP-12, Univ. Jyväkylä, Dept. Computer Science.

Tang J.C., and Minneman, S.L. 1991. VideoWhiteboard: Video Shadows to support remote collaboration. *Proceedings of CHI'91,* ACM, New York, 1991, 315-322.

Tanigawa, H., Arikawa, T. Masaki, S. and Shimamura, K. 1991. Personal multi-media multipoint teleconference system.

Thomas, J. C., & Kellogg. 1989. Minimising ecological gaps in interface design. *IEEE Software, 6,* 78-86.

Watabe, K., Sakata, S., Maeno, K., Fukuoka, H. Ohmori, T. Distributed Multiparty Desktop Conferencing System: MERMAID. *Proceedings of CSCW '90,* ACM, New York, 1990, 27-38.

Wilson, P. 1991. Computer Supported Cooperative Work (CSCW): origins, concepts and research initiatives. *Computer Networks and ISDN Systems.* 23:91-95.

CSCW Proceedings

Proceedings of the Conference on Computer-Supported Cooperative Work. September 26-28, 1988. Portland, Oregon. ACM.

Proceedings of the Conference on Computer-Supported Cooperative Work. October, 7-10, 1990. Los Angeles, CA. ACM.

Proceedings of the Second European Conference on Computer-Supported Cooperative Work. September, 24-27, 1991. The Trippenhuis Kloveniersburgwal 29, Amsterdam. Kluwer Academic Publishers.

Proceedings of the Conference on Computer-Supported Cooperative Work. October, 7-10, 1990. Los Angeles, CA. ACM.

Proceedings of the Conference on Computer-Supported Cooperative Work. October, 31 to November 4, 1992. Toronto, Canada. ACM.

Proceedings of the Conference on Computer-Supported Cooperative Work. October, 22-26, 1994. Chapel Hill. North Caorlina. USA. ACM.

** Note: This dissertation includes many references to CSCW tools and environments which are either mentioned or described within articles in the various *Proceedings*. The list is very long, and therefore, if a reference does not appear in the *journal article list*, it was taken from the Proceedings.