Easing the writing task:

designing computer based systems to help authors

Steven Robert Andrew Jones

Department of Computing Science and Mathematics

University of Stirling

Submitted in partial fulfilment of
the degree of Doctor of Philosophy

July 1994

Abstract

An increasing number of people interact not only with computers, but *through* computers. Interaction between people through computers to complete work tasks is termed Computer Supported Cooperative Work (CSCW).

The scope of activities supported by CSCW systems is described, and CSCW systems which support communication, meetings and writing are discussed. More specifically, the potential for improved computer support of the writing task is investigated. It is concluded that models of the writing task and writers are not yet sufficiently accurate to be embedded in normative computer programs or systems; individual writers and writing tasks are extremely varied. Leading on from the studies of both existing systems and writing theories, requirements for generic CSCW systems, single author support systems and multiple author support systems are presented.

The design of CSCW systems which support asynchronous collaborative authoring of structured documents is investigated in this thesis. A novel approach to design and implementation of such systems is described and discussed. This thesis then describes MILO, a system that does not feature embedded models of writers or the writing task. In fact, MILO attempts to minimize constraints on the activities of collaborating authors and on the structure of documents. Hence with MILO, roles of participants are determined by social processes, and the presentational structure of documents is imposed at the end of the writing process.

It is argued that this approach results in a workable, practical and useful design, substantiating the view that 'minimally-constrained' CSCW systems, of which MILO is an example, will be successful. It is shown that MILO successfully meets the stated requirements, and that it compares favourably with existing collaborative writing systems along several dimensions.

The limitations of work presented in the thesis are discussed, leading to suggestions for future work which will remedy deficiencies and extend the work which has been undertaken.

The nature of this thesis's contribution to CSCW in general, computer supported collaborative writing, and Human Computer Interaction (HCI) is discussed.

Acknowledgements

Without the support and encouragement of many people this thesis would not have been started, and without continued support and encouragement it would not have been completed.

Thanks must go to the Department Of Computing Science at the University of Stirling for tolerating my presence for more than seven years. To Jim Cowie and Richard Bland for initially encouraging my interest in HCI. To Muriel, Julie and Jane for the conversations.

Special thanks must go to my close friends without whom the past seven years would not have been half as rewarding or more importantly half as much fun. For conspiring to make no moment a dull one love and thanks to (in no particular order) Pete Cropper, Neil Mudd, Julie Edgar, Sara Bryant, Mark Hargraves, Chris Magennis, Chris Smith, Louise Pope, Cynthia Soong, Jane Laverick, Frank Gribben, Andrine Moran, Christine Wallis, Paul Atkins, Jerry Ward, David McGowan, Tony Weir, Graham Wilson, Bert Logan, Gordon Omand, Cathy Lewis, Liz Bradshaw, Shona Richardson, Jonathon Lewin, Cathryn O'Neill, Allyson MacCormick, Catriona Cumming, Conor Jameson, John McNally, Steve Mason, Trish Carlin, Carolyn Hilliard, Sandra Cameron, Martin Todd, Neil Morrison, Elena Groll, Rachel O'Sullivan, Steve Dakin, Paddy Minne, Dan Sturdy, Steve Marsh, Richard Weller, Ewen Robertson, John Wood, Judith Suggett.

Lynne Coventry who answered all the questions with patience above and beyond the call of office-mate duty is deserving of special appreciation.

Special thanks and best wishes to Andy Cockburn whose friendship and advice have been very much appreciated.

The advice and interest of Harold Thimbleby, my supervisor, has always been generous and valuable and has prompted many new ideas and opened many new avenues of interest to me. Many thanks.

Finally the greatest thanks and love goes to my parents and grandparents whose love and support (financial and otherwise) has always encouraged and enthused me.

O joy, O new vertigo of difference, O my platonic reader-writer racked by a most platonic insomnia, O wake of finnegan, O animal charming and benign. He doesn't help you think but he helps you because you have to think for him. A totally spiritual machine. If you write with a goose quill you scratch the sweaty pages and keep stopping to dip for ink. Your thoughts go too fast for your aching wrist. If you type, the letters cluster together, and again you must go at the poky pace of the mechanism, not the speed of your synapses. But with him (it? her?) your fingers dream, your mind brushes the keyboard, you are borne on golden pinions, at last you confront the light of critical reason with the happiness of a first encounter.

Umberto Eco, Foucault's Pendulum

Preface

This research has been undertaken between October 1988 and July 1992, being funded by the Science and Engineering Research Council for the initial three years of that period. It has been carried out in the Department of Computing Science and Mathematics at the University of Stirling, Scotland, under the supervision of Professor Harold Thimbleby.

The work presented in this thesis is composed by the author and embodies work carried out by the author and is not included in any other thesis. It should be noted, however, that part of this work has been published in a form similar to that which appears here. (Thimbleby *et al.*, 1992; Jones, 1993; Jones, 1991; Jones, 1992b; Jones, 1992a; Cockburn & Jones, 1991).

The two collaborative writing systems described in this thesis, *Sticky-Notes* and MILO, have been developed by the author of this thesis.

Appendix A (Using MILO) should be viewed as part of the author's contribution to awareness of Human-Computer Interaction issues. It is designed with the goal of providing *user-centered* documentation, and as such should be considered part of the thesis.

Appendix B (Supplementary Material) presents material which is not directly within the scope of this thesis. It does, however, demonstrate that this thesis is not the sole contribution of the author to the research and scientific communities during the period that this research has been carried out.

Part of this work has been published or developed in collaboration with other authors. The design principles for collaborative systems (see Section 2.9.2) were developed equally in collaboration with Dr. Andrew Cockburn in a co-authoring task supported successfully by MILO (see Section 4.7). Praise or criticism for this portion of the work should be shared equally between us.

Insights into the suitability of HyperCard for prototyping tasks, and its design flaws (see Section 4.6.1) were the result of intensive use of HyperCard by myself and colleagues. My contribution was initial background work, including initial drafts of the final paper.

The majority of this thesis has been written using MILO itself. In the present case, the

chosen formatting system is LaTeX but MILO is easily adaptable to provide for a variety of formatting systems.

MILO has been used by writers other than the author of this thesis and his immediate colleagues. A member of the Computing Science Department at the University of Stirling used MILO between July 1992 and June 1993 to write plays and poems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

1.1.1 Interaction with computers

It is likely that you have interacted with a computer in some way today. It is more likely that you have interacted with a computer *program* today. You may not have even been conscious that this interaction was taking place. It is no longer necessary to sit before a computer screen and keyboard in order to interact with a computer program. You have done just that by using an autoteller machine, a video cassette recorder, a microwave oven, a compact disc player or a whole host of other everyday objects.

Many years ago the mode of interaction with computer programs was more noticeable. To be able to indicate a sequence of instructions to a computer and to specify information to be operated upon using a large stack of punched cards was at one time a development for the better. As the power of computing systems has increased, so has the potential for increasing the usability of computer systems and programs. From punched cards and paper tape to teletype terminals, to cathode ray tube displays and keyboards, to high resolution graphics terminals and pointing devices, to multimedia workstations, to gesture controlled systems, technology has, in theory, paved the way for computer systems to become more sophisticated and usable. Indeed, developers of commercial hardware and software now invest a great deal of resources to ease people's interactions with computer programs.

A new-found interest in usability on the part of both vendors and consumers can be attributed, to some extent, to the greater availability of computer systems outwith the arenas of academia and large corporations. This can be seen to be the result of decreasing prices and miniaturisation of components leading to the potential for people to own an affordable, powerful desktop computer. The potential marketplace for computer systems has increased many-fold, but has also become more sophisticated. "Ease of use" of computer systems has become an issue with users and potential users—manufacturers can no longer sell products without devoting or being seen to devote resources to investigating how to make them easier to use.

Ease of use is just as important in the field of consumer electronics, but different interaction techniques have been developed to provide the most appropriate interface. So, to be interacting with a computer program, it is not necessary to be using something that looks like a computer. Many everyday electronic appliances now have computer programs embedded within them. This is a strong justification for devoting time and attention to problems of computer system usability and interface design.

1.1.2 Communication through computers

You may have interacted *with* a computer today, but it is also likely that you have interacted with other people *through* a computer or computer program. If you have used the telephone to talk to someone directly, left a message on a telephone answering machine or sent messages via electronic mail on a computer system you have been using computer technology to distribute information to others. The technology enables interaction between people, and that interaction may be synchrous, asynchronous or a mixture of both.

In some areas, such as telecommunications, a great deal of time and effort has been expended in order to increase the efficacy and likelihood of successful communication between people. This is an equally necessary effort where computer programs are concerned. Consider the example of a word processor again. In order to communicate the results of its use to others, a document must be printed out and posted, viewed directly from the computer screen, or sent by electronic media such as computer disc. The same argument holds for work done using a spreadsheet program, a drawing program, a database program, a document formatting program and many more

application areas. These examples are symptomatic of the fact that there is little integration of support for communication or collaboration in normative computer systems.

I would contend that a large proportion of work produced by using computers will be examined at some point by someone other than the creator of the work. A large proportion of that work will be produced with the initial intent of communicating it to another person. In a sense, people tend to interact *through* computers or computer programs just as much as interacting with them. There would seem to be a case then, for investing research effort in this area, so that the methods for communication via the computer are readily available to avoid the intermediate non-computer based stage of the work process. Providing support for computer-based communication which is integrated into users' everyday work tasks would empower users, increasing efficiency and effectiveness of computer systems.

1.1.3 Collaboration using computers

Computer based work involving communication between users can lead to collaboration. Collaboration via computer systems differs from communication. *Communication* is purely the transfer of information from one person to another, where one communicant creates information and passes it to another communicant, or neither communicant is the source of the information and only processes it or passes it on. *Collaboration* is the process of more than one person working together to a common goal, and will require communication between the collaborators. The common goal might be to produce something tangible such as a document or an illustration, or it might be more abstract such as developing a theory, proof, formula or computer program. Collaborative efforts can also benefit from technological developments that directly address issues of supporting computer-based collaborations.

Research into increasing the possibilities and potential of computer based communication and collaboration processes has, until recently, been limited. Computer Supported Cooperative Work (CSCW) is an emerging field of research that addresses issues involved in providing computer support for communication, collaboration, cooperation and coordination. CSCW can refer to a variety of activities: work carried out under its banner may concern investigation of sociological and/or psychological aspects of the impact of introducing CSCW systems into

existing work environments; research into the evaluation of CSCW systems; the proposal of theories of collaboration, cooperation, communication and coordination; practical implementation of CSCW systems and a whole host of other topics. CSCW practitioners may be sociologists, psychologists or computer scientists, and—as with any multi-disciplinary field of research—this can lead to synergetic discussions and collaborations but can also introduce a healthy friction between approaches.

1.2 A computational view of collaborative work and usability

This thesis is presented from a computational viewpoint. It addresses a general issue in the field of computing science:

- How computer systems might ease and enhance collaborative work processes;

More specifically this thesis explores the provision of systems which support computer-mediated asynchronous collaborative document production.

The aim of this thesis is to advance computational understanding with respect to four issues:

1. identification of desirable and undesirable approaches in existing CSCW theory and systems;
2. the potential for embedding theories of writing and writers in computer systems;
3. principles for the design of CSCW systems;
4. the application of such theories and principles to the development of a useful and usable collaborative writing support system;

The aims of this research have resulted in it being of a multidisciplinary nature. It deals, in the main, with Computer Supported Cooperative Work, and also draws upon psychological theories of writing. In addition, it is concerned with Software Engineering (Sommerville, 1992) and system design for the development of a sophisticated program to support collaborating writers. Finally, it draws upon theories of Human-Computer Interaction (HCI) in order that the design of the human-computer interface to such a writing support program is user centered.

The practical work undertaken was not considered as, nor is it presented as a rigorous software engineering exercise. However, the development of collaborative writing software was a non-trivial design and implementation task. Hence, it was necessary to be concerned with standard software engineering practices, even though such issues are not the topic of this work and are not discussed in this thesis. However, the thesis does present new ideas intended to influence the design of generic CSCW systems and collaborative authoring systems in particular.

1.3 A brief history of this work

Initial work on the content of this thesis began in October 1988 with a consideration of contemporary issues in the field of Human-Computer Interaction. By early 1990, a focus on the relationship between computers, writing and collaboration had emerged, and a literature survery was carried out. The resulting reports can be seen in chapters 2 and 3.

Sticky-Notes (see chapter 4) was developed on an Apple Macintosh, using HyperCard, as a vehicle for exploring solutions to problems and deficiences in existing writing support systems which had been highlighted by the literature survey.

Development of MILO begin in the summer of 1990. This was carried out using the most appropriate tools available at the time: in C on monochrome Hewlett-Packard Unix workstations, with an X Window System user interface. The Xt Intrinsics, Xlib, and Hewlett-Packard Widget set (Hewlett-Packard Company, 1988) of X11 Release 2 were used. No X application interface development tool was available during the development of MILO.

By July 1991 the majority of development of MILO was complete, although enhancements continued in response to feedback from users and colleagues. By June 1992 the development work on MILO as it is presented in this thesis was complete, and the thesis itself was in, essentially, its present form.

Revisions to the thesis were carried out between February 1994 and June 1994.

1.4 Structure of the thesis

Chapter1. Introduction provides an overview of the rapid increase in the amount and diversity of computer use and computer users. It presents the aims of this research, highlights its multidisciplinary approach, and provides a brief overview of the work.

Chapter 2. The scope of CSCW defines CSCW and motivates CSCW research. The apparent lack of success of CSCW tools is reviewed and potential causes are discussed. Existing computer support of communication, meetings and writing is explored, bringing to attention implications for a system designed to support collaborative writing and communication. Collaborative writing systems are considered along several dimensions such as place and time of collaboration, constraint imposition, process/product orientation and use of networks/hierarchies to represent documents. These implications are discussed, addressing social-psychological, design, and implementation issues.

Chapter 3. An investigation into writing presents and discusses an overview of theories of the writing process. This is concerned with models of writing and writers, and also models of collaborative writing processes. A selection of strategies that writers adopt are also described. Potential problems with computer based support of writing are presented and to conclude a proposal for functionality of a generic computer based writing support system, and its integration with other systems is discussed.

Chapter 4. Two computer based writing support tools presents an argument for designing writing support systems for use *now*; that it is not necessary to wait for currently vague models of writers and writing to become suitably applicable. Specific requirements for the writing support system which is to be developed are detailed. *Sticky-Notes*, a prototype system for collaborative document authoring based on annotations is described. Issues revealed by its development are considered before MILO, a useful and usable system to support asynchronous collaborative authoring of structured documents is introduced, related to other systems, and discussed.

Chapter 5. MILO: designed for the user addresses the usability of collaborative systems, highlighting the importance of interface design. Several guidelines for interface design are detailed, and their application to the development of MILO is discussed, revealing MILO's highly usable nature.

Chapter 6. Conclusions summarises the findings of the research. Conclusions are drawn from each of the preceding phases of the work about: the state of existing CSCW tools and theory, writing theory and its applicability to computer support of writing, the design of systems to support collaborating writers, and the interdisciplinary nature of CSCW research. MILO is analytically evaluated, compared to the requirements stated in chapter four, compared to other collaborative writing systems, and some insight into its use is presented. The issue of merging the contributions of multiple authors is discussed, and the limitations of this facility in MILO are analysed.

Chapter 7. Future work details how deficiencies in MILO might be remedied and considers relevant issues which are beyond the scope of the current work yet merit further investigation. Several of these concern MILO, and its potential future development proposing extensions to not only its functionality, but also its usability through amendments to the interface. Additionally further work is proposed which is relevant to CSCW in general, discussing evaluation of cooperative systems, and the potential for embedding theories of trust in cooperative systems.

Chapter 2

The scope of CSCW

2.1 Introduction

This chapter provides an overview of issues and systems related to the computer support of co-operative work, with an emphasis on those systems which support the activities of collaborating authors.

First, a description of the activities which may be part of the collaborative process are defined, in order to clarify the tasks that computer systems should enable, support and enhance. The focus of this work is placed in the context of those activities and styles of collaborative interaction.

Second, CSCW research is motivated, stressing the importance of recognising the benefits of workgroup computing and suggesting a move away from the personal computing ethos which has been predominant in recent years.

Consideration is then given to why, in the main, CSCW systems have failed to gain widespread acceptance, followed by a discussion of the social and psychological issues affecting the design of such systems.

This core of the chapter discusses existing groupware along several dimensions. First, tools which support communication between participants in a collaborative task are considered. Then, attention is turned to those systems which aim to support meetings of collaborators. We are concerned with such systems because communication and meetings are integral to collaborative exercises, and coauthoring is no exception. In developing a collaborative writing support system

we may consider the integration of support for these processes into the system itself, or may be more concerned with how a writing system will integrate with existing systems which support those tasks.

Third, a critical analysis of existing writing support systems is presented. This is organised along several dimensions

- the style of the collaboration with respect to time; whether it is synchronous or asynchrous
- the intended location of participants; whether they are colocated or distributed
- the extent to which existing systems impose constraints on participants
- whether the system emphasises the process, or creation of a product

In conclusion, drawing from the analysis described above, requirements for the design of a collaborative writing system are presented in relation to social, psychological, design and implementation issues.

2.2 Defining CSCW

For many years, researchers have been studying aspects of CSCW, yet it has only recently been recognised and discussed as an important field in its own right. It has become a major research area for workers in academic institutions and commercial organisations alike, yet the boundaries delimiting the scope of CSCW are vague, as are descriptions of what might fall within such boundaries. So what exactly is CSCW?

CSCW is a commonly accepted acronym for *Computer Supported Cooperative Work*, yet it seems from the work carried out under its banner that it could stand for any one of several terms: Computer Supported *Collaborative* Work, Computer Supported *Communicative* Work or Computer Supported *Coordination* of Work. It might then prove useful to offer some definitions of terms and outline the kind of work covered by them.

Collaboration : participants work jointly with others on a task, but specifically on producing a literary or artistic result. This covers, amongst others, document production, joint programming tasks and graphic design.

		Timing of collaboration	
		Same time (synchronous)	*Different time (asynchronous)*
Location of collaboration	*Same place (co-located)*	meeting rooms	argumentation tools
	Different place (distributed)	video conferences	email systems

Figure 2.1: The time/location matrix of types of collaborative tasks.

Communication : the imparting or exchanging of information, and social dealings. This is not task oriented but data oriented. People give and receive information in both informal and formal manners.

Cooperation : people work and act together on a task, and share the profits or benefits from doing so. This is similar to collaboration, yet implies individuals working more independently and exchanging information when necessary.

Coordination : the act of causing people or objects to function together or in a proper order. An example of coordination is a project manager assigning team members tasks and deadlines to enable them all to meet a single completion date, and encouraging cooperation, communication and collaboration.

It can be seen that these topics are disparate, yet closely interrelate. Communication is central to the success of the others, and successful completion of a cooperative or collaborative task requires successful coordination.

The profile of cooperative work tasks can differ with respect to the location of the participants and the time at which participants carry out work on the joint task. Figure 2.2 shows four different categories of cooperative work with respect to space and time (Dix *et al.*, 1993). This is the conventional view of the space/time matrix.

However, Grudin has recently proposed a three by three matrix which considers the predictability of an collaboration (Grudin, 1994). This can be seen in Figure 2.2. Grudin himself

		Timing of collaboration		
		Same time (synchronous)	*Different time but predictable (asynchronous)*	*Different time and unpredictable (asynchronous)*
Location of collaboration	*Same place (co-located)*	meeting rooms	argumentation tools	team rooms
	Different place but predictable (distributed)	video conferences	email systems	collaborative writing
	Different place and unpredictable (distributed)	interactive multicast seminars	computer bulletin boards	workflow

Figure 2.2: Grudin's three by three matrix of collaborative tasks.

points out that this matrix is not predictably accurate. Indeed it would be hard to maintain that email communications are any more predictable than collaborative writing activities, for example.

The theoretical and implementation issues to be considered differ for each of the four dimensions or styles shown in Figure 2.2. It is asynchronous, distributed collaborations which are of most interest here, specifically those concerned with collaborative document production. That is, multiple authors of a single document making contributions to the document at different times and in different places.

Within this thesis, the term CSCW is taken to refer to the entire scope of activities described above.

2.3 Motivating CSCW research

The last decade has been the age of the Personal Computer. Astounding technological advances have seen computer power rapidly increase, whilst production costs plummet. Workers can now afford to have a machine on their desk, the power of which a decade ago was only available to a select few technical experts. The use of the personal computer has, as a result, become

widespread in many areas such as business, design, research, education and recreation. However throughout this (r)evolutionary period too much emphasis has been placed on the *personal* aspect of such machines, and of computing in general. As a result, computing hardware has generally not been used to its full potential. It is undoubtedly powerful when used on a personal, individual basis, yet a wide spectrum of other uses, applications, and functionality exist beyond.

Individuals rarely work entirely alone—interaction with others is commonplace. Computers are a powerful medium for allowing new and exciting interactions with other people, and their power should be harnessed not merely to support inter-person interactions, but to *enhance* them. Of course personal computers can be connected via a network to other local computers or to other computers elsewhere in the world—there is little hardware limitation to stop people interacting via computers—but what does limit this, however, is software that does not exploit the hardware connections to their full potential. For example, there is little point connecting two geographically distant computers if the software facilities are not available for users to transfer data from one to another, and they have to rely on written mail. People should be encouraged and empowered in their interactions with others by the technology which allows computers to communicate over large distances, and the software which will exploit this technology. Existing relationships should be enhanced and the forging of new ones encouraged with the improvement in the quality, efficacy and product of interactions of prime importance. Such a belief has motivated the research described in this thesis.

2.4 Reviewing the success of CSCW tools

The majority of work on CSCW systems (commonly termed *groupware* (Ellis *et al.*, 1991a)) has been investigative and research-oriented. There are few systems which could be classified as CSCW tools which have been commercial successes or even gained a large user base. The widespread acceptance of systems such as Lotus Notes and Windows For Workgroups could cynically be attributed to corporate power rather than anything particularly desirable about those systems. The systems which could claim to have been successful are electronic mail systems and computer conferencing systems. These are purely communicative tools, though, and provide limited support for user management of information flow.

What reasons might exist for the apparent failure of systems which support specific collaborative work tasks?

There are several possible conclusions that can be drawn

- commercial system developers have not yet fully realized the importance and commercial potential of CSCW tools;
- the interaction between people via computers is an area of computing that is research-led, rather than marketplace led;
- the current state-of-the-art in CSCW[1] tools is not as yet sufficiently developed to be welcomed into widespread use;
- there is an inherent problem in the concept of CSCW tools and how they are implemented, and current evaluation methods do not result in successful ongoing development.

Grudin (1988) addresses some of these points in a discussion of the failure of CSCW applications. He states,

> Many systems, applications, and features that support cooperative work share two characteristics: A significant investment has been made in their development, and their successes have consistently fallen far short of expectations.

He identifies problems which he claims are common to many CSCW applications and have been key factors in their failure to be widely adopted. First, applications fail because they require that some people carry out additional work in order to support their use. To compound the problem, the people who do the additional work are not those who benefit directly from the use of the application. Second, the process of designing such applications fails because designers have poor intuition about applications for multiple users. They can appreciate the use of the system from their point of view but don't see the implications for other types of users. The third problem is that experiences of failure with CSCW applications are not learned from because the applications are too complex for meaningful, generalizable analysis and evaluation.

[1] for an annotated bibliography see (Greenberg, 1991). Pal Malm provides an extensive unOfficial Yellow Pages of CSCW (an extensive list of experimental and commercial groupware products), available via ftp from gorgon.tft.tele.no.

Grudin's first point is undoubtedly a valid one in some cases. To substantiate this claim he uses the example of an automatic meeting scheduler, which requires users to maintain an electronic diary when they may not otherwise do so. The users who benefit are those who arrange meetings, but there is an overhead of extra time invested by others. In order to ensure compliance with the needs of the system in such a situation each user should benefit from any effort expended in supporting it.

Grudin's second point is that developers do not fully understand the implications of using a CSCW tool. He contends that they rely heavily on intuition to commit resources to project development, and that this intuition is based on experience of developing single-user applications. He states

> Intuition may be a far more reliable guide to single-user applications—a manager with good intuition may quickly get a feel for the user's experience with a word processor, spreadsheet, or so forth. But a typical CSCW application will be used by a range of user types—people with different backgrounds and job descriptions, [...]

This is a flawed argument. There is just as wide a range of types of people using single-user systems as there are using multi-user systems. There will be just as wide a variety of user types using a multi-user CSCW application as there will be using a single-user word processor. A more valid argument would be that *all* systems are degraded by a developer's inability to account for variance in user types. CSCW application development faces additional problems, however. Development failures specific to CSCW applications do exist. What Grudin should have highlighted was the developer's potential inability to understand or support the complex network of interactions that exist in a tool for group use. Not only are there problems of making it usable on an individual basis, but the collaborative, communicative and coordinating activities that result in multiple interwoven interactions must be made possible.

The third and final problem that Grudin points out is far more valid. Single user applications have been established for many years and there is a wealth of experience in analysing and evaluating their functionality and computer-human interfaces. CSCW is an emerging field and tools for group use are relatively new. The methods for evaluating single-user applications are no longer applicable when the complex network of interaction possibilities present in group

support systems is considered. It may be possible to carry out some user interface evaluation for a single user, but aspects of social psychology must be considered in multi-user collaboration/coordination/cooperation/communication support systems. Evaluation of such systems will be a time consuming and costly process, which must largely be done in the field, and will be affected by variable group compositions and environmental factors. It will be mainly observational in nature.

Development of techniques for evaluation of such systems must be addressed in the near future. The progression of CSCW as a discipline will be retarded and new systems will find acceptance lacking until there is a framework available which allows analysis of CSCW tools.

In a case study of the implementation of a CSCW tool, the Coordinator, Carasik and Grantham (1988) suggest a further reason for the lack of acceptance of CSCW tools into the workplace. They claim that by their very nature such systems will change the way in which an organization functions, and that their acceptance is affected by existing cultures, and how adaptable they are. They recognize the fact that to be useful a tool must be adaptable for use in a variety of organizations. Their suggestions for future research include incorporation of designs which measure levels of system usage quantitatively, carefully designed methods for implementing systems, and consideration of 'affective dimensions of human-computer interaction'.

The limited success of CSCW systems to date is an additional motivator for further investigation into their design and implementation. Of special interest is the design and implementation of systems other than those to support communication and meetings of which many widely used examples already exist.

2.5 Social and psychological issues in CSCW

In order to develop effective and useful tools that enhance people's interactions with others in a work environment it is beneficial to gain an understanding of existing work practices, approaches to inter-person interaction and areas open to computer enhancement. Social and psychological issues should be considered, and the tasks to be supported specified. For instance: does inter-person communication need to be encouraged and enhanced, or should the system be more concerned with collaborative completion of tasks?

2.5.1 Organisational structure

Greenbaum (1988) provides a perspective on organization of work and management strategies, acknowledging that "The idea of designing computer support for group based work activity ... is a useful and challenging one". In support of this she cites studies during the 1980s that have highlighted the importance of communication, tacit knowledge, and the group nature of work activities. In an argument for democracy in the workplace as a better approach to computer support for cooperative work than user-centered design, she makes a distinction between several management strategies. Most management systems, she says, have top-down information flow, when lateral information flow is actually more conducive to cooperation, giving electronic mail as an example of a system where information flows laterally. If such a system, where social standing and managerial hierarchies are dispensed with encourages cooperation, then this must be a design aim of any system which intends to encourage and aid communication and cooperation. Of course there may be situations when hierarchical information flow is desirable, in a tool to support coordination say, so to strive for lateral information flow in all situations may be unwise. Greenbaum's statement "information system design that takes informal communication and group relationships into account is a step towards better user-centered, and hopefully user-oriented systems" however, is entirely reasonable.

2.5.2 Social impact of groupware

In a discussion of social and psychological aspects of communication via computers Kiesler *et al.* (1988) concur with the belief that the use of computers as communication tools was once, but is no longer, confined to technical users. People can now work and exchange information without regard for what used to be major obstacles such as geographical dispersion, energy costs and time zones. They also highlight a very important point which must be considered when developing computer based communication tools—computer mediated communication differs both technically *and culturally* from traditional communications technologies. This would indicate that a whole host of other issues must be addressed in the development of such a system, concerning not only what it will do, but how users will get it to do it, and what effect it will have on the work environment into which it is introduced. A powerful system may aid users in one

way, but hinder them in another as a result of the radical cultural change brought about by communication via computer rather than pen and paper, telephone and face-to-face meetings. Indeed Kiesler observed computer mediated work groups in a study of participation, choice and interaction where computer mediated groups took longer to reach a consensus than non computer mediated groups. This was partly attributable to time spent typing, but also because more equal participation of members of the groups took place, and interaction was far less inhibited. Hence a tradeoff must be considered: is it acceptable for the process to take more time if some individuals are encouraged to make larger contributions? Although not uncontestable evidence, this highlights possible effects which must be given consideration.

2.5.3 The importance of collaborator proximity

Kraut *et al.* have carried out in-depth studies of collaborative work in a specific environment—that of scientific research. First they have studied physical proximity and its effects on the likelihood of collaboration taking place (Kraut & Egido, 1988) and second, the structure of relationships and tasks in scientific collaborations (Kraut *et al.*, 1988). The importance of this work lies in the insight that it gives into the factors affecting the initiation of collaborations, and the protocols that are usually adopted during them. In order to encourage and aid collaboration, the designer of a system must attempt to provide the circumstances which encourage initiation. The collaboration process must then be supported and enhanced, which will not be possible if the system enforces protocols which contradict the norm. Although the work is centered on scientific collaboration, I would agree with the authors when they state that "Science is a fundamentally social process. Scientific collaboration provides a model of the way professionals in many fields construct intellectual products." Their findings could be generalized to other collaborative environments. This could, however, be a dangerous stance to take for system design. If professionals are taken as the model of a potential user, we may arrive back at the problem of such tools only being available to a select few.

According to the findings of Kraut and Egido (1988) the major obstacle to initiation of collaboration is spatial segregation. Physical proximity encourages informal discussion, out of which more formal, structured collaborations grow. In their study, researchers were physically

grouped into departments, and so it would seem that physical proximity might not have been as important as the fact that those close to each other were interested in the same topics. However they found that researchers from different departments, but on the same floor, were six times more likely to collaborate than those on a different floor. They also observed that proximity increased both the frequency and quality of interactions. A system must therefore cut across the problem of physical separation and strive in as many ways as possible to provide the benefits of proximity. This would be novel and important work, for as Kraut and Egido point out "Current communication technology available to most researchers does not allow the intensity of interaction nor the spontaneous exchange of notes on documents that are typical of face to face meetings". The richness of information exchange achieved in a physical meeting must also be provided in a computer based system. Proximity—or 'virtual proximity'—can therefore be seen to be beneficial in four ways:

- it encourages the initiation of interactions;
- it encourages interactions to happen more often;
- it encourages more worthwhile interactions;
- it saves on communication costs. Travel costs to meet a collaborator are lower, and informal, unstructured meetings are possible as opposed to intentional structured interactions.

The following two sections consider systems which either attempt to promote 'virtual proximity' through support for communication, or support actual meetings.

2.6 Computer support of communication

Person to person communication is remarkably rich. In everyday interaction with others we are not aware of the multitude of methods that are employed to express our ideas, feelings and emotions to others. The most obvious channel of communication is speech, with inflection, tone, pitch, diction all affecting the clarity and meaning of what is being said. Facial expression and gesture are also powerful channels.

Of course, these channels are rarely used in isolation, and are simultaneously used to greatly increase the richness of what we are trying to express. However confusion can arise in such

interactions, even when many channels of information are available to enable us to disambiguate what another is trying to express. Sarcasm, for example, might not be obvious without looking at the speaker's facial expression. The words used might express something in an ambiguous manner. Diction might be so unclear as to make what is being said unintelligible.

Therefore, although communication between people is a very sophisticated process which has evolved over thousands of years to its current state, it is still prone to failure. Electronic tools that support such communication are far less sophisticated, still in their developmental infancy, and even more prone to failure. This may be a failure to support the communication process successfully and/or just a technical failure.

Consider telephone, postal, video, satellite communication, which are sophisticated media, but do not support the range of expression of personal contact and are liable to technical failures such as crossed lines, lost letters, mechanical breakdown and electrical interference. Computer based tools for the support of human-human communication are less sophisticated still, with much room for improvement and development.

This section briefly addresses some issues and systems which support communication.

2.6.1 Electronic mail

By far the most widely used computer based tool for communicating with others is electronic mail (e-mail). Textual messages are sent across communication networks from one computer to another, and can be targetted at computers, groups of users or single users. E-mail messages can be sent locally (to physically close, network linked machines), nationally, and internationally. Each node of the worldwide electronic mail network has an address, and each registered user at that address can be communicated with.

E-mail has several advantages

- It is a relatively fast means of communication. Information can be sent around the world in a matter of hours;
- It is a relatively cheap medium. Messages are sent via telephone links at low cost;
- It is simple to use. Textual documents are created as per normal; all that is required is the

intended recipients electronic mail address and the correct format of command for posting the message;

- It is a well established, standardized communication medium.

Mackay (1988) states that e-mail is more than just a communication system. She claims that it also supports information management and time and task management. This is a reasonable claim as e-mail messages are not restricted to merely containing information. There is no reason why they should not be instructions—the sender delegates tasks, the recipient receives details of tasks to be performed—or timetabling details for arranging meetings. Mackay's observation that some people manage their e-mail, yet others are managed by it, highlights a major problem with it as a communication medium. In everyday personal interactions we pick out useful information and discard what we don't need. Background noise and events are ignored to concentrate on the other participant(s) in the interaction. Sometimes we even ignore the information being addressed to us if it seems irrelevant or uninteresting. E-mail can swamp a user in similarly large amounts of information, but provides little support for the extraction of pertinent information and the discarding of messages that are not required. After carrying out interviews with users of electronic mail, Mackay defines three extreme types:

prioritizers : they require support before they read their e-mail. They want support for management of incoming messages so that irrelevant ones can be filtered out as soon as possible;

archivers : they store their messages to be processed later. They require support for the extraction of relevant information from archived messages;

manager/secretary teams : the manager delegates handling of messages to the secretary.

2.6.2 Semi-structured message systems

Any e-mail system for the support of person to person communication must therefore take into account such information processing requirements. Malone *et al.* attempt to do this in their *Information Lens* (Malone *et al.*, 1986; Malone *et al.*, 1988), where they describe it as "an intelligent system for information sharing in organizations". The system helps users to filter,

sort and prioritize messages that are addressed to them, and also to find messages within the system that might be relevant but they might not otherwise have got. This is achieved by adding constraints to the structure of the messages themselves. Whereas e-mail messages are usually unstructured, apart from fields for the sender, recipient and subject, messages in the Information Lens are semistructured. Messages in the system are of identifiable types, and each of these types contains a known set of fields. The fields are used to filter, sort or prioritize the messages. Hence, it would seem that Information Lens presents a tradeoff. For increased support for the handling of incoming messages, a user must invest time in both specifying how incoming messages are to be dealt with, and in sending more complex messages.

Purely textual e-mail however, even with the facilities provided by Information Lens and similar systems (Cockburn & Thimbleby, 1993), still has disadvantages. Its power can be increased, though, by giving a user the ability to include information in a graphical form, as an illustration, or to annotate messages, or to impose a presentation order on the information. Some hypertext (CACM, 1988) environments allow this. For example Guided Tours and Tabletops (Trigg, 1988) are two tools for communication within the NoteCards hypertext environment.

2.6.3 Mixed media messaging

The Wang Freestyle system goes further in trying to increase the bandwidth of human-human communication by computer. Messages are written using a computer pencil and tablet (Francik & Akagi, 1988) in normal handwriting. This enhances the information in a message greatly. The author can emphasise data in a suitable manner; diagrams, pictures and graphs can be hand drawn, handwritten annotations can be made, and the very layout is personal and individual. Messages can also be typed, however. The work done by Thomas (1987) indicates that this ability to input text via both handwriting and a keyboard is highly desirable. In work associated with the development of a stylus, digitizing membrane and a flat screen, Thomas compared the use of such a system with that of MacWrite, a WYSIWYG (What You See Is What You Get) direct manipulation (Shneiderman, 1987) word processor for the Apple Macintosh personal computer. It was found that trained typists were faster at inputting text than those using the stylus, and that handwritten input should be retained for making alterations and annotations. Wang

Freestyle also provides a facility for adding vocal annotations to documents (Hsiao & Levine, 1988). When a document is read, the dynamic playback of the sound is synchronized with the reading of the text. This breaks away from the constraints of text-bound communication systems, and enters the dimensions of graphical and verbal communication. The effectiveness of extending the interaction process into these dimensions must be investigated however. Audio annotation, in conjunction with textual data, presents problems of synchronisation of presentation, clarity of the sound, clarity of meaning of the message without visual cues such as facial expression and gesture, and the attitudes of users to making sound recordings.

2.7 Computer mediated meetings

In addition to the computer-based tools that support inter-person communication, are those which support multi-person meetings. This section briefly considers those systems and describes exemplar systems.

2.7.1 Shared screen systems

AUGMENT (Engelbart, 1988) allows users to take part in shared-screen teleconferencing. This can be useful to allow co-authors to collaborate, but can also allow users to have a discussive meeting via the computer system. Users can enter text visible to all in a shared window, and as sometimes happens in conventional meetings, there is provision for new participants to enter the meeting, and current participants to leave it. The mode of communication available to users is constrained to text entry via the keyboard, which is undoubtedly less satisfactory than the multitude of communication media available in a conventional meeting. However, the power provided by such a system is that geographically distant participants can take part in a computer mediated meeting that may not be otherwise possible. A further drawback is that users are restricted to seeing the screen image of a single designated user, whose screen image is the one that is shared, and control of the tools involved in the process must be passed back and forth between participants during the meeting. This is a clumsy, unnatural and inefficient situation.

2.7.2 Meeting rooms

Colab (Stefik *et al.*, 1987; Stefik *et al.*, 1988) is a more recent and more sophisticated system, designed specifically to support the meeting process. Stefik *et al.* (1988) state

> Meetings are used for virtually any intellectual task that requires the coordination or agreement of several people. Statistical studies suggest that office workers spend as much as 30–70% of their time in meetings.

as part of their motivation for the development of the Colab. According to them, further motivational points are

- chalkboards[2], a commonly used medium, 'provide a shared and focused memory for a meeting, allowing flexible placement of text and figures, which complements our human capabilities for manipulating spatial memories';
- space on chalkboards is limited and items disappear when that space is needed for something else;
- rearranging items is inconvenient when they have to be manually redrawn and then erased;
- handwriting on a chalkboard can be illegible;
- it is difficult to save the information on a chalkboard from one meeting to the next.

They also highlight the functions that a computer-based tool can provide which aren't available in conventional meetings

- window systems and drawing aids provide flexibility for rearranging text and figures;
- file systems make it possible to recover information generated in previous meetings;
- independent workstations allow participants to share views and work on different aspects of a problem simultaneously.

The Colab meeting room contained six workstations, connected over a local area network that supports a distributed database. The workstations face a large touch-sensitive screen which has a

[2]Stefik *et al.* use this term to describe any wall-mounted erasable writing surface.

keyboard next to it, to allow a standing participant to present ideas facing the other members of the meeting. Users have public and personal areas of their display screens and communication takes place via these. Although Colab and its tools do attempt to provide the benefits and address the problems itemized above, it was found that users faced difficulties integrating use of Cognoter, a Colab writing tool, with common, natural work practices. Its implicit structured model of communication caused problems for users.

2.7.3 Social browsing systems

Whereas the Colab is aimed at supporting more structured, prearranged, formal meetings, CRUISER (Root, 1988) is a tool 'to enable unplanned, informal social interaction'. Root contends that the office is as much a social institution as a place where tasks are carried out, so 'critical elements for success are interpersonal communications and informal social relationships'. As discussed in Section 2.5, physical proximity can increase the likelihood of social interactions and collaborative work efforts. CRUISER proposes a virtual workspace to support such interactions. A network provides multimedia connections to other people and the interface provides virtual hallways to be browsed. From the virtual hallways a user can enter the office of others. Each office has video and audio links so as a user passes the virtual doorway of a colleague, a visual and audio peek into that office is provided. A user can socially browse by going on a planned route, or a random walk. Interaction protocols are suggested for CRUISER to constrain the ease with which others can be seen and heard at work, and to allow users to regulate who can and who can't interrupt them during their work. CRUISER even proposes the note leaving response to people who are out when visited. A digital equivalent of a post-it note can be left on the target's screen with the facility to automatically set up a dialogue with its author when the recipient returns.

CRUISER is an exciting concept which exploits to a greater extent than most other systems the power of multimedia technology. It is a valuable concept, supporting informal meetings which complements the facilities provided by Colab for a more formal group interaction. However, its use was mainly as a visual telephone, it was found to be unsatisfactory for more than two people, and users suffered from not being able to share work objects (Fish *et al.*, 1992).

2.8 Computer supported collaborative writing

This section concentrates on tools which support more than one writer working on a common authoring task. The type of support which such tools provide is diverse and will be consider along several dimensions. First a distinction is made between those which support synchronous and asynchronous work. Second, support for co-located and distributed work is discussed. Third, the level of constraints imposed by systems is addressed and finally process/product based systems and network/hierarchy based systems are considered.

2.8.1 Collaborative writing tools to support synchronous co-authoring

Earlier in this chapter a matrix of collaboration styles was presented (see Figure 2.2). This section considers existing systems which support synchronous work of multiple authors, irrespective of their location.

NLS/AUGMENT

One of the earliest *shared screen systems* was developed by Engelbart. Engelbart's work with AUGMENT (which evolved from NLS) (Engelbart, 1988) provided a single user with the ability to divide the screen into arbitrary rectangular windows and then edit between them across files. However, the other windows need not necessarily belong to the same user, hence AUGMENT provided 'shared screen teleconferencing' supporting collaboration amongst authors and colleagues.

The users who wished to collaborate via use of the same screen, issued commands indicating with whom they wished to share the screen and each user received the same view of the screen. The 'viewing' user's original screen image was replaced by that of the 'showing' user. The showing user retained control over the tool that they were using. Control of the tools could be passed back and forth between users and it was possible for other users to enter the dialogue.

This could allow users to enter into a real time, conversation-like dialogues via the keyboard and the screen, no matter how geographically distant they were. They could also collaborate on authoring a single document. However, there are potential problems with such a system. For more than two users the transferral of control could become a complicated process, with users

having difficulty in expressing their desire for control (how will they do so if they don't have it in the first place?). Users are also constrained to the view provided by a single user, whereas different users may well have different preferred methods of screen layout and presentation. Additionally, users cannot carry out simultaneous editing on a single, coauthored document, and it is unclear whether they have to continually transfer control in order to enter into a real time dialogue.

Cognoter

A more recent system is a result of the extensive work that Stefik *et al.* have carried out at Xerox PARC into computer support for collaboration. They have developed the Colab (Stefik *et al.*, 1988; Stefik *et al.*, 1987)—a meeting room where face to face meetings are supported and hopefully enhanced by computers. They introduce the term WYSIWIS (What You See Is What I See) 'which refers to the presentation of consistent images of shared information to all participants'.

> WYSIWIS creates the impression that members of a group are interacting with shared and tangible objects. ...It recognizes the importance of being able to see what work the other members have done and what work is in progress...

Whereas AUGMENT is strict in its implementation of WYSIWIS, in that users do all see a single screen image belonging to one of them, Stefik *et al.* relax WYSIWIS. Windows on a display are either *public* and accessible to all in the work group, or *private* and access is limited (for personal e-mail say). User diversity in desired screen layout is partly resolved in the Colab, as public windows can appear at different positions on different displays. Whereas AUGMENT seemed to require explicit transfer of control in the collaborative editing task, Colab seeks to increase efficiency by supporting simultaneous actions, and resolving possible conflicts by giving users a graphical 'busy signal', warning that someone else is already editing or using an item.

Cognoter, helps authors in the brainstorming, organisation, and evaluation stages of document production. Stefik *et al.* (1988) acknowledge its similarity to systems such as NoteCards (Trigg & Suchman, 1988) in its use of notes to organise ideas, but stress that its uniqueness lies in its *simultaneous*, collective use by a group of people. Members of the group initially type ideas

and supporting text into a public window. Implicit links are created between the idea labels and the text, so that the text is displayed when the idea label is clicked upon with a pointing device. Next the ideas are given order constraints by the explicit creation of links which are both transitive and distributive. Finally evaluation and deletion of unwanted material takes place.

GROVE

The GRoup Outline Viewing Editor (GROVE) (Ellis *et al.*, 1991a) provides real time private, shared and public views of a document. Each author uses a local editor and a replicated version of the group document at their own workstation. Consequently GROVE is, in the main, meant to support distributed, yet synchronous collaborations regarding document development.

Such a system provides problems in areas such as response time (for representation of activity of other participants and changes to the document which must be quickly propagated around the system), concurrent actions by participants, and data inconsistencies. To some extent these difficulties are ameliorated by supporting the development and use of social protocols.

SASE

Baecker *et al.* developed SASE in order to support 'highly interactive synchronous collaborative writing' (1993). The design requirements for SASE were that it would provide support for

- focussed collaboration and independent work
- collaborator awareness
- conflict resolution (in terms of document updates)

The shared document is replicated on each participant's workstation, and a communication server controls document updates. It was assumed that collaborators would communicate via telephone or audio/video connections. Work can be independent or users can lock views to achieve the WYSIWIS effect of AUGMENT/Colab. The activity of other users is visually represented using coloured scrollbars in the shared document.

In a usability study it was found that authors tended to divide work before beginning the shared task, reducing the need for the synchronous aspects of the tool. Also, forced synchronous

collaboration did not result in individuals manipulating the document synchronously; a scribe was appointed to make changes to the document.

This questions somewhat the necessity for synchronous facilities in collaborative writing.

SASSE

SASSE (Baecker *et al.*, 1993) was developed from SASE. It provides further support for synchronous work, including the ability to view exactly what other authors are doing. It also provides an outline editor for work involving development of hierarchically structured documents and a condensed image of the entire document.

However, due to the results gained from evaluation of the usage of SASE, more support for asynchronous work was introduced into SASSE. This will be considered later in this chapter.

SEPIA

SEPIA (Haake & Haake, 1993) is a cooperative hypermedia authoring environment. One of its aims is to provide smooth transitions between different collaborative modes, rather than support one or other of synchronous or asynchronous work as its developers claim that normative CSCW systems do. Consequently the system supports three modes of collaborative work: individual, loosely-coupled and tightly coupled. As work moves from an individual to a tightly coupled nature, support moves from asynchronous to sysnchronous. SEPIA provides four types of workspace for writing activities:

- planning
- content
- argumentation
- rhetorical

In support of synchronous work WYSIWIS principles are applied. Users are provided with shared views among concurrent browsers, telepointers, audio and video communication channels and a shared drawing tool. Individual work mode is active while an author works on a node of the document without concurrent users. Loosely coupled mode is *automatically* activated when

a node is opened that one or other authors are concurrently accessing. Authors can suggest to collaborators that they move into tightly coupled mode, and the collaborators can accept or decline.

When more than one author wishes to work on the same node of the document, concurrent access is supported. Group awareness is provided by representation of the effects of actions of other authors, and activity markers help authors to avoid collisions.

Some problems were identified with the support for tightly coupled work. The integrity of individual contributions had to be maintained, and authors had to achieve this by explicitly copying objects, Additionally there was no record of the history of the interaction in the system. These problems were addressed by inclusion of version management support in the system, using CoVer, a hypermedia version server.

2.8.2 Collaborative writing tools to support asynchronous co-authoring

This section details existing systems which support co-authoring in which contributions of different authors occur at different times.

NoteCards

NoteCards is a network based information structuring tool that can be used to support collaborative writing. It supports the generation of hypertext documents containing both textual and graphical information. Using the metaphor of physical notecards for document elements, documents are built up by addition of new cards which are linked to existing ones.

Trigg and Suchman (1988) attempted to use it to support their own collaborative work. In a similar way to AUGMENT allowed links to external files, NoteCards allows cards to be linked to external files, but in addition also allows the file contents to be put into a card. Trigg and Suchman found this useful for creating "a shared workspace so that when working independently we each have access to the product of each others work", an aim of the Memex (Bush, 1988) and AUGMENT. NoteCards also provides the facility for annotation of text, which according to Trigg and Suchman encourages dialogues within the medium. They make a general observation about hypertext and collaborative writing which goes some way to justifying basing systems

that support it on the hypertext paradigm

> Hypertext is appropriate for collaborative writing, [...]it supports the coexistence of multiple overlapping organisations of information, well suited to the annotating activities of paper draft-passing.

To some extent this statement may be considered appropriate. However, it does not reveal that NoteCards does not allow group members simultaneous access to the hypertext network. This implies that co-authors are not supported in carrying out work on the document at the same time, either on different sections, or perhaps on the same sections through document replication. NoteCards may well be suited to support of an individual's work in a collaborative writing task, but does not support the wider collaborative processes which perhaps involve writers working in parallel.

VNS

Shipman *et al.* (1989) expressly criticise NoteCards for not allowing such access but do not state whether or not their system, the Virtual Notebook System (VNS), provides it.

VNS is designed to be an electronic analogue to a scientist's notebook, which acts as a repository of data, hypotheses and notes, and enhances information sharing. The basic work unit in the system is the page. Pages can be created, mailed and linked to other pages. Groups of pages linked together associatively in a hypertext manner are called notebooks. The information in notebooks occurs only once in the underlying database, and can be accessed simultaneously by more than one user. Structuring of information, however, can be personalized, with links belonging to only one user. Hence users can be viewing the same page of the database, but it will contain differing links. Users can create new pages to add to the database at any time.

Such a hypertext system seems to be an excellent design. The user benefits from a large central data store, which is continually growing, and has access to the work of others, but can traverse and structure the information in a way which is most suited to their current needs. In order to overcome the potential problem of a user being aware of the presence of a link, but unsure of to where the link is directed, as a mouse pointer passes over a link in a page,

information pops up describing the link. On moving the pointer out of the link the information pops down again. Clicking on the link takes the user to the destination.

Three important issues arise out of the way that VNS is implemented:-

1. The data in the system is stored in a relational database on a work group server which is part of a network.

2. Pages can be sent to and retrieved from other users, but they do not necessarily have to be users of VNS, as pages are encoded into PostScript, a page description language, before they are sent.

3. The user interface to VNS is implemented in the X Window System (Scheifler & Gettys, 1986) and so VNS should be usable, independent of the hardware available.

The primary role for VNS does not seem to be either as a single or group authoring environment. Its support lies mainly in storage, retrieval and structuring of information elements. It does not specifically provide support for single or group authoring processes, yet access to the work of others, shared documents, shared information resources facilitate group work on shared artifacts.

InterNote

In work sponsored by Apple Computer, Catlin *et al.* (1989) have extended their Macintosh based hypermedia framework Intermedia, with the InterNote facility in order to support annotative collaboration. Its aim is to "support small groups of collaborators, particularly those involved in document review and revision". Like VNS it allows creation of links between objects in the system and allows information to be passed from one user to another. However, linking and information passing are done in a novel manner. Catlin *et al.* differentiate between three different types of links:

cold links are purely navigational, allowing the user to traverse the data structure

hot links are both navigational and support data transfer. They provide automatic updating of information at the ends of the links, whenever a master component is altered

warm links are also both navigational and support data transfer. They differ from hot links, however, in that they do not provide automatic updating of linked information—this has to be explicitly requested by the user

Linked elements can be one of several types. Annotations, central to the system, are automatically created with with the same type as the element to be annotated, and include explicit areas for suggested changes and for comments.

Although InterNote provides many powerful tools to support the collaborative writing process it has a major drawback. InterNote exists as an application within the Intermedia environment, and InterNote documents are restricted to use within Intermedia. Additionally Intermedia is constrained to use on an Apple Macintosh computer. Hence it is in direct contrast to VNS, in that it is not hardware independent, and InterNote documents can not be sent to others who do not use the Intermedia environment. These are major disadvantages to any system which aims to encourage collaboration and communication.

Quilt

In Quilt, a computer based tool for collaborative document production Leland *et al.* (1988) provide both textual and vocal annotation facilities. Quilt draws on ideas from social collaboration, hypertext, and direct manipulation interfaces to provide a tool which supports annotation, messaging, computer conferencing and notification facilities to support communication and information sharing among the collaborating authors of a document.

Some of its facilities support synchronous execution of group tasks, but authoring, in the main, is asynchronous.

SASSE

As stated above, SASSE was developed from SASE. Some developments were concerned with provision of support for asynchronous work. Consequently SASSE provides the facilities of SASE, but in addition provides an annotation mechanism. This allows co-authors to communicate asynchronously about document elements through the document itself. Additionally a history of contributions to the document is provided, presenting a description of which co-authors

amended which parts of the document, and when the change took place.

PREP

PREP is a "work in preparation" editor designed to help in the study of co-authoring and commenting relationships (Neuwirth *et al.*, 1993). It is exclusively meant to support the work of loosely-coupled collaborators who do not interact at the same time.

It addresses three issues

1. support of social interaction among co-authors and commenters

2. support for cognitive aspects of co-authoring and external commenting

3. support for practicality in both types of interaction

PREP's basic features are similar to other sytems. Documents are built from chunks of information or representations of ideas. The chunks can be linked, and so a network can be built up in a similar way to systems such as NoteCards. Authors have shared access to the workspace of chunks, and can group chunks into *drafts*. A draft consists of a grid of chunks, arranged in columns, which might represent plans, content and a co-author's comments. Authors can dynamically add comments to review or suggest changes to the text.

The asynchronous nature of the task requires that multiple contributions be combined, and PREP supports representation of updates through versioning. Revisions can exist as distinct versions of the document draft, yet as authors work within the same workspace, not replicated versions of it, there is no need for automated support for the integration of the work of several authors.

CES

CES is a collaborative document editing system in a distributed workstation environment (Greif & Sarin, 1987). According to its authors, it is intended to support coauthors working asynchronously on a shared document. However, the definition of asynchronous which they use would appear to entail authors not actually being able to access the same document element at the same time. This is not common with other literature addressing the issue. CES may perhaps

be more appropriately classified as a system which supports synchronous work, as changes to documents are propogated and are visible to others just after they are made, and locking occurs if simultaneous access is attempted.

A document is associated with a set of authors each of whom has specified access rights (either read only or edit) to document elements. Those elements are a table-of-contents outline and single sections. A view of the text of the whole document is also provided. Sections are held on the workstation of their author, but other users can access them, and the outline of the document is replicated and accessible to all authors.

SEPIA

SEPIA supports asynchronous interaction in addition to synchronous interaction. In fact, the default operational mode is support of individual work.

Some problems were identified with support for asynchronous work. First, two different types of work, isolated (single author access to a node) and separate (non-interfering concurrent access to a node) should be explicitly supported. Second, conflicting changes could not easily be identified by the authors.

2.8.3 Collaborative writing tools to support co-located co-authoring

This section details existing systems which support contributors who are in the same location. Co-location can be considered in a variety of ways, and becomes less necessary as network technology improves. Co-location is perhaps imposed through restricted access to a shared data store, or a requirement to use a specific machine. This may require collaborators to work in the same room. However, with local area network support, or wide area or global networks 'co-located' becomes less easily definable. Additionally support systems themselves, rather than the technology they exploit, may require co-location. Brainstorming tools are an example of this.

NoteCards

NoteCards requires collaborators to be in the same place because of access to the shared (not synchronously) data store. This constraint is a technical one, unmotivated by requirements or goals in the support of collaborative processes. Additionally authors can not simultaneously access the network of notes. Consequently the system forces co-location but does not support synchronous working.

Cognoter

Cognoter exists within Colab, a system intended to support same place, same time interactions. Participants in the group interaction can use Cognoter individually or as a group to generate and organise ideas. Cognoter was intended to complement conventional group meeting processes, yet users found difficulties in integrating its use with common work practices. This was because it imposed a structured model of communication.

ACE

The Amsterdam Conversation Environment (ACE) (Dykstra & Carasik, 1991) is intended to support group interactions in a shared workspace. It is mainly intended to support a process rather than creation of a product. However, users are unconstrained in the activities which they can use the system to support.

In conventional usage, participants have discussion related windows on the screen of their workstation, which are replicated among colleagues. Users subscribe to a discussion window, and can see who else has subscribed. A projection system can display a representation of multiple windows.

Users can incorporate text, graphics and sound into their contributions to the discussion. Consequently they are at liberty to not only exploit the process support provided by the system but can use it to create documents collaboratively in shared windows.

Other systems

The following systems support co-located working, although this was not a central design consideration in the same manner as Cognoter. Perhaps a more accurate appraisal would be that they are for use in a *locally* distributed environment, with access to the shared software and information resources required by the system. Collaborators may not by in the same room, but perhaps in the same building, with opportunity for communication by other channels.

Conceptually they may support widely distributed collaboration, but practically they support locally distributed collaboration.

- InterNote
- PREP
- CES
- SEPIA
- GROVE
- SASSE

2.8.4 Collaborative writing tools to support distributed co-authoring

This section is concerned with systems which support co-authoring when the participants in the activity are not located in the same place. This can be taken to mean 'not in the same room' with perhaps either synchronous or asynchronous computer-mediated communication. This then could concern locally distributed to globally distributed participants.

NLS/AUGMENT

As has already been stated, AUGMENT could be used over a network for collaborators to communicate and work towards a shared product. However this was not a particularly rich interaction, only one participant could have control over a shared screen at one time, a control passing protocol had to be adopted, and users could not integrate personal work and collaborative work in the same session.

VNS

VNS uses a central relational database on a networked server to store contributions to the information on hand. Consequently network access, either local or further afield, gives access to VNS as a tool for hypertext document creation. It supports communication between distributed authors, allowing pages to be passed, and acknowledges that distributed participants can not be guaranteed to have access to the same type of hardware. Consequently the user interface is implemented using the X Window System, and pages are encoded into PostScript before transmission.

Other systems

Given the distinction between conceptual and practical support that was indicated above several other systems could also be classified as supporting distributed work.

- InterNote
- PREP
- CES
- SEPIA
- GROVE
- SASSE

2.8.5 Constraint based systems

Some systems to support group working actually direct the activities which occur in the collaborative process. In these cases support actually means constraint. Roles, activities, access to documents, access rights may be constrained by the system. Such constraints may be direct implementations of models of single or group writing processes, or may exist to ease implementation and management of the process by the system.

Quilt

Quilt, a tool for collaborative document production uses user-provided information to constraint a co-authoring task.

For a new Quilt collaboration certain attributes have to be specified

- the name by which the collaborative project will be referred to;
- the people who are part of the collaborative project;
- the style of the collaboration. This specifies the types of access available to the document. Modifications can be restricted to section authors only, or any author can modify any section, or an editor can be designated to field suggested modifications from others. Customized styles can also be created;
- the roles of the members of the collaborative project then must be specified and can be one of co-author, commenter or reader.

Quilt therefore demands that rigid rules for the collaborative process are laid down at its initiation. The system must be flexible enough to allow for any changes that might take place during the evolution of the project. Someone initially designated a reader may have good ideas that could be immediately included into the document, and so it would be desirable to promote them to at least commenter, or maybe co-author. Demotions should also be possible.

In the same way that Quilt provides for different collaboration types, it provides predefined annotation types: comments, revisions, directed messages and private notes. This may save the author the effort of specifying explicitly the constraints of an annotation, but poses a problem if the needs of the author fall outwith the facilities provided by the predefined types. It is not clear whether Quilt allows for the extension of the predefined social roles of collaborators, or the annotation types.

The Writer's Assistant

Sharples *et al.* discuss a model of writing which describes the writing process in relation to the support provided by their tool the Writer's Assistant. This model describes how the system will

make processes normally internal to the writer explicit. This may be things such as setting and satisfying constraints like text length, style, and structure.

In this instance the user has control over whether the model is applied, ie constraints are not mandatory like in Quilt.

2.8.6 Minimally constrained systems

ACE

The Amsterdam Conversation Environment is, according to Dykstra and Carasik (1991), a semi-structured application "...which not only exhibits a given degree of user-modifiability, but is built with the intention of extension and development by those who use it".

They indicate that in semi-structured applications, designers don't create, adopt or impose their model of the user in the sytem: "Designing by creating a model of the user limits the potential of users, the software, and their interaction by imposing constraints".

I would agree with this in application contexts in which either no models exist at all, or available models are untested or are too vauge to be implemented convincingly. Indeed Dykstra and Carasik put faith in users to structure their group interaction successfully. They submit that "...given the opportunity, users can provide for themselves a significant amount of the task-oriented structure inherent in most computer application...".

The observations of use of early incarnations of the ACE indicated that this is indeed the case.

GROVE

The GROVE system reflects the view of its authors (Ellis *et al.*, 1991a) that

> Technological protocols can be overly restrictive: a group's idiosyncratic working style may not be supported, and the system can constrain a group that needs to use different processes for different activities.

PREP

The developers of the PREP group document development system (Neuwirth *et al.*, 1993) are critical of systems such as Quilt which present users with a predefined set of social roles, document objects and user actions, and then impose the users' selections. In their experience roles and activities were dynamic, and premature definition of or commitment to them could lead to problems for the participants.

Consequently PREP does not impose such constraints, although it is not clear how requests for simultaneous access to the workspace of chunks/drafts are dealt with without imposing some type of access control.

2.8.7 Product based systems

People may enter into a collaborative writing task for a variety of reasons, with diverse goals. One might consider two motives for participation in such a task. First, the individual wishes to produce a finished document; the product is the primary concern. Second, the individual wishes to generate, structure, discuss and revise ideas and text. In this case the process is the primary concern. Collaborative writing systems therefore may be considered as product or process based systems.

Each of the following systems (described in earlier sections) is a product based system:

- NoteCards
- VNS
- InterNote
- Quilt
- SASE/SASSE
- PREP
- CES
- GROVE

- SEPIA

2.8.8 Process based systems

When using process based systems it is the activity rather than the end product which is of central importance.

NLS/AUGMENT

AUGMENT (Engelbart, 1988) allows users to take part in shared-screen teleconferencing. This can be useful to allow co-authors to collaborate, but can also allow users to have a discussive meeting via the computer system. Users can enter text visible to all in a shared window, and as sometimes happens in conventional meetings, there is provision for new participants to enter the meeting, and current participants to leave it. The mode of communication available to users is constrained to text entry via the keyboard, which is undoubtedly less satisfactory than the multitude of communication media available in a conventional meeting. However, the power provided by such a system is that geographically distant participants can take part in a computer mediated meeting that may not be otherwise possible. A further drawback is that users are restricted to seeing the screen image of a single designated user, whose screen image is the one that is shared, and control of the tools involved in the process must be passed back and forth between participants during the meeting. This is a clumsy, unnatural and inefficient situation.

Cognoter

Cognoter, as described above, as part of the Colab environment is meant to support generation, discussion and manipulation of ideas and text. Users share views of windows, have private windows to develop work and can take control by presenting their screen image on a large screen. A document may be the product of a Cognoter sessions, but the process is of predominant interest.

ACE

The Amsterdam Conversation Environment is, as its name suggests, concerned with support for interaction between group members in a shared workspace. Its focus is placed on stimulating the communication and collaboration between participants, rather than the creation of some product, such as a co-authored document, by the group.

2.8.9 Network/hierarchy based systems

A large number of systems for supporting the writing task have been based on the concept of hypertext (CACM, 1988). In such systems authors create nodes of information which are associated with other nodes using traversable links. Hence a simple hypertext may well be sequential, or even contain only one node, but it is more likely that a hypertext document will contain a number of nodes with multiple associative links. A node can be the target of zero, one or more links from other destinations, and can itself provide zero, one or more links to other nodes.

This section considers systems which adopt this paradigm for collaborative authoring.

AUGMENT

Engelbart's AUGMENT provides a basic linking mechanism, *citation links*. Readers of documents in AUGMENT can travel via these links to the referenced bibliographic citation. The citations themselves may then be linked to the actual paper they refer to, so the user can quite quickly read the important parts of relevant papers. Of course the success of such a system is dependent on the author of an AUGMENT document investing time and effort in creating the links in addition to the plain text, and also on the presence of the additional referenced papers. If these possible drawbacks were eradicated, then AUGMENT would be a small step on the road towards the realisation of Vannevar Bush's *Memex* (Bush, 1988): a private file store and library, which a user traverses by association, creating a trail of associations or link traversals that can be reviewed at a later date.

InterNote

In a different view of annotation types to that of Nielsen's, Catlin et al (1989) define just two types

- suggested changes to the original document in the same data type as the original document;
- textual commentary for notes, comments, non-specific suggestions, and justifications of the suggested changes.

This view of annotations underlies the annotation mechanism in their InterNote system. Each annotation window in the WIMP interface that InterNote uses, is divided into two sections. An 'incorporation frame' holds the specific suggestions to replace a section in the document, and a 'commentary frame' which is used for textual commentary. This division of annotations is another method of structuring the annotation process. It is somewhat similar to the comment and revision annotation types provided by Quilt, but in addition allows the inclusion of both in a single annotation.

Annotations in the InterNote system are also heterogeneous. An author selects an object to annotate and the annotation window will allow creation of objects of the same type, such as text or graphics. Annotation type specific tools are provided in the incorporation frame, so for annotating a textual object, text editing tools will be provided, and for a graphical object, drawing tools will be provided. It is important to consider, however, that it might be more apt to annotate an object in a different type. The commentary frame will allow graphics to be annotated textually, but how can text be annotated graphically when the incorporation frame will provide text editing tools?

Other systems

Other such systems which have already been described earlier in this section are

- Cognoter
- NoteCards
- VNS

2.8.10 A taxonomy of support provided by writing tools

Figure 2.8.10 presents the writing tools discussed in previous sections and indicates the type of support that they provide.

2.9 General requirements for a system which supports collaborative writing

The preceding sections of this chapter have discussed systems and issues that have relevance to CSCW, with an emphasis on systems which support communication and collaborative writing. What general requirements are there then, for a computer based system to support such tasks? This section presents requirements with respect to three issues in the development of a generic CSCW system: social and psychological, design and implementation.

2.9.1 Social psychological requirements

The effects of introducing a computer system into an environment where there previously was not one, or of replacing an existing system can be large and disruptive. There are many issues to consider, and the process is made even more complicated when the system being introduced is to support work on an inter-person basis. A list of suggested aims and considerations follows:

- The system should enhance existing computer based relationships, and encourage the forging of new ones;
- Importance must be placed on improving the quality, efficacy and product of interactions;
- Informal communication protocols and group relationships must be taken into account. It is not sufficient to consider the protocols and relationships that should be present in the organizational hierarchy. It is necessary to investigate those that are actually there;
- The system should attempt not to enforce protocols which contradict or make it difficult to maintain those normally in place. The trade off between attempting to introduce more effective protocols, and the organizational difficulties this may cause must be considered;

- Physical separation reduces the potential for interaction and collaboration. This problem must be addressed and the system should attempt to provide the benefits of physical proximity;

- People's roles and commitments in an organization change over time. The system should be flexible and thus able to accommodate such changes;

- In addition to supporting informal interaction between people, the system should provide facilities to support more formal interactions, namely meetings. The system should support organisation, timetabling, structuring and recording of discussions and meetings;

- If such a system is a useful tool it will be used by a wide variety of people in a wide variety of organisations—it must be flexible.

2.9.2 Design requirements

An incorrect high-level approach to design can have as detrimental effect on the final product as can incorrect low-level design decisions. Cockburn and Jones (1991) have proposed principles for the design of computer systems which support collaboration. These principles will increase the chance of a system being accepted and more widely used than CSCW systems have been thus far and address the social issues of support for collaboration presented above.

These principles aim to address the problems shown in Figures 2.4 and 2.5—major transitions are required between work processes involving tool use for either personal or group, and between personal and group work processes.

Maximise the likelihood of system acceptance at a personal level. There is a similarity in how users view systems for personal work (a word processor, for example) and those supporting group work—a common question users ask about both types of tool is "what can it do for *me now*?".

In answering this question adoption of the "Reflexive Perspective" of CSCW (Cockburn & Thimbleby, 1991) is advocated; the individual's satisfaction is of paramount importance. Figure 2.6 shows how the reflexive perspective can bring about a blurring between group and personal work processes;

Minimise system imposed constraints on users. Through avoidance of embedding models of specific human tasks in systems, users are provided with the opportunity to form their own communication mediating protocols and to adopt personal and flexible working methods;

Minimise system requirements imposed on the user. Systems requiring additional work are likely to be unpopular and under-used, particularly when there is a disparity between those people carrying out the work and those gaining the benefit. This perspective argues that systems should not *depend* on users executing additional work, and yet retain benefits for those who commit to the extra work.

Figure 2.7 shows the more fully integrated environment that these principles encourage. There is an increased overlap between group and personal work processes across all types of tools, greatly decreasing the transitions between work processes which users are required to make.

2.9.3 Implementation requirements

At the implementation stage of the system there are many issues to be considered. The actual implementation techniques and facilities provided by the system will have a direct effect on its usability and success. There will be a multitude of implementation considerations, which will undoubtedly lead to trade-offs.

- The software should exploit the most suitable technology and hardware to maximum benefit for the users. For example, there is little point in providing text based technology and software to an environment in which the object of information exchange is a graphical image, and vice versa;

- Exploit electronic mail as it is a relatively cheap, powerful and widely used medium for computer based communication. It should be considered as the tool for the communication aspect of the system;

- Exploit annotation, making use of several media such as textual, graphical, auditory and video. Whether any or all of these are apt is a matter for consideration;

- The system should not be restricted to manipulation of a single information type. It should be flexible and allow for text, graphics, voice, video images, and handwritten messages to be used. If the current system only uses a subset of these, the new system should provide easy and powerful use and access to the new facilities;

- Entry of information into the system should be easy and flexible, whatever form it may be in. Special purpose editors may be provided for text, graphic, sound or handwritten entry, for example;

- The output from the system should be in a useful and manipulable form. Output from the system should also be suitable input for the system;

- Hypertextual facilities such as annotation, directed links and associative organisation of information may well be apt. Thought should be given to the benefits and drawbacks of a hypertext based system or one which has hypertextual facilities;

- The system should co-exist with other systems, access external resources and allow access to its own resources, rather than be 'stand-alone' with no facilities for making use of external resources.

- The unit of information exchange should not be dependent on system specific implementation. There are benefits to the system having its own information representation and exchange protocols (this is easier for the designer), but this renders the information stored and manipulated by the system useless without it. Data storage and transfer facilities should be potentially independent of the system;

- The system should be as hardware independent as possible. This may lead to wider availability and use, and allows users to retain control over the hardware used.

2.10 Summary

A description of the disparate, yet interrelated, topics within the area of CSCW have been presented, and it has been shown that there are many issues, including social and psychological ones, which motivate CSCW research.

Three CSCW application areas have been discussed. Communication supported by computer can be via many channels; the challenge is to make it as rich and effective as face to face contact. Meetings mediated by computer should be more effective than those not, and effective, supportive tools should be provided. Support for both formal and informal meetings is necessary. The support of writing itself can be viewed along several different dimensions such as location, time of collaboration, imposition of constraints and process or product orientation.

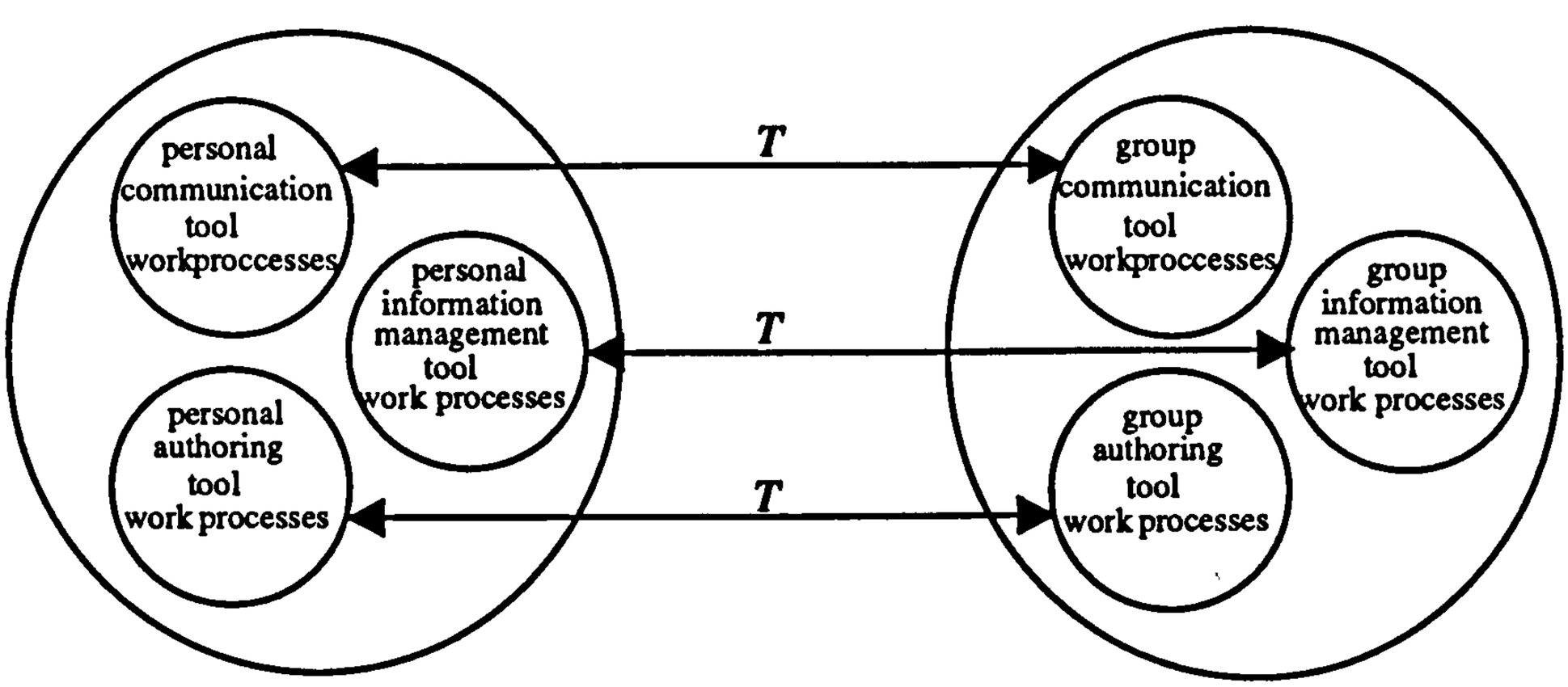
Possible reasons for the relative lack of success of CSCW systems have been discussed, with suggestions that might help to make future systems more successful, including principles for the design of groupware.

Finally, general requirements for a system to support collaborative work were presented. They relate to social, design and implementation issues.

The following chapter will consider more closely the writing and collaborative writing processes themselves, with the aim of identifying requirements for a system to specifically support collaborative writing.

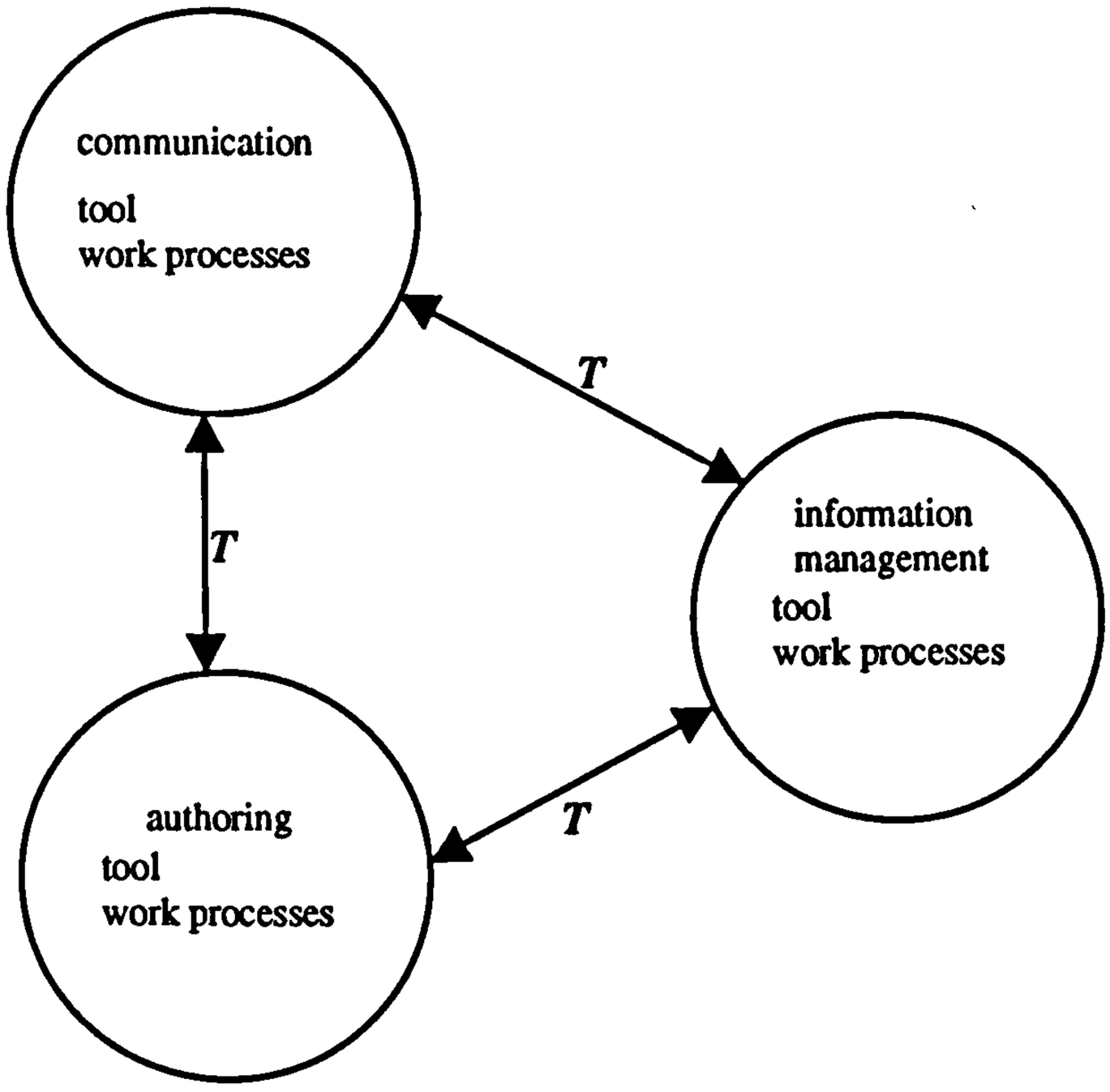
	Type of support provided													
	Synchronous	Asynchronous	Colocated	Distributed	Constrained	Minimally constrained	Product based	Process based	Network based					
NoteCards	✓	✓	✓	✗	✗	✗	✓	✗	✓					
NLS/ AUGMENT	✓	✗	✗	✓	✗	✗	✗	✓	✓					
Cognoter	✓	✗	✓	✗	✗	✗	✗	✓	✗					
SASE	✓	✗	✓	✗	✗	✗	✓	✗	✗					
SASSE	✓	✓	✓	✗	✗	✗	✓	✗	✗					
GROVE	✓	✗	✓	✗	✗	✗	✓	✗	✗					
SEPIA	✓	✓	✓	✗	✗	✗	✓	✗	✗					
VNS	✗	✓	✓	✓	✗	✗	✓	✗	✗					
InterNote	✗	✓	✓	✓	✓	✗	✓	✗	✗					
Quilt	✗	✓	✗	✓	✓	✗	✓	✗	✗					
PREP	✗	✓	✓	✓	✗	✗	✓	✗	✗					
CES	✗	✓	✓	✓	✗	✗	✓	✗	✗					
ACE	✗	✗	✓	✗	✗	✓	✗	✓	✗					

Figure 2.3: A taxomony of writing tools and the support that they provide.



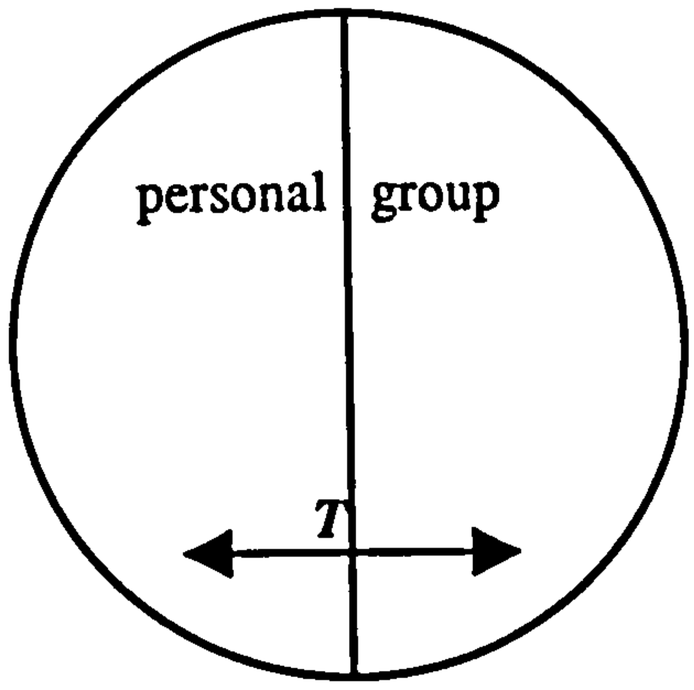
T denotes a transition from one set of work processes to another

Figure 2.4: Users are currently required to make multiple and major transitions between personal work processes and group work processes.

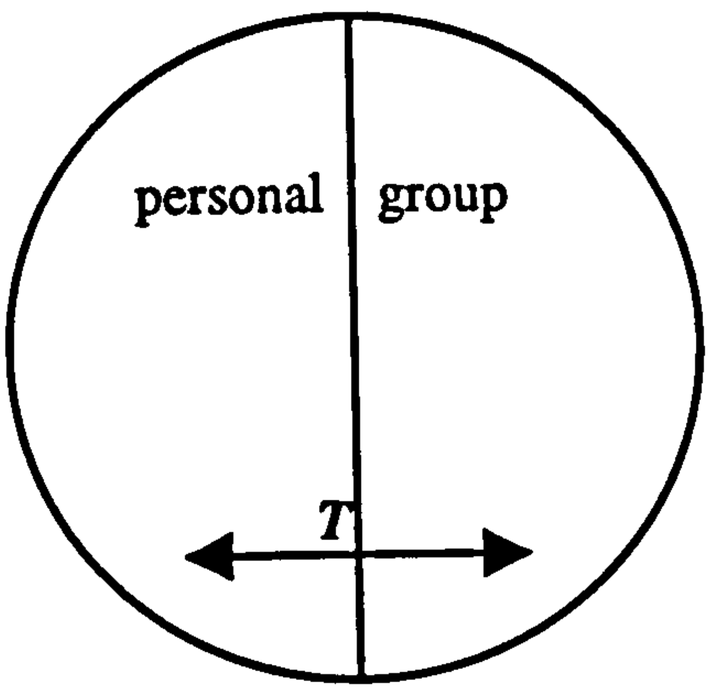


T denotes a transition from one set of work processes to another

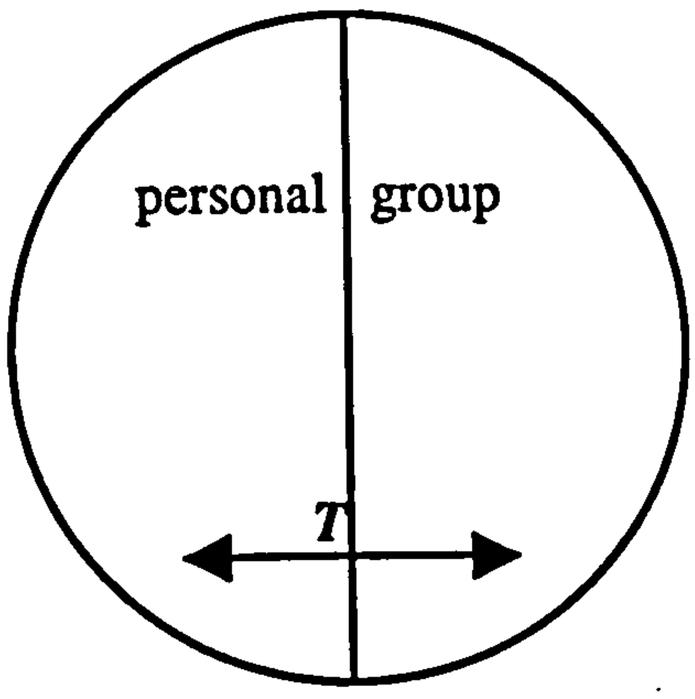
Figure 2.5: Users are currently required to make many major transitions between the work processes surrounding tools in both personal and group work environments.



Communication Tool Work Processes



Information Management Tool Work Processes



Authoring Tool Work Processes

T denotes a transition from one set of work processes to another

Figure 2.6: Designing from a reflexive perspective blurs the distinction between personal and group work, reducing the number and scale of transitions users are required to work.

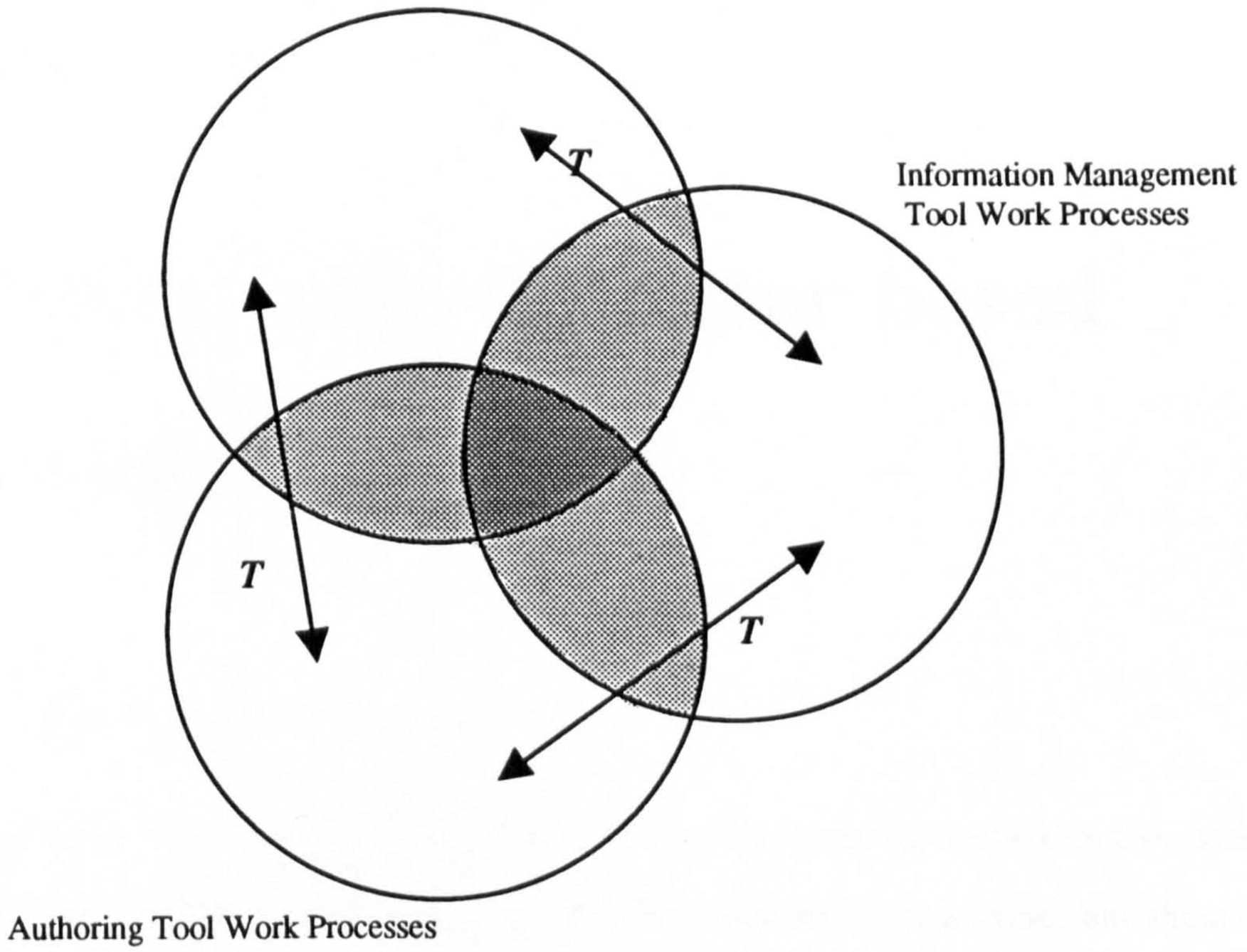


Figure 2.7: By reducing system imposed constraints on users, and making systems hardware independent there is potential for work processes to overlap. Combined with systems designed from the reflexive perspective this approach will lead to more integrated environments where work processes are generalizable.

Chapter 3

Writing and computer based writing systems

3.1 Defining a 'writer'

Most of us are not thought of, and do not think of ourselves as 'writers' but we most definitely are. Such a title should not be reserved for those who write for a living, but should be applied with equal validity to a much wider population. Writers of novels, essays, letters, memoranda all combine plans, thoughts, ideas, opinions and knowledge to produce text.

Of course, this introductory paragraph makes no mention of the true complexity of the writing process, nor is it concerned with the diverse pressures that different types of writing task have associated with them. However, writing plays an intrinsic part in the lives of most people, at a variety of levels.

If most of us are writers, by this definition, what constitutes 'writing'? It can be thought of as a wide ranging activity, that constitutes generating ideas, forming opinions, drawing on knowledge, and as a result creating a text which will communicate to ourselves or others those ideas and opinions and that knowledge at a later date. The resulting text must be in a form accessible by others, such as pen on paper, computer printout, paintings on cave walls or newspaper type. The production and dissemination of the text are also considered to be parts of the writing activity.

This chapter considers writing in order to gain an understanding which may be useful in

development of a computer based writing support tool. It is concerned with the writing activity, specifically the writing process, which is considered to exclude the mechanics of creating a tangible product, such as applying pen to paper, and the dissemination of the result, yet include the process of generating ideas, drawing on knowledge and creating a text which may require review.

First we consider why computer support of writers may be necessary, and then look at models of the writing process and collaborative writing, view writing as constraint management and consider the importance of revision. Finally, after investigating models of writers and strategies which they employ we propose requirements for a generic writing support tool.

3.2 Why might writers require computer based support?

If Flower and Hayes (1981) are correct when they say that "Writing is among the most complex of human mental activities" then the provision of computer based tools to support it would surely be worthwhile.

In 1980 Corbett (1981) contended that the activity of writing was proliferating rather than vanishing and that the widely emerging computer technology had not made it obsolete. He has been proven correct: in the decade since, there has been a major advance in the spread of computing on a personal basis, and many more people have greater access to computer based tools, which can be used in the writing activity or otherwise.

In fact, the writing task has been very widely supported by computer systems since their inception. There is a problem, however, with the level of support that has been provided. Computer based tools provide little support for the writing process.

3.2.1 Text entry

Computers have been used to store and retrieve information since early in their development. The information was initially only numeric, but systems quickly developed which allowed text to be stored and retrieved. The ability to do this created a niche for a tool which would allow a user to alter and manipulate the retrieved text and then store it again. Hence the proliferation and widespread use of interactive editing systems. Such editors were originally line oriented,

where the user could only see and manipulate a single line of the text being edited at any one time. Then came stream editors, and then display editors, which would allow several lines of the text to be seen at any one time. The context of the line being edited could be more easily established, and text traversal was easier and more directly indicated.

Graphics based editors were developed with the advent of powerful hardware that made use of bitmapped graphics displays. The aim of such tools was to extend the functionality of conventional text editing systems, whilst making the interface less complicated. Syntax directed editors were developed which would constrain the user to adherence to a certain syntax. These were generally used for program development where the strict syntax of a programming language is of upmost importance, in order to eliminate errors at the text entry stage rather than after compilation. Sophisticated word processors then appeared which greatly increased the manner in which a user could manipulate text. Spelling could be automatically checked, word substitutions done automatically throughout the whole document, different typefaces and fonts selected and tables of content and indexes automatically generated. Meyrowitz and Van Dam provide excellent, detailed histories of interactive editing systems in (Meyrowitz & Van Dam, 1982a) and (Meyrowitz & Van Dam, 1982b).

3.2.2 Text output

Tools of the types described above are very useful for entering and manipulating text, but in general do not address the problems of obtaining a printed version of the text in a formatted or typeset manner[1]. Hence the development of tools which will take text and format it for output onto paper. Such formatting tools are surveyed by Furuta *et al.* (1982). Many of these systems require authors to 'mark up' a document to describe how they wish it to be formatted. Coombs *et al.* (1987) describe methods of textual markup for formatting. The three most common are:

presentational : the markup of higher level entities such as horizontal and vertical spacing, page breaks and enumeration. WYSIWYG editors such as MacWrite or Microsoft Word require, or encourage authors to carry out this markup method;

[1]Some word processing systems are also partly text formatting systems. For example WYSIWYG editors allow the user to format text on a display, using different typefaces, fonts etc, and the printed version will correspond directly to the display version.

procedural : the author includes commands recognized by the formatter in the text to indicate how the text should be formatted. The nroff formatting system, for example, is based on procedural markup;

descriptive : text tokens are given element types by the author, such as chapter, section, figure, and enumerated list, and the formatter refers to a rule base for the element types to produce the formatted document. LaTeX (Lamport, 1986) is an example of a descriptive markup language.

3.2.3 The writing process

The preceding two sections described how systems have been developed to support stages of document production. The text entry and manipulation tasks are supported and other tools support formatting of the text for output. Many such systems exist and are widely used. The same cannot be said, however, for tools to support the actual writing process. There are few widely used systems that try to help an author in actually writing a document, and fewer that attempt to support more than one author in writing a single document. The next section will describe some of those that exist.

3.2.4 High level support of writing

There are many examples of systems which allow for the entry of generated text such as word processors and text editors (Meyrowitz & Van Dam, 1982a; Meyrowitz & Van Dam, 1982b), and for the production of formatted tangible versions of the text (Furuta *et al.*, 1982). Few support the actual process of generating ideas and text, and revising the result. Dissemination of the product of the writing process also has relatively little computer-based support. Dauite points out

> Writing on the computer means using the machine as a pencil, eraser, typewriter, printer, scissors, paste, copier, filing cabinet, memo pad, and post office. Thus the computer is a communication channel as well as a writing tool. (Daiute, 1985, page xiv)

This is somewhat of a generalisation, but is correct in that the computer *can* support some parts of the writing activity, and *can* be used as a communication tool and thus for disseminating the product of the writing process. These possibilities need to be fully exploited.

However, as one would expect when faced with a complex cognitive activity, people experience a variety of problems. Williams (1990) suggests several writing-specific problems which technology may be able to reduce

- mastery of language, such as mechanics of spelling, grammar and punctuation;
- writing clearly with unfamiliar material;
- getting over writer's block;
- text may seem fixed once written;
- organizing information;
- getting information in the first place;
- meeting the needs of an audience;
- being too critical or not critical enough;
- frustration.

These are high level problems, however. They offer little indication of the lower level cognitive activities that might be supported in helping writers to solve these kinds of problem. Collins and Gentner (1980) state that an advantage of a precise theory of writing is "...the possibility of embedding it in computer technology". To look more closely at theories of writing, then, may well prove to be useful.

3.3 Modelling the writing process

Just as the term 'writing activity' is not specific enough for our purposes if we are to provide computer based support, the term 'writing process' is equally non-specific. A deeper understanding of the cognitive processes of a writer are required. Understanding the writing process

and hence its many complex and interrelated subprocess is important if it is to be supported by computer based tools.

First, consider a more abstract view of the writing process, which is not concerned with specific and detailed subprocesses. Frederiksen and Dominic (1981) propose four different perspectives on the process of writing. They propose that it can be considered as

a cognitive activity : cognitive processes underlie and enable the use of knowledge and expression of meanings in writing;

a particular form of language and language use : writing processes are influenced by language forms and principles for selecting and using them in different contexts;

a communicative process : writers need to be aware of the need to alter content and expression to suit the audience;

a contextualized, purposive activity : the writing process is influenced by the the situations that writers write in and their motivations for writing.

These are four separate perspectives on the writing process but are not mutually exclusive. They undoubtedly intersect, introducing complexity when attempting to view the process at a high level. To clarify the issues and be more specific, attempts have been made to model the underlying lower levels and show how they interrelate.

3.3.1 Hayes and Flowers' three phase model of the writing process

Models of the writing process have been proposed in an attempt to represent the complexity of, and interrelation between lower level tasks. Hayes and Flower (1980) contend that writing is really a process that takes place in three main phases which themselves consist of subtasks. The first of these three is the *Planning Phase.* It consists of three subprocesses: generating, organising and goal setting. The planning phase is preparation for the next phase which is concerned with generating the actual text. They refer to this as the *Translating Phase.* The third phase that they describe is *Reviewing*, which consists of two subprocesses: reading and editing.

1. **Planning** consists of three subprocesses, and its function is to take information from the task environment and memory, and set goals for a writing plan.

   generating retrieves relevant information from long term memory. When a suitable item is found, a note is generated which is usually from one word to a sentence long.

   organising selects the most useful items from the generating phase and organizes them into a writing plan.

   goal setting identifies and stores criteria by which to judge the text;

2. **Translating** is a subprocesses during which the writer is guided by the writing plan to produce language corresponding to information in memory;

3. **Reviewing** consists of two subprocesses, and its function is to improve the quality of the written text. The text is read and editing takes place to improve the quality of the text produced in the translating process, and can interrupt any ongoing process.

Hayes and Flower go on to say, however, that this model is recursive and not necessarily sequential. Therefore the three main processes can in turn become subprocesses of other processes. Also, those processes could be carried out in any order. They are describing, then, elements of the writing process but not time or ordering constraints, and so what appears initially, to be a useful and detailed model, proves to be a loose description of the writing process. Figure 3.3.1 provides a graphical description of this model.

The model of Hayes and Flower was provisionally proposed to guide research into writing processes, which it indeed has done. Experience gained from many protocol analyses resulted in the development of their model, which was successfully applied in the analysis of a writer's protocol. Although limited in scope, the core of their model is widely held to be a solid building block for representations of the writing process.

Holt (1989), however, criticizes their model because it was developed using problem solving analysis techniques. His contention is that writing is not a 'solution' to a fixed problem. Additionally he questions their sample size, subject profile and lack of insight into subjects.

Given these limitations their model is still useful if considered with them in mind.

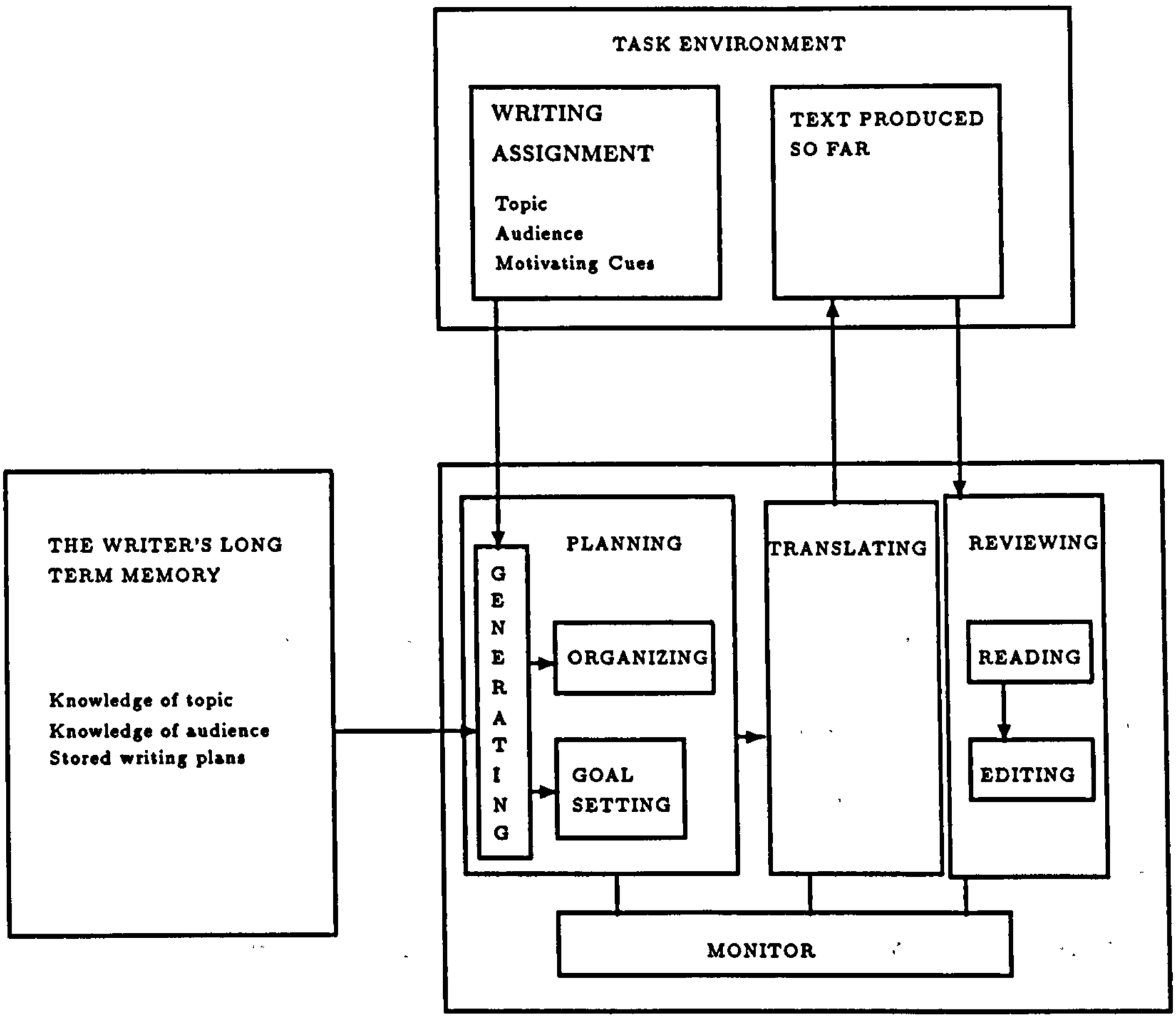


Figure 3.1: Hayes and Flowers' model of the writing process.

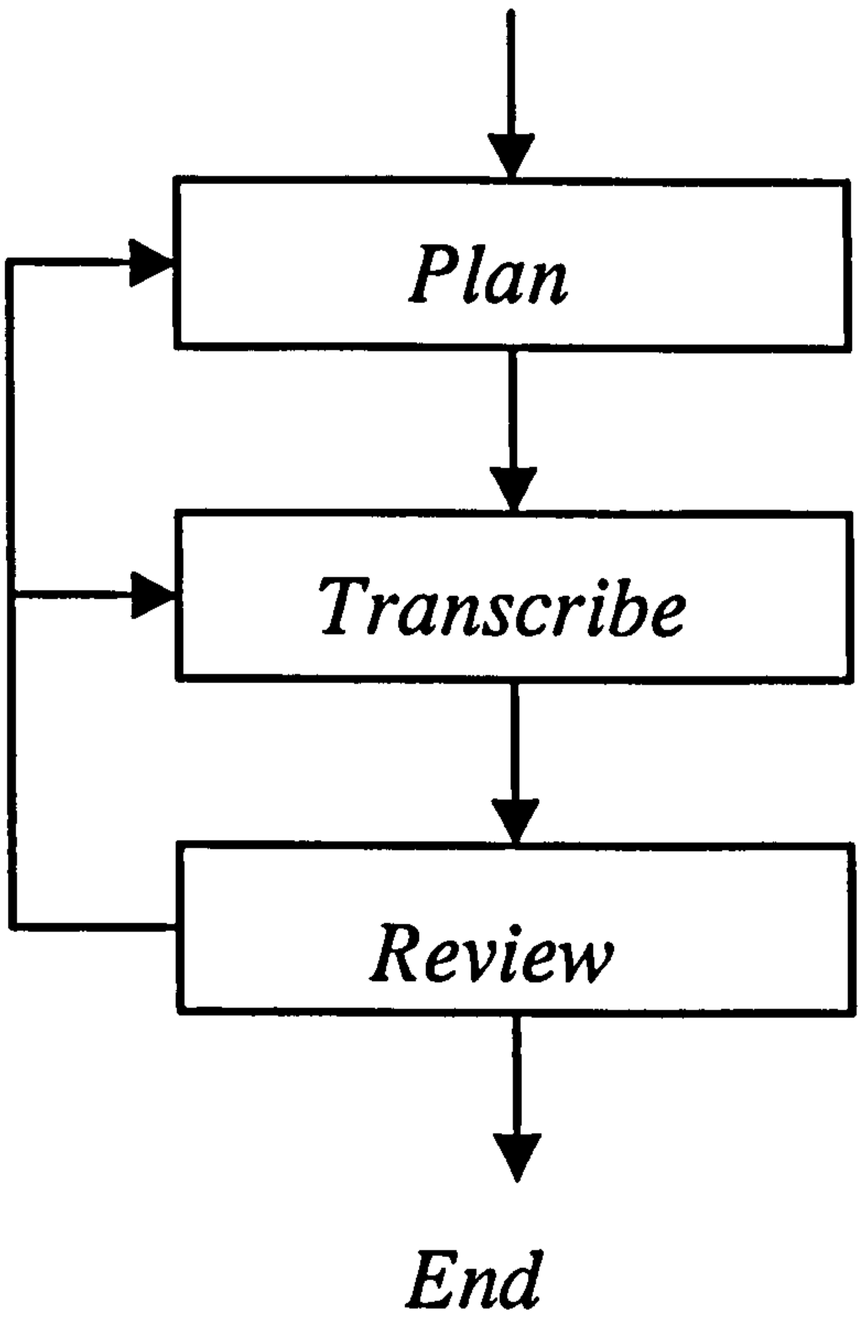


Figure 3.2: Nold's simple model of the writing process.

3.3.2 Nold's three phase model of the writing process

Nold (1981) offers a model of the writing process which is similar to that of Hayes and Flower. It differs slightly, though. Whereas Hayes and Flower provide descriptions of subprocesses but explicitly state that the model can be recursive and has no strict sequence Nold, as can be seen from Figure 3.2, presents a model which imposes a strict ordering on the execution of processes. It doesn't appear to allow for an author who immediately reviews a plan having derived it, or wishes to return to the planning process immediately after transcription. This would be acceptable if data had been provided to show that such actions were not carried out by writers, but this is not the case and doubt must be cast on the accuracy of the model's procedural constraints.

Elaborating on this model, Nold describes its subprocesses to be very similar to those of Hayes and Flower, with three main processes: planning, transcribing and reviewing.

1. **Planning** stores three products in memory. They are representations of

- the writer's intended meaning
- the writer's intended audience
- the persona the writer wishes to project;

2. **Transcribing** draws from language knowledge to produce writing;
3. **Reviewing** the raw materials for this process are memories of the text refreshed by reading. Elements of the text which are evaluated as faulty are deleted and altered and new ones inserted.

Hayes and Flower's model and that of Nold will be seen to be similar, yet conflict. That of Hayes and Flower is more complete as it does not deny the potential for close integration between the phases of the writing process. However, Nold's model is not necessarily incorrect, it may correspond exactly to the activities of some writers. Hence the two models serve to both provide a basis for consideration of the writing process, yet highlight the difficulty in providing an exact description of it.

3.3.3 A representation space for writing

The models described above could be categorised by the fact that they deal with only the mental processes of writers. They do not address how writers interact with available external media during the document development process.

Sharples *et al.* (1989) point out that it is necessary to consider the operations, strategies, and techniques carried out on some external medium, if a tool is to be developed which supports manipulation of external representations rather than mental structures. The *representation space* for writing, or 'six box model' that they propose can be seen in Figure 3.3.3.

Sharples *et al.* indicate that writers create three types of text item. Uninstantiated items are, they say, incomplete representations such as section headings. These are likely to be expanded later. Instantiated items are pieces of connected prose, and annotational items are meta-text elements. Notes are described as uninstantiated items, yet in the model are instantiated but unorganised items. The latter definition will be adopted here. The role of annotational items is not shown in the model.

Organization of items \ *Type of item*	**Uninstantiated**	**Instantiated**
Unorganized	**1** *Techniques*: Brainstorming *Representations*: Idea-labels	**2** *Techniques*: Note-taking (verbatim) Collecting quotes *Representations*: Notes
Non-linear organization	**3** *Techniques*: Following a thread Writing as dialectic *Representations*: Network of idea-labels	**4** *Techniques*: Organizing notes Filing *Representations*: Network of notes
Linear organization	**5** *Techniques*: Linear planning Outlining *Representations*: Lists of idea-labels Table of contents	**6** *Techniques*: Drafting text Revising text Copying text *Representations*: Linear text

Figure 3.3: The representation space for writing proposed by Sharples et al.

Adopting this model, the conventional goal of a writer is to reach box 6. However, for different writers the path to box 6 may vary. The path taken by authors using the models suggested by Hayes and Flower and Nold, which have the *plan-translate-review* process at their core, would be box3-box5-box6. The model itself, however, imposes no start point, end point, or path through the boxes.

The representation oriented model and the process oriented models are complementary. Together they provide a more complete representation of the writing task, addressing both cognitive and representational issues.

3.3.4 Linguistic models of the writing process

Cookson (1989) takes an alternative view of the derivation of models of the writing process which may be embedded in computer tools. The approach which she recommends considers both the

differences and similarities between spoken and written language.

Some example differences are

- speech is generated in a sequence over time, whereas writing entails making marks arranged in space

- speech is, in general, generated in a face-to-face communication, or with some other available feedback from the audience such as the telephone. Writing does not generally take place in the presence of the intended audience. Immediate feedback is lost, but there is more opportunity for review

- part of the importance of speech is in its social role (in addition to its role in imparting information. Writing is mainly used to record information, ideas etc.

There are also similarities:

- speech is driven by constraints derived from its role as a communication medium, as is writing

- speech is influenced by constraints derived from the nature of speech perception, and writing is influenced by those derived from reading.

The model described was in its infancy, and far from concrete. It is not clear whether it will be appropriate for embedding in computer support of writing.

3.3.5 Writing as constraint management

There are many factors to be considered during the writing process, rendering it, as Hayes and Flower have said, one of the most complex mental activities that we perform. In order to be successful, writers have to satisfy many demands on their abilities. They have to convey their meaning to the intended audience, write in a style that suits the audience, adhere to rules of grammar and spelling, include salient points and exclude spurious ones, work under time limitations within the scope of their knowledge and so on. These demands may well be complementary such as the first two or may well contradict each other such as the need to write

1000 words in 30 minutes on an unfamiliar subject. To be successful, a writer must attempt to fulfill each demand as successfully as possible.

Such a view may well provide an alternative formalisation of the writing process. Indeed Flower and Hayes (1980) state "...the act of writing is best described as the act of juggling a number of simultaneous constraints." This could become a fifth perspective to complement the four offered by Frederiksen and Dominic in Section 3.3. Flower and Hayes describe the three major constraints as being

Knowledge when it is not in a form acceptable for the task in hand;

Written speech : spoken language conforms far less to rules of grammar and correct delivery, and contains many mistakes which are not usually of great importance. Written language, however, must be far more accurate and so turning verbal thoughts into text is a demanding task;

Rhetorical problem : what writers write must conform to their purpose, their sense of audience and how they wish to project themselves.

The third constraint can be seen to break down into further constraints. Success during the writing process depends, then, on the writers ability to juggle these constraints successfully without becoming overloaded with attention demanding tasks.

Collins and Gentner (1980) concur that constraints play an important role in writing and claim that writing is a process of generating and editing text constrained by:

- structure (what are good forms for sentences, paragraphs etc);
- content (what ideas need to be expressed and how can this be done);
- purpose (the writer's goals and model of the reader).

These are not quite as high level as those presented by Flower and Hayes, and overlap somewhat with their constraints of written speech and the rhetorical problem.

Nold is more detailed in describing the constraints which are active during the process of text production, describing those that operate at a far lower level:

- semantic layout (the need to highlight crucial information and subordinate that which is incidental);
- syntax (the text needs to correspond to the grammar of the chosen language)
- grapholect (a dialect of English used in formal written texts aimed at a wide and/or educated audience);
- word choice (must mirror the writer's intended meaning and create the correct association with the audience, and the function of the word in the context must be considered);
- physical layout (such as the use of headings, paragraphs and lists);
- orthographics (conventions of spelling and punctuation);
- motor skills (actually handwriting or typing etc).

Figure 3.4 shows Nold's view of the relationship between these constraints in both the planning and transcribing stages.

Having acknowledged that varying levels of constraints exist, and that ability to cope with them affects quality of performance during writing, what can writers do to minimize their detrimental effect? Flower and Hayes and Nold agree that relying on well learned procedures can reduce the cognitive demands of overcoming the many simultaneous constraints. Of course, many people have learned to do this to some great extent without thinking about it. Imagine adding how to spell, how to construct sentences and how to punctuate to the tasks to be considered. Fortunately over time, with practice and experience, these tasks have become routine and reliably accurate. Flower and Hayes (1980) offer other strategies for reducing constraints

- throw constraints away;
- partition the problem;
- set priorities and attempt to 'satisfice' them.

It is likely that writers will in some way succumb to the pressures of constraints regardless of, or perhaps due to, the number of strategies that they need to employ in order to reduce their effect.

It is therefore unlikely that the translation process will be completed without the introduction of errors. Revision can help to rectify them.

3.3.6 Revision as part of the writing process

If a writer overcomes constraints to produce text, revision may take place. During this process writers add and delete elements of the text because they have evaluated them as faulty, and can think of a way to improve them (Nold, 1981). Nold places a great deal of importance on revision, contending that one of the major differences between skilled and unskilled adult writers is in the way that they revise.

Nold's view is lacking, however, in that it purely considers revision as a means for rectifying mistakes and making improvements. Matsuhashi (1987) considers it as a more complex task which can be viewed in the context of

repair when it can reflect cognitive strategies when used to achieve correctness and make text fit intentions, when errors in the text of others can be rectified but not an author's own, or additional necessary information is inserted;

reading it can be ineffective if it is merely re-reading the text;

shaping at the point of inscription constant re-evaluation of text, and its direction and emphasis.

Nold and Matsuhashi are in agreement that in order to be fully effective revision should be applied to both content and structure. *Downsliding* (becoming preoccupied with more local levels of problem resolution) is a common problem. Revision as reading will generally only result in local (mainly spelling, grammar, punctuation) problems being rectified ie surface corrections rather than those concerned with content.

An advantage of word processors is that computer based text is revised more because it has fluidity rather than the permanence of pen marks on paper (Bridwell-Bowles *et al.*, 1987). Matsuhashi points out that the presence of text can hinder the revision process, however. It might not hinder the amount of revision that takes place, it might indeed increase it, but it can have a detrimental effect. She claims that it encourages more local revisions, distracting

the writer from higher-level revisions. This may result in text that is more grammatically accurate with fewer surface mistakes, but it is less likely to have structural and content mistakes successfully rectified.

Haas and Hayes (1986) concur with this hypothesis, but attribute it to the fact that the task of actually reading from a computer screen hinders the writer. Performance on location, comprehension and revision tasks was, in general, significantly better when hardcopy was used instead of computer based text. However, just as importantly, it was found that computer based performance could be improved with different spatial representations of the text.

3.4 Strategies used during the writing process

As stated in Section 3.3.5, writers can reduce the cognitive load placed upon them by the demands of the writing task by resorting to tried and tested routines. In addition they can employ strategies to achieve success, and each writer is likely to have his/her own strategies which are best suited to them. The choice and implementation of a suitable strategy could be a factor in the distinction between good and bad writers.

In his paper "Specific Thoughts On The Writing Process", Wason (1980) actually has some rather general strategies for successful writing. He proposes two rules: the complete text must as far as possible be written in a single session, and during this session no sentence should be re-read or altered. This is his personal strategy, however, and one that has been tried and tested many times and proven to be successful for him. It could be disastrous for other writers. This highlights the fact that each writer will have a personal approach to writing. But is there an optimal approach, and are general strategies of benefit to writers?

Flower and Hayes (1981) view writing as essentially a problem solving process, and claim that people draw on heuristic procedures or strategies to achieve a successful solution.

> Good writers have a repertory of powerful heuristics which might include *brainstorming*, *planning*, or simulating a *reader's response*...(Flower & Hayes, 1981)

If it is the case that individuals have unique strategies, or at best use one of many available strategies then to support such strategies in a writing tool will be difficult.

Kellog asked

> ...whether strategies such as preparing an outline or composing a rough draft improve the efficacy of writing and the quality of the resulting document (Kellogg, 1989)

He hypothesized that such strategies could provide benefits for writers by reducing the aforementioned attentional overload that hinders the writing process. After experimentation he found that outlining did improve the quality of the product but did not improve efficiency and that writing rough drafts had no benefit to either aspect.

Collins and Gentner (1980) propose detailed strategies for both idea production and text production:

Idea production They propose seven available options for stimulating idea production: identify the dependent variable, critical case generation, compare to similar cases, compare to dissimilar cases, simulate, taxonomize, dimensionalize.

Text production This task can be stimulated and eased by choosing a suitable

objective which could be make the text enticing, or comprehensible, or rememberable, or persuasive. Each of the many possible objectives will have different levels of importance in different types of texts.

structural device such as pyramid form, narrative form, argument form and process of elimination form

stylistic device such as metaphor, suspense, or pictures

content device such as hierarchical structure, connective flow

Flower and Hayes (1980) say that "Plans allow writers to reduce cognitive strain" and that the most suitable plans "...appear to be the sketchy flexible sort that recognizes priorities and defines the writers high level goals". As Collins and Gentner proposed plans for idea generation so do Flower and Hayes (1981): procedural plans stop the generating process being diverted, pursue interesting features even if it is not evident where such a pursuit will lead, think by conflict, say what you really mean in order to reduce a body of information to its essential features, find

a focus for what is being written. They also propose higher level plans for producing a paper: organising ideas and implementing lower level plans to deal with them, using reader-based plans to render writing a process driven by a continual consideration of the audience which influences both global and local decisions, product-based plans using which writing is a process driven by continual consideration of features of the final product. They also consider how plans interact. It is possible and highly probable that a writer will switch from one plan to another. From pursuing interesting features a writer might feel it is time to find a focus and switch from one to another. Plans can map onto one another so that one plan can accomplish both tasks. However, plans can also be in conflict, a situation which must be resolved.

Hartley and Branthwaite (1989) carried out a questionnaire survey of 88 productive British psychologists and identified patterns and characteristics of productive writers.

When methods of composing were considered, the respondents were seen to fall into two main clusters: 'thinkers' who planned ahead, worked on elements of the text in any order, worked on different sections and produced more drafts; and 'doers' who completed one section at a time, in sequence and were more systematic. It was also found that 'thinkers' were more enthusiastic and confident about their writing than 'doers', and in fact, more productive.

The more productive writers of books were less likely to write sporadically, and more likely to claim to write the text in a single draft, with no clear sequence, than were less productive book authors. Productive chapter writers were more likely to accept commissions, be more confident about the quality of their work, concentrate on one main theme and write one section at a time than their less productive counterparts. When productive paper writers were considered it was found that they thought their writing important but did not enjoy it very much, rarely had writer's block and seized any opportunity to write.

These insights perhaps hint that authors have different priorities and strategies dependent on the task, and can perhaps be divided into two broad categories, each favouring a different approach. However, as Hartley and Branthwaite point out "It was not possible to find overall patterns of writing in responses to the questionnaire as a whole" and there are "...still many idosyncratic individuals".

3.5 Models of collaborative writing

This chapter has thus far considered models of writing processes and writers as individuals. This section takes a broader view of the writing activity, presenting insights into how people write together, which may prove to be appropriate as an underpinning for the development of a computer based tool which supports collaborative writing.

Although the models discussed so far consider writers in isolation, there are many occasions when writing tasks involve others in the process. Co-authoring may be explicit, or may take the form of review, annotation, research, illustration and so on. I would suggest that it is rare that ostensibly single author publications are, in fact, the result of only one person's labours. Published texts, from academic papers to popular novels to newspaper articles are subject to review and suggestions for change. Although the ideas within this thesis and the words used to express them are mine, it would be egotistical to claim that they have not been influenced by the praise, and indeed criticism received from friends and colleagues. Without this feedback, it would be a very different piece of work. Of course, it is still possible to claim single authorship because I have considered which of these suggestions should be considered or ignored and the final work reflects my view alone of the finished document. However, consider if by necessity this document were to be revised to adhere to specified requirements concerning both content and structure. The words contained within would still be mine, but the issues they describe would not, nor would the structure adopted to do so. Yet it will still be viewed as a single author document.

This raises the question of how to define single author and collaborative authoring tasks. What scale and type of collaboration is required for a document to have co-authors? No answer is presented here. However, this section reports on studies of what have been referred to in those studies as collaborative writing tasks. It provides an insight into the prevalence of collaborative writing activities and the strategies that groups adopt in order to produce a document whose final form is influenced by several individuals.

Subramanyam (Subramanyam, 1983) carried out bibliometric studies of collaborative writing in scientific disciplines in the early 1980s. This research showed that a great deal of collaboration takes place, but the amount can vary greatly between fields. This perhaps indicates that social

protocols regarding acknowledgment of contributions through shared authorship vary from one discipline to another. Subramanyam also notes that since the start of the 20th century there has been a increasing trend towards multiple authored papers. This does not necessarily indicate that a great deal more collaboration is occuring; there may just be a growing willingness to acknowledge the contributions of others, even if it does not entail completely shared responsibility and workload. However, given the rapid increases in communication technologies it is safe to assume that the opportunities and support for collaboration are far greater than ever before.

Indeed, Ede and Lunsford (Ede & Lunsford, 1990) and Couture and Rymer (Couture & Rymer, 1991) have recently carried out extensive surveys of writers. Ede and Lunsford received 700 responses to their survey of people in seven professions in the U.S.A. They observed a pervasiveness of joint writing activities in the work situation of those who responded. In all, 87% of respondents indicated that they wrote as members of a team or as a group. Couture and Rymer's study, which entailed a survey of 400 professional in the U.S.A., produced results which show less joint writing activity, although they may result from different definitions of collaborative writing. In their study 24% of respondents sometimes contributed to team authored documents or did so often or very often - a much smaller number than the survey of Ede and Lunsford suggests. However, Couture and Rymer's study also shows that 76% of respondents sometimes or more often talked over their writing with others before drafting.

Such studies serve to quantify collaborative writing activities, but as emphasised above, authorship in print does not necessarily reflect authorship in practice. To guide the design of a computer based collaborative writing system it is necessary to investigate the collaborative writing process in greater detail. Such investigations may reveal strategies that groups use to write together, the most effective group size, roles that are adopted and attitudes toward group document production.

Beck (Beck, 1993)provides a quite constrained but nevertheless enlightening survey of activities in collabrative writing tasks. She surveyed 23 academic co-authors. The survey consisted of 27 questions about co-authoring, mainly concering co-authoring in an academic context. In terms of motivation for the collaboration it was found that the most popular reason for establishing a writing group was to get a specific paper written, followed by a desire to work with other

group members. The most popular individual motivation for joining the group was enjoyment. In terms of the process, the study revealed that discussions regarding content and structure of the document took place mostly during writing and were believed to be adequate both at the time and with hindsight. The same response was received regarding discussions about organization of the work. Very few discussions after writing was completed were reported. Responses indicating inadequacy of discussions also indicated infrequency of discussions.

Although group sizes were, in general, small (just over half had only two members) the size was quite dynamic. 39% of respondents indicated that the number of authors in their group had changed during the process. In some cases this change was drastic; from 9 members to 1 member, and from 20 members to 3 members. Group organization was diverse, with a relatively even spread between group with a self-appointed leader, no general leader and an agreed leader.

In general the collaborative task was viewed positively, yet this conflicts with the fact that almost half the respondents had at some point during the collaboration changed their minds about remaining in the group. This may indicate crisis points within the collaborative process which were resolved.

The small sample size, omission of some results from the analysis because of ambiguous questions, and confusions in presentation of the results serve to undermine the contributions of this survey. However, it indicates general trends such as the dynamic nature of such groups, lack of common group organization and the positive way that authors view participation in group writing tasks.

Posner and Baecker (Posner & Baecker, 1993)used a smaller sample size still in their study of how people write together. They interviewed ten people about a total of 22 joint writing projects that they had been involved in. In general, subject's comments corresponded with the findings of Beck. They revealed a diversity of opinions about expectations about joint writing, the quality of the written document, authorship conventions, group size, the status of collaborators and criticism.

Posner and Baecker present a taxonomy of roles, activities, document control methods and writing strategies evinced by their study. Roles consisted of writer, consultant (who did not acutally produce text), editor and reviewer. Roles were seen to be changeable and interchange-

able and were sometimes imposed by the technology available to the participants. The size of the groups concerned ranged from two to five members.

Document control methods consisted of centralized (one person controls the document throughout the process), relay (one person in control at a time with control passing from person to person throughout the process), independent (each member works on a separate part of the document and maintains control of it) and shared (simultaneous and equal access to the document are available to several team members at once). The most popular methods were centralized and independent with over half the projects using them, sometimes in conjunction. However, such control methods are not static once established. 77% of groups experienced changes in control methods.

Writing strategies consisted of single (one person generated the text), scribe (and individual writes down what happens at meetings), separate (each member writes and is responsible for a different part of the document), joint (the group writes together deciding on structure and wording). All but three of the projects used the separate writer strategy at some point. Just over a half of the groups used the single writer strategy at some point.

3.6 Computer support for the writing process

What can be learned from this study of writing? Evidently, writing is a very complex task, during which the writer is subject to multiple external and internal influences which may be complementary and contradictory, necessitating prioritization. Most importantly there is not a consensus on how people actually write, what constraints must be balanced, and what strategies may be successful.

The proposal of multiple plans and complex strategies to reduce pressures and ease constraints may well complicate matters, introducing cognitive strain during the choice and implementation of strategies, rather than reducing it. There is a need for a tool which can reduce pressures and ease the process. Three possible approaches to the design for such a tool are apparent:

1. Recognize the complexity and diversity of approaches to writing and the diversity of techniques available to writers to ease the process, and attempt to embed them in a computer

system. This will lead to a large, complex system which may allow writers to select elements from a database of constraints and strategies to which the system constrains them. It is likely that this approach will not cater for all types of writer or writing task, nor the full range of complex interaction between constraints, plans and strategies;

2. Wait for more accurate and generalizable models and frameworks of writing tasks and writers before creating a system to support writers. This might require a lengthy wait;

3. Take useful indicators from studies of writing such as the importance of revision, placing as much stress on structural as surface issues, allowing brainstorming and idea generation, and allowing interaction between the basic planning-translation-revision subtasks. Build a basic support framework according to such indicators on top of which writers can bring their own strategies to bear. A socially mediated system such as this will be more useful in support of collaborative writing tasks where the diverse constraints are increased through the presence of multiple authors.

It is the third approach which is advocated here.

3.6.1 Existing systems

It has already been stated that there is computer based support for both entry and printing of text and other information, but little for the writing process itself. Dauite's (1985) taxomonomy of systems (see Table 3.6.1) would seem to support this.[2] Williams (1990) proposes a similar list of classes of computer application to support the writing process (see Table 3.6.1). It differs slightly from that of Daiute but presents several differing classes. In this case I have provided instances of the classes described.

3.6.2 Problems inherent in computer support of the writing process

Like any application area, the application of computer technology to the writing process has inherent problems. These are general problems, not necessarily linked to the context of writing.

- hardware failure;

[2] Dauite provides details of hardware required and distributors

Table 3.1: Dauite's taxonomy of systems to support writing related tasks.

CLASS	INSTANCES
Word Processors	Apple Writer II, Atari Writer, Bank Street Writer, Final Word, Homeword, Microsoft Word, PeachText, Perfect Writer, PIE Writer, Word Perfect, WordStar, PFS:Write, Screenwriter II, SuperScripsit, The Correspondent, Volkswriter Deluxe, Word Juggler, Word Vision
Integrated Programs	AppleWorks, QUILL, Writer's Helper
Mail Programs	E-Com, E-Mail, Electronic Messenger, Micro-Courier, Transend
Outlining and Organization Aids	Questtext III, Superfile, ThinkTank, ZyIndex
Prompting for Prewriting and Planning	QUILL Planner, SEEN, TOPOI, Prewrite
Prompting for Revising and Text-Analysis Programs	Catch, EPISTLE, Grammatik, Homer, Punctuation and Style, Writer's Workbench
Readability Programs	Readability Analysis, Readability
Spelling Checkers	Electric Webster, Lexicheck, Perfect Speller, Random House Proofreader, Sensible Speller, SpellStar, Spell Wizard, The Dictionary, The Word Plus, Word Proof
Typing Programs	Smartype, Typing Tutor, MasterType, Type Attack, Touch Typing Tutor, Typing Teacher

Table 3.2: Williams' list of classes of computer application to support the writing process.

CLASS	INSTANCES
Hypertext and Hypermedia	NoteCards (Trigg & Suchman, 1988; Trigg & Irish, 1988), VNS (Shipman III *et al.*, 1989), Intermedia (Catlin *et al.*, 1989), AUGMENT (Engelbart, 1988)
Networks for collaborative writing	Diamond (Thomas *et al.*, 1988), VNS (Shipman III *et al.*, 1989), Wang Freestyle (Hsiao & Levine, 1988)
Systems for dissemination of material	e-mail (Mackay, 1988; Pliskin, 1989), Information Lens (Malone *et al.*, 1986), Guided Tours and Tabletops (Trigg, 1988), Diamond (Thomas *et al.*, 1988)

- software failure;
- insufficient functionality to successfully carry out the task in hand;
- overwhelming functionality, confusing the user with facilities that are rarely used;
- poor technical and user documentation;
- poor user interface.

There are further potential writing-oriented problems:

- the writer may suffer from increased cognitive load by trying to incorporate the computer based system into already existing plans;
- constraints on the writer may well be increased rather than lifted. For example, a system which formats documents may result in formatting constraints being placed on the writer, diverting attention from the writing process itself:
- surface rather than structural revision may be promoted, because the text is continually accessible and surface changes can be easily and quickly made, with facilities such as spell checkers directing attention to surface rather than structural considerations;
- the system needs to be sufficiently flexible to support the multitude of approaches to writing employed by writers.

3.6.3 Integration with other systems

A fact to be considered when implementing such a system is how it can exist alongside the already existing systems to support the other stages of the writing task. There are two possible approaches. First, it could be developed to exist independently, with writers using other systems for text entry and formatting, and the purpose designed system for the creative process. This approach would require emphasis to be placed on compatibility issues—the tool would need to allow the user to access other tools and transfer data between them and itself. Second, an entirely new system could be developed which has its central emphasis on the writing process, providing integrated facilities to support the other stages of the writing task. This will reduce

compatibility problems, but unless the tool provides sophisticated functionality it is unlikely to meet the needs of all users, while at the same time denying them the opportunity to use other tools hand-in-hand with itself.

3.7 Requirements for a generic writing support tool

What support should a computer system give for the writing process? Sharples and Pemberton (1990) suggest support should be given for

- hierarchy of structure and tasks;
- interleaving of tasks;
- constraint management;
- basic operations such as cut, paste and undo;
- the reuse of previously generated text;
- predefined text styles such as letters;
- multiple views of the same information;
- creation of unordered items;
- creation of notes and networks of notes;
- providing drafts of linear versions of the text;
- manipulating a simulation of the final output;
- easy movement between views of the information;
- merging document text and 'meta-text' (annotations, elaborations etc).

I would concur the majority of this list with the exception of two points. First I do not believe that explicit embedding of constraint management techniques will cater for the needs of all writers. Second, predefined text classes are driven by the system designer's belief of which styles

are required and how those classes should be embodied. The overlap with the needs of writers is unpredictable.

Posner and Baecker (1993) identify implications for collaborative writing system design which result from their study of ten collaborative authors:

- preserve collaborator identities;
- support communication among collaborators;
- make collaborator roles explicit;
- support brainstorming, planning writing, editing, reviewing;
- support transitions between activities;
- provide access to relevant information;
- make plans explicit;
- provide version control mechanisms;
- support concurrent and sequential access;
- support write, comment and read access
- support separate document segments;
- support one and several writers;
- support synchronous and asynchronous writing.

I would agree with all of these design requirements except for the explicit nature of roles within the system. The following chapter will detail, with justification, which requirements will be appropriate in the design of an asynchnonous, distributed support tool.

3.8 Summary

Providing a computer based support system for writers is a difficult task. Such a system will need to be sufficiently general and tailorable to cater for the many diverse approaches of writers, yet provide functionality which will be of tangible use, otherwise few writers will use it.

One way to achieve this may be to gain a greater understanding of models of the writing process in order to embed them in the technology.

An alternative approach may be to consider writing as a process of managing many constraints, which draws attention from the creative task in hand and makes it far more difficult. If reduction of these constraints can be automated, the writing task may be eased.

Automation of strategies and plans for writing might also be beneficial. A system could provide support for predefined plans and strategies.

However, models, constraints and strategies are many and diverse and as yet insufficiently generalizable to cater for the needs of all writers. Dangers of embedding them in technology now are that they may provide insufficient support, overwhelm writers with the need to select from a large choice or constrain writers to behaviours which they would not normally adopt.

Although the provision of a system to support writing is undoubtedly a difficult task, it could yield many benefits:

> Computers, like other writing instruments, change the nature of written communication. [...] Such a writing instrument can blur the distinction between thinking, talking, and writing in a way that the pencil and typewriter have not. (Daiute, 1985, page vi)

and several desirable features of such a system have been identified.

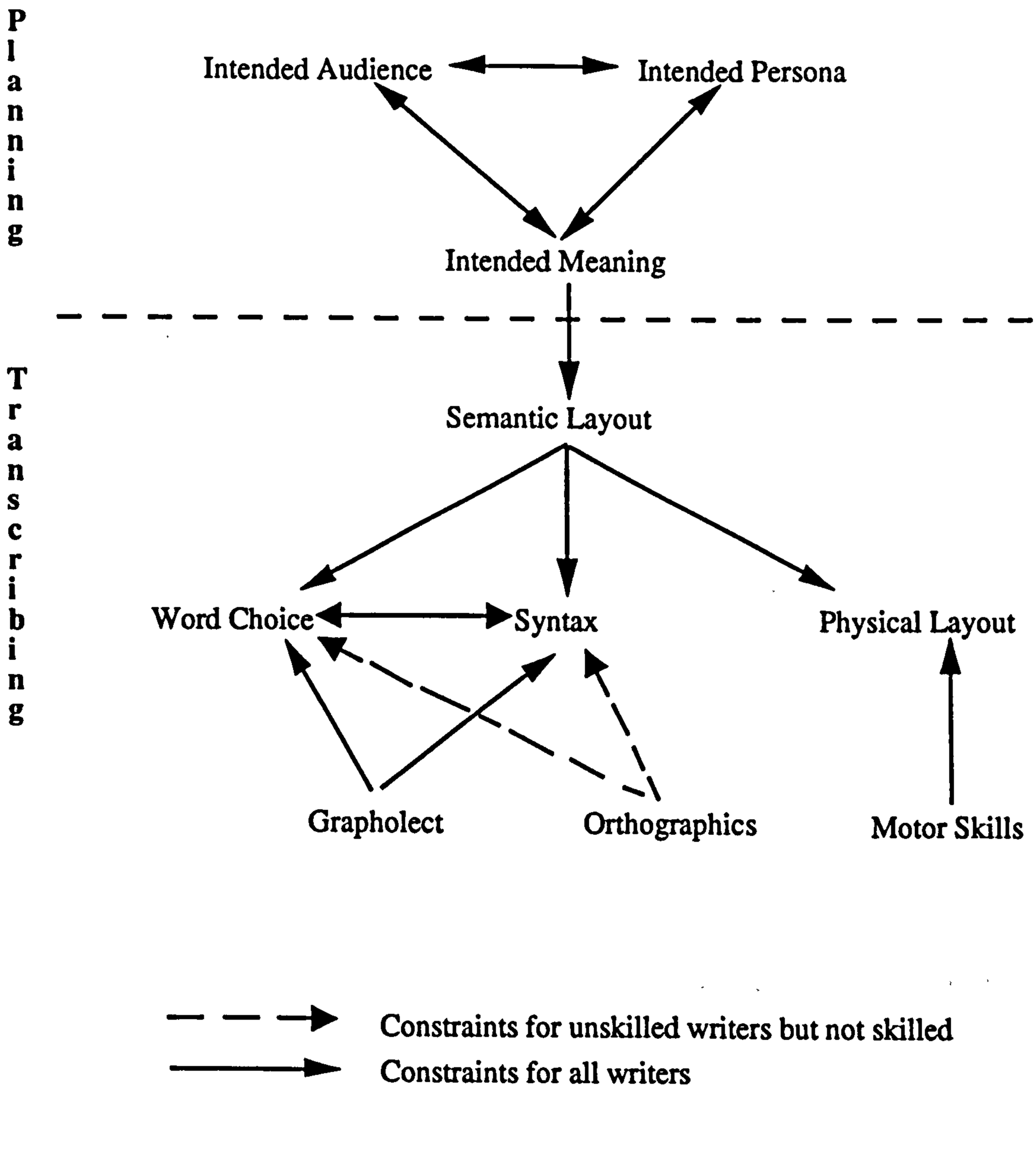


Figure 3.4: Nold's proposal for constraints on the writing process and the relationship between them.

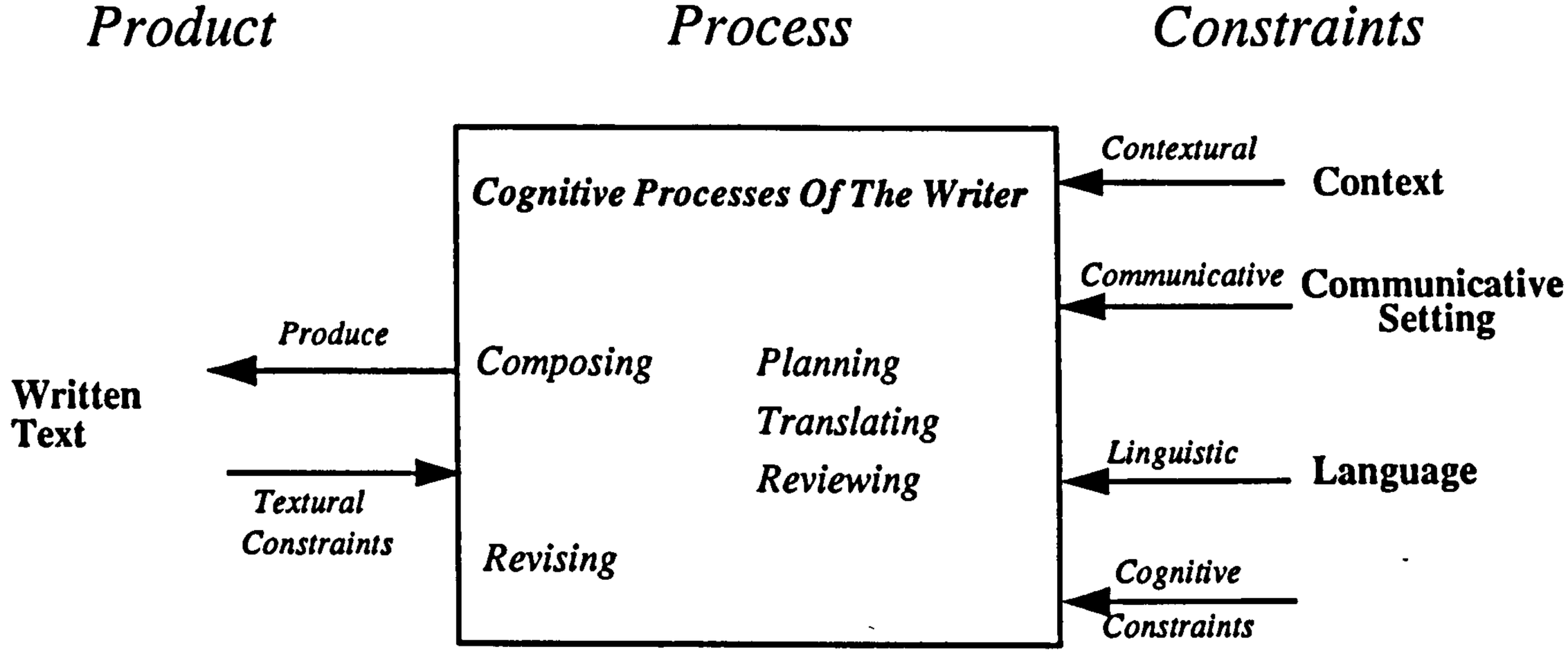


Figure 3.5: Frederiksen and Dominic's image of the integration of various parts of the writing process.

Chapter 4

Computer based writing support tools

Computer based tools such as text editors, graphics editors, idea processors, hypertext authoring systems, communication systems and document formatting systems can be useful to both individual and collaborative authors. Unfortunately, these tools are diverse in appearance and behaviour, and are unlikely to be compatible. Jones (1990) has highlighted the fact that although powerful tools are available, no single tool can as yet satisfy the majority of an author's needs.

As a result, authors can be faced with the daunting task of using several different systems during the writing process in order to cater for specific tasks such as idea processing, text entry, diagram production and document formatting. These systems may not even be available on the same hardware platform. Collaborating authors face more difficulties. They need to find a suitable communication system if they are geographically distant, and the tools used by one co-author may not be available to another. Kraut *et al.* (1988), for example, have observed that collaborators using computer based tools "...had difficulties with the incompatibilities among programs and computing environments".

This chapter discusses two computer based writing support systems as part of the contribution of this thesis. They are designed to support a range of tasks associated with writing and reduce technological limitations and constraints on writers. The systems are *Sticky-Notes* and

MILO, and both support distributed, asynchronous authoring of structured documents. The design of both systems is driven by the findings of Chapters 2 and 3. *Sticky-Notes* is used for an initial investigation into the utility of both personal and collaborative annotations, in addition to being a vehicle for implementation of concepts proposed to be useful in Section 3.7. Lessons learned from the prototype, *Sticky-Notes*, drive the design of MILO, a fully usable system to support collaborating authors. MILO is also designed from perspectives which increase the possibility of integrated use with other systems; see (Cockburn & Jones, 1991).

4.1 Designing systems for use *now*

A common belief in research of computer supported collaborative writing is that computer based systems should not be designed and built until the processes involved in collaborative writing are more clearly understood. There is no guarantee, however, having waited for understanding to be sufficiently advanced, that embedding it in a computer based system will benefit collaborators. It would be wrong to assume that the technology will be transparent and not affect the process. I would suggest that to provide computer based support for collaborative writing *now*, systems can and should be designed and implemented, drawing on ideas from sociological/psychological research, but also on concepts, attributes and experiences of existing systems. Evaluation will indicate successful implementation details, those that have failed, and those which need to be added or amended. As Sharples *et al.* (1993) point out, the design of highly interactive systems which are intended to support complex tasks cannot be fully specified in advance of implementation.

The 'implementation now' approach tangibly addresses issues such as user interface design for collaborative systems, computer support for individual writers, co-authors and communication between collaborators. It is the approach which has been adopted in this work.

4.2 Requirements for a system to support distributed, asynchronous collaborative authoring of structured documents

Chapters 2 and 3 of this thesis have presented overviews of CSCW, groupware (with an emphasis on writing systems), and writing theory. They have concluded with requirements for collaboration support systems in general, and collaborative writing systems in particular.

However, not all of those requirements will be applicable to the development of a distributed, asynchronous tool to support collaborative authoring. Figures 4.1, 4.2 and 4.3 indicate which of the requirements will be adopted or rejected and the reason for the decision.

4.3 Adopting user interface design guidelines

The design of a collaborative writing support tool is concerned with two main issues

- provision of appropriate functionality for the task
- provision of appropriate access to the functionality

Earlier chapters and sections in this chapter have been concerned with identifying appropriate functionality. To some extent this dictates aspects of the user interface. However, for an interactive system to be both useful and *usable* it is necessary to consider the design of the user interface explicitly. The design and implementation of the interfaces to the two systems described later in this chapter have been driven by commonly accepted interface design guidelines drawn from a variety of sources. The guidelines which were adopted are listed below. Chapter 5 provides a more detailed discussion of the guidelines and how they influenced the design of MILO.

- allow the user to see and point rather than remember and type
- user actions should be reversible
- provide consistency within the application
- provide consistency with other applications
- provide the user with feedback

- require user effort commensurate with the potential effects of actions
- do not allow the user's display to become cluttered
- allow the user to directly manipulate objects on the screen
- allow the user to draw on real world experiences and knowledge
- support the use in maintaining an accurate an current mental model of the data being manipulated
- protect users from hardware and software failure

4.4 *Sticky-Notes*

Sticky-Notes is a computer based simulation of Post-It notes, which partially supports an author's annotation and communication requirements.

The motivating factor behind the development of *Sticky-Notes* is the desire to investigate the applicability of the Post-It note metaphor in the field of computer based document creation and document related communication between multiple authors. It provides much of the support which is indicated as useful in Section 3.7: hierarchy of structure, interleaving of tasks, cut/paste/undo, text reuse, multiple views of information, creation of notes, easy movement between views, and document merging. This work also reinforces and extends the desktop metaphor used in the interface of many personal computers. This extension introduces computer-based representations of actual real-world objects—Post-Its—which can be used in a manner comparable to the real, tangible Post-Its.

Post-Its have become omnipresent in offices, and come in varying shapes, sizes and colours and their uses are many. They can be drawn on or written on in order to convey or store information. They can be used for personal reminders and storing information over the short term. For example, they can be used for noting dates of meetings, the content of telephone messages, telephone numbers and lists of things to do. They are normally positioned so that they are noticeable, perhaps attached to computer screens, desks, shelves or doors. They can also be used for inter-person communication, conveying requests or information to others. A

use which can be both reflexive or inter-personal is in document annotation. Post-Its can be attached to documents as either personal reminders or as information for others. They might contain such information as a distribution list, general comments on the document, comments on specific parts of the document (in which case the note is placed in the vicinity of the document section to which it refers), the name of the person who is to be the recipient or future action to be taken regarding the document.

The use of Post-It notes as an annotation medium allows information to be added to, or comments made on documents without the need to physically alter the documents. This contrasts with annotation of the actual document by hand using pen or pencil, when additions are of a more permanent nature. The content of a note may well merit transferral to the document at a later date, when it will be physically altered.

Additionally Post-It notes can be used to attach information or comments to existing notes, providing the potential for the development of a dialogue where contributors can add to or comment on both the document text and any other notes attached to the document, or indeed any notes attached to those notes, and so on.

Sticky-Notes allows creation of multiple document elements, creation of relationships between those document elements, multiple views of document content, annotations for both communication and reminders and doesn't constrain users to work methods or document types.

4.5 An implementation of *Sticky-Notes*

Sticky-Notes is implemented using HyperCard[1], a hypertext application for Apple Macintosh computers that also supports rapid prototyping of applications which utilise graphical user interfaces. Although designed to be a hypertextual application, HyperCard is widely used for rapid prototyping of graphical user interfaces because of the ease with which interface objects can be created and manipulated (Nielsen, 1989). In general, little programming is required, although the underlying functionality of *Sticky-Notes* is programmed in HyperTalk, the HyperCard programming language. The implementation of *Sticky-Notes* concentrates on the use of

[1]Version 1.0 of HyperCard was the only version available at the time. All further mentions of HyperCard refer to version 1.0

Post-It notes as a medium for annotation of documents and communication between multiple authors and commenters on documents.

The utilisation of notes in this computer-based implementation is intended to mirror closely the use of their real life counterparts. They can be attached to any part of a document and repositioned when required, they can contain text which either suggests change to the document, or comments on the document. Notes can be attached to other notes, and they can be easily disposed of.

The implementation of a metaphor, must, almost by definition, not correspond exactly to the real-world. The facilities provided by *Sticky-Notes* go beyond the potential of their real-world counterparts. Annotations within the system can be filtered, do not obscure the text they refer to, can be automatically transferred between communicants, maintain their context and can be automatically merged. Figure 4.4 shows a typical *Sticky-Notes* screen.

4.5.1 Filtering

Notes attached to text as annotations or communications can be filtered to select those which an author has a specific interest in. For example, any notes created by a collaborating author or notes created after a certain date. Through use of this facility the user gains automated access to multiple information items, rather than being required to search through potentially large amounts of information sequentially, selecting items of interest. Access to the information about authors and the timing of their contributions is part of the provision of a context-history of the interation which Miles *et al.* recommend (1993).

4.5.2 *Sticky-Notes* don't obscure the text they refer to

A drawback of Post-It notes is that they obscure the text to which they are attached and need to be removed or repositioned in order to gain access to the text. It is therefore possible that the context of notes referring to specific text elements may well be lost. Computer supported annotation must clearly maintain the context of notes in a manner which doesn't obscure the text which is referred to. It is possible to remove the body of a *Sticky-Note* from the document leaving a position indicator so that the underlying text can be accessed and the context of the

note is retained.

4.5.3 Automatic text transfer between notes and documents

Text within Post-It notes may be comments on document text and/or suggestions as to how the document text may be altered. If it suggests changes to the document text the author may wish to incorporate those suggestions directly into the document, requiring rewriting, a potentially time consuming process.

Text within *Sticky-Notes*, however, can be transferred automatically into the document text, taking the point at which the note is attached as the insertion point. The note will then become empty and can be disposed of. If the author wishes to maintain a record of the contents of the note (if they intend to amend the suggested text once in the main document, for example) he or she can copy the contents of the note into the document text leaving the note intact for later reference.

4.5.4 Automatic transfer of notes to colleagues

Sending Post-It notes to colleagues either as reminders, informational messages or as annotations on documents requires a certain amount of effort on the part of the sender, and there is usually a time delay before the message is received. Going to the recipient's office, hand-delivering documents or sending documents via surface mail require additional effort on the part of the note creator after the important task of creating the information to be imparted has been completed.

Using *Sticky-Notes*, an author is provided with a list of other *Sticky-Notes* users with whom it is currently possible to communicate. Cockburn (1993) has highlighted the usefulness of providing information on the availability of potential collaborators. The user can select from the list, and the document and notes are communicated to the selected colleague, appearing directly on their computer screen. Communication effort is reduced to a minimum and (if the selected colleague is currently active at their computer) the information is almost immediately available to the recipient.

4.5.5 Maintaining the context of annotations

The context of an annotation can be lost if a Post-It note is removed or repositioned. This may have confusing or even destructive consequences if its meaning is specific to a small document element but is thought to refer to a larger portion of the document. For example, a note stating 'This is rubbish!' may be attached to a diagram on page three of a document, but if the note is transferred to the title page of the document the meaning is substantially altered with a potentially destructive effect on the collaborative relationship.

Sticky-Notes allows the author of a note to indicate the degree of specificity of an annotation by allocating a level of adhesiveness to the glue with which the note is attached to the document. A high level of adhesiveness indicates that a note refers to a specific part of the document and consequently should not be moved away from its initial position. A low level of adhesiveness allows the note to be repositioned with freedom. This might be allocated if the note itself contains non-specific references or explicitly indicates that it refers to the document as a whole. The tangible result of this is that notes can not be physically moved away from their initial position if attached with a strong glue.

4.5.6 Automated merging of annotations

An author may well distribute a document to several colleagues for annotation. On return of the document from each source of annotations it is necessary to collate all additions in order to more easily assimilate suggestions or comments. This can be a time consuming process. *Sticky-Notes* provides the user with a facility which automates the merging of annotations from multiple incarnations of a document. Two *Sticky-Note* documents are selected to be merged leaving the user with a new version on which notes from both versions are visible.

4.6 Conclusions

4.6.1 *Sticky-Notes*

Sticky-Notes has been developed to investigate several issues related to the provision of writing support systems:

- the usefulness of annotations as a personal reminder utility;
- the usefulness of annotations as an author-to-author communication medium;
- multiple views of document content;
- automated merging of contributions of multiple commenters;
- representation of annotations;
- automation of communication to reduce the time and effort required for collaboration.

The system served to quickly clarify these issues.

The development of *Sticky-Notes* resulted in the following conclusions:

- the functionality described above, in addition to that described in Section 3.7, *can* be embedded in a computer system with relative ease;
- although few systems have brought together such functionality into a single system, there is no indication that this is because of technological limitations, or because the functions are mutually exclusive;
- the ideas under investigation (detailed earlier in this section) appear to be appropriate and useful and merit thorough evaluation;
- other potentially desirable support such as multiple views of documents, document drafts, manipulation of output, multimedia documents and multimedia annotations need to be investigated;
- regardless of the sophistication of a computer based system, it is realistic to assume that authors will require hard copy of documents such as those created using *Sticky-Notes*. Creation of printed versions of annotated document must be considered. For example, how will annotations be represented and contextualised in the printed version?
- how will authors collaborate with colleagues who do not use the same system? They may not even using the same type of hardware. *Sticky-Notes* unsatisfactorily requires collaborators to be using *Sticky-Notes* at the same time to facilitate communication of annotations, hence also requiring use of Apple Macintosh computers;

- collaborating authors are likely to be physically dispersed. Communication beyond the local network (as supported by *Sticky-Notes*) must be provided for;

- a richer communication medium than purely textual is likely to benefit collaborators, providing the potential for more expressive and natural communications;

- annotations are not necessarily limited in size or content. There is no reason why a single annotation could not hold an entire document. *Sticky-Notes* treats annotations as subordinate to the main text of the document. Presenting annotations as the only possible document element yet making them more powerful (multimedia content, no restriction on size, allowing annotation of annotations) would simplify document creation and manipulation for authors.

4.6.2 HyperCard as a prototyping tool

HyperCard has benefits and limitations for use as a prototyping tool for systems with graphical user interfaces (Thimbleby *et al.*, 1992). Positive aspects of it for such use are that

- interfaces can be built and amended by direct manipulation of interface objects;

- common operations can be programmed 'by example' without the need to know details of the underlying programming language HyperTalk;

- HyperTalk code is relatively easy to understand and existing programs can be easily tailored to other uses;

- applications can be built bit by bit, and in no particular order, unlike conventional system design.

However, there are also many limitations of HyperCard and HyperTalk

- the pseudo-English style of HyperTalk is inconsistent both in its English and its syntax;

- object reference through names, IDs or numbers is unreliable either because of HyperCard imposed changes during system development, or interface requirements;

- structuring programs is difficult, and the ability to 'hide' objects can lead to lost code;

- more complex programming (rendered necessary because of the limitations of HyperTalk) can be done using external commands in a language such as Pascal or C and then integrated into HyperCard. This is the type of task that users of HyperCard for prototyping are trying to avoid;

- despite its popular and accepted (by Apple) use it does not allow users to create applications which 'look and feel' like true Macintosh applications. It fails to support standard Macintosh interface objects, and standard Macintosh user actions.

4.7 MILO

Sharples and Pemberton (1990) state that "Support for writing should embrace the entire process, from registering the task to producing a finished manuscript". There is a need for a single tool which provides such support, eases communication difficulties and the collaborative process for co-authors, and is hardware independent. MILO is such a system.

MILO is intended to be a useful tool for authors of structured documents, into which they may wish to include both text and graphics. It provides facilities both for an individual and for multiple authors.

At the most basic level MILO can be used by an individual author as a text editor to construct a simple linear document. At the other extreme it can meet the needs of geographically distant co-authors who are writing a continually revised document of complex structure containing text and graphics. It also caters for needs anywhere else along this broad spectrum.

MILO can be used as an idea processor, allowing the user to note distinct ideas and create links between them, providing a graphical representation of the resulting structure and allowing its content and form to be edited.

Construction of structured documents is achieved through creation of a hierarchy of logical units called *notes*. The author is provided with alternative representations of the document structure and can take different views of document content. Fast access methods to elements of interest in potentially large document structures are provided.

The use of MILO can ease communication and information exchange between distant col-

leagues. Although MILO has an advanced graphical user interface, MILO documents are stored in a purely textual manner, allowing geographically distant collaborations to take place via electronic mail, and making exchange of MILO created documents system and hardware independent.

MILO also allows automatic updating of a document that has multiple authors, drawing together the contributions of all them into a single version of the document.

Figure 4.5 shows an example screen layout from a MILO session. It includes the main MILO window containing the menu of functions available for use on the whole document and the hierarchical overview. It also shows several note windows containing both text and graphics.

4.7.1 Related systems

Chapter two presented a variety of group work/writing support systems, and viewed them along several dimensions.

Along the time dimension MILO should be placed with any other asynchronous systems. Along the situation dimension it should be situated with any other systems which support distributed group working. Along the constraints dimension it should be situated with other minimally constrained systems. It is also a product rather than a process oriented system.

Therefore MILO is related to other systems in part but is not *directly* comparable to any single system.

The use of a note based metaphor, and organisation of items of information (in this case document elements) into a structure (in this case hierarchical) reflects influences from systems such as NoteCards (Trigg & Irish, 1988; Neuwirth *et al.*, 1988), Cognoter (Stefik *et al.*, 1987; Tatar *et al.*, 1991), InterNote (Catlin *et al.*, 1989) and Notes (Neuwirth *et al.*, 1988).

Multiple views of the document (structure based, outline based, time based, private, public) reflect systems such as Writer's Assistant (Sharples *et al.*, 1989) and GROVE (Ellis *et al.*, 1991b). The minimal imposition of constraints by the system on the user is similar to that of GROVE (Ellis *et al.*, 1991b) and CES (Greif & Sarin, 1987). Emphasis on document structure can also be found in GROVE, SASSE (Baecker *et al.*, 1993), the Writer's Assistant. Provision of a context history of the interaction is similar to that in the proposed system of (Miles *et al.*,

1993). Hardware indepence through use of the X Windows System is also a feature of VNS (Shipman III *et al.*, 1989).

However, MILO is novel in its implementation and representation of these features. The majority of comparable systems fail to combine all of these features, adopting one or two as the core support provided by the system. MILO draws together these features into a single tool.

Additionally MILO presents novel facilities such as semi-automated merging of documents co-authored in an asynchronous and distributed manner, hierarchical structure driven document creation, integrated document markup.

4.7.2 MILO and models of writing

The study of writing in chapter 3 highlighted the diversity of views on the writing process. Several models were presented, and dealt with both the writer's mental processes and external representations of those processes. Strategies for single author writing and collaborative writing were also described.

As stated in section 3.6, the approach adopted here is to provide a basic support framework, allowing authors to implement their own strategies and to enter into socially mediated collaborations. No model is *directly* reflected in the implementation of MILO, but commonalities of models have been considered and supported.

The abstract plan-translate-review model of Hayes and Flower, and Nold, and variations resulting from repetition of phases in those models is supported by MILO. The 'six box model' of Sharples *et al.* is also supported with representations of ideas, notes, table of contents, and instantiated, uninstantiated, organized and unorganized document elements. It does not, however, provide a network structure. The linguistic model considered in chapter 3 does not impact on the design of MILO.

Lower level activities integral to the models, such as orthographics, syntax and so on are not supported. Higher level issues such as structural considerations are supported, however. An aim of MILO is to emphasise structural issues and reduce problems such as downsliding.

Strategies such as outlining are supported but explicit support for stimulation of idea production or text generation, for example, are not.

The models that have been considered have influenced the selection of requirements which have already been stated in this chapter. Aspects of the models create requirements such as constraint management, explicit plans, write, comment and read access and so on. Hence, the models have been considered with regard to development of an actual system. Although they are not explicitly implemented, the models guide the requirements specification in order to provide the basic support framework which is the goal.

Section 4.7.9 considers how MILO supports the models of collaborative writing and group writing strategies which were discussed in chapter 3.

4.7.3 Notes

A MILO user manipulates a single data element type called a *note*, to create a document. MILO documents are built from any number of individual notes, and so a valid MILO document could consist of a single note or many interrelated notes existing in a hierarchical structure. Each note occupies its own screen space and there is no limit to the number of notes which can be visible at any one time. The complexity and form of the hierarchical structure is user defined.

All notes have exactly the same look, feel and behaviour, and consist of three elements. These are

1. A text field which is intended to hold the header or title for the individual note;

2. An EMACS like text editor which is intended to hold the main body of text for the note. The text editor behaviour is exactly the same for the two text elements of a note;

3. A simple graphics editor allowing the user to draw lines, rectangles, circles and text in a variety of pen patterns.

A user can also access information about each note, including who created it and when, and who last amended it and when. The information is continually updated providing users with information about the work patterns of themselves and their co-authors.

Some systems provide multiple document element types, such as annotations, graphics and document text. MILO takes the opposite approach providing a single, consistent element type.

This will maintain conceptual simplicity of document content. A note can serve as a document element, an annotation, a personal reminder or a communication medium.

Note relationships

Notes are organised into hierarchical structures. Most notes will have a hierarchical relationship with another note. This relational structure allows each document element to have any number of subordinate elements providing an author with many possible structures which could underlie a MILO document. Additionally notes can be created which are not subordinate to other notes, from which a hierarchical structure can evolve. Authors are therefore not constrained to a single structure which represents the whole document, and can integrate and separate independent structures as they wish.

There are therefore three classes of MILO document structure

1. The simplest MILO document will contain a single note. In theory, this should not restrict the user in the size of the document that she wishes to create, allowing anything from an empty note to a full paper within one note. In this instance the system is being used in a similar manner to a conventional text editor outwith MILO, but with the added power of the integrated graphics editor, and automatic mapping to a LaTeXdocument (see Section 4.7.5). Additionally authors can access information about creation and amendment times and authors and amenders of the note (see Section 4.7.3), and view more document text at any one time than most text editors allow.

2. A more complex single structure where notes have zero, one or more children, indicating a greater segmentation of the document into smaller, logical, hierarchical units.

3. A grouping of structures where notes have zero, one or more children. This enables authors to develop ideas, notes, reminders and document sections in parallel. A new note need not be immediately integrated into the main document structure; it may remain independent or may be integrated into any existing hierarchies in the current document.

4.7.4 Creating documents

An author creates a MILO document by adding notes to the existing document structure, or adding unattached notes as they are required. In the case of adding to an existing structure, an author will specify the position that the new element will occupy in the structure.

Figure 3.3.1 shows Hayes and Flower's (1980) three phase model of the writing process. This indicates that although planning, translating (text creation) and revision are central to the writing process, they are repeated and have no constraints on order of execution. This provides a loose description of how people write—approaches vary greatly. To embed a model of an author's approach to creating a document in a computer based tool may benefit those writers who adopt that specific approach, but is likely to alienate those who do not.

MILO addresses this problem by catering for diverse approaches to creating a structured document. At one extreme the author might create the structure of the document before entering any text or graphics. This indicates the initial hierarchy of the notes, and the relationships between document elements before the author concentrates on text generation. At the other extreme, the author may begin by 'writing to discover what she has to say' and generating a large amount of unstructured text which is then manipulated into a structured document. Of course it is possible for an author to want to use an approach somewhere between these two extremes, and MILO will allow this, catering for the diversity of writing styles which exist along the continuum.

The author can move the focus of her attention from one part of the structure to another, from one note to another at any time, and all notes are always easily accessible. During the creation of a document, text and graphics can be updated, added or deleted at any point.

4.7.5 Storing documents

A MILO document can be saved to a file. The file is a standard text file, accessible to other text based applications available to the user and its contents will reflect the entire structure of the document. The information written to the file can take one of three forms:

1. LaTeX (Lamport, 1986) format. When this option is selected, the hierarchical structure of notes in the MILO document is emulated in a LaTeX document. The LaTeX commands

to create a structured document are generated automatically and impose a predefined mapping of MILO document structure to LaTeX structure. LaTeX picture environment commands are also generated into order to create figures in the LaTeX document which correspond to the contents of the graphics editors of each note in the MILO document. *Sticky-Notes* highlighted the importance of providing hardcopy, or the potential for hard-copy, of documents which include non-linear elements. Through the automated generation of a descriptively marked-up version of the document, authors can not only easily create a printed version, but also do not have the overhead of marking-up the text themselves.

2. Plain text. This option will store only the textual contents and the title of each note. It imposes no structure on the document and contains no information about the notes or any of the graphical information contained within the MILO document. A file stored in this format allows an author to mark up the document for formatting in any way she wishes, outwith MILO.

3. A textual representation of the MILO document and its structure and attributes is stored. This allows the user to retrieve a document's text, graphics and structure from a text file and it has exactly the same appearance as it did when saved. An author can also email the exact contents and appearance of a MILO document to someone else using this format. This format allows another author to alter the document from within MILO at another site. New notes, text and graphics can be introduced and the structure itself can be changed. A development from *Sticky-Notes*, this allows for dissemination and viewing of multiple authors' work which can be seen in the context of the entire document.

4.7.6 Amending MILO documents

The importance of revision in the writing process is stressed by Nold (1981), and has been discussed in Chapter 3.To be a useful writing tool, MILO must then provide facilities for repeated revision of a MILO document during its lifetime. The author must be able to easily revise not only the textual and graphical content of a document but also its structure. There should also be no time constraint on revisions to a MILO document. It is always possible to revise a document during the current MILO session, but it should also be possible to return to the document at a

later date in order to carry out more revisions.

Hence, an author can read a MILO document from a file, restoring both the content and the structure of the document, and not only the textual and graphical content of each note but also the relationship between notes in the structure.

MILO also address the problem of finding relevant files from within a potentially large store of documents. When using MILO authors will in general manipulate MILO documents—other files become temporarily irrelevant. Hence, MILO provides users with a list of currently available MILO files from which a file of interest can be directly selected for amendment. Alternatively an author can enter the name of any file via the keyboard.

Editing content

The title and textual body of each note can be edited using the text editor provided. They support commands for movement, text deletion, insertion and selection. Text can also be selected using a pointing device with movement and button clicks. Once text has been selected it can be deleted or copied into another note within the MILO document currently being edited. Additionally it can be copied to any other application running under the X Window System (see Section 4.7.13 for details of MILO implementation) which recognises the X selection mechanism. Similarly, information can be selected in another X application and copied into a document being created or edited using MILO. This eases the problem of incompatibility with other systems, which many writing tools present the user.

The graphical content of each note can be edited in a basic manner. Objects can be added to the canvas at any time. The objects available are lines, rectangles, filled rectangles, circles, filled circles and text. A pattern in which drawing and filling takes place can be selected from a palette.

Editing structure

Matsuhashi (1987) has found that the continual presence of text (as is the case with conventional word processors or text editors can encourage low level revision (such as correction of spelling errors) of the text. However, it also reduces the likelihood of more global changes, such as

amendments to the structure. The reduced awareness of context and structure in computer based text is highlighted by Haas and Hayes (1986). MILO therefore encourages the author to consider both structure and content by providing representations of structure, through which the content is accessed. This structural representation can then be used for addition, deletion and relocation of document elements, without the need to manipulate large portions of text.

An author can alter the structure of a MILO document by one of three actions:

1. Adding notes. Notes can be added at any point in time to any position in the note hierarchy.

2. Deleting notes. Individual notes or groups of notes within the structure can be deleted. MILO provides an undo facility for deletion, which allows an author to backtrack through all delete operations which have been carried out during the current session.

3. Altering the position of notes. A user wishing to move a note to another position in the hierarchical structure can do so by indicating the new position that the note is to take in the structure. Subelements of the relocated note remain so and hence the effect of this action is to alter the position of a subtree of the structure. Relocation is achieved by directly manipulating elements within the hierarchical overview using a pointing device.

Although it is straightforward for an author to amend the document structure at any point in the document creation process, it is likely that she will not always know where a document element should be initially situated.

For this reason MILO allows an author to create document elements which do not immediately become part of a structure on creation. There is no limit on the number of unattached notes which can be created, and the provision of this utility does not mean that an author can not introduce new elements directly into an existing structure. It is possible to integrate the independent notes into a structure at any point. It is possible to integrate notes from a structure with independent notes. The appearance and behaviour of the independent notes will be exactly the same as any notes within a structure.

This will allow an author to build a document in an unconstrained way delaying committment to document structuring actions for as long as is required.

4.7.7 Collaboration

Cockburn and Thimbleby have highlighted the importance of a reflexive perspective of Computer Supported Cooperative Work (CSCW), 'an emphasis on the importance of catering for the individual's requirements and preferences in cooperative environments' (Cockburn & Thimbleby, 1991). By blurring the distinction between individual and collaborative tools there is less overhead in transferring from one work mode to another (Cockburn & Jones, 1991) and the group work environment is more predictable. This can encourage collaboration.

MILO is designed from this perspective, ensuring useful and usable facilities for non-collaborative authors so that collaborative use of the system can be eased into.

The facilities described so far indicate that MILO may well be a useful tool for the creation and manipulation of structured documents containing text and graphics that is as independent of hardware as possible and can easily be used in conjunction with other tools for text manipulation. This is a firm base on which to build collaborative aspects.

4.7.8 Communicating via MILO

In order to achieve its aim of being a useful co-authoring tool, MILO supports the author in communicating with colleagues. Communicating authors using MILO transfer information via purely textual electronic mail. Pliskin (1989) describes problems posed by the use of electronic mail, and drawbacks of e-mail systems in general: users experience addressing difficulties, it is unreliable, the medium has limitations and interface problems exist. E-mail is used in MILO, however, for two reasons. Firstly it is an easily accessible, widely used and cheap method of information exchange. Secondly, a user need not directly interact with an electronic mail system as MILO indicates when a MILO document has arrived in the user's mail box, and such documents can be read directly into MILO from the mail box.

The author is supported in selecting a mail target by the system. A reasonable assumption is that authors will have a set of people with whom they regularly communicate, and when writing a document collaboratively, will in the main be sending the document to their coauthors.

As a result when the author has indicated that she wishes to send a document to a colleague, the names of people who appear in her mail alias file appear as buttons. Clicking on a button

produces the electronic mail address for the person whose name appears in the button. Additionally the names of people who have contributed as authors to the current MILO document appear. This goes some way towards alleviating the problems of remembering complex electronic mail address, and keeping track of who has contributed to the document. This is useful when a document has been distributed to numerous people for review, say, and consequently may have many contributors at different parts of the document structure. Alternatively the user will enter any e-mail address via the keyboard, as she will do with the name of the file to be sent.

The author is free to select any file rather than just the document currently being worked on to allow greater flexibility. The current document can then be sent (if it has been saved to a file), or any other document the user desires. It may not necessarily have been created by MILO. MILO will structure the mailing command for the user, alleviating the need to remember its format.

4.7.9 Supporting collaborative writing strategies

Collaborating groups of writers can adopt varying approaches to creation of documents. Sharples *et al.* (1993) outline three approaches that may be used in the kind of distributed, asynchronous task which MILO is intended to support.

Sequential partitioning of the task entails dividing it into a sequence of stages, each of which is completed by a different individual or group. Each author or group in the sequence may contribute a new section of text, or amend the existing document to produce a new draft.

Parallel partitioning of the task entails different individuals or groups producing different sections of the document at the same time, and then undertaking a merging process to combine their contributions.

Reciprocal partitioning entails different individuals or groups working together to create a common product, and amending their activities to take into account the contributions of other participants. The definition implies that a single version of the document is updated by all participants.

The strategies described by Sharples *et al.* are closely related to the document control methods that Posner and Baecker observed in their study of 22 joint writing projects (Posner & Baecker, 1993) (see Section 3.5). *Centralised* control corresponds most closely to reciprocal partitioning but additionally requires that a single person controls the document throughout the process. *Relay* corresponds to sequential partioning, *independent* to parallel and *shared* to reciprocal.

MILO supports sequential and parallel partitioning through its encouraged division of the document into logical elements, the information it provides about individual notes' author/amender history and the filtering of notes of interest (see Section 4.7.12). The context of each groups' work can be made clearer through these facilities. Documents are distributed by electronic mail.

MILO less evidently supports reciprocal partitioning. MILO does not support shared access to a single instance of the document. However, Sharples *et al.* do not clearly state whether contributions are made synchronously, asynchronously or both. MILO would support asychronous contributions but not synchronous. Consequently it could be said to support the centralised strategy described by Posner and Baecker.

When parallel partitioning is adopted merging document sections to form a single, cohesive text may be a slow and difficult process. If the co-authors were communicating via paper, not only could this be a very slow process if they were a great distance apart, but generating a new document from the multiple versions would be laborious.

Similarly, communication via purely textual e-mail will have its disadvantages although it will be somewhat faster than written communication. A co-author will still be faced with multiple versions of a document which need to be amalgamated into the single entity. This is even more difficult using text held on a computer system because, unlike paper, it will not generally have comments, scribbles and directions that are usually present on paper, in the form of a standard markup language or informal notes.

A useful facility would be semi-automated merging of versions of a document created with MILO to produce a single, up-to-date version. This process is described as semi-automated as the receiving author will have to, in some cases, intervene to make decisions about which changes will prevail. This facility will be described in the following section.

Posner and Baecker found that centralised and independent control were the most frequently adopted methods, and MILO is most suited to centralised, independent or relay working. MILO also supports easy transfer between methods as happened in 77% of group in Posner and Baeckers' study.

MILO is also suited to three of the approaches to text generation which Posner and Baecker term writing strategies (see Section 3.5). It supports single, separate and scribe strategies but not joint.

4.7.10 MILO and Liveware

Liveware (Witten *et al.*, 1991) is a novel approach to sharing data in social networks. It is designed to take maximum advantage of irregular communication for information exchange. Liveware is a concept rather than a system, and its principles can be applied effectively to a coauthoring environment. Witten *et al.* describe a Liveware database system where automatic updates occur in order to maintain consistency and promote information sharing.

There are similarities between a structured document and a database. New elements are introduced into a database, and existing elements are amended or deleted or not acted upon. This same principle can be applied to MILO, with the notes that make up a document corresponding to the data elements of a database. There is therefore a potential advantage in introducing Liveware concepts into the functionality of MILO.

Liveware is used in MILO to automate the process of merging two versions of a single document into a new updated version. This is a useful facility for authors. First, it benefits coauthors of a single document as the work of each individual can be easily drawn together into a single cohesive text. There is no limit to the number of coauthors whose work can be amalgamated in such a way. Second, the facility can be used by the single author to combine work from multiple incarnations of the same document. Many drafts of a document are usually produced during the authoring process and automation can allow the author to easily retrieve elements from previous versions.

The implementation of Liveware in MILO goes beyond a one-to-one correspondence with the application of Liveware concepts to database manipulation. In addition to handling additions

and deletions of notes, and amendments to note contents, the merging process embedded in MILO also takes into account restructuring of the hierarchical document.

A central principle of Liveware is that information exchange is symmetric; two Liveware versions of a database are identical after an update. MILO relies on authors sending information to each other via electronic mail, and so a single author is concerned with merging different versions of the same document. There is no need to update both versions as long as the author is clear as to which is the new version to be sent to co-authors.

Witten *et al.* advise access rights to units of information with alterations only possible by the owner. This might be suitable for a database implementation, but is not desirable for an unconstrained co-authoring system where free access to all elements of the document is vital.

4.7.11 Issues in document merging

It is perhaps useful to contextualize the approach to merging multiple versions of a single document that is taken in MILO.

Factors such as available technology, geographical location, work patterns and so on result in a variety of collaborative working styles. One way to view these styles is to consider them in terms of the times and the places that the work is carried out. Such a view provides four models of how a collaborative task may be carried out: co-located- synchronous, co-located-asynchronous, distributed-synchronous and distributed- asynchronous. Of course, such a rigid view is not always appropriate and there may be an intersection between styles as, for example, group members travel to meet each other or working hours change for group members.

Collaborative writing activities fit into this view. It may be said that MILO supports all four of these styles as each writer using MILO has their own individual copy of a document prepared with MILO. However, it is mainly intended to support the distributed-asynchronous model of collaboration.

Regardless of the style adopted in a co-authoring task, a necessity is to make group members aware of amendments that are taking or have taken place, and to make those amendments accessible. The manner in which a co-authoring system supports this is, to some extent, dictated by the strategy adopted for storage and access to the document or elements of the document.

One strategy is to allow group members shared access to a single copy of the document. In this case the document may be stored as a whole or as several files that correspond to document elements. These files may be stored in a single location or be stored on the machines of their authors.

This strategy is most suitable when collaborators are co-located, with access to the same hardware, and may be updating the document at the same time. It is adopted by Greif and Sarin (1987) in their collaborative document editing system CES.

A second strategy may be also be used in such a scenario. This entails user access to a replicated version of the shared document and is the approach adopted by Ellis et al. (1991b) in their group outline viewing editor GROVE. In this case a centralised controller manages the updates that occur on the replicated documents.

A third strategy is more suitable when collaborators are geographically distributed, may not have access to the same hardware, and will be working on the document at different times. This entails each participant updating their own copy of the document or document element.

In each of these cases it is necessary to merge the contributions of the multiple authors.

When a single, shared document is updated (the first strategy), a merge operation should take place automatically - the amendment is also the merge operation. This can, however, present problems in the management of concurrency of document elements and simultaneous access by multiple participants to the same document element. This means that steps must be taken to ensure that consistency is maintained after each amendment to the document, that multiple attempted simultaneous updates have the same effect as carrying them out one at a time, and that participants are not in conflict over access to document elements. Such steps must also be taken in the second strategy outlined above.

The third strategy is that which is adopted in MILO. This results in the necessity to solve problems similar to those faced when the first two strategies are implemented, although not in real time as is required in CES and GROVE.

CES uses technological protocols to control data sharing and concurrency. Co- authors have specified access rights to document elements that enable them to carry out read only or editing operations. In the case of conflict over access to elements CES allocates a 'lock' to a single author

that may be explicitly relinquished or negated by the system after an idle period. Changes are propagated to other users a little after they have been made. The coordination of such activities is directed by roles of the individuals involved, the activities that are appropriate to the roles involved, and the ordering of the relationships among the operations carried out. However, the coordination is dictated by the system itself.

Ellis *et al.* (1991b) take a different approach to this, stating that "Technological protocols can be overly restrictive: a group's idiosyncratic working style may not be supported, and the system can constrain a group that needs to use different processes for different activities".

More specifically they consider that "many of the approaches to handling concurrency in database applications—such as explicit locking or transaction processing—are not only inappropriate for groupware but can actually hinder tightly coupled teamwork". They found that social protocols took only a short time to develop during collaborative authoring tasks, and the ability for everyone to see and edit shared information without locking did not result in chaotic attempts to complete the task.

Although MILO is different to both CES and GROVE in that they are primarily concerned with real time co-authoring , it addresses a similar issue; the contributions of multiple authors must be integrated in a sensible manner to result in a current version of the document. CES and GROVE manage this at the time that updates take place, MILO manages this when an amended document is communicated from one writer to another, using Liveware.

Like GROVE, MILO allows authors to freely create, amend and delete any part of a document, even if they are not the author of that element. This requires consideration of several issues when the results of such actions must be merged. How might conflicting operations on the same document element be resolved? Are timestamps alone a suitable indicator of which contributions prevail in the final version? Should authorship of a section be considered more important than contributions of an amender? Do deletions prevail regardless of who carried them out?

These issues are complex and dynamic. Solutions which might be appropriate in one group scenario might not be so in another, as Ellis *et al.* have pointed out.

The liveware merge as currently implemented in MILO has limited functionality. It serves

as an indicator that a new approach to merging of asynchronous, distributed contributions to a co-authored document can be useful. It certainly requires development and extension, but such issues are not the central thrust of this thesis. The following paragraphs describe a study carried out on the merge operation of MILO (Chen, 1993).

The merge function as currently implemented adheres to the following rules (document A2, which is the product of a non-co-located coleague, is to be merged with document A1 to produce document B):

1. notes from A2 are added into the updated document B if they are not present in A1
2. if a note appears in both A1 and A2, the most recently amended prevails in B
3. the position of a note in B is dictated by its most recent position in either A1 or A2
4. notes deleted from A2 will prevail in B if they are present in A1

This results in several problems.:

1. notes deleted from A1 but not from A2 will be present in B and vice versa
2. the rule that most recently updated versions of notes prevail does not account for the status or size of changes that occur within notes. A minor change to punctuation, for example, may prevail over a major revision
3. structural changes have equality with changes to content and so a more recent change to a note's postition only will prevail over any less recent revisions of its content

Solutions to these problems would be more obvious if MILO used roles to guide the merge operation, or imposed access control when amendments were made. However, unlike systems such as Quilt and CES, MILO does not direct users into protocols embedded in the technology. If it did, note author contributions may prevail, amendments may not be permitted by other than those on access lists, and so on. In essence, the flexibility of MILO renders the merge more complex. However, as Jones and Cockburn have highlighted (1993) it is important to avoid easing the design of the merge function by imposing inappropriate effort or protocols on the user. Hence, in development of the merge the flexibility of MILO should not be compromised.

Chen (1993)outlines alternative designs for MILO's Liveware merge. The most appropriate strategy that she suggests for coping with deletion problems is as follows: all users can delete notes (the current situation) but only deletions by the authors of notes prevail, other deleted notes appear in the merged version as meta-notes. This indicates that amenders of notes have wished to delete some notes, removing them from the document per se but allowing them to be easily integrated at a later data. However, it does not offer up for discussion deletions carried out by note authors themselves. An improvement may be to turn all deleted notes into meta-notes.

Chen's alternative strategy for merging amended notes does not always result in the most recently amended version prevailing: an author's amendments to his/her own note prevail over other amendments, and notes amended by co-authors appear alongside the original version in the resulting document. Suggested changes to a note can all therefore be seen in conjunction as the most appropriate one or an amalgalm of one or more can be selected.

Users may also be aided by appropriate presentational developments such as colouring notes differently for different authors and amenders, shading notes as the period since they were last amended increases and so on.

4.7.12 Viewing MILO documents

If an author is to successfully create and manipulate structured documents with MILO then that structure and its content must be accessible and easily understood. When creating a document of structural complexity it is important for the author to be supported by the system in maintaining a correct model of the document at all times. Developing such a document on paper or with an ordinary text editor can be difficult as no matter the model the author has of the structure, the document has no other appearance than linear. Additionally, text editors and word processors generally only show the author a small portion of the whole text, making contextualisation of current work difficult. This may well degrade the quality of the final product.

To maximise an author's ability to manipulate document structure and content to best effect MILO provides different kinds of structure overview, allows an author to search out sections fulfilling certain criteria, and search for instances of specific text patterns.

Different views of document structure

MILO provides two distinct types of graphical structure overview for documents.

Firstly a tree-like graphic represents the hierarchical relationships between all notes contained within the document. Notes are represented using the text which has been entered in the title bar for each one. The graphical presentation reinforces the hierarchical model of a MILO document showing elements and their subelements.

Secondly, a graphic resembling an indented table of contents presents a linear representation of the structure. This corresponds to the mapping from hierarchical structure to linear document which is implemented if the author stores the structure in a LaTeX format file.

The first overview aids the author in manipulating the structure for optimal effect, arranging and rearranging ideas and sections of text. The second shows the author a potential linear form for the document. The two overviews should complement each other in helping the author refine the document.

Figure 4.5 shows the two overviews.

Applying filters to documents

In addition to wishing to take different views of the structure of a MILO document, an author may also wish to take different views of the content of the document. MILO allows a user to view portions of the document selectively by filtering notes within the document based on rules provided by the user.

Consider an example. A tutor has produced a document within MILO and as part of their assessment her students must comment on the document. She sends each student a version of the original via electronic mail, they complete their assignment by adding, removing and amending notes and return the MILO document, again via e-mail. The tutor then merges the work of each student into a single document using MILO's Liveware facility. Now she is faced with a very large document, containing many notes and has the task of assessing the work of each student.

The tutor might well then wish to view the work of one individual at a time, or to see the work of more than one student at a time if she suspects collusion. She might wish to see when

certain students began their project, or when they completed work on it. Using the filtering functions of MILO it is possible to do each of these things. The user can specify individual keys such as note creation/amendment time, or note author/amender to be used for searching for and highlighting notes. Combinations of keys can also be used to find sections of the document satisfying more specific criteria.

The filters can also be used by a single author in order to keep a track of her own work, providing the ability to access parts of a document based on when work was last done on them.

Any notes satisfying criteria specified for the filtering process are brought visually to the users attention.

Time ordered view

It is sometimes difficult for an author to keep track of work on a document, especially if there are gaps between the work phases. Remembering which elements were being worked on, or the order in which tasks were being dealt with may pose problems which MILO helps to alleviate. It provides a view of the document elements labelled with the time that they were last amended, and ordered from most to least recent amendment. The elements are represented by their title. An author can therefore very quickly see a history of work on the document and via this view access elements of interest.

The provision of this view, the ability to filter elements of interest from the document and the information available about each note goes some way towards providing the context history of interaction which Miles *et al.* (1993) indicate is important in asynchronous collaboration.

Searching documents

There are two reasons why an author using MILO will require a facility to search the contents of a document. She may have forgotten in which note certain subjects were being discussed and need to find it or them to re-read or amend what has been previously written. It may also be desirable to consider elements of the document which are related in subject matter.

The facility for finding text within a MILO document provides for both reasons. The user can specify not just individual words but a text string which is to be searched for. Any notes

which contain the text are brought visually to the user attention as with the filtering process. The text being searched for is highlighted and further searches for the same text within a note result in the next instance being highlighted.

Automating the finding of document elements provides accuracy and efficiency benefits for the user.

Spell checking

MILO provides a spell checking facility that can be used within MILO, without the need to save the current document and access such a tool from outside. The spell checker conforms to British spellings, and errors are reported to the user within MILO, indicating in which note they have occurred.

4.7.13 Implementation of MILO

The interface to MILO is implemented in the X Window System (Scheifler & Gettys, 1986) using the Xt Intrinsics and Hewlett Packard X Widgets (Hewlett-Packard Company, 1988). It incorporates many standard graphical user interface objects such as multiple windows, menus, buttons, dialogues and scrollbars to provide a powerful direct manipulation graphical user interface.

X is suitable for the task of developing the complex user interface to MILO for three reasons. Firstly extensive libraries of high-level user interface objects which can be combined into a cohesive interface are provided, using libraries of routines to manipulate those objects. These libraries are abstracted from the most basic programming level in X, simplifying the development task, but still allowing the programmer access to lower level but useful routines. Secondly, the use of X is becoming more widespread and with the support of many major hardware manufacturers it is almost provided by default with powerful graphical workstations. Its use in MILO will increase the potential user base for the system. Thirdly, because of its client-server architecture, any applications developed under X will be device independent. This again increases the potential user base by allowing users (even collaborating ones) to utilise it on many different hardware platforms.

4.8 Summary

Two novel systems, *Sticky-Notes* and MILO, have been developed as media for investigation of computer based support for both the personal and the collaborative requirements of writers. Their design has been driven by:

- adapting desirable attributes of other cooperative work support systems to the support of writing;
- the lack of suitability of current models of writers and the writing process (both collaborative and personal);
- consideration of strategies and approaches adopted by writers;
- design principles for CSCW systems (Cockburn & Jones, 1991).

The implementation of *Sticky-Notes* was carried out in HyperCard in order to enable rapid prototyping of a system with a graphical user interface. This process provided useful indicators as to:

- the suitability of functionality provided by *Sticky-Notes*;
- shortcomings in the functionality of *Sticky-Notes*;
- the suitability of HyperCard for development of interactive systems.

MILO has been developed in order to further investigate the issues considered via *Sticky-Notes*, to investigate the new avenues of potential interest highlighted through the development of *Sticky-Notes*, and to provide a fully working widely usable hardware-independent tool. A novel design approach has been applied to MILO and a novel range of functionality has been embedded within it.

MILO is in use and initial, informal, observations indicate that it is a useful tool for writers in both a personal and collaborative context. It has been successfully used to write and amend much of this thesis (a single author task!)[2] and also to write and amend a co-authored paper (Cockburn & Jones, 1991).

[2]MILO was used to plan, structure, write, review and amend chapters 1, 4, 5, 6 and 7 of this thesis.

General Requirements		
Requirement	**Adopted**	**Comment**
Enhance existing relationships, encourage new ones	yes	General requirement achieved by meeting of more specific requirements described below.
Improve quality, efficacy and product of interactions	yes	General requirement achieved by meeting of more specific requirements described below.
Provide benefits of physical proximity	?	Difficult to achieve for an asynchronous distributed authoring system. Could be facilitated by use of high bandwith media for communications, such as synchronous video links (requiring same-time participation) or video and/or audio annotations to the document (for a richer asynchronous communication).
Accommodate changes in roles and commitments	yes	Achieved by avoiding imposition of roles or adherence to commitments. System is minimally constrained to allow development of social protocols.
Support formal interactions such as meetings	no	Beyond the scope of the discussion.
Maximise likelihood of acceptance at a personal level	yes	Achieved by providing powerful support for single authoring tasks.
Integrate existing social protocols and relationships Do not enforce protocols or restrict use of existing ones Minimise system imposed constraints on users	yes	Allow social protocols to prevail. Avoid imposition of roles, communication styles, access rights, document formats etc.
Minimise system imposed requirements on users	yes	Avoid requests for user information which aid the system. Gather information from other sources such as the operating system environment, previous user interactions etc.
Exploit the most suitable hardware	?	The available hardware and software tools for implementation of the system will affect the way in which the system meets the requirements. Existing options rule out investigation of the use of video/audio facilities, for example.
Use electronic mail as a communication medium	yes	Easily and cheaply accessible.
Output should be useful and manipulable	yes	The output from the system should be useful for the individual (represent the data accurately and appropriately, be manipulable by other systems), and for collaborators (provide details of co-authors' work etc)
Provide hypertextual facilities	no	The system to be developed will support authoring of hierarchically structured documents. Elements will be linked in the document structure but it will not be a network.
Allow for integrated use with other systems	yes	The system will not provide all the facilities a writer or group of writers could want during the authoring process. Consequently it must be possible to integrate its use with other systems such as document formatters, email systems, spell checkers and so. This can be achieved through implementation on an appropriate platform, and representation of the data.
Information exchange should not be dependent on the system	yes	Documents created with the system will be amendable and exchangable without the system, as co-authors may not have access to it.
The system should be hardware independent	yes	Distributed collaborating authors are not likely to be using the same type of hardware. The system must be developed to be easily portable to other platforms.

Figure 4.1: General requirements for a collaboration support system.

Writing support tool requirements		
Requirement	**Adopted**	**Comment**
Hierarchy of structures and tasks	yes	Object to manipulate is a hierarchical document. Presentation and manipulation of documents should support document development as authors progress through tasks and subtasks (eg chapter-section-subsection development).
Interleaving of tasks	yes	Allow user to easily move between tasks in the system (planning/structure development/text generation/communication etc). For example structure development and text generation should not have fixed order of completion. The user should be able to move freely between the tasks. Additionally the user should be able to move freely between use of the system and other systems in their environment.
Constraint management	no	Allow social protocols to prevail, providing a more flexible system. Imposition of set constraints will not offer a flexible or always appropriate system.
Basic text manipulation operations	yes	Standard text editor to be provided. Additionally it should be possible to manipulate text between this system and others that are in the user's environment.
Reuse of previously generated text	yes	Documents created by the system will act as input to the system. It should be possible to read other documents into the system.
Predefined document styles	no	Unlikely to be appropriate in all cases. User empowered to build up library of document structures and styles.
Multiple views of the same information	yes	Views should be structure (hierarchy) based, and provide linear views of the document, its contents, and a formatted version of the document. Additonally provide views based on the history of contributions to the document.
Creation of unordered items	yes	Allow any number of single document elements to be created outside of the main document structure. Allow any number of hierarchical document substructures to be created outside of the main document structure.
Creation of notes and networks of notes	yes	Documents will be built from a structure of notes (document elements). Notes will be hierarchically related, but will not form a network.
Provision of linear drafts of the text	yes	Provide both editable and formatted linear versions of the hierarchical structure.
Manipulation of a simulation of the final output	no	Emphasis will be placed on document structure and content rather than formatting.
Easy movement between views of the information	yes	Movement should be swift and obvious, and it should be possible to see the multiple views simultaneously.
Merging document text and meta text	yes	It should be possible to integrate unordered text into the document at any point during the interaction, and at any position in the document.
Preservation of collaborator identities	yes	Authors should be able to easily access information about co-authors and their contribution to the document.
Support for communication among collaborators	yes	Provide an interface to electronic mail within the system. Automate dissemination of documents between authors. Integrate with email conversation support tools.
Make collaborator roles explicit	no	Imposition of roles implies imposition of constraints on activities. This does not allow for an open and flexible collaboration. Allow social protocols to develop and change to meet role requirements as appropriate.
Support brainstorming, researching, planning, writing, editing, reviewing	yes	Activities will be possible in the system but not explicitly supported.
Support transitions between activities	yes	Similar to interleaving of tasks.

Figure 4.2: Requirements for an asynchronous distributed writing support system.

Writing support tool requirements		
Requirement	**Adopted**	**Comment**
Provide access to relevant information	?	No explicit support for access to other information sources, but use of the system will integrate with use of other tools in the user's environment.
Make plans explicit	?	User can record plans and view them, but no explicit support.
Provide version control mechanisms	no	No explicit support will be provided but use could be integrated with use of a version control application. However integrated use with an email conversation web support tool may help.
Support concurrent and sequential access	?	Access will be concurrent to replicated versions of the same document in either a co-located or a distributed scenario. Access could also be sequential in both scenarios.
Support write, comment, read access methods	no	Authors will not be constrained to a single access method, and document elements will not impose single access methods. Again social protocols will prevail.
Support separate document segments	yes	Document segments will be represented by notes. These may be chapters, sections, subsections, paragraphs, diagrams and so on.
Support one and several writers	yes	Facilities will be provided for a single author, with easy progression to use of facilities for multiple authors.
Provide easy, flexible information entry	yes	There will be no imposed ordering on document creation (either structure development or text generation).
Support synchronous and asynchronous writing	?	The process will not be synchronous in terms of multiple authors editing a document at the same time and immediately seeing the results of the work of their colleagues (eg shared screen editing). Authors will be able to work at the same time or at different times on replicated versions of the document.
Provide spatial representations of the text	yes	Provide graphical representations of the document structure. Elements in the representation will be directly manipulable.
Provide automated merging of contributions to the document	yes	Provide basic merging algorithm for combining multiple instances of a document into a single coherent structure.
Maintain an easily accessible history of the interaction	yes	Record contributions with timestamps and author identifiers. Integrate use with a converstation web visualisation system such as Mona (Cockburn, 1993).
Provide structure manipulation tools	yes	Support additions, deletions and other manipulations at a structure level in addition to a text level.
Provide multiple types for document components	yes	Support both textual and graphical document elements. Video and audio components are not practically possible.
Provide multiple types for annotations	yes	Support annotations in a way consistent with document elements: textual and graphical annotations. Annotations will not be represented in a different manner to document components: authors deal with a single element type.
Provide utilities eg spelling checker	yes	Integrate with exisiting system utilities.

Figure 4.3: Further requirements for an asynchronous distributed writing support system.

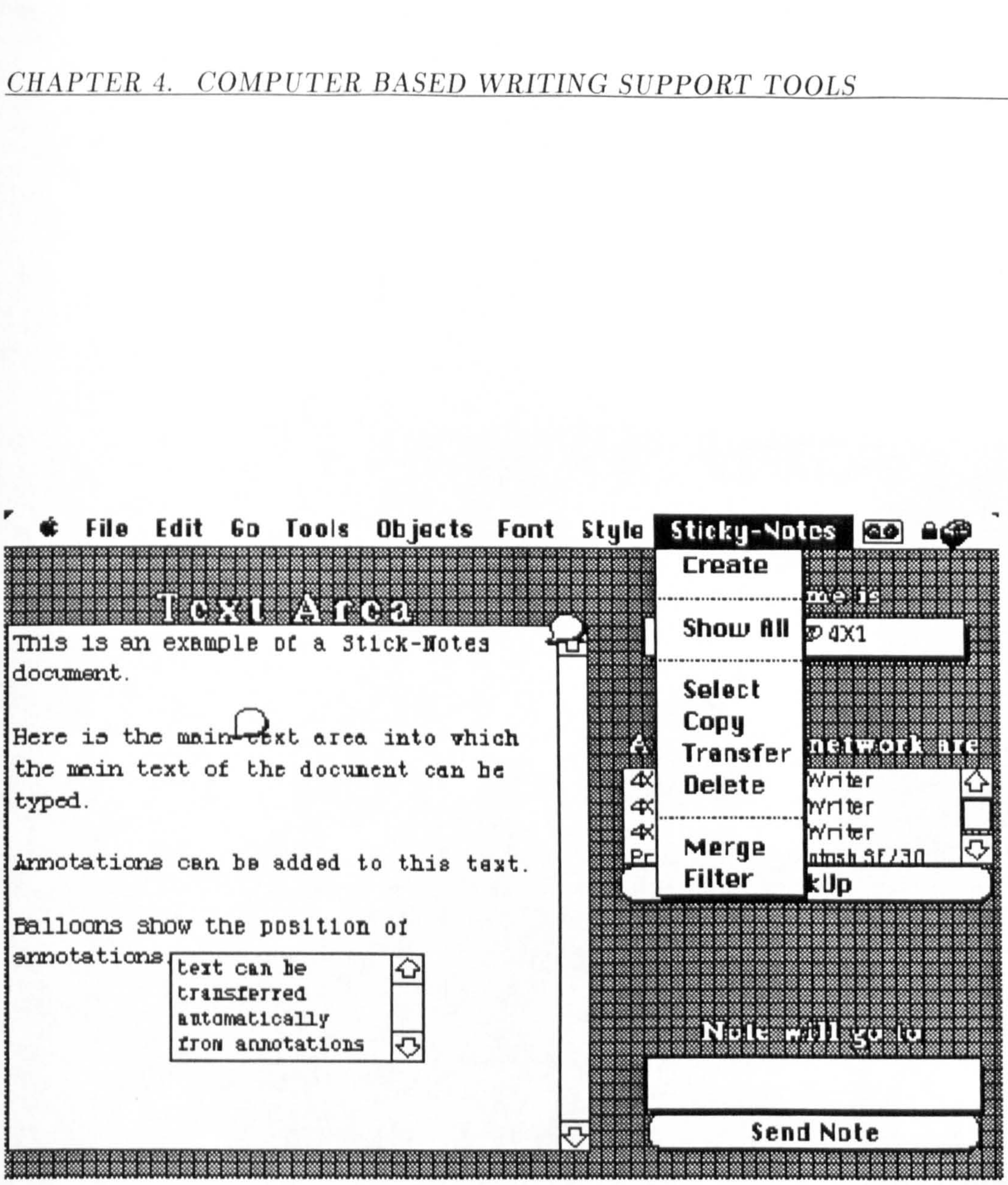


Figure 4.4: A typical *Sticky-Notes* user's screen. On the left of the screen is the main text area into which the document is entered. This is a scrollable window. Balloons represent the position of expandable notes. The smaller window is a note, again in a scrollable window. A pulldown menu provides access to *Sticky-Notes* functions. A list of other nodes on the local network appears on the right of the screen. Entries can be selected as the target for a document transfer.

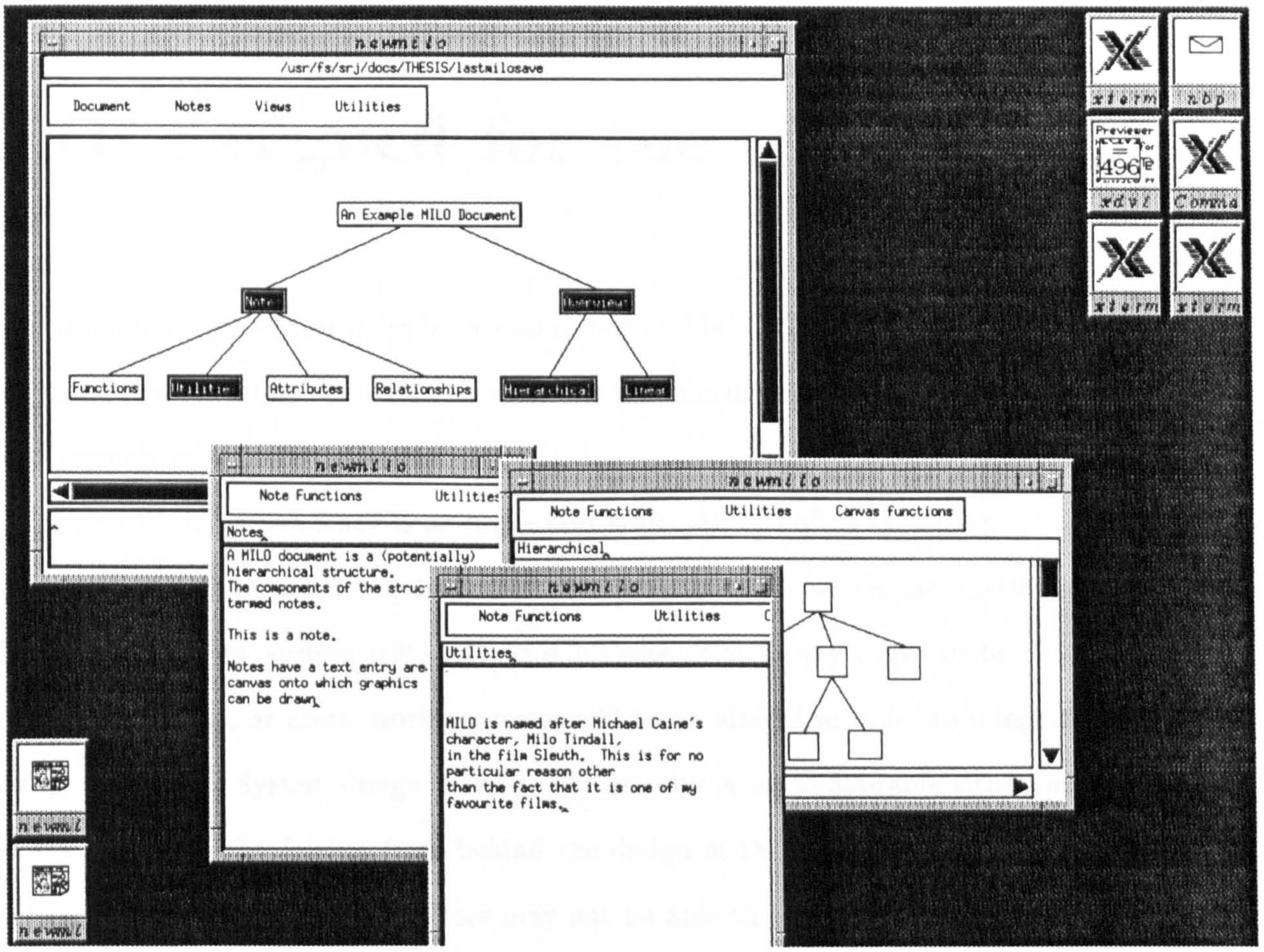


Figure 4.5: A typical MILO users screen, showing the main MILO window, several note windows containing either text or graphics, and other iconified tools.

Chapter 5

MILO: designed for the user

Interactive computer systems must be both useful and usable. The usefulness of both *Sticky-Notes* and MILO is dictated by the extent to which the functionality which they offer corresponds to the requirements of potential users.

The direction of this dependency is an important issue. All too often systems provide functions which users rarely or never require and fail to support tasks which users wish to carry out. In this situation the system will either not be used, which may prove to be disastrous for commercial products, or users' work processes will to be altered in order to integrate with what the system offers. System design driving user activity is an undesirable situation. User requirements should be the driving force behind the design of the system. This, however, can prove to be a difficult goal to satisfy. Users may not be able to adequately specify what they will use the system for, or foresee how their needs may change over time. A system designer has the unenviable task of satisfying immediate user requirements, preempting the evolution of those requirements and designing a system which can accommodate such changes.

The usability of a system concerns how users access the functionality which is provided, and is discussed in this chapter. A system designer must consider which interaction techniques are most appropriate for the application which is being developed. These may include graphical user interfaces, command line interfaces, speech driven interfaces, gesture driven interfaces, menu driven interfaces, or a hybrid of any of these and other approaches. Such a high level decision on appropriate interaction techniques leads to a multitude of lower level decisions about how exactly to implement the user interface to the system. These issues may concern placement of

objects on the screen, how many elements a menu should contain, what form commands should take, what the system should tell the user about its current state and a host more. During this process the goal should always be to make the interface to the functionality transparent so that the user effort is concentrated on completing their tasks and not on the actions required to do so.

The relationship between usefulness and usability is a complex one, which affects the potential success and acceptance of an interactive system. A system with a high level of functionality which corresponds closely with user requirements is not going to be used if user input and system output take unacceptable forms, and the effort on the part of the user to access the functionality is so great that it will negate the potential benefits that it might bring. Conversely, a system with a transparent interface where the user invests minimal effort in accessing the functionality will not be successful if that functionality does not meet user requirements, or gives erroneous results.

Of course the majority of systems fall between these two extremes where a compromise takes place. The compromise, however, is generally on the part of the user, who is required to accept systems which meet neither functionality nor usability requirements fully. The onus, then, is on designers to avoid the need for such a tradeoff and design for the user.

5.1 The usability of collaborative systems

Designing standalone systems well is a difficult task. Designing interactive systems which support a user's work processes in a certain domain *and* support collaboration with colleagues is even more difficult. The two dimensions of usability and usefulness are complemented with the need to consider how to support the interpersonal communication and collaboration phases of a user's work.

Systems explicitly designed to support communication/collaboration alone (such as electronic mail systems or computer based conferencing systems) must have the social and psychological issues involved as a central concern. They may be easy to use and provide relevant functionality, but if they do not support socially imposed protocols sufficiently or require unnatural behaviour to be adopted they will have failed in their goal. The designer's emphasis is shifted from catering for interaction with the computer to catering for interaction *through* the computer.

Other systems, of which MILO is an example, are designed to support collaboration as integral to other aims. A system to support collaborative writing, for example, must primarily address issues related to support of a single writer to be useful. It will not succeed if it provides excellent support for collaboration, but inhibits a user's ability to write successfully.

MILO is designed from the perspective of reflexive computer supported cooperative work (CSCW) (Cockburn & Thimbleby, 1991). This approach holds that CSCW systems must firstly be useful and usable for a single user so that the progression to collaboration requires minimal additional effort and learning. During the design of MILO, emphasis has been placed on producing a system which can be used easily and effectively by a single user to create and amend structured documents. Functionality and usability have been considered, so that a practical and usable system was created into which collaborative aspects could be integrated. This is not to say that an holistic view was not taken during the design process. To simply attach collaborative facilities to an existing system can prove ill advised, as they are likely to be constrained by the design of the original system. The collaborative aspects of MILO were considered from the outset to avoid conflict with the single user facilities developed.

5.2 Guidelines for interface design

If the design of interactive systems is such a difficult task, how might a designer increase the chances of producing a 'usable' system? To a certain extent the design of the human-computer interface will be driven by the knowledge and experience of the designer, which makes usability of the final product highly dependent on the skills of the designer. The results of this fact are evident in the highly variant quality of interfaces to todays interactive systems.

A way of increasing the likelihood of designers achieving a higher level of system usability is for designers to adhere to certain rules or guidelines about the design of interfaces. As long as the guidelines used have a sound basis the quality of the interface can be improved regardless of the quality of the designer. They still provide scope, however, for designers to apply their own knowledge and to create an interface suitably tailored to the application domain under consideration.

Apple Macintosh computers are an example of the effectiveness of guidelines (Apple Com-

puter, Inc., 1987). Although many hundreds of Macintosh applications have been developed, their appearance and behaviour is basically very similar. Although variations occur and the guidelines are sometimes broken, experience of Macintosh applications can allow new ones to be learned and used in a short space of time. This is because the appearance and behaviour of Macintosh interface objects is constrained, and guidelines suggest less tangible but more desirable attributes of application interfaces.

5.2.1 What are the guidelines?

Several widely accepted and used user interface design guidelines exist. The adopted guidelines were introduced in Section 4.3. The remainder of this section describes how they can positively influence the design of interactive systems, and the following sections describes how they have influenced the design of MILO. The following list is drawn from several sources (Apple Computer, Inc., 1987; Thimbleby, 1990; Monk, 1988; Shneiderman, 1983) but is not comprehensive. Some guidelines can conflict with others, but their use *can* influence design for the better.

Allow the user to see and point rather than remember and type. This ensures that objects and functions within the system are readily visible and accessible. It is applicable to systems such a MILO, where interaction is through a graphical user interface where objects are manipulated using a pointing device. It means that commands are continually present, most likely in the form of menu entries or buttons, so that the whole range of functionality is at hand. The user's memory load is reduced in comparison to command line interfaces for example, where commands must be remembered and entered via a keyboard. The potential for forgetting command names, or even that certain functionality exists is eradicated.

User actions should be reversible. This has two benefits. First, if the user is aware that actions are reversible she is more likely to explore the functionality of the system more fully with less anxiety. Second, users make mistakes and need to take remedial action in order to correct them. The extent to which actions can be reversed may vary. Many systems allow only the previous action to be 'undone'; unless the user notices a necessary correction immediately it may become far more difficult to carry it out. It is more desirable

to allow the user to backtrack through previous actions to a state where recovery is easier or even possible. It is most important to allow destructive actions such as data deletion to be reversed.

Provide consistency within the application. This also can reduce user anxiety and helps the user to become familiar with systems both more quickly and more easily. Identical behaviour of objects throughout the system, and actions having the same effect regardless of system state, in addition to attributes such as consistent terminology, labelling and prompts all serve to ease learning and increase usability.

Provide consistency with other applications. Users are likely to have knowledge and experience of other systems, which can be exploited to the user's advantage by designing the interface to conform to interaction techniques which they have already learned. Of course, this should not interfere with the use of the most effective interface for the system, but the utilisation of commonly used interface objects such as buttons, menus and windows can engender a immediate sense of familiarity.

Provide the user with feedback. User actions should result in some indication by the system about the effect and the success of those actions. This can give users confidence that actions are having the appropriate effect and they are progressing towards task completion, or warn of errors at the earliest opportunity. Feedback should concern the state of the system itself. Consider a word processor. The system state whilst carrying out a search and replace, i.e. busy, should be reflected in order to explain possible lack of response or other side effects and inhibit possibly destructive actions to which the system will respond later.

Require user effort commensurate with the potential effects of actions. Simple, low-risk, often needed actions such as inserting or deleting a single character should be easy to access and carry out. Potentially harmful actions such as delete or merge should require more effort to access and execute (Thimbleby, 1990). This can protect the user from errors such as miskeys, or mis-selections and indicates the potentially destructive effect of certain actions.

Do not allow the user's display to become cluttered. A confused display containing many interface objects can confuse the user. The amount of information on show can become overwhelming and irrelevant information can distract from that which is of interest. Information may be hidden and the user may be required to spend a disproportionate amount of time manipulating the screen contents to enable useful work to be done. Presenting the user with less information may well be to their benefit, if what is presented is of use. System dictated methods of reducing screen complexity also has the potential to hinder the user. There may be circumstances when a cluttered screen is necessary or suits a user's work processes. For this reason it is more appropriate to support the user in tidying the screen and managing the complexity when required.

Allow the user to directly manipulate objects on the screen. This is generally achieved using some kind of pointing device such as a mouse or trackball. Objects can be repositioned and operations such as copy, or delete can be carried out by calling on skills from the real world such as pointing and moving. The results of actions are less abstract as objects are visible on the screen, providing the user with a greater sense of satisfaction.

Allow the user to draw on real-world experiences and knowledge. Users come to systems with a wealth of knowledge and experience from other domains, which can be exploited to ease the process of both learning and using a new system. Systems which do this are generally regarded as employing metaphor, analogies or models. A prime example is the Apple Macintosh which utilises a so-called desktop metaphor containing computer representations of real-world objects such as folders, wastebaskets, calculators and calendars. The knowledge a user has of a wastebasket, for example, can enable them to use the computer based version correctly and with confidence. They expect to be able to throw away documents, folders and pages by putting them into the wastebasket, but also to be able to retrieve them if necessary. This is a far better approach than simply providing a delete function, the result of which is unknown and unpredictable. To take the representation of real-world objects to its extreme, however, fails to fully exploit the potential which the computer promises. What is the point of using a computer based desktop which behaves *exactly* like a real one? The metaphor should provide familiarity

for the user, *and* extend the user's abilities.

I would also propose further guidelines:

Support the user in maintaining an accurate and current model of the data being manipulated. In addition to providing the user with feedback about system activity, the system should also provide feedback about the state of the data which they are manipulating. It is important that a user's model of their data should be both accurate and current. Consider the confusion which would be caused by a word processor which presents paragraphs in a random ordering, and only reflects alterations to the text when the file is saved. In general, interactive systems such as text editors and word processors do reflect the current state of data.

Systems like LaTeX, however, can create confusion between the current marked-up source version of a document and the printed version because of the need for intermediate processing of the document by the LaTeX program. Additionally, it is difficult to see if the data as the system requires it corresponds to the correct user-required form.

A more difficult aim is to provide an accurate user model of the data, in MILO's case, document text and structure. Haas and Hayes (1986) have shown that writers find it more difficult to maintain a sense of the whole text when writing on a computer. In this case the accuracy of the user's structural model of the data is lacking.

Presentation techniques such as graphical representations, outline views and fish-eye views (Furnas, 1986) may be appropriate in implementation of this guideline.

Protect users from hardware/software failure. Users are at the mercy of the reliability of both computer hardware and software, which is virtually impossible to expect or guarantee. Unexpected software or hardware crashes can result in the loss of large amounts of data. This guideline is intended to provide systems which allow users to recover from such occurences during their use. Such protection really means protecting what the user is most interested in, their data. This may be a document, picture, program, spreadsheet, game state and so on. Provision of regular backups of user data during sessions can in part do this.

Care must be taken over the design of such a feature. HyperCard, for example, automatically overwrites old data with its new version during each session. This does not account for sessions

the user wishes to abort and requires arbitrary back-up regimes to be developed. What is required is a system which provides regular back-up during a session without inconveniencing the user, and makes a clear distinction between user and system saved versions of the data.

5.3 MILO and the guidelines

This section discusses the extent to which MILO conforms to the user interface design guidelines described in Section 5.2.1. It will be seen that the adherence to the guidelines is quite strict, the aim of which is to positively affect the usability of the system.

5.3.1 See and point

A MILO user is not required to remember any commands at all in order to use the system. Functionality is accessed by using a pointing device to make selections from menus of commands, and so the user is only burdened with the need to remember what actions are required to access a menu. If this aspect of the system is remembered the user can traverse the menus within MILO to find the appropriate function for their requirements (Mayes *et al.*, 1988). Of course, in many cases the user will remember in which menu the desired command is situated and be able to access it directly.

In order to aid this process, functions have been separated into several menus containing logically related entries. One menu, for example contains functions concerned with creating new documents, and writing and reading documents to and from files. Another menu is concerned with functions which allow a user to take various views of the elements within a MILO document. The menus which concern the document as a whole are implemented as pulldown menus where the title of the menu, which indicates its content is always visible.

Other functions within MILO, however, are only appropriate for a single note. These are functions such as removing a note window from the screen, moving a note within the document structure, deleting a note or accessing the graphics editor attached to each note. These functions appear in pulldown menus within each note and within the elements in the overviews which represent notes. The scope of the effect of these functions is only the note from which the menu was popped up. This makes MILO different from many systems which implement a 'see

and point' approach. Such systems actually require two steps to carry out an action on an object: select the object and then select the function from a menu which may be distant from the object on the screen. MILO allows the selection of object and function to be carried out by a single action via its object specific pulldown menus. This is an enhancement to the usual implementation of 'see and point'.

5.3.2 Actions should be reversible

User actions within MILO are easily reversible by, in the majority of cases, carrying out an action which is the converse of the one to be undone. This is applicable to the many levels of action possible within the system. Character entry within a text editor can be undone by character deletion, and vice versa. Deletion of a larger portion of text by selection and cutting can be undone by issuing a paste command, and unwanted pastes can be undone via selection and cut. This is extended to notes. Existing notes which are unwanted can be deleted but deleted notes can be integrated into the document again at a later date. Notes which are repositioned in the structure can be moved back to their original position by carrying out the same repositioning action in reverse.

Emphasis has been placed on providing the user with the ability to reverse potentially destructive actions such as deletion and reordering of data. Like many applications MILO's implementation of undo is mainly concerned with reversing the last action the user carried out. In the majority of cases this is acceptable as the user can recover successfully from cut, paste and rearrangement errors without explicit system support. However the reversal of deletion of notes within MILO does have explicit system support. This is because unlike cut, paste and rearrangement the removed data is no longer accessible for the user to manage reversal. MILO therefore maintains records of deleted notes so that a single user action can restore them to the position which they occupied before deletion.

MILO provides the facility to replace not only the most recently deleted note, but all the notes which were deleted during the current session, providing the user with the ability to backtrack through multiple document states. The ability to backtrack through previous system or data states is a facility which interactive systems rarely provide for users.

MILO does not force the user along a course of action once it has been selected. If a user were to select the function to save a document to a file, for example, it would then be possible to escape from this action without having to save a file. The user is given the opportunity to terminate several actions of this kind before time and computing power is wasted.

5.3.3 Consistency within the application

MILO is designed to be consistent. The intention is that once a user has learned to carry out an action the gained knowledge can be transferred to carrying out similar actions within the system.

All text entry areas within MILO behave in almost exactly the same way throughout regardless of their function. The main text areas and titles of notes and text entry areas within dialogues all respond in the same way to the same keystrokes and mouse actions. Information can also be easily transferred between each text area using cut, copy and paste. This can be done with the confidence that the text will be treated in the same way wherever it appears. A minor inconsistency has been introduced to the benefit of the user. Some dialogues require the user to enter a file name. If such items of information were entered onto multiple lines the user might be faced with the problem that the operating system (as with most) does not allow anything other than single line, or single word file identifiers. For this reason carriage return operations are not permitted within these single line text entry areas—an inconsistency to the user's benefit.

Similarly, all pulldown menus within the system behave in the same manner, activated by the same mouse action. Additionally, as detailed in Section 5.3.1 each element of the structure overview has an attached menu, each of which contains the same functions. To know what note specific functions are available for a single element is to know what are available for all other elements.

All notes, the only data element within MILO, appear and behave in the same manner. All contain the same areas into which the user can enter information, all have the same attached functions and all note windows can be manipulated in the same way using window manager functions.

All dialogue formats, into which the user is required to enter necessary information, are the same, containing an information entry area, a button to click to initiate an action and a button to cancel the action.

In addition to interface objects within MILO having consistent appearance and behaviour, the behaviour of MILO as the result of user actions is also consistent. Selecting a function will always have the effect expected by the user from experience, providing a feeling of predictability and confidence in the dependability of the system.

5.3.4 Consistency with other applications

MILO utilises interface objects which are common to many applications employing a graphical user interface. These objects include push buttons, pulldown and popup menus, text entry areas, scrollbars and windows. As with other systems these objects are manipulated using a pointing device. The adherence to other interface design guidelines attempts to ensure that MILO is learnable and usable for a user new to such graphical user interfaces. Adherence to this guideline, however, eases the burden of learning and using a new system for users who already have prior experience of applications employing a similar style of interface.

The interface objects within MILO also behave in a manner similar to comparable objects in other systems. A single mouse click over a button, for example, will activate it, and a double mouse click over an object provides access to the entity which it represents. Menu entries are selected by a drag and release action with the mouse as in other systems

Additionally MILO's interface is implemented using the X Window System (Scheifler & Gettys, 1986), a tool which is now a widespread standard for implementation of graphical user interfaces. Consistency with other systems developed using X will be high with the additional advantage that information exchange between X based systems is eased via the X selection mechanism; text selected from one X application can easily be pasted into another X application.

5.3.5 Provide feedback

MILO provides feedback concerning the current state of the system. This is done in an informative and consistent way in order that the user is not misled by differing forms of feedback when

the system is in different states.

For example, reading a large document into MILO from a file, for example, can be a time-consuming task. The user is informed of progress in two ways: a textual message indicates how much more of the file remains to be read, and the mouse cursor changes shape to a watch to indicate that the system is busy. This reassures the user that some activity and the correct one is taking place, but also serves to indicate that other actions with the system are not possible or will be delayed until the current task is completed.

5.3.6 Commensurate effort

MILO users are required to invest more effort in selecting commands with potentially damaging results such as deletion of document elements, amending document structure and quitting the system. This is implemented in the ordering and format of command menus within MILO. Two factors have motivated menu design: often used commands should be easily accessible and potentially harmful commands should require more effort to access even if they are commonly used. Often used commands are placed at the top of menus in order to protect the user from the possibility of mis-selection. To illustrate an extreme, placing the often-used command to add a new element to the document between the deletion and session termination commands would not be acceptable because of the potentially harmful effects of mis-selections. Each menu, then, is designed to allow easy access to non-harmful commonly used commands.

The commands which are deemed potentially 'dangerous' are placed at the bottom of menus. This requires more user effort to select them and reduces the possibility of mis-selection. Additionally separators of differing types are placed between menu entries. The most 'dangerous' command, deletion, is separated from other menu entries by a more noticeable separator, while the restructuring command is also separated but less noticeably.

As additional protection for the user, commands such as session termination, reading a new document from a file or creating a new document require the user to confirm that this is their wish as they will result in the current document being overwritten. MILO will also prompt the user if the contents of the current document have not been saved to a file, allowing the user the opportunity to do so before they are lost.

Requiring commensurate effort of the user is valuable in a system. It must be noted, however, that within MILO most actions are reversible. What the extra effort is really protecting the user from is not damage resulting from actions, but the need to invest effort invoking system support to reverse those actions.

5.3.7 Avoid a cluttered display

Systems which encourage users to utilise multiple windows at any one time risk encouraging users to create untidy screens and become overwhelmed with windows and information. MILO has a single window containing a hierarchical document overview and global document commands, but many other windows containing note contents and different document views can be present. For this reason the user is supported by MILO in maintaining the number of windows at a useful minimum.

Individual note windows can be brought onto the screen and removed from the screen independently of other windows at any time. This can help the user maintain their current working set (Card *et al.*, 1984) of note windows explicitly. Part of the power of MILO, however, is that many notes can be visible on the screen at any one time in order to facilitate easy cross-referencing between document sections. Moving to a new working set of windows will require a great deal of window manipulation actions by the user unless supported by the system. MILO does this by allowing the user to remove *all* note windows from the screen by issuing a single command. Conversely windows for all notes within the document can be displayed on the screen by issuing a single command.

Windows within MILO can also be iconified as can those belonging to any other application the user is simultaneously using under the X Window System. In order to allow the user to differentiate between icons belonging to diverse applications each is labelled with the name of the application to which it belongs. Additionally the user can differentiate between icons corresponding to MILO owned windows as each different type contains a different image. Note windows, the main overview window and the mail dialogue window each have a unique icon design depicting the class of window.

A large MILO document of complex structure can lead to a large and complex hierarchical

overview. MILO provides the user with the ability to take detailed or abstract views of this overview by 'folding' and 'unfolding' objects representing elements and subordinate elements of the document structure.

5.3.8 Direct manipulation

The user can directly manipulate interface objects within MILO using a pointing device such as a mouse. All windows generated by MILO can be moved, resized, iconified and repositioned within the stack of windows on the screen via actions with the mouse. Scrollbars can be operated using the mouse, with dragging actions allowing the user to view different parts of windows containing more information than can displayed within the current window area. Window separators can be dragged using the mouse in order to directly alter the size of the separated windows.

Elements of the hierarchical structure overview can also be moved within the structure by directly indicating with a dragging action of the mouse to which position they are to be moved.

5.3.9 Exploit real-world knowledge

One aspect of exploiting real-world knowledge has already been touched upon in Section 5.3.4 where it was described how advantage could be taken of a user's previous experience of other systems in order to make a system more learnable and usable. MILO exploits aspects of other domains of user knowledge.

The notes within MILO, for example, can be thought of as loosely based on Post-It notes. They adhere to whichever part of the screen they are positioned, yet can easily be picked up and repositioned or disposed of. They can contain both text and drawings and can be used to write notes, reminders or larger passages of text. However, as pointed out in Section 5.2.1 to strictly adhere to the functionality of real world objects is pointless. MILO seamlessly brings other facilities such as the ability to create relationships between the notes, authoring/amendment history, filtering, different overviews and spell checking to the basic functionality that the user might expect from his or her real-world knowledge.

5.3.10 Support the user's model of the data being manipulated

Several views can be taken of a MILO document. At the text level, additions, deletions and amendments are made immediately visible to the user, as is the case in the majority of text editing systems. This helps to maintain a correct and current low level model of the data.

If the structure is amended in any way, the structural overview of the document is immediately updated to reflect the change. This occurs whenever elements are added to, removed from, or moved about within the document structure. A higher level view of the data is presented in order to maintain the currency and accuracy of the user's sense of the document as a whole.

5.3.11 Protect users from hardware/software failure

MILO achieves this by automatically saving the current version of the user's document to a back-up file at short, regular intervals. This does not impair system performance, and it is not an intrusive activity during usage. The back-up file does not conflict with the user generated version of the data but can be easily recovered when required.

5.4 Summary

This chapter has stressed the importance of considering the user in the design of useful and usable interactive computer systems. This is a difficult task which can benefit from guidelines for interface design.

CSCW systems, must, above all, even before consideration of cooperative issues be usable by a single user. Section 2.4 discussed why many CSCW systems fail. In order that MILO would not fail at the first usability hurdle, user interface design guidelines were adopted and this chapter has discussed how they directed the design of MILO.

Chapter 6

Summary and conclusions

This chapter summarises the contents of this thesis from literature reviews of CSCW and writing, through to the development of novel collaborative writing systems. It also details the contribution of this thesis to the areas of computer supported collaborative writing and human-computer interaction.

6.1 Summary of the thesis

This research has been motivated by a desire to investigate the provision of improved support for collaborating authors in computer based tools. Conclusions have been reached, not just in the design and implementation of support facilities for computer based writers, but also with respect to groupware tools in general and with respect to writing theory.

In chapter 1, the introduction to the thesis, the fact that people interact not only with computers but *through* computers was emphasised. The increasing exploitation of computers for person-to-person interaction, and their potential for support of collaborative work served to motivate the work presented in the following chapters.

The research initially involved a survey of exisiting CSCW theory and systems, taking account of meeting, communication and writing support systems. This survey, presented in chapter 2, enabled identification of social, design and implementation issues in the provision of computer based support of collaborative work (see section 2.9), and statement of requirements for such a system. It also led to the proposal of design principles for groupware: maximise the likelihood of

system acceptance at a personal level, mimimise system imposed constraints on users, minimise system requirements imposed on users.

Chapter 3 presented an investigation into theories of single author and collaborative writing with a view to their integration into computer tools. This investigation provided insight into the scope and applicability of such theories. Providing computer based support for writers is a difficult task, and as yet, models of the writing task are many and diverse. Functionality requirements for a computer based writing support system were suggested, avoiding dependence on models of writers or writing processes, but providing a basic framework for writers to utilise their own writing and collaborative writing strategies.

Many of these proposals were then exemplified in a prototype system to support collaborating authors, *Sticky-Notes*, which is described in chapter 4. This tool revealed details about the implementation platform, HyperCard, and more importantly about the design of a collaborative writing system. Personal reminders, author-to-author annotations, multiple views of document content, and automated merging of contributions were seen to be useful functions.

Requirements for a system to support asynchronous distributed collaborative writing tasks were presented, drawn from the surveys of Chapter 2 and 3.

MILO, a computer based tool to support collaborating authors of structured documents, was developed with respect to the requirements. It is described in chapter 4 where it is also related to other systems and models of individual and group authoring tasks. Appendix A presents a user's guide to MILO. MILO is a useful system, rendered usable by the consideration of human-computer interface design guidelines, and has several attributes novel to computer based writing systems, such as a Liveware merge of document elements, an emphasis on structure over surface details, and a context history of the collaborative writing task.

Usability is a central issue in the development of all interactive computer systems, including groupware. Chapter 5 describes how usability issues and interface design guidelines impacted on the design of MILO.

6.2 Limitations

This section acknowledges the limitations of this thesis and the work that it documents. The work presented here, as with any academic endeavour, could be refined, developed, extended or approached in an alternative manner. Chapter 7 (Future Work) will propose suitable avenues for exploration with respect to this—work for the future to consolidate and expand upon what has already been achieved.

However, there have been both circumstantial and conceptual limitations on the work which should be discussed and contextualised.

6.2.1 Circumstantial limitations on the work

First, at the outset of the software development aspect of the project, the X Window System was relatively sparsely used. For this reason local (ie departmental) expertise was non-existant, and other sources were difficult to access. This limited the initial speed and scope of development of MILO, requiring a large learning curve to be climbed. This resulted in the author becoming the local expert, but more importantly highlighted the need for a direct manipulation graphical interface design tool for the X Window System.

Such tools are now becoming more widely available, yet the locally available example, X Designer, still leaves much to be desired. Naive X developers are required to manipulate the underlying hierarchical structure of their interface objects, from which the corresponding interface is built. A simple question is 'Why can users not manipulate the interface objects themselves in the first instance, and turn their attention to lower level implementation issues when necessary?'. Other tools such as Builder Xcessory[1] show more promise as users can directly manipulate interface objects into position.

Second, and more generally concerned with successful completion of postgraduate research is the issue of funding. Financial hardship has been a constant for the duration of this research. Over the previous four years the government has systematically imposed ever increasing financial hardship on students, postgraduate students being some of the most badly affected. With studies showing that the majority of higher education students graduate in debt (a situation

[1]Integrated Computer Solutions, Inc. Broadway, Cambridge, MA 02139, USA

now actively encouraged through the Student Loans Scheme), the repercussions for the potential number wishing to take up postgraduate research are worrying. Those who wish to do so may find themselves unable to devote as much time as they would like to their studies because of the need to supplement their income via additional work. The situation has therefore arisen where both access to, and quality of postgraduate research are threatened.

6.2.2 Conceptual limitations on the work

Chapter 2 of this thesis ('The scope of CSCW') presents a survey and critical appraisal of several CSCW theories and systems. This is not exhaustive in that it does not discuss *every* contribution in the field. Indeed, as one would expect with any literature survey, it becomes outdated almost as soon as it is written. There are undoubtedly many appropriate papers published since the survey was carried out, and many which have not been included. However, this is not to say that it does not serve its purpose. It provides a solid introduction to the diversity of interests and systems relevant to CSCW, with an emphasis on collaborative writing support.

It serves to direct the reader to more detailed contributions in associated areas, and to underpin the work that follows in the thesis both theoretically and with an understanding of practical issues.

Chapter 3 ('An investigation into writing') may be viewed in the same way; it does not deal with every model of writing or writers. It does, however, indicate the diversity of approaches to sole or collaborative writing, and highlight basic phases of the writing task. This has revealed problems with normative writing support systems and enabled suggestions to be made for the design of new systems. As with the principles for groupware design proposed in chapter 2, these may not be the only means by which the desired end may be achieved. For the moment they can be thought of as a subset of all potential positive developments.

It has been claimed in preceeding chapters that *Sticky-Notes* and MILO, two collaborative writing support systems (described in chapter 4), are both useful and usable. It is important to note that this statement has no experimental or formal evaluative results to underpin it. It is made with the benefit of observations of usage of the systems by several authors, and with reference to their feedback on them. This may be viewed as a limitation. However,

the two systems in question are *point* systems, and embody a novel approach to provision of computer support for sole and collaborating authors. This thesis is not primarily concerned with evaluation techniques for groupware. In fact, appropriate techniques for CSCW system evaluation are lacking, and their development has been suggested as a suitable topic for research (see section 7.2.1).

As Tatar *et al.* (1991) have stated, "...CSCW systems almost always touch on many different research areas, not all of which can be pursued actively".

6.3 Conclusions relating to the design and implementation of MILO

6.3.1 HyperCard as a prototyping tool

HyperCard has become a widely used tool for prototyping graphical user interfaces. This is in spite of many shortcomings in its design, underlying programming language HyperTalk and its actual suitability for such a task (for further details see (Thimbleby *et al.*, 1992)). It might be concluded that Apple Computer, Inc., authors of HyperCard have done little to provide a more suitable yet similarly accessible application because of their investment in software development and links with third party developers. If users could develop *real* Macintosh applications using a system similar to HyperCard one can imagine power and profits being devolved from software houses to users. There is therefore a need for a tool which builds on the positive attributes of HyperCard, but learns from its developers' mistakes.

6.3.2 Designing and implementing a collaborative writing tool

At the outset of system design and implementation, it was not evident how the aims of designing from a reflexive perspective of CSCW, minimising constraints, minimising requirements, promoting integration and avoiding hardware dependency would interact. In practice there was little conflict between the aims, and they did not present difficulties in either the design or implementation stages of the software engineering task undertaken in developing the two systems.

There are a variety of differences in the requirements for asynchronous and concurrent collaborative authoring. Asynchronous collaborative authoring is primarily concerned with task division, viewing the work of others, integrating contributions and easing communication between co-authors. Concurrent collaborative authoring (as supported by shared screen systems) is more concerned with granularity of locking of data objects, protocols for access to objects, conflict in desire for different views of data, communicating intentions and systems response time.

6.3.3 Does MILO meet the specified requirements?

Chapter 4 presented requirements for the system to be developed, MILO. These resulted from a study of existing relevant theories and systems. Figures 6.1 , 6.2 and 6.3 present the requirements with an analysis of whether or not they have been met and why this is, or is not the case. This analysis reveals that the majority of requirements were indeed met.

However, it is difficult to determine without detailed evaluation if requirements such as enhancement of existing relationships, increased quality and efficacy of interactions and provision of benefits of physical proximity have been met. Environmental constraints, such as availability of hardware and software development environments also compromised meeting requirements related to hardware suitability and independence.

Additionally, linear drafts of the text have not been provided .

6.3.4 A comparison of MILO with other collaborative writing systems

Figure 6.4 provides a comparison of MILO with other collaborative writing systems. It is based on the requirements and comparison table presented by Posner and Baecker (1993).

Unfortunately, Posner and Baecker are not always clear about what each requirement specifically entails. The following presumptions have been made when suggesting how MILO meets the requirements which are not explained in detail in their paper:

'Support a variety of activities'. *Researching* is taken to mean provision of support for access to on-line information sources. *Reviewing* is taken to mean provision of support for reading, commenting on and editing work of oneself or others.

'Access to relevant information'. This is taken to mean provision of information about the group writing process such as current and past activities of group members in relation to the document being produced.

It can be seen that MILO compares favourably with the other systems considered. In several of the requirements MILO provides no support, namely make roles explicit, provide synchronous communication, read only access methods, and synchronous document access. These are not limitations but conscious design decisions; ie do not constrain writers to roles embedded in the technology, do not impose different levels of access and support distributed, asynchronous collaboration.

It could be claimed that MILO actually provides synchronous access to the document, as multiple users can work on copies of the document at the same time. However, in this context, synchronous access is taken to be real time shared editing of the same document.

6.3.5 Use of MILO

Anecdotal evidence has already indicated that MILO is both useful and usable for single author and collaborative authoring tasks. Its use has lead to the following insights:

- it is tangibly useful;
- its emphasis on structure can ease document creation, review and amendment tasks;
- multiple views of document structure support a better sense of the document as a whole, and the time-ordered view can be useful in regaining the thread of work left in a previous session;
- the ability to refer to several document sections at the same time (via multiple windows) is very useful for cross referencing and review. It is a great improvement on the limited number of lines from a single location within a document which standard screen editors provide;
- support for unattached document elements allows authors to delay commitment to structure and use MILO as an outline/idea processor, include as yet unplaced elements into the document when appropriate;

- the protection against hardware/software crashes has proven useful;

- the ability to amend structure by simple direct manipulation of document elements is a great improvement on single screen display editors, easing and encouraging structural changes;

- the windows in which document elements are displayed can be resized to any dimensions. This is useful in that it allows the display area for data to be the most suitable size;

- the single mapping to a LaTeX document is restrictive. The suggestion of Section 7.1.5 to provide a library of alternatives is one which should be undertaken;

- the use of the X Window System as the interface development platform for MILO has allowed integration with other applications. Text can easily be transferred between applications;

- MILO has been effectivley utilised to create documents other than those exhibiting the hierarchical nature that was originally envisaged. It has been used to author short stories and plays, and during this process its division of documents into logical elements and emphasis on structure has been useful (Kane, 1993);

- the groupware design principles had a positive impact for MILO users. It makes mimimal requirements of users, places minimal constraints on their activities and provides aupport for personal activities within the group context.

MILO has also been used in a collaborative document production task to produce (Cockburn & Jones, 1991). This also revealed some insights

- Use of MILO can be successfully integrated with use of other systems supporting collaborative work. For the co-authoring task in question, MILO was used in conjunction with Mona (Cockburn & Thimbleby, 1993), a system designed according to the same principles as MILO, but which provides conversational context in electronic mail. Mona provided a high level view of the structure of the collaboration, allowing the collaborators to see and review a communication history. MILO provided a lower level view, that of contributions to the product of the collaboration, the document;

- the filtering facilities provided in MILO are highly useful for identification of amendments and additions to the co-authored document, which in a conventional text editor would require explicit marking by the authors to render them visible;

- the element based structure of MILO documents lends itself well to a variety of collaborative document production techniques. Both longitudinal and parallel partitioning of the task (Sharples *et al.*, 1993) took place during writing and was successfully supported by MILO, which eased the process of combining the work of individuals;

- the Liveware merging of versions of a document is a promisingly useful facility. However, it requires further development. This would deliver a more sophisticated merging algorithm which provides greater flexibility in the manner in which document elements are merged (see Section 4.7.11).

6.3.6 Potential uses of MILO

MILO is a flexible system, and its use can be generalized beyond that of collaborative authoring of textual and graphical documents.

A bulletin/discussion board

MILO has common attributes with bulletin or discussion board systems. Such systems build conversations and discussions over time, providing for diversification of subjects along different conversational paths. Communicants search for items of interest, and post their own contributions. Such systems, in general, provide only a text based interface to the discussion structure.

The hierarchical document elements of MILO, however, could be viewed as contributions along conversational branches. Contributors can create new branches for diversification of topics, or comment on existing elements. MILO has several advantages in such an application:

- it provides a graphical overview of the structure of the discussion;

- it provides sophisticated filtering functions to allow identification of objects of interest;

- multiple contributions can be visible at any one time allowing of ease of cross reference and assimilation of several related contributions;

- alternative views can aid in location of most recent additions or amendments;
- conversational strands are easily distinguishable and closely related spatially in the representation, rather than their elements being intermingled with contributions to other strands;
- the Liveware merge can automate integration of other people's contributions.

Program development environment

Computer programs can be viewed as logically hierarchical documents where code segments (perhaps subroutines) correspond to document elements. MILO notes could be used to hold code segments. Such an approach would lend itself to a variety of software development methods. A single developer can see a graphical representation of the code structure, access a historical view of changes to the program, easily amend the program structure and view several code segments at the same time.

MILO would also support a variety of approaches for software development teams. Task subdivision could occur along parallel or longitudinal lines (as with collaborative document production) and individual contributions could be incorporated using the Liveware merge facility. Filtering would reveal amendments and additions of interest.

6.4 General conclusions

6.4.1 The failure of CSCW tools

It is evident that the majority of CSCW tools have failed to gain any widespread acceptance. A noticeable exception is in the area of computer supported communication—electronic mail. Applications in meeting support, coordination and writing have been notable for their lack of success, and are, in general, developed in academic research environments rather than by industry. The most persuasive arguments for this are that commercial software developers have yet to commit sufficient resources to the widespread development of groupware; groupware applications are inherently very difficult to design successfully, and evalution of groupware is not yet sufficiently developed to learn from "experiences of failure". The recent emergence of high

profile systems such as Lotus Notes and Windows For Workgroups augers well for the future. CSCW will gain a higher profile and more systems will follow in their wake. However, a note of caution must be introduced. These systems will mould those yet to emerge and it is important that both their shortcomings and good points are recognised in the design of new systems. This thesis has proposed principles to guide that design.

6.4.2 Lack of integration of CSCW tools

There is little integration or potential for integration between CSCW systems. This is surprising. One would imagine (or perhaps hope) that designers of systems to support cooperative or collaborative tasks would recognize the need for integration of systems supporting different aspects of a task. In a collaborative environment one would hope that transition between systems to support collaborative writing, communication and coordination would be straightforward.

This is not the case. It has been found that few groupware systems are designed with integration in mind. They constrain users to a specific type of hardware, impose unique formats on data making transfer more complex than necessary and require non-transferable user behaviours. This thesis has proposed principles of groupware design which aim to reduce user transitions between systems and work processes.

6.4.3 Principles for the design of groupware

The efficacy and usability of groupware, and hence its acceptability, can be increased. Transitions required of the user between programs, hardware platforms and work processes serve to increase the user effort required to exploit groupware. In addition, groupware tools may not provide support for personal activities within the group context, may constrain users to inappropriate behaviours or require them to provide information or carry out unecessary actions.

Groupware design principles may help to avoid such undesirable aspects. The following principles: maximise the likelihood of system acceptance at a personal level, mimimise system imposed constraints on users, minimise system requirements imposed on users have been used in the design of several groupware applications including MILO (see also (Cockburn & Thimbleby, 1993)). Strategies for implementing the principles have been also been proposed ((Jones &

Cockburn, 1993; Cockburn & Jones, 1991)).

6.4.4 The complexity of writing and writing theory

Writing is a highly complex cognitive task. It is possibly for this reason that there are a variety of models of writing and writers, and a consensus on how to view writing is lacking. Several models are loosely based around the tasks of planning, translating and reviewing, yet fail to provide a definitive model of a generic writer. Perhaps there is just too much diversity among individual writers to enable a useful, generalizable model to be developed. This conclusion would certainly seem to be borne out by the large number and variation of strategies which writers have been seen to use.

The complexity of a single writer's task might go some way to explaining the lack of theory which is applicable to collaborative writing. The complexity is increased many-fold when considering the strategies of each individual writer, and the interaction between them. Insights into such tasks are currently at a high-level, concerning the distribution of writing of document segments over space and time (Sharples *et al.*, 1993).

6.4.5 Writing theory and computer systems

The evident complexity and diversity present in writing tasks leads to the conclusion that attempts to embed writing theory or models in computer systems are unlikely to be successful. Constraints imposed by writing support software are likely to conflict with the strategies and approaches of many writers. Consideration of this statement in conjuction with the inference that it will take a long time to develop suitable theories of collaborative writing leads to the conclusion that in order to design useful systems now, indicators can be taken from writing theory to *support* but not constrain writers.

Minimal constraints can allow social protocols to prevail, allowing collaborative authoring tasks to be mediated and driven by the participants rather than a computer system.

MILO can be viewed as a *point system* as described by Olson *et al.* (1993), who highlight that fact that CSCW is, as yet, an emerging discipline. Consequently many systems are being developed, some for academic purposes, some for commercial exploitation, some are evaluated and

some are not. They argue that all such systems are useful contributions, promoting discussion and criticism. This is one of the purposes of MILO.

6.4.6 Transferring emphasis to higher-level issues in computer based writing systems

The majority of computer systems for writing support emphasise low-level issues such as spelling, word counting and typefaces. Research has shown, however, that computer based text tends to degrade a writers sense of the whole document. Increased support for the viewing and manipulation of document structure needs to be investigated. MILO provides this support and could be used as a vehicle for investigation of these issues.

(via multiple windows) is very useful for cross referencing and review.

6.4.7 CSCW and HCI

Design of collaboration support systems is inherently more complex than design of systems to support personal work. Certainly systems which support collaborative writing must truly provide the best of both worlds, supporting writing on a personal basis and providing facilities which support collaboration with co-authors. The distinction between these two types of work task must, as far as possible, be blurred so that neither is inhibited, and transitions from one to another are eased.

An interdisciplinary approach

As in any area of computing which is concerned with the development of interactive systems, groupware designers must be aware of human-computer interaction issues. However, guidelines of groupware design and interface design may well impact upon each other.

For example, the groupware design aim of hardware independence to provide increased accessibility and increase the number of potential collaborators for users can impact on interface design. Currently, provision of hardware independence requires use of the X Window System for interface development. However, this requirement restricts the developer in choice of both look and feel of the interface, and may rule out use of the most appropriate interaction techniques

for the application. Such potential trade-offs need to be identified early in the design process in order that the most suitable design decision for the application in question can be made.

In contrast, however, the groupware design aims proposed in Section 2.9.2 may have a beneficial impact on the design of the user interface to CSCW systems. The drive to provide unconstrained systems which support flexibility in work processes and interactions with others and the system may help to avoid undesirable aspects of interactive systems such as preemption and modes. Users will be supported in more flexible interactions without system imposed orderings on activities or actions.

6.5 In conclusion...

This thesis has made several theoretical and practical contributions to the understanding of Computer Supported Cooperative Work, specifically the area of computer supported collaborative writing.

New principles for the design of groupware have been proposed. Existing models of individual and group writing have been found to be inappropriate for direct inclusion in systems because of their diversity. They can, however, drive the design of a basic support framework. General requirements for groupware and writing systems have been identified. Requirements for a tool to support asynchronous distributed collaborative writing have been presented. Two novel authoring systems *Sticky-Notes* and MILO have been developed and documented as media for expression of the stated design requirements. Conclusions regarding collaborative writing theories and systems were stated earlier in this chapter.

The following (and final) chapter indicates how this work may be improved, continued and extended.

General Requirements			
Requirement	**Adopted**	**Met**	**Comment**
Enhance existing relationships, encourage new ones	yes	?	To be determined by evaluation of the system in a field trial.
Improve quality, efficacy and product of interactions	yes	?	To be determined by evaluation of the system in a field trial.
Provide benefits of physical proximity	?	?	Doesn't support the rich interactions made possible by physical proximity. However, it is intended to support collaborators who *cannot* achieve physical proximity.
Accommodate changes in roles and commitments	yes	yes	Users are free to change roles at any time and the system does not record or enforce commitments. This has been observed to be empowering in a collaborative authoring task mediated by social protocols. Roles/commitments were highly dynamic due to external pressures.
Maximise likelihood of acceptance at a personal level	yes	yes	Anecdotal evidence would seem to suggest that MILO is useful for single authors for a variety of tasks.
Integrate existing social protocols and relationships Do not enforce protocols or restrict use of existing ones Minimise system imposed constraints on users	yes	yes	Social protocols have prevailed, resulting in successful task completion, in a multiple author task.
Minimise system imposed requirements on users	yes	yes	Authors are only requested to provide the system with a single piece of information: their personal identifier. User names or email addresses could be retrieved from the environment, but this allows some personalisation by the user. Other information not retrievable by the system such as names of save files, document to email etc is requested of the user, but in a manner which minimises typing and suggests commonly used options.
Exploit the most suitable hardware	?	?	The most suitable available hardware has been used. This did indeed rule out the use of video/audio, but the powerful graphical workstation that the system was designed for supports high quality graphical interfaces, and provides high performance.
Use electronic mail as a communication medium	yes	yes	The system has been successfully interfaced with electronic mail.
Output should be useful and manipulable	yes	yes	File based output is of three types (all ASCII): plain text, LaTeX format, MILO format. Each type is editable and mailable without use of MILO. Plain text allows users to include their own formatting commands. LaTeX format requires no knowledge of LaTeX to produce a typeset document. MILO format allows MILO generated documents to be read back into the system.
Allow for integrated use with other systems	yes	yes	Development of an X Window System interface allows data to be cut and pasted between MILO and other X applications. Both MILO and Mona (Cockburn, 1993) have been developed according to the principles for groupware design proposed in Chapter 2 of this thesis. Their use has been successfully integrated in a collaborative writing support tool. MILO integrates with an email system, spell checker, document formatter, document previewer.
Information exchange should not be dependent on the system	yes	yes	MILO documents can be exchanged without the use of MILO.
The system should be hardware independent	yes	?	Development of an X Window System interface should promote hardware independence. Recompilation should be the only necessary step to install MILO on any hardware supporting the C language and the X Window System. However, the development of X and adoption of standard widget sets/toolkits means that some widget sets (particularly proprietary ones) are not available on a variety of hardware. This is the case with MILO. It was developed on Hewlett Packard hardware using Hewlett Packard's widget set. This severely reduces portability as the widget set is not widely available. If Motif has been available it should and would have been used.

Figure 6.1: An evaluation of MILO relative to the adopted equirements for a generic groupware system.

Writing support tool requirements			
Requirement	**Adopted**	**Met**	**Comment**
Hierarchy of structures and tasks	yes	yes	Document creation and manipulation in MILO centres around a representation of the hierarchical structure of the document. Several hierarchical structures relating to the single document can be visible and manipulable at the same time. This can lead to a more evident top-down/bottom-up approach to document development (or indeed a hybrid approach) where subtasks are directly related to the document hierarchy. The provision of this hierarchy is also intended to promote higher level tasks such as considering structure over lower level tasks such as spell checking.
Interleaving of tasks	yes	yes	MILO does not impose an ordering on task execution or completion. During any task, the user is free to transfer attention to any other task supported within MILO. Additionally the user is free to transfer attention to tasks requiring use of tools other than MILO. The time-ordered view of additions and amendments may help to remind users of what they were doing before interleaving took place.
Basic text manipulation operations	yes	yes	Each note contains a basic text editor (supplied with the HP Widget Set). The functions provided are listed in Appendix A. Additonally a search facility to locate text within a single note or any of the notes in a MILO document has been provided. Text can be cut and pasted between MILO and other tools in the user's environment which support the X protocol.
Reuse of previously generated text	yes	yes	Documents saved in MILO format can be used as input to MILO to recreate structure and content. Documents saved in plain text ASCII format can be used as input to any other ASCII file based tool. Documents saved in LaTeX format can be used as input to the LaTeX document preparation tool. Any ASCII file can also be loaded into any note of a MILO document.
Multiple views of the same information	yes	yes	MILO provides hierarchical, linear, time ordered views of the document. It also supports filtering of elements which conform to specified criteria. It does not provide a linear, editable version of the complete document, or a representation of the final format. The later can be accessed via the LaTeX system, and a previewer such as *xdvi*.
Creation of unordered items	yes	yes	Any number of single document elements can be created outside of the main document structure. Any number of hierarchical document substructures can be created outside of the main document structure.
Creation of notes and networks of notes	yes	yes	Documents can be built from a structure of notes (document elements). Notes can be hierarchically related, but do not form a network.
Provision of linear drafts of the text	yes	no	MILO does not provide either an editable or a formatted linear version of the text.
Easy movement between views of the information	yes	yes	Each view can easily be selected from a menu and all views are simultaneously accesible. Editable views are still editable when other views are active.
Merging document text and meta text	yes	yes	Unordered text can be integrated into the document at any point during the interaction, and at any position in the document.
Preservation of collaborator identities	yes	yes	Each document element provides access to historical information about authors and amenders of the element.
Support for communication among collaborators	yes	yes	MILO provides and interface to electronic mail. The user is presented with a list of regular email contacts, and contibutors to the system. The current document and/or other files can be mailed to any of the individuals or groups by direct manipulation.
Support brainstorming, researching, planning, writing, editing, reviewing	yes	?	Explicit support is provided for writing (text generation) and editing through the text editor. Brainstorming is supported through creation of unordered notes, ease of document restructuring and interleavable tasks. Researching planning and reviewing are not explicitly supported.
Support transitions between activities	yes	yes	Similar to interleaving of tasks.

Figure 6.2: An evaluation of MILO relative to the adopted requirements for a writing support tool.

Writing support tool requirements			
Requirement	**Adopted**	**Met**	**Comment**
Support separate document segments	yes	yes	Notes represent individual document segments of unconstrained size, organised in one or several hierarchies, each containing one or several notes.
Support one and several writers	yes	yes	Designed from the Reflexive Perspective of CSCW, MILO supports the activities of a single author. This is essential for asychronous, distributed co-authoring. It also provides facilities which support the single author's activities within the context of a group authoring task.
Provide easy, flexible information entry	yes	yes	Textual data can be entered into any note at any time via the keyboard, or can be imported from an ASCII file. It can also be pasted from other application supporting the X protocol. Diagrams can also be input using the drawing tool which is integrated into each note.
Provide spatial representations of the text	yes	yes	The document structure is represented by a hiearchy in which elements are directly manipulable. Additionally the user is free to impose any spatial relationships required between individual notes (the note windows in practice).
Provide automated merging of contributions to the document	yes	yes	A Liveware merge has been implemented which integrates two versions of the same document. The function and shortcomings of the merge are described in chapter four of this thesis, and later in this chapter.
Maintain an easily accessible history of the interaction	yes	yes	A time-ordered view of activity in the document is available to users. Activities which amend the document are recorded (times/author responsible), and can be accessed for individual document elements. MILO's use has been integrated with that of Mona (Cockburn, 1993), a conversation web visualisation system, to provide a richer history of interaction between collaborators. This complements the document level detail provided by MILO.
Provide structure manipulation tools	yes	yes	Elements can be added to any point of the structure, deleted from any point in the structure, and moved in the structure by direct manipulation. Branches of the hierarchy can also be compressed to provide views at different levels of abstraction.
Provide multiple types for document components	yes	yes	Each document component can contain a textual element and a graphical element.
Provide multiple types for annotations	yes	yes	There is not an specific annotation object in MILO. Authors deal with a single document element type, the note. Notes can also be used for annotation and consequently can be textual or graphical and are consistent with document elements.
Provide utilities eg spelling checker	yes	yes	Spell checker, text search.

Figure 6.3: An evaluation of MILO relative to further adopted requirements for a writing support tool.

Requirement	Aspects	ForComment	GROVE	PREP	Quilt	SASSE	ShrEdit	MILO
Preserve identities	✓✓	✓✓	✓✓	✓	✓✓	✓✓	✓✓	✓✓
Enhance communication								
Annotations	X	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓
Asynchronous	X	✓✓	X	X	✓✓	✓✓	✓	✓✓
Synchronous	✓✓	X	X	✓	X	✓✓	✓✓	X
Make roles explicit	✓✓	X	✓✓	X	✓✓	✓	X	X
Variety of activities								
Brainstorming	✓✓	X	✓	✓	✓	✓✓	✓✓	✓
Researching	X	X	X	X	X	X	X	X
Planning (outline)	✓	X	✓✓	✓✓	✓	✓✓	✓	✓✓
Writing	✓✓	X	X	✓✓	✓	✓✓	✓✓	✓✓
Editing	✓✓	X	X	✓✓	✓	✓✓	✓✓	✓✓
Reviewing	X	✓✓	X	✓✓	✓✓	✓✓	✓	✓✓
Transitions between activities	✓	X	X	✓✓	✓✓	✓✓	✓	✓✓
Access to relevant information	✓	?	?	✓	✓✓	X	✓	✓✓
Make plans explicit								
Process plans	X	X	X	✓	X	X	X	X
Outline plans	X	X	✓✓	✓✓	✓	✓✓	X	X
Version control mechanisms	✓✓	X	X	✓	X	✓✓	X	X
Document access								
Synchronous	✓✓	X	✓✓	X	X	✓✓	✓✓	X
Sequential	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
Several access methods								
Write	✓✓	X	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
Comment	X	✓✓	X	✓✓	✓✓	✓✓	✓	✓✓
Read only	✓✓	X	✓✓	✓✓	✓✓	✓	X	X
Separate document segments	✓✓	X	?	✓	✓✓	X	X	✓✓
Number of writers								
One	✓✓	✓✓	?	✓✓	✓✓	✓✓	✓✓	✓✓
Several	✓✓	X	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
Writing approach								
Synchronous	✓✓	X	✓✓	X	X	✓✓	✓✓	✓✓
Asynchronous	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓

Notation	
✓✓	system provides support
✓	system can handle but does not explicitly support
X	system does not support
?	not clear if support is provided

Figure 6.4: Posner and Baecker's comparison of group authoring support tools, amended to include MILO.

Chapter 7

Future work

MILO is a useful and usable system. This has been witnessed through its use in writing the majority of this thesis, a collaboratively authored paper, and documents with very different structures such as plays and poems. Extensive usage of MILO has prompted changes in the system. Usage indicated the efficacy of existing functionality and stimulated ideas as to how it could be extended or improved. Usage has also shed light on the usability of MILO, leading to further development of the user interface.

What MILO can offer a user (a single or collaborating author) is constrained by the time available to implement and evaluate new ideas about, and different approaches to provision of facilities to aid a writer or multiple authors in creating a structured document. MILO has been seen to be both a useful and usable system in its current state, yet this should not halt the desire to improve it, or to test the validity of further ideas.

This chapter suggests further work to be carried out on MILO. In some cases this is to address deficiencies in the system. In others it is to provide additional facilities to users, and in others to evaluate further ideas. Avenues meriting further investigation which are applicable to CSCW systems in general have become evident through this work. They are also discussed.

7.1 MILO

7.1.1 Provide a linear editable version of the document

The only stated requirement that MILO definitely does not meet is provision of an editable linear version of the document being developed. This can be rectified in a straightforward manner. The required output will be either the plain ASCII text or LaTeXform of the document already output to a user selected file. Modification of this process to give the user the option of displaying a screen based linear version of the document is a trivial software modification task.

Provision of an editable final draft of the document is a greater challenge. This implies a WYSIWYG-style editor. However, such a facility is at odds with adopted design considerations such as promotion of structure and content over low level issues such as document layout.

7.1.2 Provide a non-graphical menu based interface

The benefits which MILO provides authors are currently only available to those with access to hardware with the capability of providing the user with an X Window System based graphical display. Although the access to and the provision of X based facilities is quickly becoming more widespread over a range of diverse hardware platforms, authors who only have access to text based hardware are not yet able to benefit from the facilities which MILO offers. Authors using MILO are also therefore limited to a certain extent in their pool of potential co-authors. To extend MILO so that users of text based hardware can access its functionality would therefore benefit not only them, but also authors currently using MILO.

For this reason I suggest that an interface to MILO be developed which would exist alongside the current graphical interface but would not be dependent on the user having access to graphical hardware. Much of the current functionality could be accessed via a menu based or command line interface. The menu based interface would be the more appropriate approach as it could be developed to closely correspond to the graphical menu driven interface which already exists. This would ease use of MILO for both users who have been using MILO on text based hardware and gain access to graphical hardware and for those who in some instances do not have access to the graphical hardware which they normally use.

```
$ milo

                                NO CURRENT FILE

DOCUMENT                    NOTES                       VIEWS                     UTILITIES
Register user identifier ^I Display all notes      ^D   Filter notes         ^E   Spell check              ^K
Create new document      ^C Access single note     ^N   Find text            ^F   Use mailer               ^M
Read document from file  ^R Add a note             ^A   Outline view        ^O   Liveware merge           ^L
Save document to file    ^S Delete a note          ^X   Time ordered view   ^T   Print document overview  ^P
                            Reparent a note        ^V
                            Undo most recent deletion ^U                          Quit                     ^Q
```

Figure 7.1: A possible initial MILO menu in a text based interface.

MILO currently ascertains whether an author is using a display with X based facilities, and if not will indicate that this is necessary. A trivial extension would be to invoke the textual menu based interface if this is the case.

A suggested menu based interface to MILO

The format of a menu based interface to MILO is suggested which conforms closely to the format of the existing graphical menu interface. It allows access to much of the functionality offered to users of the graphical interface. The user initially has access to menu choices which globally affect the current document, and these functions would form the basis of the text menu.

The keyboard commands should match any keyboard accelerators provided by MILO. Figure 7.1.2 shows a possible initial MILO menu in a non-graphical interface. Only one of the graphical menu entries are not pertinent in this situation, **Hide all notes** as there are no note windows on the screen which may require tidying. Several of these functions—**Register user identifier, Read document from file, Save document to file, Filter notes, Find text, Liveware merge** and **Print Document Overview**—require additional information from the user. In this case the user will be prompted for the information by a text based dialogue.

Unlike the graphical interface dialogues, such dialogues will be pre-emptive; the user will not be able to select other commands whilst the dialogue remains to be completed. This serves to protect the user from the confusion of multiple textual prompts interspersed on the screen.

The textual interface to MILO, would not be totally pre-emptive, however, as the user could interleave MILO tasks with other tasks through the provision of a command line escape sequence (!⟨*command*⟩ is a commonly used sequence).

A hierarchical overview of the current document structure could be provided textually. Each document element is now uniquely numbered so that the user can refer to them easily as the

directness of the graphical method is lost. The note functions **Access single note, Add a note, Delete a note** and **Reparent a note** will require the user to indicate a note by its unique numerical identifier.

The hierarchical overview will be updated after each amendment to the document structure as the graphical overview is, in order to maintain the user's model of the document.

Note text entry could be achieved by invoking a default or user specified text editor into which document contents could be entered. On completion of text entry, control would return to the main text menu and document hierarchy view.

The functions providing different views of the document contents and structure would provide a textual list of note headings. Selecting **Outline view** for the current document, for example, would produce output of the form show above.

This section has suggested a menu based interface to MILO, indicating that it is feasible to provide access to the functionality of MILO in such a manner. The implemented version should, of course, be designed with regard to the wide body of literature concerning the design of such interfaces, such as (Whiteside *et al.*, 1985; Callahen *et al.*, 1988; Shneiderman, 1983).

7.1.3 Explicit annotation note type

Sharples and Pemberton (1990) stress the importance of providing for introduction of meta-text into computer based documents. This text is not part of the document text but is an external addition to the document, either referring to the structure or content. In general, meta-text takes the form of an annotation, which can comment on the document, suggest alternative phrasing or structure, or pose questions.

MILO makes no explicit provision for inclusion of meta-text. Authors using MILO can easily include both textual and graphical annotations as new notes within the current document. This is a conceptually simple approach, in which the user need make no differentiation or cope with any difference in behaviour between notes—there is always only one document element type. With this approach, however, the onus is on the recipient of a MILO document to discriminate between the intended purpose of each document element—is it part of the document content or a comment upon the content or structure?

Further development of MILO could provide meta-text facilities which eradicate such potential confusion, yet maintain the conceptual simplicity of a single document element type. Annotations would appear and behave in exactly the same way as notes containing document text and graphics, except when the document were printed out. Annotations would be requested via the menus attached to each note or overview object representing notes. Hence an annotation would be attached to the note from which the menu had been selected in the same way that new notes are attached to existing notes.

Annotations would be recognised as a result of the way they are represented in the hierarchical overview.

On saving a document the user would be able to specify whether or not annotations should be saved with the document content. Representation of annotations in LaTeX format documents could follow simply by using the *marginpar* command to provide annotation text in the document margin alongside the document element which is being annotated.

Provision of this functionality would allow exploration of whether the suggestion that computer based writing systems need to provide explicit support for meta-text is valid. MILO currently does not provide this explicit support but does allow users to annotate document content.

7.1.4 Integrate MILO with alternative document formatting systems

A MILO author has the option to store a structured document in a file which can be used to prepare a formatted version of the document. MILO currently creates a file which will allow the document to be formatted using the LaTeX document formatting system. This is achieved by mapping elements within a MILO document to elements within a formatted structured document. LaTeX commands are therefore integrated with the contents of the structured MILO document in the created file which can then be processed using the LaTeX document formatting application to create a device independent file representing a formatted version of the document.

As a result of this approach authors using MILO can incorporate LaTeX formatting commands into the text of the document being created or edited if they plan to use LaTeX to format the document. This is useful for inclusion of document elements such as tables, itemised lists and

citations and indicating the desire for effects such as change of typeface or font.

Not all MILO users, however, will have access to LaTeX. Although those that do not have acess to LaTeX could save the MILO document in the plain text form and enter the appropriate document formatting commands it is more desirable to provide the user with a wider range of choice of formatting systems which MILO can prepare a file for. For example, in addition to LaTeX, MILO might provide the user with SGML or nroff as alternatives. In order to achieve this, the process which creates the formatted file will be required to abstract the document element types from the control sequences or commands for each of the formatting systems. Hence MILO, when mapping a document element to a section might interrogate a file to retrieve the appropriate text to place in the output file.

Such a file could easily lend itself to user configuration, so that an author could alter the default format for document elements which is output when a file is created by MILO for processing by their favoured formatting system. Authors could also add commands for formatting systems not initially supported by MILO. MILO could therefore provide integration with both procedural and descriptive markup systems (Furuta *et al.*, 1982; Coombs *et al.*, 1987).

7.1.5 Alternative linearisation heuristics

MILO currently utilises a basic heuristic for producing a linear document from the hierarchical structure created by an author. This heuristic maps the root note of the document hierarchy to an abstract, elements at the next level to sections, those at the next level to subsections and so on. This has been found to be a useful mapping for creation of structured, hierarchical academic documents but there will undoubtedly be cases where authors require an alternative approach to linearisation. For example, a book written with MILO will require a mapping to chapters, sections and subsections, or a letter would require a mapping to addresses, salutation, letter contents (which may contain sections) and closing message.

Whereas Sharples and Pemberton (1990) suggest that the user should select from different document types before creating a document and then be constrained to the structure provided by the system, I would suggest that the user should create a structured (or unstructured) document and then indicate which document type they wish the structure to be represented by. This could

be done by providing the user with linearisation options to select from when the document is saved to a file. MILO would then select the appropriate control sequences to include into the file to produce the document type required by the user.

Integrating this approach with the provision of various document formatting options would allow a MILO author to create various forms of document utilising various formatting tools from a single hierarchical structure.

7.1.6 Provide context sensitive help

As described in Chapters 4 and 5, MILO is designed to be both useful and usable. Many of the utilities which it provides are readily available via pulldown menus which a new user can investigate and try out. The ease with which actions can be reversed within MILO should encourage new users to become familiar with the system with minimal apprehension.

Although clear documentation is provided to help users become acquainted with MILO (see Appendix A, A User's Guide To MILO), not all users will always have access to the documentation. It is also sometimes difficult to transfer instructions or descriptions from the written page to actions on the computer screen. New users of MILO could benefit from help or instructions on system usage which are readily available on the screen. Some systems, however, provide this help to users as a distinct and separate module where it is still difficult to associate the help information provided with actions to be carried out or objects in the interface. In some cases the help information can even conceal the objects it refers to.

Context sensitive help can strengthen the association between the information and the referenced object or action and relieve the user from the need to search through information about the whole system to find the sections of interest. Context sensitive help in MILO will, in part, take the form of information available from within menus which describes the effect of menu entries. Figure 7.2 provides an example of how such information may be presented. Each menu entry will have a subelement which contains a textual description of what the user will be required to do and will see happen if that function is selected.

MILO could additionally print the help information that appears in menu subelements in the message box below the hierarchical structure overview. This would allow it to easily be

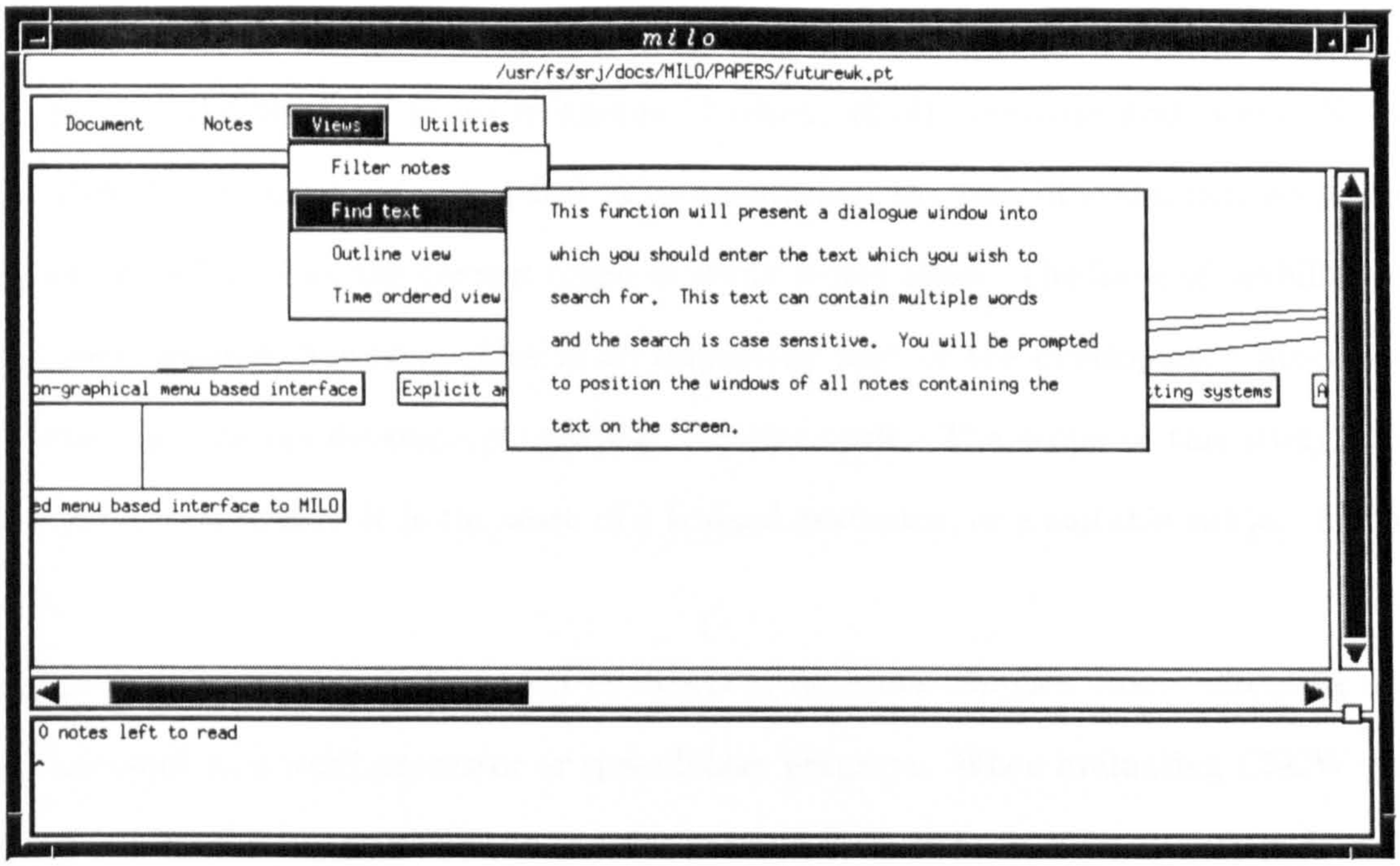


Figure 7.2: This figure shows how context sensitive help to indicate the effect of menu entries might be presented.

returned to and referred to repeatedly, without the transience of the menu entry.

An additional help menu in the main menu bar could contain entries corresponding to utilities which are not activated from menus. This would provide a similar method of access to information about activities such as the moving of elements in the document structure overview by direct manipulation, use of scrollbars and paned windows, traversal and manipulation of text in text entry areas of notes, use of the graphics editors within notes and how to access note windows from the hierarchical overview.

7.2 CSCW in general

7.2.1 Extensive evaluation

MILO has not undergone extensive evaluation. It has been informally evaluated by a handful of users, and been seen to be a useful and usable tool for writing documents. It has not been fully evaluated for its effectiveness in collaborative writing tasks. To undertake full evaluation of a powerful, sophisticated system such as MILO which exploits a graphical interface would be a major, long-term project in itself.

Usability testing is a research area in itself, and even using existing techniques to evaluate MILO, such as the thinking aloud technique (Nielsen, 1988), heuristic evaluation (Nielsen & Molich, 1990) or more rigorous empirical experimentation, the task of evaluation would require more time and effort than the current scope of study would allow. The issue of usability testing has not been ignored, however. This is an important part of the development process which should exist in a design-development-testing iterative cycle. The scope of this study excludes such intensive evaluation—it is the work of a trained evaluator, or a suitable subject for further research.

Evaluation of a CSCW system such as MILO is far more complex than evaluating a single user system such as a word processor or spreadsheet program. When evaluating CSCW systems two areas of usability must be investigated. Firstly the system must be usable and useful for a single user. There is little point designing a collaborative system that a single user cannot utilise; there will be no scope at all for collaboration. CSCW systems should undergo the same intensive usability evaluation that should be applied to all systems. For systems such as MILO which provide a sophisticated graphical user interface to ease the use of complex underlying functionality there is an increased complexity of evaluation.

Additionally CSCW systems require a further phase of usability analysis which is of a less tangible nature. Computer systems supporting collaboration must successfully address a variety of issues. They must provide a natural mode of communication between the entrants in a collaboration, they must not adversely affect collaborators' normal working practices, the costs in time and effort for using the system must be outweighed by the benefits, and there should not be additional work involved for people who do not benefit from the system's facilities. Social and psychological aspects of introducing a CSCW system into an otherwise habitual working environment must be considered. There are limited existing approaches to evaluating the collaborative aspects of computer systems. Those approaches which do exist are not sufficiently advanced as yet to be widely applicable to CSCW systems in general. A system such as MILO can be informally evaluated to ascertain the efficacy and usability of the collaborative aspects. I would suggest that further work in the area of evaluation of CSCW systems is required.

7.2.2 Trust in cooperative systems

Previous chapters have highlighted the extremes of approaches to providing support for collaborative writing in computer systems. At one extreme, inflexible systems impose roles and behaviour on authors, and at the other authors are faced with minimal constraints. It has been argued that MILO (a minimally constrained system) is more desirable because it supports a diversity of collaboration styles and document structures, is highly flexible and allows social protocols to prevail. It has also been argued that models of writing and collaboration are as yet insufficiently developed to embed them in writing support systems, but valuable support in the form of systems such as MILO can be provided *now*.

However, MILO is intended mainly for use in a collaborative environment where participants have mutually agreed to enter into the collaboration. Of course, this will not always be the case (Newman & Newman, 1993). It is possible in either an academic or industrial context that undesired collaboration need be entered into. A possible reason why a participant may not wish to enter the collaboration is because he/she does not trust or does not wish to work with other participants. In this case neither the constrained nor the minimally constrained approach appears suitable. Participants may be assigned roles at the outset, rather than selecting a role, potentially leading to conflict over perceived value of contributions, responsibilities and group hierarchy. Perceptions of collaborators may change, however, requiring a flexibility in role definition which is not present. It may be deemed desirable to promote or demote individuals during the course of the project. This is only a global view, concentrating on a individual's role in the context of the entire group. It may be the case that a participant A is vindictive and destructive towards the work of participant B but to all others appears to be a valuable contributor. In this case participant B wishes that participant A's role can be amended with respect to B's work but not with respect to the group as a whole.

Of course such behaviours and relationship will be manyfold within the group. This presents a complexity which conventional systems which impose roles cannot support; i.e. that of increasing and decreasing trust between participants. In addition to trust changing directly between collaborators, it can be affected transitively by the effect behaviour has on other participants. In the example above, participant A may become less trusted in general because of his/her attitude

to B. Trust can also be amended in a positive way during collaborations.

The approach which minimally constrains roles and behaviour is also unsuited to this type of situation. The dependence on social protocols provides the opportunity for anti-social behaviour to take place, because of its dependence on high levels of trust and cooperation. Socially mediated methods of redressing the situation such as threats or censure may be attempted but with unpredictable results.

Therefore, I suggest that research into systems between these extremes is necessary and valuable. One approach is to allow collaborators to update the system with changes in roles at any time. This poses two problems. First, relationships between group members in addition to global roles will be difficult and tedious to specify and amend. Second, collaborators must make prompt amendments in order to discourage further negative action or encourage and allow further positive action.

I would propose an investigation into the application of theories of trust in collaborative systems. Writers themselves are aware of its importance. Posner and Baecker (1993) report that in their interviews with authors about joint writing projects, the one factor which was mentioned the most often was the level of trust between participants. Lack of trust was believed to adversely affect the process.

The goal of such an investigation would be the return the work of maintaining roles and relationships on group and participant levels to systems. Such systems would be equipped to cope with changing roles and levels of trust between individuals and within the group as a whole. The system will maintain levels of trust to allow and forbid actions, and trust levels will be amended with each action. This approach allows the system's model of group relationships to change with the group. It may be necessary to indicate initial roles or trust levels but if rules about positive and negative actions have been embedded in the system, trust levels will begin to change immediately and continue to do so. Such a system does not rule out intervention by participants. For example trust in another participant may be affected by their behaviour outwith the current collaboration.

Such a flexible, secure, evolving system has further benefits. Levels of trust developed within one collaboration can be taken to other collaborations, and participants may be prepared for

collaborators who they have not previously experienced, through 'second hand' trust values. Groupings of collaborators can be made to provide inter-group, in addition to intra-group trust levels.

From this initial consideration of the subject, embedding trust in cooperative systems would appear to present a variety of exciting possibilities which merit further, detailed invesitgation.

7.3 Summary

This chapter has conluded the thesis with recommendations for future work which will extend that which is presented here. These recommendations primarily concern further development of MILO, but also address more general issues in CSCW such as evaluation and provision of more appropriate and dynamic group support.

References

Apple Computer, Inc. 1987. *Human interface guidelines: the Apple desktop interface.* Addison-Wesley.

Baecker, RM, Nastos, D, Posner, IT, & Mawby, KL. 1993. The user-centred iterative design of collaborative writing software. *Pages 399–405 of:* Ashlund, S, Mullet, K, Henderson, A, Hollnagel, E, & White, T (eds), *Human factors in computing systems. interchi '93 conference proceedings.* ACM Press/Addison-Wesley.

Beck, EE. 1993. A survey of experiences of collaborative writing. *Pages 87–112 of:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Bridwell-Bowles, L, Johnson, P, & Behe, S. 1987. Composing and computers: case studies of experienced writers. *Pages 81–107 of:* Matsuhashi, A (ed), *Writing in real time. modelling production processes.* Ablex, Norwood, New Jersey.

Bush, V. 1988. As we may think. *Pages 17–34 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

CACM. 1988 (July). *Communications of the ACM.* Hypertext issue.

Callahen, J, Hopkins, D, Weiser, M, & Shneiderman, B. 1988. An empirical comparison of pie vs linear menus. *In: Proceedings of CHI'88 conference on human factors in computing systems.* ACM Press.

Carasik, RP, & Grantham, CE. 1988. A case study of CSCW in a dispersed organisation. *Pages 61–65 of: Proceedings of CHI'88 conference on human factors in computing systems.* ACM Press.

Card, SK, Pavel, M, & Farrell, JE. 1984. Window-based computer dialogues. *In:* Shackel, B (ed), *Human computer interaction - interact 84.*

Catlin, T, Bush, P, & Yankelovich, N. 1989. InterNote: extending a hypermedia framework to support annotative collaboration. *Pages 365–378 of: Hypertext '89 Proceedings.* Pittsburgh, Pennsylvania November 1989. ACM Press.

Chen, M. 1993. *An exploratory study of merge functions for collaborative writing systems.* MSc. thesis, University of Stirling, Stirling, Scotland.

Cockburn, AJG. 1993.. *Groupware design: principles, prototypes, and systems.* Ph.D. thesis, University of Stirling, Stirling, Scotland.

Cockburn, AJG, & Jones, SRA. 1991. *Four principles for groupware design: encouraging adoption and easing system use.* Presentation at CSCW SIG Workshop on Implementation perspectives on CSCW design.

Cockburn, AJG, & Thimbleby, H. 1991. A Reflexive Perspective of CSCW. *SIGCHI Bulletin*, **23**(3), 63–68.

Cockburn, AJG, & Thimbleby, HW. 1993. Reducing user effort in collaboration support. *Pages 215–218 of:* Gray, WD, Hefley, WE, & Murray, D (eds), *Proceedings of the 1993 international workshop on intelligent user interfaces, orlando, florida. january 4–7.* New York: ACM Press.

Collins, A, & Gentner, D. 1980. A framework for a cognitive theory of writing. *Pages 51–72 of:* Gregg, LW, & Steinberg, ER (eds), *Cognitive processes in writing.* Lawrence Erlbaum Associates.

Cookson, S. 1989. Designing computational writing tools within a linguistic model of the writing process. *Pages 17–21 of:* Williams, N, & Holt, P (eds), *Computers and writing: models and tools.* Intellect.

Coombs, J, Renear, AH, & DeRose, SJ. 1987. Markup systems and the future of scholarly text processing. *Communications of the ACM*, **30**(11), 933–947.

Corbett, EPJ. 1981. The status of writing in our society. *Pages 47–52 of:* Whiteman, M (ed), *Writing: the nature, development, and teaching of written communication. Volume 1 Variation in writing: functional and linguistic-cultural differences.* Lawrence Erlbaum Associates, Inc.

Couture, B, & Rymer, J. 1991. Discourse interaction between writer and supervisor: a primary collaboration in workplace writing. *In:* Lay, MM, & Karis, WM (eds), *Collaborative writing in industry: Investigations in theory and practice.* Beywood, Amityville, NY.

Daiute, C. 1985. *Writing and computers.* Addison-Wesley.

Dix, A, Finlay, J, Abowd, G, & Beale, R. 1993. *Human-computer interaction.* Prentice-Hall.

Dykstra, EA, & Carasik, RP. 1991. Structure and support in cooperative environments: the Amsterdam Conversation Environment. *International journal of man-machine studies*, **34**, 419–434.

Eco, U. 1989. *Foucault's Pendulum.* Martin Secker & Warburg Limited.

Ede, L, & Lunsford, A. 1990. *Singular texts/plural authors: perspectives on collaborative writing.* Southern Illinois University Press, Carbondale.

Ellis, CA, Gibbs, SJ, & Rein, GL. 1991a. Groupware: some issues and experiences. *Communications of the ACM*, **34**(1), 38–58.

Ellis, CA, Gibbs, SJ, & Rein, GL. 1991b. Groupware. some issues and experiences. *Communications of the ACM*, **34**(1), 38–58.

Engelbart, DC. 1988. Authorship provisions in AUGMENT. *Pages 107–126 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Fish, RS, Kraut, RE, Root, RW, & Rice, RE. 1992. Evaluating video as a technology for informal communication. *Pages 37–48 of: Proceedings of CHI '92* Monterey, California, May, 1992). ACM, New York.

Flower, L, & Hayes, JR. 1981. Plans that guide the composing process. *Pages 39–58 of:* Frederiksen, CH, & Dominic, JF (eds), *Writing: the nature, development, and teaching of written communication. Volume 2 Writing: process, development and communication.* Lawrence Erlbaum Associates, Inc.

Flower, LS, & Hayes, JR. 1980. The dynamics of composing: making plans and juggling constraints. *Pages 31–50 of:* Gregg, LW, & Steinberg, ER (eds), *Cognitive processes in writing.* Lawrence Erlbaum Associates.

Francik, E, & Akagi, K. 1988. *Designing a computer pencil and tablet for handwriting.* Tech. rept. Wang Laboratories, Inc. Lowell, MA, USA.

Frederiksen, CH, & Dominic, JF. 1981. Perspectives on the activity of writing. *Pages 1–20 of:* Frederiksen, CH, & Dominic, JF (eds), *Writing: the nature, development, and teaching of written communication. Volume 2 Writing: process, development and communication.* Lawrence Erlbaum Associates, Inc.

Furnas, GW. 1986. Generalized fisheye views. *In: Human factors in computing systems III. Proceedings of the CHI'86 conference.*

Furuta, R, Scofield, J, & Shaw, A. 1982. Document formatting systems: survey, concepts and issues. *ACM Computing Surveys*, **14**(3), 417–472.

Greenbaum, J. 1988. In search of cooperation: an historical analysis of work organization and management systems. *Pages 102–114 of: Proceedings of the 2nd conference on computer supported cooperative work (CSCW '88). Portland, Oregon. September 1988.*

Greenberg, S. 1991. An annotated bibliography of computer supported collaboratvie work. *SIGCHI Bulletin*, **23**(3), 29–76.

Greif, I, & Sarin, S. 1987. Data sharing in group work. *Acm transactions on office information systems*, **5**(2), 187–211.

Grudin, J. 1988. Why CSCW applications fail: problems in the design and evaluation of organizational interfaces. *Pages 85–93 of: Proceedings of the 2nd conference on computer supported cooperative work (cscw '88).* portland, oregon. september 1988.

Grudin, J. 1994. CSCW: History and focus. *Computer*, **27**(5), 19–26.

Haake, A, & Haake, JM. 1993. Take cover: Exploiting version support in cooperative systems. *Pages 406–413 of:* Ashlund, S, Mullet, K, Henderson, A, Hollnagel, E, & White, T (eds), *Human factors in computing systems. interchi '93 conference proceedings.* ACM Press/Addison-Wesley.

Haas, C, & Hayes, JR. 1986. What did I just say? reading problems in writing with the machine. *Research in the teaching of english*, **20**(1).

Hartley, J, & Branthwaite, A. 1989. The psychologist as wordsmith: a questionnaire study of writing strategies of productive British psychologists. *Higher education*, 18, 423–452.

Hayes, JR, & Flower, LS. 1980. Identifying the organization of writing procecesses. *Pages 3–30 of:* Gregg, LW, & Steinberg, ER (eds), *Cognitive processes in writing.* Lawrence Erlbaum Associates.

Hewlett-Packard Company. 1988. *Programming with the HP X Widgets and the Xt Intrinsics.* Hewlett-Packard Company.

Holt, P. 1989. Models of writing: a question of interaction. *Pages 50–60 of:* Williams, N, & Holt, P (eds), *Computers and writing: models and tools.* Intellect.

Hsiao, CC, & Levine, SR. 1988. *Voice annotation on Wang Freestyle System.* Tech. rept. Wang Laboratories, Inc. Lowell, MA, USA.

Jones, S. 1990 (September). *A discussion of issues and systems relevant to computer supported cooperative work.* Technical report. Department of Computing Science and Mathematics, University of Stirling, Stirling, Scotland. T.R. 64.

Jones, S. 1991 (November). *MILO: a computer based tool for (co)authoring structured documents.* Technical report. Department of Computing Science and Mathematics, University of Stirling, Stirling, Scotland. T.R. 81.

Jones, S. 1992a. Designing computer systems to support collaborating authors. *Pages 134–141 of:* Rees, MJ, & Iannella, R (eds), *OZCHI'92, Interface Technology: Advancing Human-Computer Communication.* Ergonomics Society of Australia.

Jones, S. 1992b. MILO. *ESRC data archive bulletin*, October, Software Bulletin, S1–S3.

Jones, S. 1993. MILO: a computer based tool for (co)authoring structured documents. *Pages 185–202 of:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Jones, S, & Cockburn, A. 1993. *Reducing requirements in computer supported writing tasks.* Presentation at the 6th UK Conference on Computers and Writing. University of Wales, April 13th–15th, 1993.

Kane, T. 1993. Personal communication.

Kellogg, RT. 1989. Attentional overload and writing performance: effects of rough draft and outline strategies. *Journal of experimental psychology: learning, memory and cognition*, 14(2), 355–365.

Kiesler, S, Siegel, J, & McGuire, TW. 1988. Social psychological aspects of computer-mediated communication. *Pages 657–682 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Kraut, R, & Egido, C. 1988. Patterns of contact and communication in scientific research collaborations. *Pages 1–12 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Kraut, R, Galegher, J, & Egido, C. 1988. Relationships and tasks in scientific research collaborations. *Pages 741–769 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Lamport, L. 1986. LaTeX*: a document preparation system.* Addison-Wesley.

Leland, MDP, Fish, RS, & Kraut, RE. 1988. Collaborative document production using Quilt. *Pages 206–215 of: Proceedings of the 2nd conference on computer supported cooperative work (CSCW '88).* portland, oregon. september 1988.

Mackay, WE. 1988. More than just a communication system: diversity in the use of electronic mail. *Pages 344–353 of: Proceedings of the 2nd conference on Computer Supported Cooperative Work (CSCW '88).* Portland, Oregon. September 1988.

Malone, TW, Grant, KR, & Turbak, FA. 1986. The Information Lens: and intelligent system for information sharing in organizations. *In: Proceedings of the chi '86 conference on human factors in information systems.* ACM Press New York.

Malone, TW, Grant, KR, Lai, K-Y, Rao, R, & Rosenblatt, D. 1988. Semistructured messages are surprisingly useful for computer-supported coordination. *Pages 311–331 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Matsuhashi, A. 1987. Revising the plan and altering the text. *Pages 197-223 of:* Matsuhashi, A (ed), *Writing in real time. modelling production processes.* Ablex, Norwood, New Jersey.

Mayes, JT, Draper, SW, McGregor, AM, & Oatley, K. 1988. Information flow in a user interface : the effect of experience and context on the recall of MacWrite screens. *In:* Jones, DM, & Winder, R (eds), *People and Computers IV. HCI '88.* CUP.

Meyrowitz, N, & Van Dam, A. 1982a. Interactive editing systems: part I. *ACM Computing Surveys*, 14(3), 321–352.

Meyrowitz, N, & Van Dam, A. 1982b. Interactive editing systems: part II. *ACM Computing Surveys*, 14(3), 353–409.

Miles, VC, McCarthy, JC, Dix, AJ, Harrison, MD, & Monk, AF. 1993. Reviewing designs for a synchronous-asynchronous group editing environment. *Pages 137-160 of:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Monk, A. 1988. *Getting to known location in a hypertext.* Tech. rept. Dept. of Psychology, University of York, England.

Neuwirth, C, Kaufer, D, Chimera, R, & Gillespie, T. 1988 (March). The Notes program: a hypertext application for writing from source texts. *Pages 121-141 of: Hypertext '87, University of North Carolina, Chapel Hill, North Carolina, November 13-15 1987.*

Neuwirth, CM, Kaufer, DS, Chandhock, R, & Morris, JH. 1993. Issues in the design of computer support for co-authoring and commenting. *Pages 537-549 of:* Baecker, RM (ed), *Readings in groupware and computer supported cooperative work: assisting human-human collaboration.* Morgan Kaufmann.

Newman, R, & Newman, J. 1993. Social writing: premisses and practices in computerised contexts. *Pages 29-40 of:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Nielsen, J. 1988 (July). *Evaluating the thinking aloud technique for use by computer scientists.* Paper presented at IFIP Working Group 8.1 International Workshop on Human Factors of Information Systems Analysis and Design. London July 28–29.

Nielsen, J. 1989 (June). Prototyping user interfaces using and object oriented hypertext programming system. *In: Proceedings of NordDATA '89 joint Scandinavian computer conference, Denmark.*

Nielsen, J, & Molich, R. 1990. Heuristic evaluation of user interfaces. *Pages 249-256 of: Proceedings of CHI'90 conference on human factors in computing systems.* ACM Press.

Nold, EW. 1981. Revising. *Pages 67-79 of:* Frederiksen, CH, & Dominic, JF (eds), *Writing: the nature, development, and teaching of written communication. Volume 2 Writing: process, development and communication.* Lawrence Erlbaum Associates, Inc.

Olon, JS, Card, SK, Landauer, TK, Olson, GM, Malone, T, & Legget, J. 1993. Computer supported co-operative work: research issues for the 90s. *Behaviour & information technology*, 12(2), 115–129.

Pliskin, N. 1989. Interacting with electronic mail can be a dream or nightmare: a user's point of view. *Interacting with computers: the interdisciplinary journal of human-computer interaction*, 1(3), 259–272.

Posner, IR, & Baecker, RM. 1993. How people write together. *Pages 239–250 of:* Baecker, RM (ed), *Readings in groupware and computer supported cooperative work: assisting human-human collaboration.* Morgan Kaufmann.

Root, RW. 1988. Design of a multi-media vehicle for social browsing. *Pages 25–38 of: Proceedings of the 2nd conference on Computer Supported Cooperative Work (CSCW '88). Portland, Oregon. September 1988.*

Scheifler, RW, & Gettys, J. 1986. The X Window System. *ACM transactions on graphics*, **5**(2), 79–109.

Sharples, M, & Pemberton, L. 1990. Starting from the writer: guidelines for the design of user-centred document processors. *Computer assisted language learning*, **2**, 37–57.

Sharples, M, Goodlet, J, & Pemberton, L. 1989. Developing a writer's assistant. *Pages 22–37 of:* Williams, N, & Holt, P (eds), *Computers and writing: models and tools.* Intellect.

Sharples, M, Goodlet, J, Beck, E, Wood, C, Easterbrook, S, Plowman, L, & Evans, W. 1993. A framework for the study of computer supported collaborative writing. *In:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Shipman III, FM, Chaney, RJ, & Gorry, GA. 1989. Distributed hypertext for collaborative research: the virtual notebook system. *In: Hypertext '89 proceedings. Pittsburgh, Pennsylvania November 1989.* ACM Press.

Shneiderman, B. 1983. *Designing the user interface: strategies for effective human computer interaction.* Addison-Wesley.

Shneiderman, B. 1987. Direct manipulation : a step beyond programming languages (excerpt). *In:* Baecker, RM, & Buxton, WAS (eds), *Readings in human-computer interaction: A multidisciplinary approach.* Morgan Kaufmann.

Sommerville, I. 1992. *Software engineering.* Addison-Wesley.

Stefik, M, Bobrow, DG, Foster, G, Lanning, S, & Tatar, D. 1987. WYSISIS revised: early experiences with multiuser interfaces. *ACM transactions on office information systems*, **5**(2), 147–167.

Stefik, M, Foster, G, Bobrow, DG, Kahn, K, Lanning, S, & Suchman, L. 1988. Beyond the chalkboard: computer-support for collaboration and problem solving in meetings. *Pages 335–366 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Subramanyam, K. 1983. Bibliometric studies of research collaboration. *Journal of information science*, **6**(1), 33–38.

Tatar, DG, Foster, G, & Bobrow, DG. 1991. Design for conversation: lessons from Cognoter. *International journal of man-machine studies*, **34**, 185–209.

Thimbleby, H. 1990. *User interface design.* ACM Press.

Thimbleby, H, Jones, S, & Cockburn, A. 1992. Hypercard: An object oriented disappointment. *In:* Gray, PD, & Took, R (eds), *Building interactive systems: Architectures and tools.* Springer-Verlag.

Thomas, C. 1987. Designing electronic paper to fit user requirements. *In:* Diaper, D, & Winder, R (eds), *People and Computers III. Proceedings of the 3rd conference of the BCS HCI SG. University of Exeter, September 1987.*

Thomas, RH, Forsdick, HC, Crowley, TR, Schaaf, RW, Tomlinson, RS, Travers, VM, & Robertson, GG. 1988. Diamond: a multimedia message system built on a distributed architecture. *Pages 509–532 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Trigg, RH. 1988. Guided tours and tabletops: tools for communicating in hypertext environment. *Pages 216–226 of: Proceedings of the 2nd conference on computer supported cooperative work (CSCW '88).* Portland, Oregon. September 1988.

Trigg, RH, & Irish, PM. 1988 (March). Hypertext habitats: experiences of writers in NoteCards. *Pages 89–108 of: Hypertext '87, University of North Carolina, Chapel Hill, North Carolina, November 13–15 1987.*

Trigg, RH, & Suchman, LM. 1988. *Collaborative writing in NoteCards.* Paper delivered at Hypertext theory into practice, HyperText I. March 1988, University of Aberdeen, Scotland.

Wason, PC. 1980. Specific thought on the writing process. *Pages 129–137 of:* Gregg, LW, & Steinberg, ER (eds), *Cognitive processes in writing.* Lawrence Erlbaum Associates.

Whiteside, J, Jones, S, Levy, P, & Wixon, D. 1985. User performance with command, menu and iconic interfaces. *In: Human factors in computing systems II. Proceedings of the CHI'85 conference.*

Williams, N. 1990. Writers' problems and computer solutions. *Computer assisted language learning*, **2**, 5–25.

Witten, IH, Thimbleby, HW, Coulouris, G, & Greenberg, S. 1991. Liveware: a new approach to sharing data in social networks. *International journal of man-machine studies*, **34**(3), 337–348.

Appendix A

A User's Guide To MILO

Contents

List of Figures

A.1 Introduction

This document describes the use of MILO, a computer program which supports authoring of documents on both a personal and collaborative basis.

Documents created by MILO have a structure which you, the author, impose. You do not have to make decisions about document type or formatting until you are happy with both structure and content.

MILO provides a wide range of powerful facilities to view, manipulate and alter documents for a single author. Little additional effort is required to use MILO in a collaborative authoring task.

All of MILO's facilities are accessed via a graphical, window based user interface which is described in later sections. To exploit this interface you should use MILO on hardware which supports the X Window System.

This document is task oriented rather than presenting a list of the systems facilities and how they are accessed. It is intended as a reference which decribes how to carry out individual tasks, such as reading an existing file into MILO, or creating a new MILO document. It can be read from beginning to end leading you through a MILO session, and also via the index as a reference to refresh your memory.

Let us begin.

A.2 Getting started

You can start MILO by simply typing

`milo` ⟨*Return*⟩

You will then be required to position a window on the screen. After a brief pause, you will see a window outline appear on the screen. Move the mouse (or pointing device) to position this window where you want it to be, and then click the leftmost mouse button. A window will now appear in full and is the main MILO window from which the majority of your interactions with the system will be driven.

Figure A.1 shows the initial form of this window.

A.2.1 The main MILO window

The main MILO window consists of four areas

- the title bar which contains the name of the current document (in the form of a full path name). The title bar will initially contain the text – **No Current File** – as no file has as yet been read into or saved from MILO;
- the main command menu bar which is positioned below the title bar. This contains four main menu headers: **Document, Notes, Views** and **Utilities.** To select elements from these menus first place the mouse pointer over the chosen menu header. Then press the leftmost mouse button; you will see a menu of operations appearing on the screen. This menu appears only for the time that the leftmost mouse button is depressed. Whilst keeping the mouse button depressed move the mouse pointer over the menu elements. Elements become highlighted as the pointer passes over them. To select an element release the mouse button while the element is highlighted;
- the hierarchical structure overview which is positioned below the title bar. This will reflect the hiearchical structure of elements in the current document in a graphical manner. When MILO is first started up, this part of the window is empty, as in Figure A.1;

- the MILO message area in which useful information will appear. These might be error messages or reports on MILO activity. A small square at the top right of the message area separates it from the hierarchical overview. On dragging[1] this upwards with the left mouse button the message area is revealed.

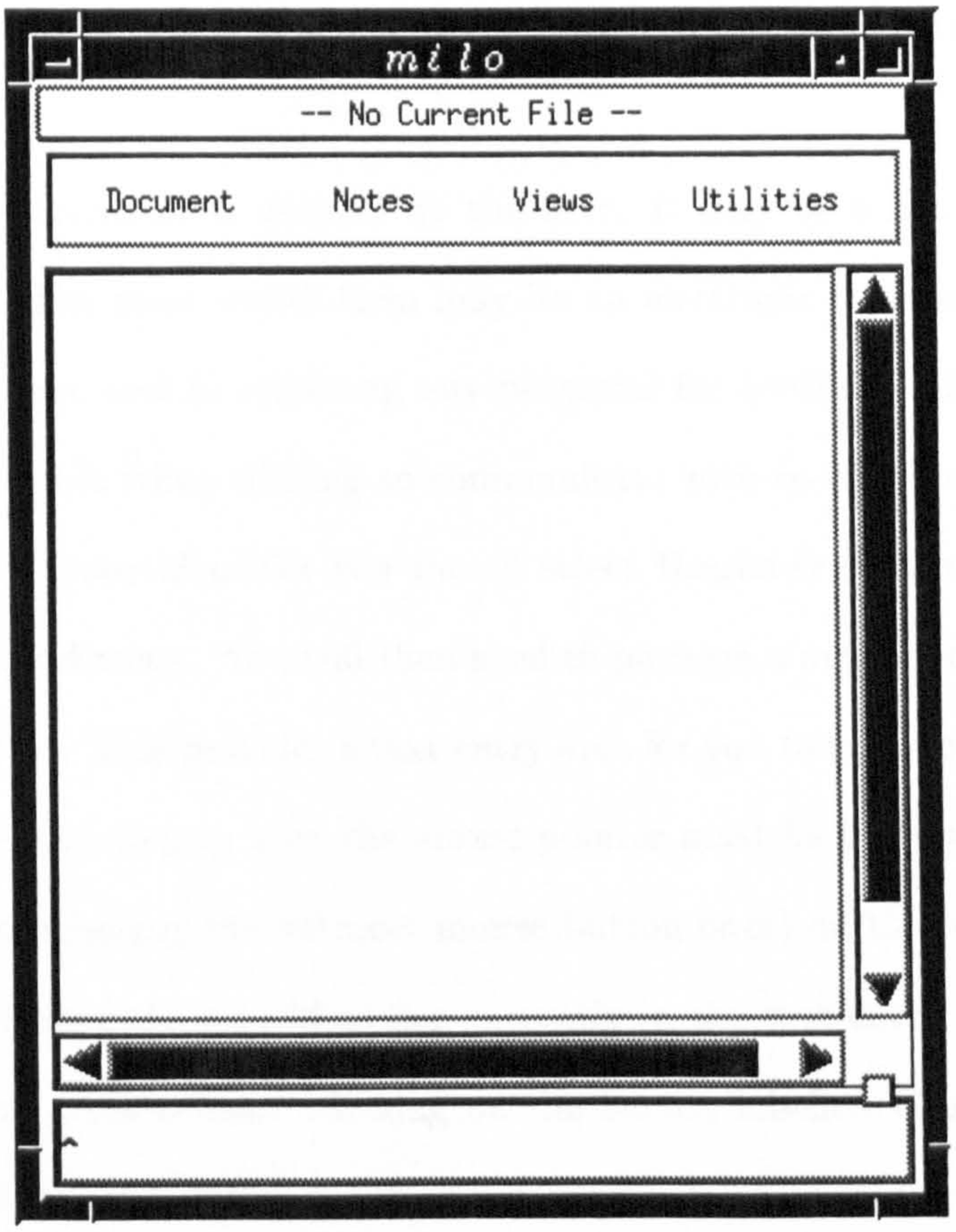


Figure A.1: The initial state of the main MILO window just after the system has been started up.

[1] 'dragging' means depressing a mouse button whilst the pointer is within the object, and then moving the mouse pointer whilst the mouse button remains depressed. To end the dragging action, release the mouse button.

A.3 Registering a new user identifier

When reading or amending a document which has been created using MILO, it may be useful to see who was responsible for creating or amending specific elements of it. This is made possible by the fact that a user can register their own individual identifier with MILO at any point during a MILO session. This identifier is then associated with creation and alteration of document elements.

The form of a user identifier is decided by the user. It may be a full name, first name or surname for example. The most useful form may be an electronic mail address. This has the advantage of being unique and so removing any potential for confusion about author identity, and is also readily available when wishing to communicate with co-authors (see Section A.10).

In order to register a user identifier you should select **Register new user identifier** from the **Document** command menu. You will then need to position a new window on the screen (as described in Section A.2). This provides a text entry area for you to type your identifier, and two buttons. To type into a text entry area the mouse pointer must be positioned in the text area. Clicking (pressing and releasing the leftmost mouse button once) on the button labelled **Click when correct** will register the user identifier currently in the text entry area with MILO and remove the window from the screen. Clicking on the button labelled **Cancel** will not register an identifier with MILO and will remove the window from the screen.

If notes are created or amended before a user identifier is registered, then the author or amender is recorded as **unknown**.

Figure A.2 shows this dialogue.

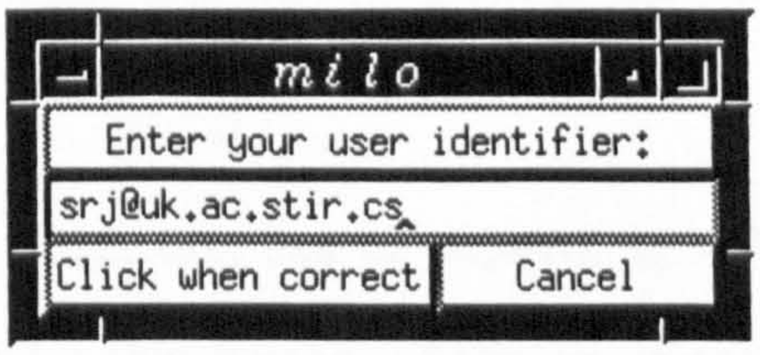


Figure A.2: The dialogue window into which a new user identifier is entered.

A.4 Creating a new document

To initiate the process of creating a new document select **Create new document** from the **Document** command menu. Two things will then happen

1. a small rectangle will appear in the hierarchical structure overview which represents the first element in the new document. This displays the title of the corresponding note. At the moment it will contain no text as the new element has not yet been given a title.

2. you will be prompted to place a new window on the screen. This is the first document element (called a *note*), and you can enter text and graphics into it. A MILO document will consist of one or more notes.

A.4.1 What does a note consist of?

A note consists of four areas

- the note command menu bar. This contains the menu headers **Note functions, Utilities** and **Canvas functions,** and these pulldown menus are used in the same way as those in the main MILO window;

- the title bar which is an area into which text can be entered, and is intended to contain the title of this element of the document. The text which appears in this area also appears in the corresponding elements of the hierachical overview. This area is fully editable, and when amended the corresponding element of the hierarchical is immediately updated. Try typing something into the title bar now (remember that the mouse pointer must be within it to do so);

- the main text area into which text can be entered. It is fully editable and is intended to contain the body of text for this element of the document;

- a graphics editor into which objects such as lines, circles, rectangles and text can be drawn. This area is not initially visible.

Figure A.3 shows the initial appearance of a note.

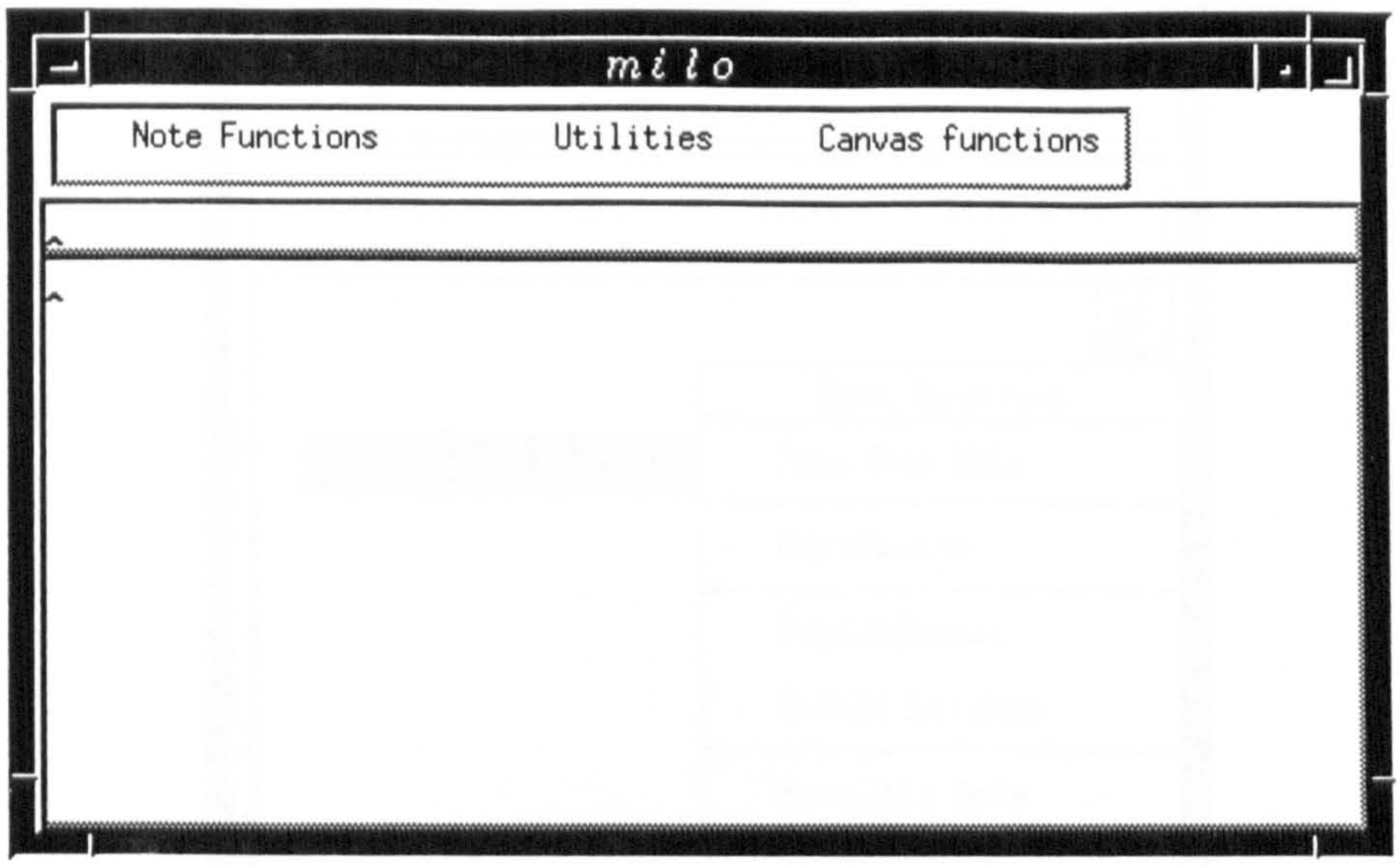


Figure A.3: A newly created note, with menu headers, title bar and main text area.

A.4.2 Adding notes to the document structure

A new note can be added to the document at any time. This action can be invoked in any of three ways, but you are always required to indicate to which existing document element a new sub element should be added. The three ways to add a new note are

- via the pulldown menu attached to each element of the hierarchical overview;
- via the pulldown menu attached to each note window;
- via the pulldown menu attached to each element of the linear overview (see Section A.8.3).

Each of these menus is accessed by placing the pointer within the object of interest, and depressing the leftmost mouse button. They can be seen in Figures A.4, A.5 and A.11.

They are very similar, and can be accessed from each element of the two overviews and from each note window. The functions within the menus, however, *are specific to the document element from which the menu was accessed* and are invoked by releasing the mouse button when the pointer is above the required function (which will be highlighted).

So, to add a new note to the document select the **Add subnote** function from a pulldown menu attached to an element within either of the overviews, or to a note window. The hierarchical and linear overviews will be immediately updated to include the new addition to the document. Try adding several more notes to your document.

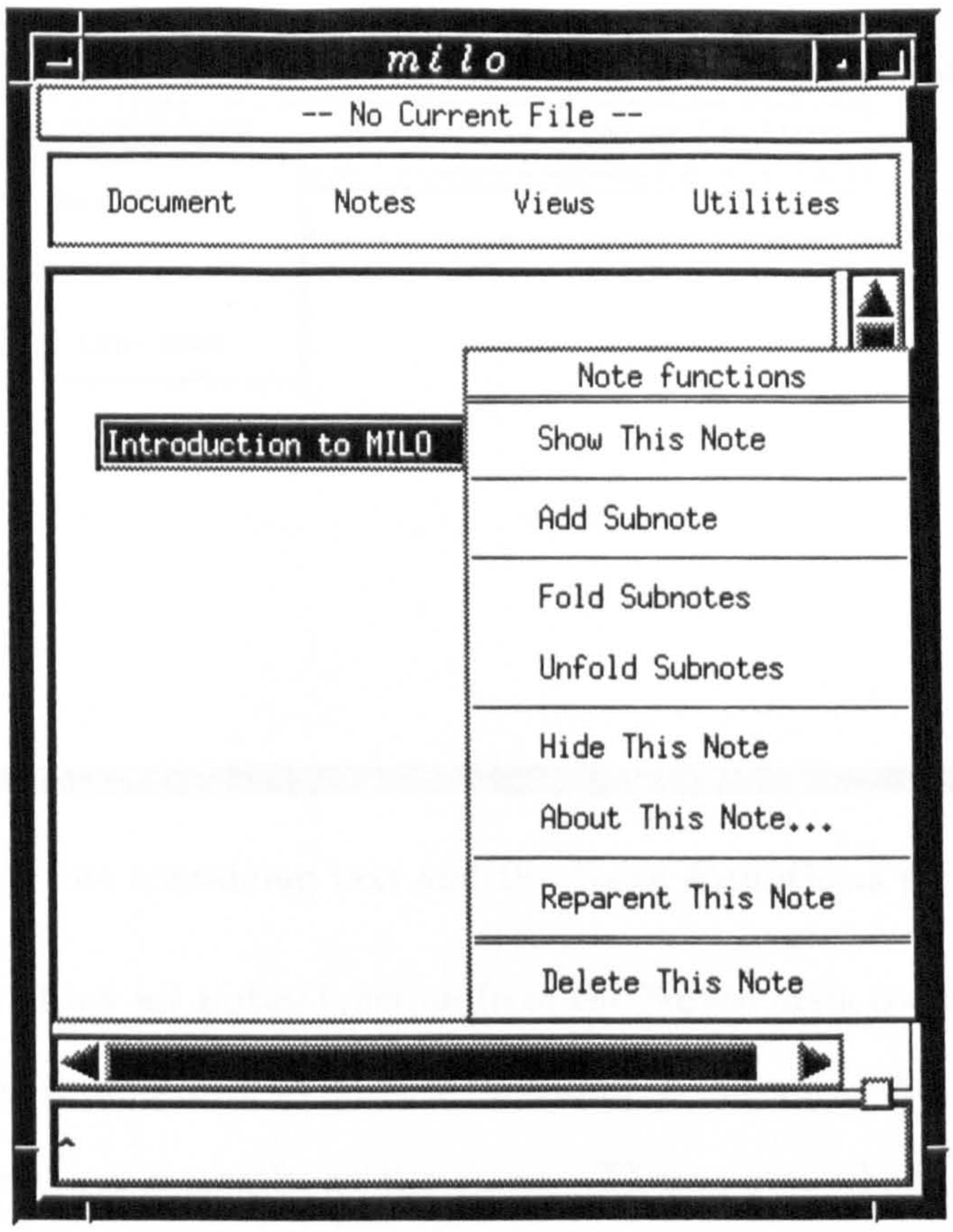


Figure A.4: An element in the hierarchical overview which represents a note in the document. The pulldown menu is accessed by pressing the leftmost mouse button within the element in the overview. The entries in the menu are specific to that element.

A.4.3 Adding notes *outside* the document structure

Of course, you may not always know at what position in the document structure you want to add a document element. You may wish to defer the decision until later. MILO allows you to to do this, creating as many notes as you wish which are not integrated into the structure.

To create this type of note select **Add unattached note** from the **Notes** command menu. Unattached notes are like ordinary notes in every respect; they can have subnotes, be moved about and deleted. They can be integrated into the main document structure at any time.

Elements representing unattached notes in the hierarchical overview appear in a row at the top of the overview. Try adding some unattached notes.

A.4.4 Accessing notes

Notes can be accessed for reading or amending in any of four ways

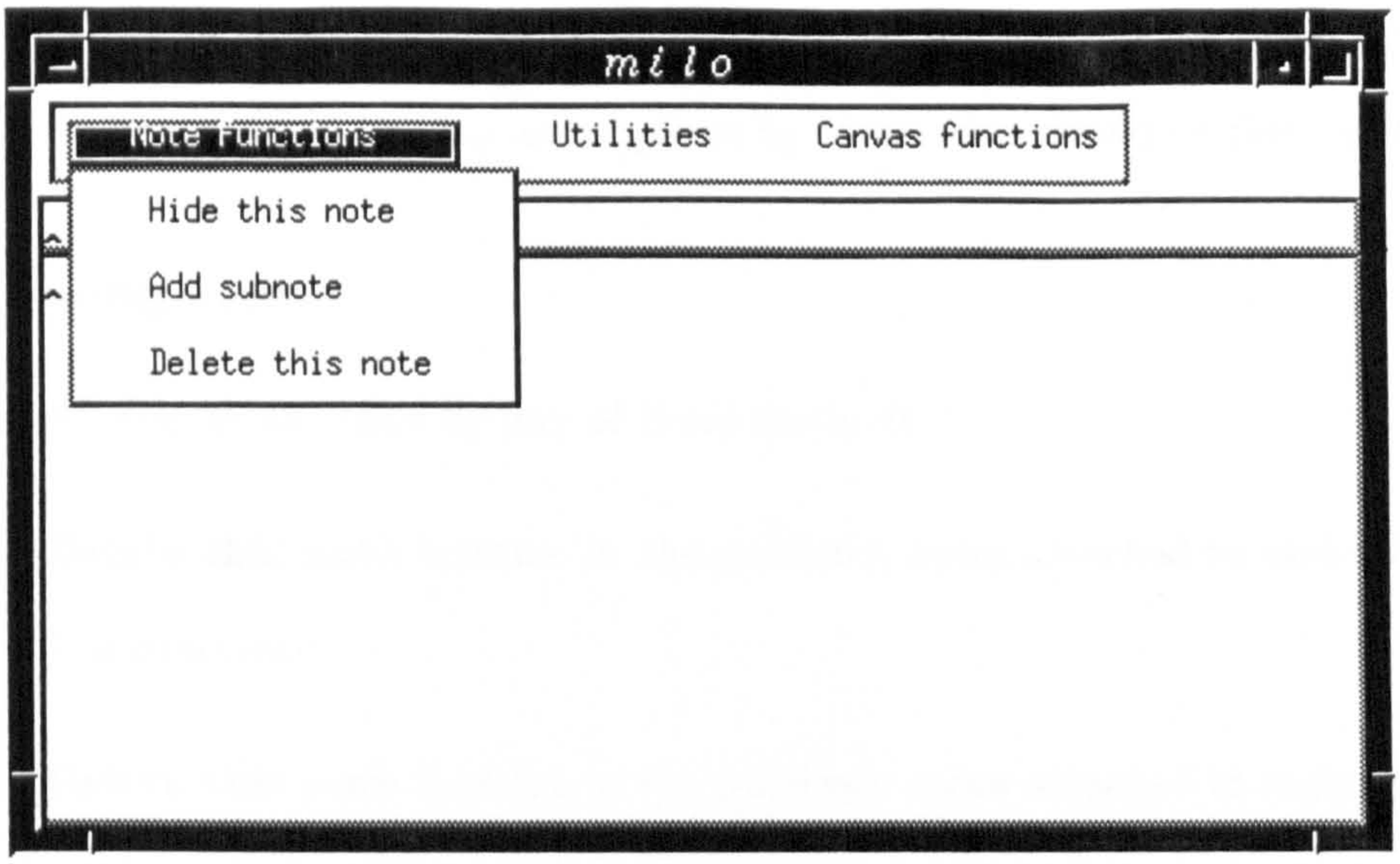


Figure A.5: A note containing text and the **Note Functions** pulldown menu.

- by selecting the **Show all notes** function from the **Notes** main command menu. This will result in you being prompted to position windows for all notes that make up the current document and are not currently on the screen. Those notes which are on the screen will be made visible;

- by selecting **Show this note** from the pulldown menu attached to the element in the hierarchical overview which corresponds to the note of interest. This will result in you being prompted to position a window for the note of interest if it is not currently on the screen. If it is on the screen it will be made visible;

- by selecting **Show this note** from the pulldown menu attached to the element in the linear overview (see Section A.8.3) which corresponds to the note of interest. This will also result in you being prompted to position a window for the note of interest if it is not currently on the screen. If it is on the screen it will be made visible;

- by selecting **Show this note** from the pulldown menu attached to the element in the time ordered view of notes (see Section A.9.3) which corresponds to the note of interest. This will also result in you being prompted to position a window for the note of interest if it is not currently on the screen. If it is on the screen it will be made visible.

For a document containing many notes the first method will be slow and tedious, and it will be more efficient to access notes as they are required by either the second or third method.

A.4.5 Deleting notes

Notes can be deleted at any time by any of three methods

- via the **Delete this note** function in the pulldown menu attached to each element of the hierarchical overview;
- via the **Delete this note** function in the pulldown menu attached to each element of the linear overview;
- via the **Delete this note** function in the pulldown menu attached to each note window.

The deletion applies to the document element from which the function was invoked, and the hierarchical and linear overviews are immediately updated to reflect the deletion. The note window is removed from the screen.

If a note has subnotes and is deleted its subnotes are also deleted.

A.4.6 Undoing deletions

Deleted notes can be recovered easily regardless of whether other deletions have taken place since.

One of the functions in the **Notes** main command menu is labelled **Undo most recent deletion.** Once a note has been deleted this command will become **Replace deleted note:** followed by the name of the most recently deleted note. Selecting this function at this point will replace the deleted note in the document and the function will be labelled **No deletions to undo.**

If, however, note *a* was deleted followed by note *b* the function would be labelled **Replace deleted note: a** and then **Replace deleted note: b.** Selecting this function at this point will replace deleted note *b* in the document and the function will be labelled **Replace deleted note: a.** Selecting this function at this point will replace the deleted note *a* in the document and the function will be labelled **No deletions to undo.**

You can 'backtrack' through any number of deletions in this manner, and each time a deleted note is replaced the hierarchical and linear overviews are immediately update to reflect the change.

A.4.7 Altering the position of notes in the document

During the process of creating and amending a document you may wish to alter the position of document elements within the structural hierarchy. This can be done easily at any time by either of two methods:

1. By indicating directly with the mouse the new position of the note to be moved. This is done by pressing the *rightmost* mouse button in the element in the hierarchical overview corresponding to the note to be moved. While still holding down the mouse button move the pointer to within the element in the hierarchical overview which the moved note is to become a subnote of. Then release the mouse button. The hierarchical overview is immediately updated.

   Additionally you can reposition a note between other notes by carrying out the action described above, and releasing the mouse button when the pointer is between elements of the hierarchical overview.

   Try to move notes about in the structure in both of these ways.

2. By specifying the two notes which are involved in the operation. The first note which you specify is the one which is to be moved. The second note is the one for which the first is to become a subnote. The first note is indicated by any of three methods

   - via the **Reparent This Note** function in the pulldown menu attached to each element of the hierarchical overview;
   - via the **Reparent This Note** function in the pulldown menu attached to each element of the linear overview;
   - via the **Reparent This Note** function in the pulldown menu attached to each note window.

The appropriate note is then registered as the one to be moved. The note which it is about to become a subnote of is then indicated by clicking the left mouse button in the title bar of its note window. Each of the overviews is then updated to reflect the new structure of the document.

If the repositioned note has subnotes, those subnotes will remain its subnotes and therefore also be repositioned in the document structure.

A.4.8 Entering text

Every text entry area in MILO (note titles, note text, dialogues) is an individual text editor. Each behaves in virtually the same manner (see Section A.14 for a list of editor commands). To enter text into one of these areas, place the mouse pointer within it and begin typing. This text is fully editable.

The single line text editors in MILO will not allow commands which would take the insertion point or extend the text beyond a single line.

A.4.9 Copying and pasting text

Text can be copied and pasted between textual objects within MILO, and also between MILO and textual objects outwith it.

To select text to be copied you should use the mouse. This requires you to drag with the left mouse button across the range of text to be selected. Once you release the mouse button the currently highlighted text is that which will be copied (see Figure A.6). To paste this text into another location (either within or outwith MILO) move the mouse pointer to the required window and position the insertion point at the appropriate place. Then click the middle mouse button. The previously selected text will be inserted.

Text can be copied and pasted between MILO and any other X Window System based application.

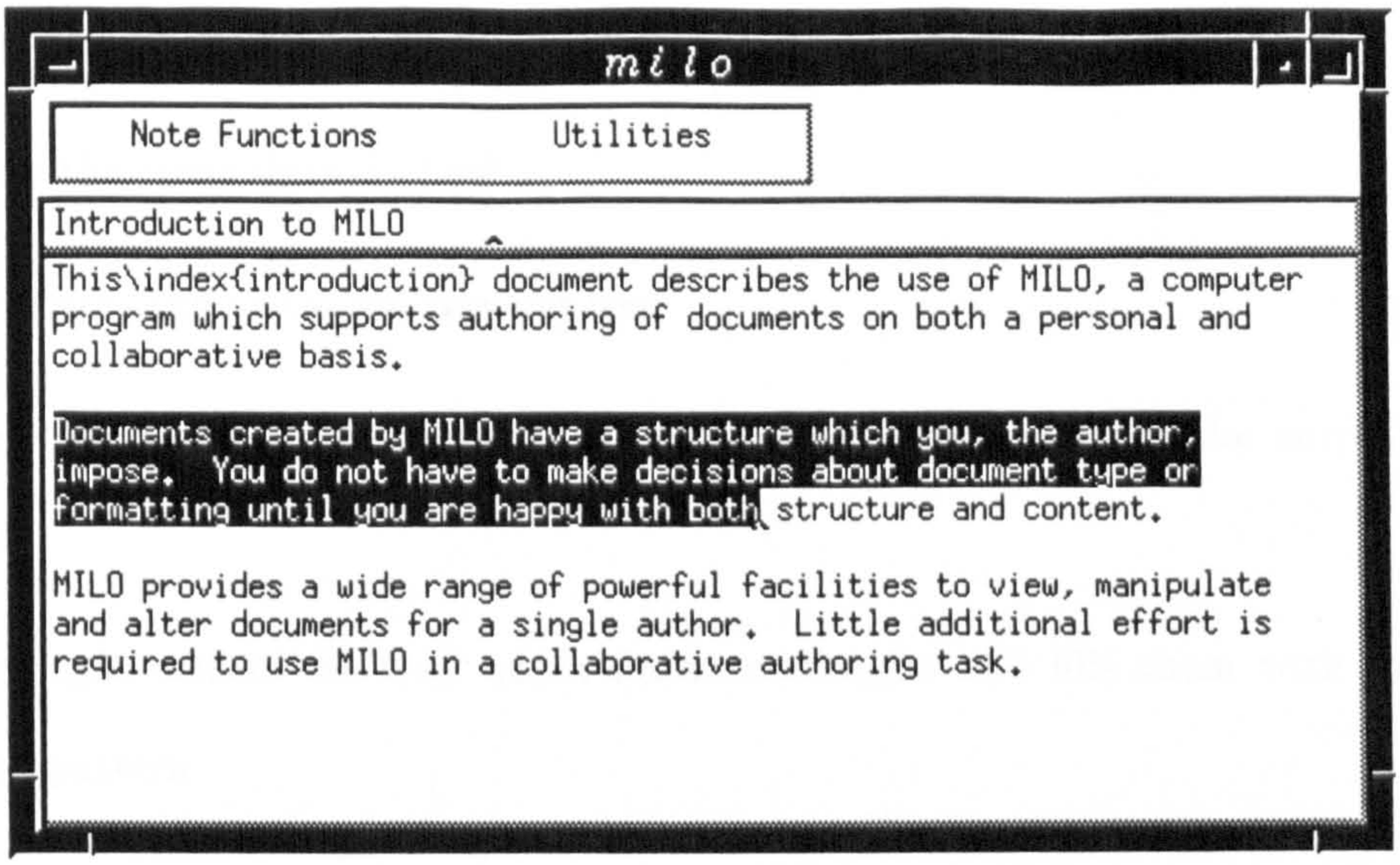


Figure A.6: A note containing selected text, which is highlighted. Text is select by 'dragging' the mouse over the required text. Selected text can be pasted into other notes or windows.

A.4.10 Entering graphics

Each note within MILO has an attached graphics editor. This allows document elements to consist of both text and graphics. A note's graphics editor is not initially visible—it is hidden below the note's text area—but it can be easily made visible by selecting the **Raise canvas** function from the **Utilities** pulldown menu within the note window.

Once this has been done the note text will disappear and a blank canvas will appear with scrollbars down the right hand side and along the bottom. The graphics commands are accessed via the pulldown menu **Canvas functions** which is in the note window.

This menu contains two entries

- **Functions →**
- **Patterns →**

The arrows indicate that each of these menu items has an associated submenu. A submenu is accessed by moving the pointer over the arrow while still holding down the left mouse button.

The **Functions** submenu contains several items

Refresh Canvas selecting this item will redraw all objects on the canvas should it become

corrupted

Line selects the line drawing tool

Circle selects the circle drawing tool

Rectangle selects the rectangle drawing tool

Filled Circle selects the tool which draws circles and fills them with the currently selected pattern

Filled Rectangle selects the tool which draws rectangles and fills them with the currently selected pattern

Draw Text provides a dialogue into which the text to be placed on the canvas should be entered. To place the text on the canvas click at the appropriate point

Erase removes all objects from the canvas

Figure A.7 shows this menu. The **Patterns** submenu contains a choice of patterns to be used

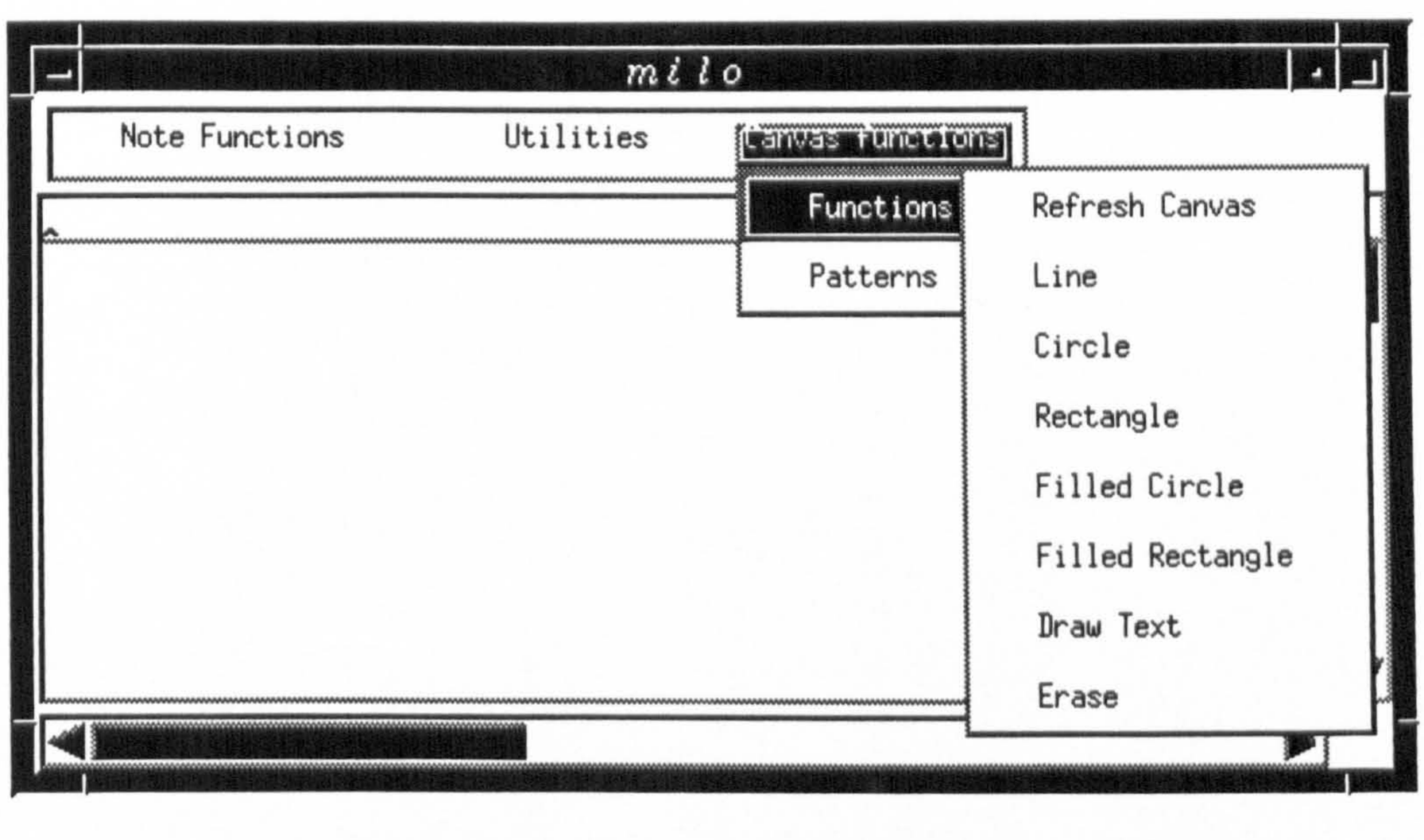


Figure A.7: A note showing the canvas functions menu.

when objects are drawn on the canvas. Both lines and filled objects will be drawn with the currently selected pattern.

Once an object and/or pattern are selected all subsequent drawing actions will be with that object and pattern until a new one is selected.

A drawing operation is carried out by pressing any mouse button while the pointer is within the canvas, and moving the pointer to indicate the extent of the object whilst keeping the mouse button depressed. Release the mouse button to complete the drawing action.

Try drawing several graphics objects. Remember to select the function, the drawing pattern and then to drag the pointer across the canvas to draw the shape.

A.5 Keeping the screen tidy

MILO offers facilities to aid in preventing the screen becoming cluttered. When working on a large document containing many notes it is likely that the working set of notes will grow as you refer from one section of the document to another.

Note windows can be removed from the screen in several ways

- by selecting the function **Hide this note** from the pulldown menu attached to each element of the hierarchical overview, each element of the linear overview, and each note window. When selected, the corresponding note window will be removed from the screen. If it is not currently displayed no action is taken.

- by iconfiying them using the method provided by whichever window manager you are using. Icons which represent note windows are easily distinguishable from other icons that might be on the screen.

- by selecting the **Hide all notes** function from the **Notes** main command menu, which removes all note windows from the screen.

A.6 Saving MILO documents to a file

A document created using MILO can be saved to a text file. This process is initiated by selecting the **Save document to file** item in the **Document** command menu. On doing this you will be prompted to position a dialogue on the screen (see Figure A.8). This dialogue contains four areas

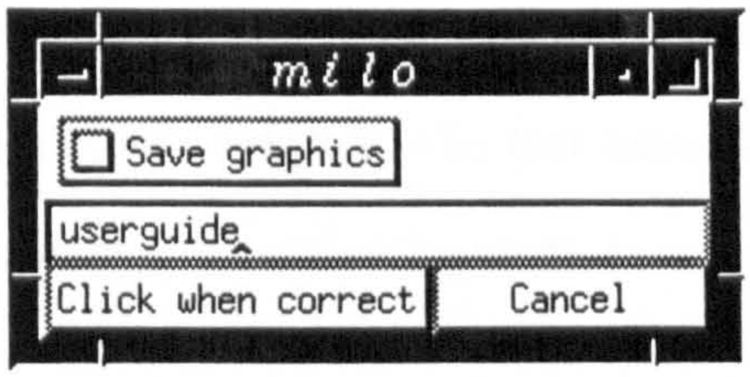


Figure A.8: The dialogue into which you enter the name under which the current file will be saved.

- the first area contains a single toggle button which allows you to select whether information from the graphics editors of each note should be saved to the file;
- the second area is a text entry area into which you should enter the name of the file to which the document will be saved;
- the third area contains two buttons. The first is labelled **Click when correct** which should be clicked on when both the correct file format and file name have been entered. The second is labelled **Cancel** which should be clicked if you wish to abort the save operation.

MILO will actually generate three files from your MILO document. One contains a mapping of your document onto a file in LaTeX format, the second contains only the textual elements of your document, and the third contains a MILO specific version of the document. When specifying the name of the file in the save dialogue you do not need to provide an extension to the filename. If you were to enter *introduction* as the filename, files called *introduction.tex*, *introduction.plain* and *introduction.milo* will be created.

A.6.1 LaTeX format

MILO will automatically generate a LaTeX file reflecting the hierarchical structure of your document. This is done by applying a mapping of levels of your document to LaTeX document elements. Hence the note at the root of your hierarchy becomes the abstract, its subnotes become sections, their subnotes subsections and so on.

MILO will insert information at the start of each document element to indicate who the author of that section was, when it was created, who last amended it and when the last change took place. The history of the document's development can therefore be seen from a printed version.

If you indicate in the **Save document to file** dialogue that you wish any graphics in the document to be saved to the file, MILO will attempt to approximate MILO graphics objects in a LaTeX picture environment.

If you know that you will wish to save your document as a LaTeX file you can enter LaTeX commands into the text held in individual notes. This allows you to create more complex LaTeX document elements such as itemised lists, tables of contents, indices, footnotes and so on. The LaTeX document can also be edited outwith MILO allowing any 'fine-tuning' of document layout that you wish.

A.6.2 Plain text format

MILO automatically creates a file containing only the headings of notes and the textual contents of notes. This allows you to create documents using the facilities provided by MILO, and then mark the document up in an appropriate manner selected by you.

A.6.3 MILO format

MILO also saves the document in a MILO-specific format. This enables the document to be retrieved by MILO, maintaining the document hierarchy, and the textual and graphical contents of each note.

Although this file contains information about graphical objects and the relationship between document elements, it is purely textual. It can therefore be edited like any text file and elec-

tronically mailed to colleagues. This enables them to reconstruct your document using MILO, and view it exactly as you had been doing.

A.7 Reading MILO documents from a file

A MILO document can be read from a file at any time. This is achieved by selecting the **Read document from file** entry in the **Document** command menu. Having done this you will then be prompted to place a dialogue window on the screen. This dialogue contains two areas (see Figure A.9). The first contains a list of MILO format files present in the current directory. The second is text entry area into which you should type the name of the file which you wish to read in. You can select the name of a file from the given list using the mouse, and paste it into the text area.

Figure A.9: The selection dialogue for the file to be read into MILO. You can select one of the filenames (MILO format files in the current directory) or type a name into the text box.

The filename entered should have the **.milo** extension, (e.g. introduction.milo).

If you select this function during a MILO session the current document will be destroyed and a new one created from the contents of the specified file. You should therefore ensure that you have saved the current document (if required) before reading a new one from a file.

A.8 Getting different views of document structure

MILO provides alternative views of the structure of the current document, showing the relationship between the notes within the document. The main view, a hierarchical representation of the document structure, is always visible. A linear view can be presented on demand. If the structure of the document is altered the two overviews are immediately updated to reflect the change.

A.8.1 The hierarchical overview

The hierarchical overview presents a tree-like graphical representation of the document structure. Each node of the tree corresponds to a note of the current document, and contains the title of that note. If the note title is amended, its corresponding node in the tree structure is changed character by character to reflect the alteration.

The node at the top of the hierarchical overview corresponds to the first note which was created for this document. The next level down represents subnotes of the first note and their relationship to it is represented by lines connecting the nodes. These nodes can also have subnotes and these note-subnote relationships are also represented by connecting lines. Figure A.10 shows a hierarchical overview for a large MILO document.

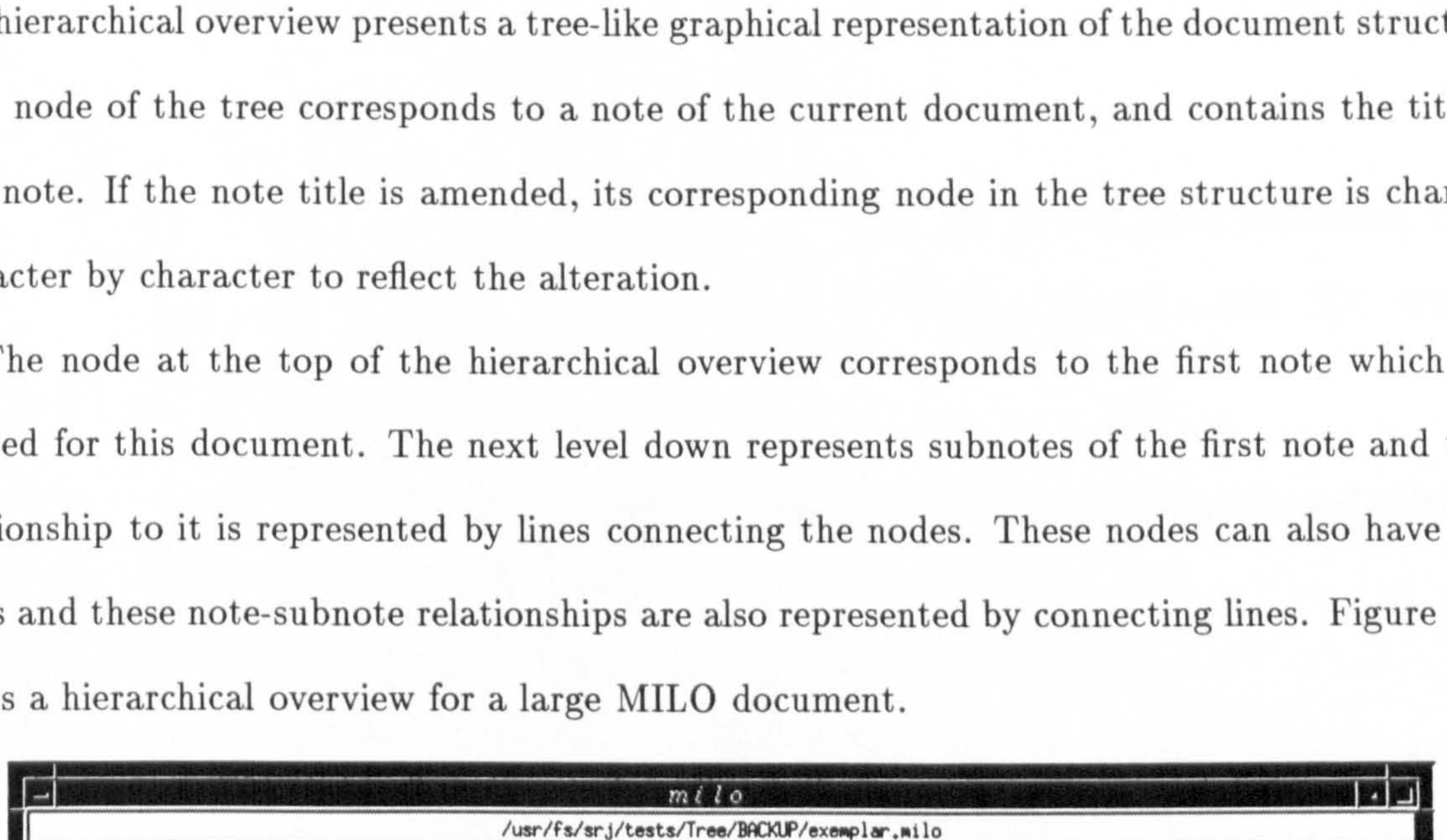


Figure A.10: The hierarchical overview representing a larger document.

Each node of the overview has an attached pulldown menu providing functions that can be

carried out on the corresponding note.

Whatever the breadth, depth or complexity of the document structure, the hierarchical overview will always represent the exact state of that structure.

Scrollbars are provided down the right hand side and along the bottom of this overview. This allows you to view parts of the overview which are not currently visible in the overview window, which may happen as you add more notes. The scrollbars are used by

1. placing the mouse pointer in an arrow at either end of the scrollbar and clicking the left mouse button once to move the window a small amount, or keeping the left mouse button depressed within the arrow in order to move the window a large amount. The window will move in the direction of whichever arrow you do this in.

2. placing the mouse pointer in the bar at the centre of the scrollbar, depressing the left mouse button, and keeping it depressed whilst moving the mouse pointer. This will allow you to move the window more rapidly.

In both cases release the mouse button when the window contents have been moved sufficiently.

A.8.2 Folding subnotes

All the notes contained in the current MILO document are, by default, represented in the hierarchical overview. However, it may be useful at times to take a more abstract view of the structure of the document. This means removing low level entities temporarily, so that you can view the higher level entities which remain visible in the hierarchical representation.

It is easy to simplify the hierarchical overview in MILO. If you wish to remove subnotes of a note from the overview, select **Fold Subnotes** from the pulldown menu attached to the parent element in the overview. The overview is then redrawn without the subnotes. To replace them, simply select Unfold Subnotes, again from the pulldown menu attached to the parent element in the overview.

A.8.3 The linear overview

The linear structure overview (see Figure A.11) provides an alternative representation of the

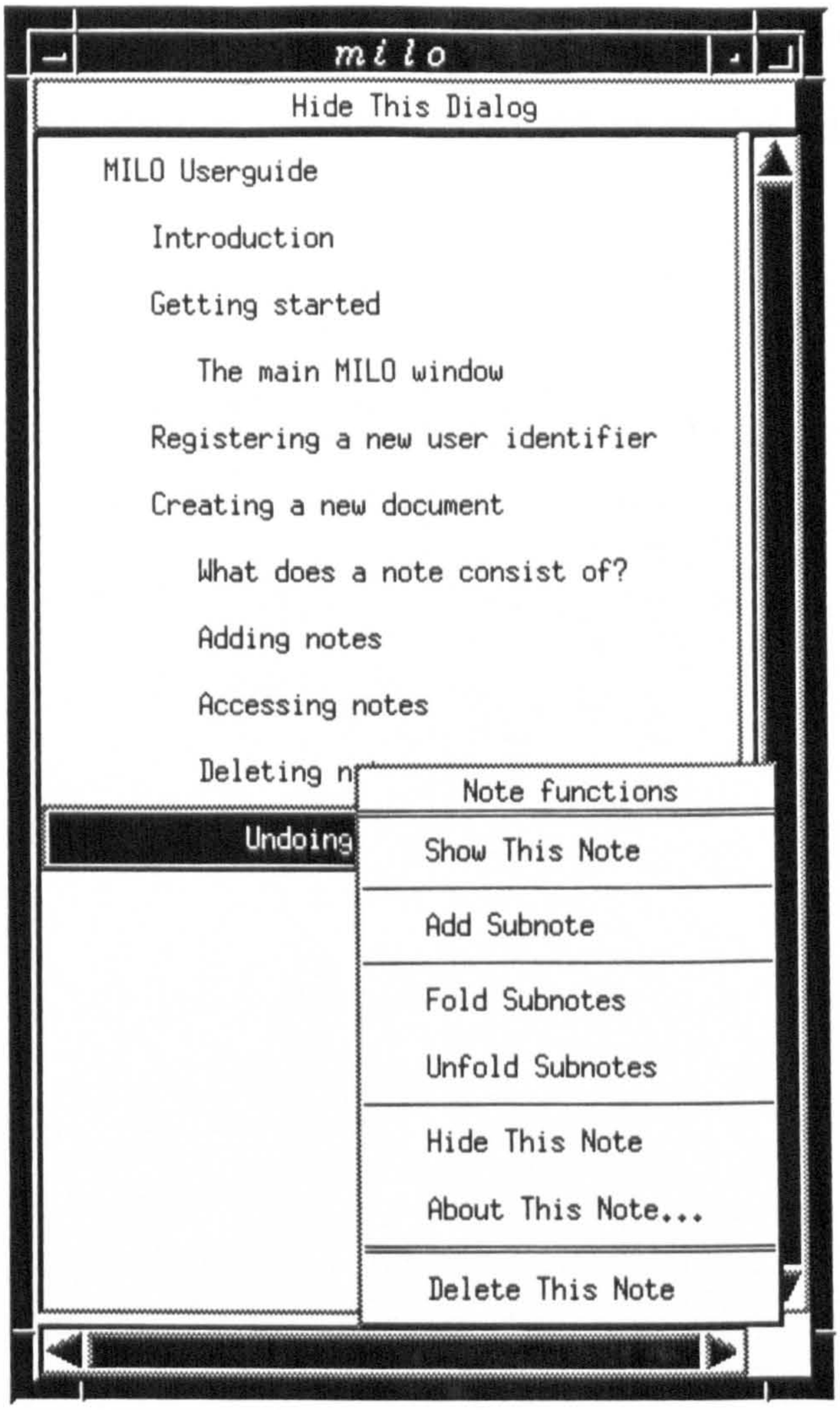


Figure A.11: The linear document overview presents an outline-like representation of the document structure. Each element has an attached pulldown menu.

structure of the current document. The representation provided is in the form of a table of contents/outline containing text labels which are the titles of each of the notes in the structure. It is accessed by selecting the **Linear overview** entry from the **Utilities** command menu. The text labels are indented in order to correspond to the level of the document hierarchy at which the corresponding note is situated. This linear representation corresponds exactly to the ordering of the document when stored and printed in LaTeX format.

When the document structure is altered, the linear overview is immediately updated to reflect the new structure. When the titles of notes are amended the corresponding label in the linear overview is also immediately updated.

The linear overview also has scrollbars which are used in exactly the same way as the scroll-

bars attached to the hierarchical overview. Each element of the linear overview also has a pull-down menu attached which provides functions which can be carried out on the corresponding note.

A.9 Getting different views of document content

In addition to providing different views of document structure, MILO also allows you to take different views of the content of the document. Facilities are provided to find elements of the document which were created by a specified author, or after a certain date, to find document elements containing specified text, and to see which elements have been amended most recently. These functions are accessed via entries in the **Views** command menu which are labelled **Filter notes**, **Find text**, and **Time ordered view**.

A.9.1 Filtering notes

It is possible to filter the notes within the document so that those with specified attributes are made visible.

To initiate the filtering process select the **Filter** entry in the **Utilities** command menu. You will then be prompted to position a dialogue on the screen. This contains four labels, four text entry areas and three buttons. The labels are **Author to show**, **Notes created after (dd/mm/yy)**, **Amender to show** and **Notes amended after (dd/mm/yy)**, and indicate which attribute is to be entered into which text area. You can enter values for any combination of the attributes (see Figure A.12). The buttons are labelled

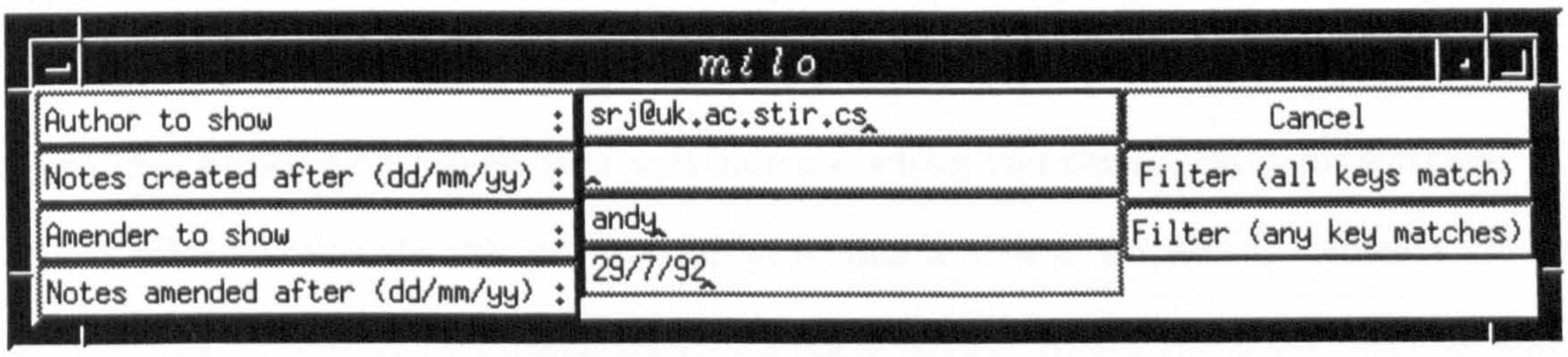


Figure A.12: The dialogue into which you enter information about which notes you require to be filtered from the document.

Filter (any key matches) when this button is clicked, all notes in the current document whose attributes match any one of those specified will be made visible. Blank attributes are ignored, and if no matches are found no notes will be made visible.

Filter (all keys match) when this button is clicked, all notes in the current document whose attributes match *all* those specified will be made visible. Any blank attributes are ignored.

Cancel when this button is clicked the filter dialogue is removed from the screen.

A.9.2 Finding text

MILO allows you to find which document elements contain a specific text pattern. To achieve this select the **Find text** entry from the **Views** command menu. You will be prompted to position a dialogue which contains a text label, a text entry area and two buttons. The label is **Enter text to find:** and indicates that you should enter the text pattern you wish to search for into the text entry area. Once you have entered the text correctly click on the **Click when correct** button. All notes within the current document which contain exactly the specified text in their main text body will then be made visible. The search for text is case sensitive, but can contain spaces, which allows you to search for phrases in addition to single words.

The first occurrence of the text in each note will be highlighted. You can search for further occurrences within a note by selecting the **Find text in this note** entry in the **Utilities** menu of the note in question.

The **Cancel** button will remove the dialogue from the screen.

A.9.3 Time ordered view

Keeping track of your work on a document may be difficult, especially if there are gaps between the work phases. It is sometimes difficult to remember which part of the document you were most recently working on, which may well indicate where you should carry on working.

MILO helps to alleviate this problem by providing a view of document elements ordered on the time which they were last amended (see Figure A.13). Hence the most recently amended element is the first in the list and least recently amended is the last in the list.

In order to use this facility you should select the **Time ordered view** function from the **Views** command menu. You will then be prompted to position a new window on the screen. This window contains a button at the top labelled **Hide This Dialog** which should be clicked on when you have finished with the time ordered view window. It will then be removed from the screen. The window also contains several other buttons. There will be one button for each note in the current document structure. Each of these buttons will contain the title of the note

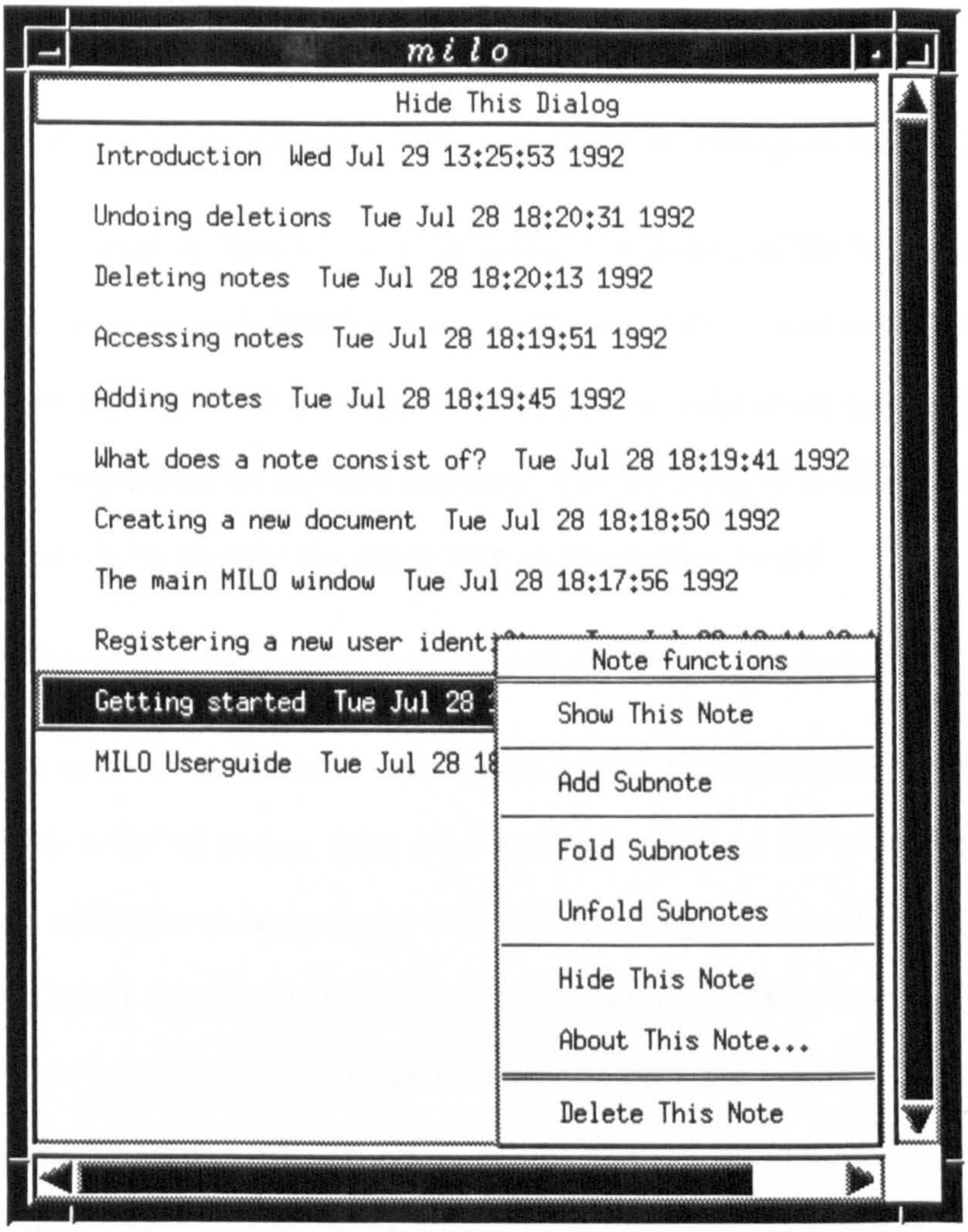


Figure A.13: A view of document elements ordered on time of amendment. The most recently amended are shown at the top of the list, the least recently amended at the bottom. Each element has an attached pulldown menu.

to which it corresponds, and the time at which that note was last amended. The button closest to the top of the window corresponds to the most recently amended note, and the button at the bottom to the note which was amended the longest time ago.

A.10 Collaborating via electronic mail

MILO supports you in your interactions with co-authors or colleagues via electronic mail by

- allowing MILO created documents to be saved to a variety of file formats, all of which are purely textual even though MILO creates a structured document containing both text and graphics. This allows MILO documents to be sent to colleagues via e-mail and then read or amended using MILO at another location. The recipient of a MILO document can read and manipulate it in exactly the same way as the author could.

- supporting the selection of e-mail destinations and structuring of e-mail commands.

MILO reduces the need to remember complicated e-mail addresses and command formats by presenting you with a list of names from your mail alias file[2], a list of authors of the current document and the ability to automatically create the mailing command.

To achieve this select the **Use mailer** function from the **Utilities** command menu. You will then be prompted to position a window on the screen (see Figure A.14). This window contains two text entry areas labelled **Mail target is :** and **File to send is :**. The first is where the appropriate e-mail address will appear and the second is where the name of the file to send will appear. Below this is a button labelled **Click To Send Now** which should be clicked once both the mail target and file name are satisfactorily specified. When this is done the command to mail the specified file to the specified target is created and executed.

Below this is a button labelled **Hide This Dialog** which should be clicked on to remove the mailer dialogue from the screen. Below this areas are two sets of names placed in a window with attached scrollbars. The left hand set of names are retrieved from your mail alias file. When you click on a name with the left mouse button, the corresponding e-mail address is placed in the mail target entry area. The names on the right are the user identifiers of authors of notes in the current document. These may or may not be e-mail addresses.

Although you can retrieve e-mail addresses by clicking on the name buttons, it is also possible to enter addresses via the keyboard in to the mail target area. This means that documents

[2]This is expected to be in $HOME/.mailias

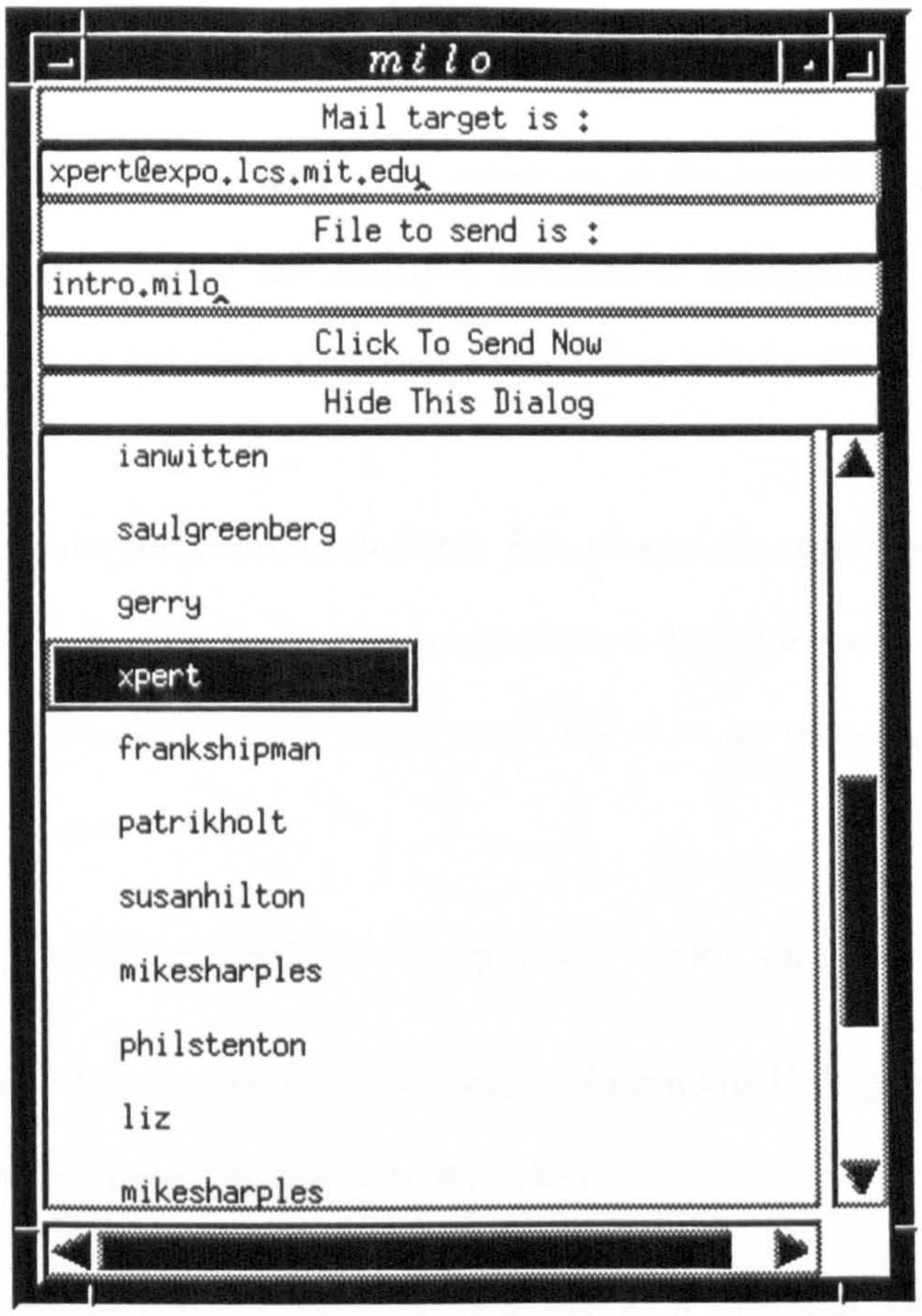


Figure A.14: The mailer dialogue. When one of the aliases is selected the electronic mail address in retrieved.

which you send need not have been created by MILO and the target need not be a regular correspondent.

A.11 Merging MILO documents

MILO provides facilities for you to merge two versions of the same document which have been created using MILO. This is useful for multiple authors of documents, as collation of changes made by different authors is automated rather than being a long process of editing a previous version.

To merge two documents you should select the **Liveware merge** function from the **Utilities** command menu. You will then be prompted to position a dialogue on the screen which asks you to enter the name of the file with which the current file is to be merged.

The rules for merging are thus:

- the current MILO document will be the updated version on completion of the merge.
- new notes will be added into the current MILO document if they are not present in it, but are present in the document being merged into it.
- notes will be moved in the structure if they are at a different position in the document which is being merged in and they were updated more recently.
- if a note appears in both versions of the document, the text and graphics of the most recently amended version will prevail.
- if a note exists in the current MILO document, but not in the document being merged in, it will not be deleted.

When the merging process has been completed, the hierarchical and linear overviews will reflect the new structure of the document.

A.12 Other utilities

MILO offers another utility which has not yet been mentioned.

A.12.1 Spell checking

MILO will check the document for incorrect spellings. If you select **Spell check** from the **Utilities** command menu a list of spelling errors will appear in the MILO message area. This information indicates in which note the spelling errors have occurred.

A.13 Ending a MILO session

In order to end a MILO session you should click on the **Quit** function in the main command menu. One of two things will then happen

1. MILO will shut down and any windows currently belonging to MILO will be removed from the screen. This will happen if no changes have been made to the document since it was last saved to a file.

2. If any changes have been made to the document since it was last saved to a file you will be prompted to position a dialogue on the screen. This will contain two buttons: **Quit without saving** which should be clicked if you wish to quit without saving the changes, and **Don't quit** which cancels the quit function, giving you the opportunity to carry on with MILO session and perhaps save the current document to a file.

A.14 MILO Text Editing Functions

forward 1 character	Ctrl F
	→
backward 1 character	Ctrl B
	←
forward 1 word	⟨*Meta Key*⟩ F
backward 1 word	⟨*Meta Key*⟩ B
forward 1 paragraph	⟨*Meta Key*⟩]
backward 1 paragraph	Ctrl [
go to beginning of line	Ctrl A
go to end of line	Ctrl E
go to next line	Ctrl N
	↓
go to previous line	Ctrl P
	↑
go to next page	Ctrl V
	Next
go to previous page	⟨*Meta Key*⟩ V
	Prev
go to beginning of text	↖
go to end of text	Shift ↖
scroll text 1 line upwards	Ctrl Z
scroll text 1 line downwards	⟨*Meta Key*⟩ Z
delete next character	Ctrl D
	Delete char
delete previous character	Ctrl H
delete next word	⟨*Meta Key*⟩ D
delete previous word	⟨*Meta Key*⟩ H
kill word	Shift ⟨*Meta Key*⟩ D
backward kill word	Shift ⟨*Meta Key*⟩ H
kill selection	Ctrl W
kill to end of line	Ctrl K
kill to end of paragraph	⟨*Meta Key*⟩ K
newline and carriage return	Ctrl J
	Insert line
newline and back up	Ctrl O
newline	Ctrl M
	Return
redraw text	Ctrl L

On the HP Workstations the Meta Key is the Extended char key

Index

Appendix B

Supplementary material

B.1 Jones, S. 1989. Of Mice And Moaning. *BCS HCI Specialist Group Newsletter*, 13, pp10–11.

B.2 Jones, S. 1989. A Mouse's Tale (Continued). *BCS HCI Specialist Group Newsletter*, 14, pp7–9.

B.3 Jones, S. 1990. Friend Or Foe? *New Scientist*, August, p55.

B.4 Jones, S. 1990. Inside The Postgraduate Poverty Trap. *New Scientist*, December, p76.

B.5 Jones, S. 1991. MILO. *ESRC Data Archive Bulletin.* Software Bulletin, S1–S3. p51, October.

B.6 Thimbleby, H, Jones, S, & Cockburn, A. 1992. Hypercard: An object oriented disappointment. *In:* Gray, PD, & Took, R (eds), *Building interactive systems: Architectures and tools.* Springer-Verlag.

B.7 Jones, S, & Cockburn, A. 1993. *Reducing requirements in computer supported writing tasks.* Presentation at the 6th UK Conference on Computers and Writing. University of Wales, April 13th–15th, 1993.

B.8 Thimbleby, H, Marsh, S, Jones, S, & Cockburn, A. 1993. Trust in CSCW. *In:* Scrivener, S (ed), *Computer-Supported Cooperative Work.* Ashgate Publishing. In press.

B.9 Cockburn, AJG, & Jones, SRA. 1993. Four principles for groupware design: encouraging adoption and easing system use. *In:* Dourish, P (ed), *Implementation perspectives on CSCW design.* Springer-Verlag. In press.

Of Mice and Moaning

Introduction

I first used an Apple Macintosh approximately eighteen months ago, as part of a short undergraduate HCI course at Stirling University. My classmates and myself had been reared on VMS and Unix based systems - mice and WIMPs were alien to us. As a result a class of twenty year olds ended up grinning like kids at Christmas as soon as we got our hands on MacPaint. We loved the novel interface, and thoroughly enjoyed using the machine.

Since then I had used the Mac only occasionally to create short documents using MacWrite. However, Christmas did come early and in December I got a MacPlus in my office. Up to this point I had used a Mac in the same room as other more experienced users, but now I was on my own and couldn't ask the person next to me for advice. The more I used it, the more problems I experienced, and the bugs and inconsistencies present in the interface became evident. Harold Thimbleby (who also got a Mac at the same time) and myself started to note these and managed to fill well over fifteen pages of A4 paper (in a reasonably small font of course). Nevertheless, we could still use machines effectively, as we asked advice of each other or colleagues in order to overcome problems. Then, however, I started to wonder what would have happened if I had been the first person in the department to get a Mac - what would have happened if it had been just me (or any user), and the manual, versus the system?

Answering the question

To get a reasonably quick impression of what the answer to the above question might be, I coerced three naive Mac users into my office. "Volunteer" A had never used a computer before, B had programmed in Pascal under VMS for eleven weeks, and C has a BSc in Computing Science and works with PC compatibles ever weekday.

I presented them with my single drive Mac Plus (without a hard disc - as the system was initially tolerated until the hard disc arrived), a System Tools disc and a Mac Write disk and told them to work through Chapter One of the user manual. In spite of the sometimes pitiful pleadings, I gave no advice during any of the sessions.

Observations

For a start, all three had difficulty even switching the machine on! Each felt at the bottom back right for the switch, and two had to peer around the back (narrowly avoiding bookshelf related head injuries to find it. Has anyone used hardware other than a Mac with the on/off switch situated back-middle-left? Not me. All managed to then boot the system, and follow the manual's instructions on window manipulation. C was more adventurous and investigated some functionality without prompting from the manual.

Eventually subjects A and B read from the manual that they had been using the 'Finder' (C skipped that bit). I still don't know why it is called the Finder or what exactly it comprises and they certainly didn't. "Is it some sort of index perhaps?", "Is it the operating system stuff?". Anyway the manual decided that it was time to use an 'application'. This is fine if you know what the jargon 'application' means but A didn't. C is fluent in computer jargon, but still

didn't understand a paragraph about system software, version numbers, installers, "About the Finder", etc.. B didn't understand it either and as a result thought that it must be important! In the paragraph it prominently said to select 'Shut Down' (although if you could decipher the jargon, it was only to be done if you wanted to check which version of the system software was on the application disk) - so he did!. Fortunately he managed to restart.

Time to use MacWrite - all three ejected the system disk and inserted the MacWrite disk successfully, although they did wonder why the System Tools icons and windows should still be displayed when the disk was out of the machine and in their hand. They all double-clicked on the MacWrite icon and then the fun started (well for me anyway). The Mac spat the MacWrite disk out and prompted for the System Tools Disc to be inserted. The manual made no reference to a single drive Mac nor to the fact that in this case about twenty further disk swaps would be required to start up MacWrite. A small selection of resulting comments - "I'm stuck swapping discs", "it seems something has gone wrong", "it's highly irregular", "I could be doing this [disc swapping] all day", "I'm completely lost now". A and B had no idea what was happening but at least C knew that "it is copying bits of the disks into memory, but also "I would have thought it would have been more intelligent". Disks started to be inserted with somewhat more than the necessary force after the sixteenth swap.

A conceded defeat after 30 minutes of spasmodic disc swapping/manual reading (1 hour 30 minutes into the session), but didn't want to switch the machine off because he was unsure of the consequences. B had got into MacWrite but selected Close to 'close the window' to see what was underneath. Too bad that here Close means' close the file'. He couldn't get it back and after 1 hour 20 minutes of the session gave up, also not wanting to switch off. C got into MacWrite but because it started with a document entitled 'untitled' and New was inactive in the File menu he thought "I can't create a new document". Wrong. He thought that Page Setup would initialise the page for typing. Wrong. He thought he could type at the pointer and not the I-beam cursor. Wrong. He quit MacWrite and after the disc swapping, had System Tools in the Drive. "I've completely lost MacWrite - permanently". Wrong. After 1 hour and 28 minutes he ejected the disk, selected Shut Down and switched off.

Comment

I have pages of problems and difficulties that the subjects came across and misconceptions that resulted. Maybe it was cruel to provide only one floppy disk drive, but that was how the Mac Plus arrived and how it stayed for two weeks. The Macintosh interface has been the "state-of-the-art" for years, but now the use of WIMP interfaces is becoming more widespread. I wonder if the Mac's 'easy to learn' and 'easy to use' reputation is quite so well deserved, and hope that the new breed of WIMP interfaces will have learned from it rather than just copied it.

Steve Jones, Stirling University.

Features

A Mouse's Tale (Continued)

Introduction

In the previous issue of the Newsletter, I reported on my observations of first time users of the Macintosh, and the (numerous) problems they experienced. I then started to ask myself to what extent they had learned from that experience. An underlying principle of the Mac is that knowledge of the basic interaction techniques will transfer between applications, and make the process of interacting with a new application easier. I wondered, if asked to use the Mac again they would remember the fundamental, underlying concepts of Macintosh interaction, and if they did, would their knowledge help them in trying to use an application such as MacWrite intuitively? I also wondered if they might swear loudly and refuse to touch a Mac again.

Hoping not to get the latter response, I asked my three 'volunteers' to commit themselves to another session to carry out a further task. Fortunately, they all agreed willingly! (To recap, A had never used a computer before, B had eleven weeks experience of programming in Pascal under VMS, and C works with PC compatibles every weekday.)

The Task

In the previous study, none of the subjects managed to do any useful work with MacWrite, having enough trouble trying to master the Finder. In this study I asked them to create a MacWrite document, specifying certain stylistic points such as font, point size, margins, spacing and justification. I asked them to both enter text in the desired style and also to enter it as plain text and change it later. They were instructed to save the document under a certain name, in a certain folder, then alter it and save it under a different name.

In a moment of compassion I presented them with my single drive Mac Plus *with* the hard disk attached. This was to relieve them of the tedious disk swapping, which both physically and mentally exhausted them last time (and also to avoid physical abuse). I did not give them a manual, and once again refused to give any advice during the sessions.

Observations

Each subject showed an immediate familiarity with window and menu manipulation techniques. They each traversed the menus to remind themselves of their content, and carried out window manipulation to some extent. A and B opened, moved, resized and closed windows without hesitation, but C entered MacWrite immediately by double-clicking on its icon. This, plus their apparent increase in dexterity with the mouse, would seem to indicate that they had in fact remembered the fundamental interaction techniques, even though it was *six weeks* since their previous sessions.

However, it soon became obvious that even though this was the case, they would not find creating a document with MacWrite, for the first time, particularly easy. Their problems were many and varied, so I shall restrict my report here to major *misconceptions* which I observed.

Misconceptions

Firstly, all three subjects believed that within MacWrite all operations would be controlled by menu choices, as they had previously seen to be the case in the Finder. They could set the font, point size, justification and type style via the

menus, yet had difficulty finding out how they should set the margins and line spacing (even though the ruler sits directly below the menus). C said "Spacing will be under Format. It's not!?", A said "I'll type it in and *then* try and sort it [spacing and justification] out", and B started typing without correct margins set, with the default justification and thinking that he had to double-space explicitly! He did not realise that he would still have opportunity to change them.

The ruler deceived all subjects too. Each eventually realised that there were margin markers, but having been told to create a left margin of 2 inches, B and C moved it in by 2 to give a margin of 3 inches. How many readers who use MacWrite immediately realised that the ruler had an inch missing? Not me for one. A and C weren't even convinced that they were inches - "I presume these are inches", "I'm assuming the scale on the ruler is inches". C thought that 6 lines per inch referred to the number of divisions per inch on the ruler! By the way - what *does* 6 lines per inch mean or do?

Is MacWrite really a WYSIWYG editor? Well, sometimes the hard copy *does* correspond to the screen, but is what you see on the screen *really* what you get to edit? In the case of blank space it certainly isn't - it isn't possible to insert text where text has not already been inserted. All subjects perceived blank space, within and without the boundaries of the text entered, to be exactly the same, because they are led to do so by MacWrite, which displays them as exactly the same! If it treats them so differently, then why not indicate so?

A developed interesting yet misconceived behaviour. He realised that after choosing *Select All* he could operate on all of the document. However, he didn't really want to do so yet, but couldn't find how to 'deselect all'. Simple, *if you know how*, but imagine trying to intuit clicking on the selected text to unselect it! He thought that double-click on *Select All* was the answer. Well it was, but only because after the first click the menu disappeared, and the second click occurred in the selected text! He double-clicked too quickly to notice this and this behaviour became ingrained, and was repeated several times more.

B wanted to underline a word that he had just typed. He moved the pointer to before, after, and to the middle of the word, selecting underline each time. Nothing happened and he gave up. He later decided to experiment with the *Find* menu choice. He did so and closed the Find window, only to discover that the word he had found was now highlighted and he could change its style. From this point he believed that this was the correct procedure for changing the style of some text.

Feedback

A large number of problems arose as a result of no feedback to user operations, little feedback which didn't serve its purpose, and misleading feedback.

An example of the first is the *Save as...* dialogue. All subjects had difficulty believing that the document had actually been saved once the dialogue disappeared - "I *think* my document has been saved, but I don't know where.", "I have no confidence it has saved it". All subjects repeated the *Save as...* operation in order to verify that the file had in fact been saved. C actually had to do the operation a third time, because he thought that when he chose *Cancel* from the dialogue, his file might well have been deleted. Surely all that is required to solve this problem is to show the name of the saved file in the dialogue window *before* removing the dialogue from the screen. The user could then have a feeling of successful completion of the subtask, and concentrate on the next part of the editing task.

The operation of selecting a font results in feedback which does not fulfil its purpose. A user can only be certain that the correct font has been selected by looking at the font menu again (and then there is always a chance of a mismouse, and choosing the wrong font when only wanting to peruse the menu). C did exactly this. Why not place the tick next to the font and pause a moment before removing the menu from the screen?

Typing italics in MacWrite results in misleading feedback. It requires more than one space to be inserted after the italicised word for a single space to manifest itself on the screen! The subjects who italicised as they typed made this 'mistake', and their hardcopy did not correspond to what they saw on the screen.

Comment

It is not possible to convey in a short article such as this, the multiplicity of problems the subjects experienced, and the terrible frustration that resulted. It took them an average of 1 1/4 hours

to create a five line document *incorrectly*. I don't know how long it would have taken had I pushed them to continue until the document was correct.

I have highlighted some problems with MacWrite in this article, but the principles can be extended to any system/application. Each subject now has misconceptions about the system - it is important to support the building of a *correct* mental model of the system by the user. Each subject was misled, or under-informed by the system - feedback should be used to ensure that the user has a correct understanding of the system state at all times.

I finish, as I finished my previous article, by urging that emerging WIMP interfaces not only adopt the Macintosh's good points, but attempt to identify and rectify the bad ones.

Steve Jones, University of Stirling.

Friend or foe?

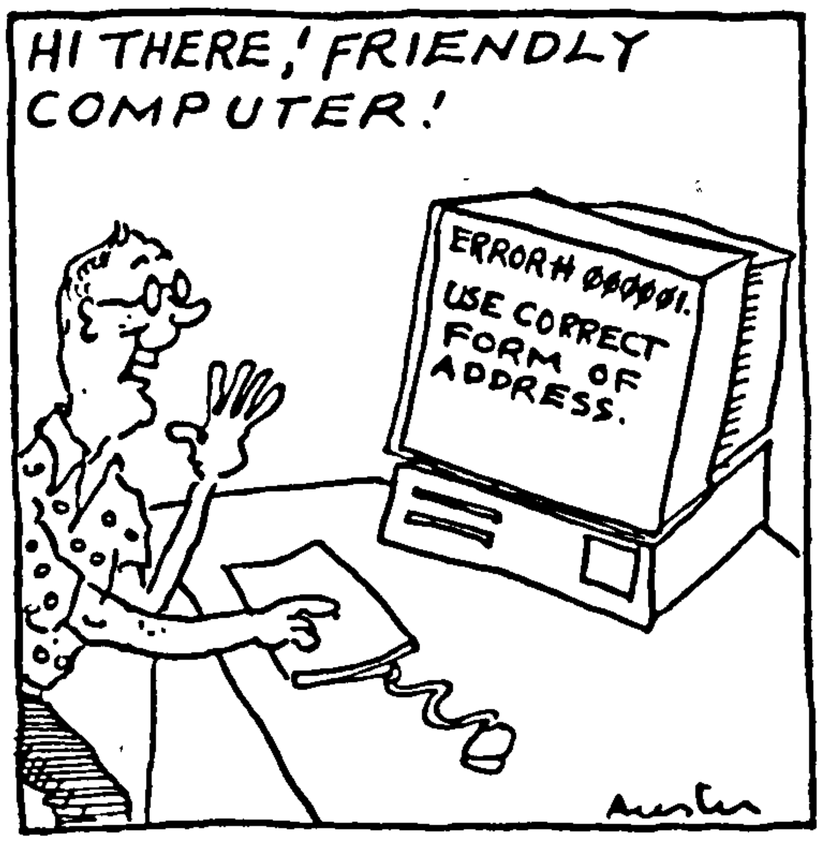
Steve Jones finds computers are not all they claim to be

WHEN manufacturers of computing hardware and software try to sell us their products, they are keen to point out how many things those products can do, how quickly they do them and how much of a bargain they would still be, even at twice the price. However, a new selling point has entered the picture: ease of learning and ease of everyday use.

Consumers perceive that they can avoid the expense of training staff to use otherwise unusable computer products if they buy a system that is sufficiently "user friendly" in the first place. In turn, manufacturers have realised that consumers think like this, hence the proliferation of products accompanied by claims of user-friendliness.

A year ago, a colleague and I took delivery of a highly regarded machine. Over several years it has established a reputation for being both easy to learn and to use, while maintaining sufficient raw computing power to make it apt for a variety of applications. We were quite excited about using it. This feeling turned to disappointment when, after a few weeks, we had a long list of problems with its interface which had a direct and adverse effect on our ability to understand or use the machine effectively and efficiently.

Either we were particularly inept (my colleague is a professor of information technology and I am a graduate in computing science, so one would hope not), or others had also experienced problems similar to, or the same as, the ones that we encountered. I coerced three "volunteers" with varying degrees of computer experience into learning to use the machine. I presented them with my machine, its user manual and the required floppy discs, and set them to it. I then observed their efforts and, despite pitiful pleadings, gave no advice.



How did they fare? Disturbingly badly. All three had difficulty even switching the machine on. The switch is out of sight, and they searched for it at the bottom back right of the case—it is in the middle back left. Two of my subjects had to peer around the rear before discovering it. To make matters worse, the switch conforms to the American standard of up for on, and down for off.

When it came to the user manual, they couldn't understand a large proportion of this because of the use of jargon such as system software, installers and system tools.

Starting up a word-processing application involved physically swapping two floppy discs in and out of the machine almost 20 times. After a few swaps, the volunteers were showing signs of removing and inserting the discs with a little more force than necessary. All gave up at this point, and they hadn't even begun to do anything constructive. An "easy to learn" computer had lasted an hour before the naive users came across what they thought to be an insurmountable problem.

A few weeks later, I managed to persuade the subjects to come back and try the word processor once more. This time I attached a hard disc to the machine, which removed the need for disc swapping (sighs of relief). They all remembered the basic interaction techniques required—pointing, clicking, dragging, selecting. This wasn't much use, though, when it was not obvious to them that the screen's ruler was marked in inches, or that an inch of it was missing from the left of the screen. Nor was it apparent that typed spaces and blank spaces are treated differently even though they look the same on the screen, or that the command to save a file had actually done so.

I could continue listing problems for the rest of this page that these naive users encountered. But the point is that I would be surprised if there were anyone who had learnt to use a new computer and new applications without experiencing numerous, unnecessary difficulties. Manufacturers claim user-friendliness for their machines, and some come closer to achieving it than others, but "user tolerant" is probably a more apt description. Some systems are still downright "user hostile".

Computer hardware and software is a multimillion pound worldwide industry. Millions of pounds are spent each year on research and development of new products. These improvements, however, seem to be mostly in the area of functionality. "Upgraded" and "improved" generally seem to mean "We have added another 25 features" rather than anything to do with usability. Having spent thousands of pounds developing the functionality of a product, surely it makes sense for a company to invest in making those features easy to learn and use so as to increase its share of the market as much as possible?

The word processor I am now using has a screen test facility which draws nice patterns on the screen. I haven't used it; I don't need to use it; it does not help me in writing and formatting documents. The system also has a spell checker which is very useful. However, it doesn't check the spelling as I type and it contains spelling mistakes itself. It seems obvious which facility deserves the development effort.

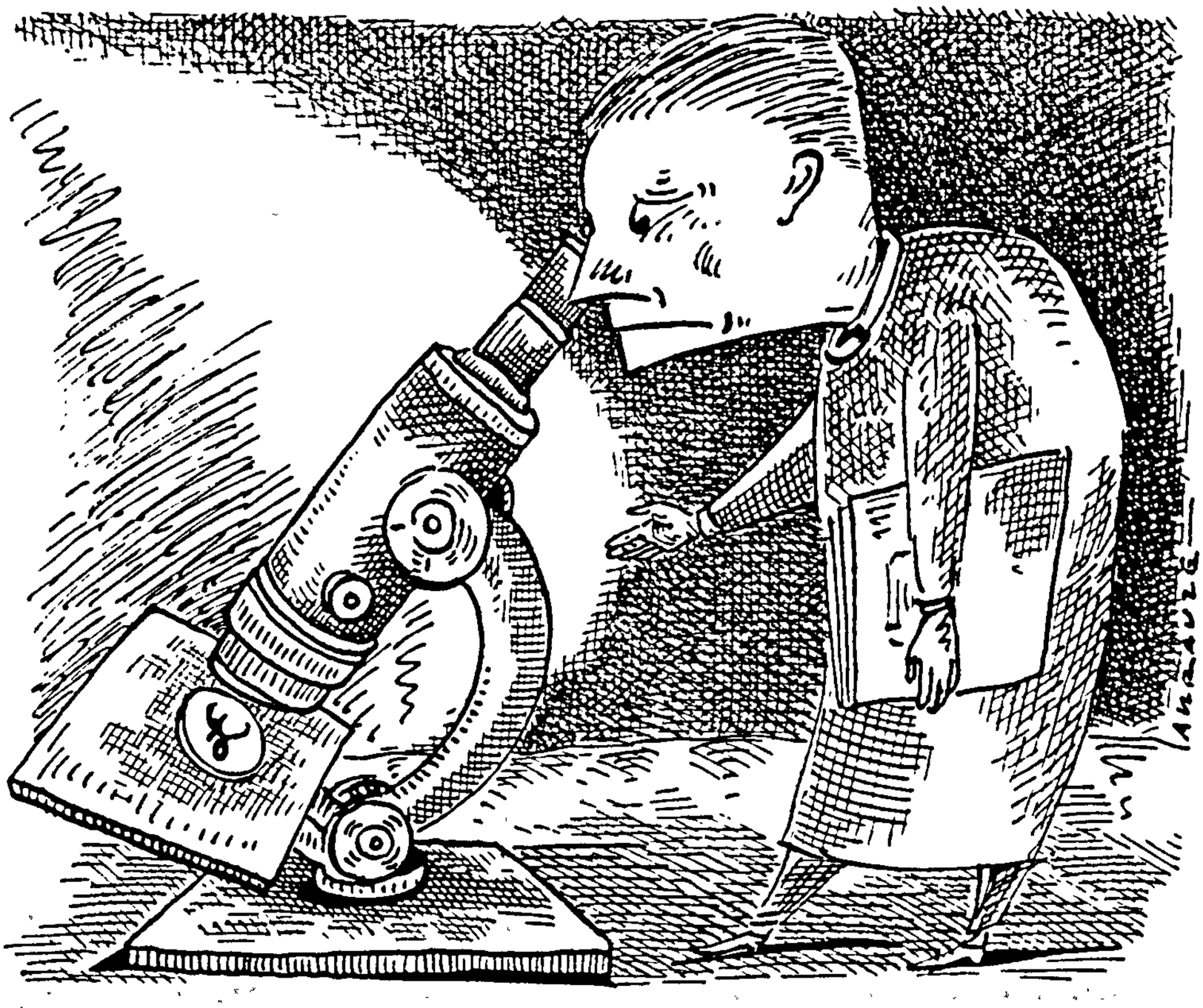
This is not to denigrate the advances that the industry has made in computer usability, but there is still much to be done. I look forward to the day when everyone will be able to walk up to a computer and use it effectively without surrounding themselves with manuals, and investing much time and effort in gaining the required knowledge. Until that day, I wish manufacturers would restrain from making subjective and usually inaccurate claims about their system's amicable attitude towards the user. □

Steve Jones is a PhD student in the Department of Computing Science at the University of Stirling.

FORUM

Inside the postgraduate poverty trap

It's not just undergraduates who have a raw deal, says Steve Jones



LIFE can be many things to a postgraduate research student: fun, depressing, exciting, mundane, challenging, dull, relaxed and exhausting. Bouts of lethargy and intensive work come and go. But there is one consistent aspect to all this, and that is continuing and worsening financial hardship. Not only is postgraduate research financially unrewarding, but it is increasingly financially non-viable.

In recent years the value of the undergraduate grant has steadily decreased. Now it is frozen at this year's level. Postgraduate grants from research councils and other bodies have tended to keep in line with the weekly value of the undergraduate grant and have also been dropping in value, although fortunately they have not been frozen at the current level.

More and more undergraduate students have been turning to part-time jobs during term time, and full-time jobs during the holidays. This option isn't available for research postgraduates, as they are committed as full-time students to work all but six or so weeks a year, and so they lose on two fronts.

Well, actually, they lose on far more than two fronts. To counteract the effects of the freezing of the grant level, the government has set up a scheme whereby undergraduates can now take out Student Loans from the Glasgow-based Student Loan Company.

Postgraduates, however, are not eligible for a student loan, and so they cannot take advantage of the preferential interest rates and extended repayment schemes.

A further solid financial blow was delivered by the recent housing and benefit "reforms" for those living in rented accommodation. Last year's relaxation of control over private landlords and rent levels meant that the concept of a fair rent was removed, leaving landlords free to demand whatever amount they wanted.

With accommodation problems in universities and polytechnics across the country, postgraduates are generally left to fend for themselves in the private housing market, unable to get places in university-subsidised accommodation.

The situation became even worse with the withdrawal of Housing Benefit (the rebate payed to those paying higher rents) from all students—this is probably the worst of the adversities facing postgraduates.

Consider this alongside the abolition of the rating system and the introduction of the poll tax, the Community Charge.

Previously, rates were usually payed as part of the rent, but after the abolition of the rating system few landlords reduced the rent to pass this saving on to the tenant. So tenants, including students, will now be paying the old rates in addition to the new Community Charge.

Many scientific research students are now also being affected by the delay in increasing the grants to those funded by the Science and Engineering Research Council and the Medical Research Council, as David Gray reported in a recent letter to *New Scientist* (3 November).

The increase would normally take place in October, but it will now only happen in April next year, helping the councils to live within their budgets, but also depriving those students of several hundred much-needed pounds.

So what does this mean for a typical research postgraduate at, say, the University of Stirling? The weekly value of the grant is £69, and that is your total income. Outgoings are £31 for rent, £23 for food, £2 for bills, £4 for travelling to work, and £1·50 for the Community Charge. That comes to £61·50, leaving a mammoth £7·50, and the student hasn't done anything but sleep, eat, keep warm, and go to work! If you buy a couple of books, you spend a couple of months' disposable income!

Morale, then, cannot be anything other than low among research postgraduates, and little wonder. Consider the friends of our typical research postgraduate.

They graduated with the same degree as you at the same time, two years ago. They now both earn in the region of £13 000 to £15 000, and one is buying his own flat, and the other drives a nice car and goes to foreign climes for her holidays.

Material wealth is obviously not that important to the average postgrad, but this example just shows what major incentives there are for not going on to further study.

As the "demographic time bomb" takes effect, it is employers that are placing more and more value on graduates, offering ever-increasing incentives to entice them into industry. For a science graduate the temptation is usually even more pronounced, with much higher salaries being offered than to arts graduates.

The crux, then, is the effect this will have on the future of academic research in this country. For someone to commit themselves to a three- or four-year undergraduate course is a big step, when they know that at the end of it they are likely to have a large debt hanging over them. To consider further study under even greater financial hardship may be thought by some to be the first signs of mental instability.

Yet people do still want to extend their academic career, and undertake research. How long this will continue to be the case is another matter.

Unless the most important resource, people, are encouraged in a manner that reflects their value, the likelihood of an increase, or even maintenance, of the numbers of research students is nil. □

Steve Jones is in the Department of Computing Sciences at the University of Stirling.

Number 31

Software Bulletin

Edited for the ESRC Research Resources Advisory Group by Richard Bland, Department of Computing Science, University of Stirling. Contributions and comments are always welcome.

MILO

A tool which supports asynchronous co-authoring of structured documents

Introduction

Computer based tools such as text editors, graphics editors, idea processors, hypertext authoring systems, communication systems and document formatting systems can be useful to both individual and collaborative authors.

Unfortunately, use of these tools presents authors with two main problems: no single tool can as yet satisfy the majority of an author's needs; existing tools are diverse in appearance and behaviour, and are unlikely to be compatible.

As a result, authors can be faced with the daunting task of using several different systems during the writing process for support of tasks such as idea processing, text entry, diagram production and document formatting. These systems may not even be available on the same hardware platform. Collaborating authors face more difficulties. They need to find a suitable communication system if they are geographically distant, and the tools used by one co-author may not be available to another.

This paper discusses MILO, a hardware independent computer based tool which supports distributed, asynchronous authoring of structured documents. A wide variety of approaches to creating documents containing both text and graphics is supported.

At the most basic level MILO can be used by an individual author as a text editor to construct a linear document. At the other extreme it can meet the needs of geographically distant co-authors who are writing a continually revised document of complex structure containing text and graphics.

MILO can be used as an idea processor, allowing the user to note distinct ideas and create links between them, providing a graphical representation of the resulting structure and allowing its content and form to be edited.

Construction of structured documents is achieved through creation of a hierarchy of logical units called notes. The author is provided with alternative representations of the document structure and can take different views of document content. Fast access methods to elements of interest in potentially large document structures are provided.

The use of MILO can ease communication and information exchange between distant colleagues. Although MILO has an advanced graphical user interface, MILO documents are stored in a purely textual manner, allowing geographically distant collaborations to take place via electronic mail, and making exchange of MILO created documents system and hardware independent.

MILO also provides automatic updating of a document which has multiple authors, drawing together the contributions of all them into a single version of the document.

Notes

As we have seen, a MILO user manipulates a single data element type called a note, to create a document. MILO documents are built from any number of individual notes, and so a valid MILO document could consist of a single note or many interrelated notes existing in a hierarchical structure. Each note occupies its own screen space and there is no limit to the number of notes which can be visible at any one time. The complexity and form of the hierarchical structure is user defined.

All notes have exactly the same look, feel and behaviour, and consist of three elements. These are:

- A text field which is intended to hold the header or title for the individual note;
- An EMACS like text editor which is intended to hold the main body of text for the note. The text editor behaviour is exactly the same for the two text elements of a note;
- A simple graphics editor allowing the user to draw lines, rectangles, circles and text in a variety of pen patterns.

A user can also access information about each note, including who created it and when, and who last amended it and when. The information is continually updated providing users with information about the work patterns of themselves and their co-authors.

Some systems provide multiple document element types, such as annotations, graphics and document text. MILO takes the opposite approach providing a single, consistent element type. This will maintain conceptual simplicity of document content. A note can serve as a document element, an annotation, a personal reminder or a communication medium.

Creating documents

An author creates a MILO document by adding notes to the existing document structure, or adding unattached notes as they are required. In the case of adding to an existing structure, an author will specify the position that the new element will occupy in the structure.

MILO caters for diverse approaches to creating a structured document. At one extreme the author might create the structure of the document before entering any text or graphics. This indicates the initial hierarchy of the notes, and the relationships between document elements before the author needs to concentrate on text generation. At the other extreme, the author may begin by 'writing to discover what she has to say' and generating a large amount of unstructured text which is then manipulated into a structured document. Of course it is possible for an author to want to use an approach somewhere between these two extremes, and MILO will allow this, catering for the diversity of writing styles which exist along the continuum.

The author can move the focus of her attention from one part of the structure to another, from one note to another at any time, and all notes are always easily accessible. During the creation of a document, text and graphics can be updated, added or deleted at any point.

Amending MILO documents

Revision is crucial in the writing process. MILO provides facilities for repeated revision of a MILO document during its lifetime. The author is able to easily revise not only the textual and graphical content of a document but also its structure. There is no time constraint on revisions to a MILO document. It is always possible to revise a document during the current MILO session, and it is also be possible to return to the document at a later date in order to carry out more revisions.

An author can read a MILO document from a file, restoring both the content and the structure of the document, and not only the textual and graphical content of each note but also the relationship between notes in the structure.

The title and textual body of each note can be edited using the text editor provided. They support commands for movement, text deletion, insertion and selection. Text can also be selected using a pointing device with movement and button clicks. Once text has been selected it can deleted or copied into another note within the MILO document currently being edited. Additionally it can be copied to any other application running under the X Window System and which recognises the X selection mechanism. Similarly, information can be selected in another X application and copied into a document being created or edited using MILO. This eases the problem of incompatibility with other systems.

The graphical content of each note can be edited in a basic manner. Objects can be added to the canvas at any time. The objects available are lines, rectangles, filled rectangles, circles, filled circles and text. A pattern in which drawing and filling takes place can be selected from a palette.

MILO therefore encourages the author to consider both structure and content by providing representations of structure, via which content is accessed. This structural representation can then be used for addition, deletion and relocation of document elements, without the need to manipulate large portions of text.

An author can alter the structure of a MILO document by: adding notes to the note hierarchy, deleting notes or altering the position of notes in the structure.

Collaboration

Cockburn and Thimbleby have highlighted the importance of a reflexive perspective of Computer Supported Cooperative Work (CSCW), 'an emphasis on the importance of catering for the individual's requirements and preferences in cooperative environments'. By blurring the distinction between individual and collaborative tools there is less effort required in transferring from one work mode to another and the group work environment is more predictable. This can encourage collaboration.

MILO is designed from this perspective, ensuring useful and usable facilities for non-collaborative authors so that collaborative use of the system can be eased into. Communicating authors using MILO will transfer information via purely textual electronic mail.

The author is supported in selecting a mail target by the system. A reasonable assumption is that authors will have a set of people with whom they regularly communicate, and when writing a document collaboratively, will in the main be sending the document to their coauthors.

As a result when the author has indicated that she wishes to send a document to a colleague, the names of people who appear in her mail alias file appear as buttons. Clicking on a button produces the electronic mail address for the person whose name appears in the button. Additionally the names of people who have contributed as authors to the current MILO document appear. This goes some way towards alleviating the problems of remembering complex electronic mail address, and keeping track of who has contributed to the document. MILO will structure the mailing command for the user, alleviating the need to remember its format.

Liveware

If collaborating authors simply exchanged chunks of document via e-mail, it would be difficult to keep track of different versions. It would be a great advantage if separate versions of a document, being worked on by collaborating authors at different sites, could automatically update themselves. This is the key idea of Liveware, developed by Witten et al. It is a novel approach to sharing data in social networks, designed to take maximum advantage of irregular communication for information exchange. Liveware is a concept rather than a system, and its principles can be applied effectively to a coauthoring environment. Witten et al. describe a Liveware database system, where automatic updates occur in order to maintain consistency and promote information sharing. This principle can be applied to MILO, with the notes that make up a document corresponding to the data elements of a database.

Liveware is used in MILO to automate the process of merging two versions of a single document into a new updated version. This is a useful facility for authors. Firstly it benefits coauthors of a single document as the work of each individual can be easily drawn together into a single cohesive text. There is no limit to the number of coauthors whose work can be amalgamated in such a way. Secondly, the facility can be used by the single author to combine work from multiple incarnations of the same document. Many drafts of a document are usually produced during the authoring process and automation can allow the author to easily retrieve elements from previous versions.

Viewing MILO documents

MILO provides two distinct types of graphical structure overview for documents. Firstly a tree-like graphic represents the hierarchical relationships between all notes contained within the document. Notes are represented using the text which has been entered in the title bar for each one. The graphical presentation reinforces the hierarchical model of a MILO document showing elements and their subelements.

Secondly, a graphic resembling an indented table of contents presents a linear representation of the structure. This corresponds to the mapping from hierarchical structure to linear document which is implemented if the author stores the structure in a LaTeX format file.

The first overview aids the author in manipulating the structure for optimal effect, arranging and rearranging ideas and sections of text. The second shows the author a potential linear form for the document. The two overviews should complement each other in helping the author refine the document.

In addition to wishing to take different views of the structure of a MILO document, an author may also wish to take different views of the content of the document. MILO allows a user to view portions of the document selectively by filtering notes within the document based on rules provided by the user.

It is sometimes difficult for an author to keep track of work on a document, especially if there are gaps between the work phases. Remembering which elements were being worked on, or the order in which tasks were being dealt with may pose problems which MILO helps to alleviate. It

provides a view of the document elements labelled with the time that they were last amended, and ordered from most to least recent amendment. The elements are represented by their title. An author can therefore very quickly see a history of work on the document and via this view access elements of interest.

MILO provides a spell checking facility that can be used within MILO, without the need to save the current document and access such a tool from outside. The spell checker conforms to British spellings, and errors are reported to the user within MILO, indicating in which note they have occurred.

Future Work

MILO is in use and initial, informal, observations indicate that it is a useful tool for writers in both a personal and collaborative context. It has been successfully used to write and amend the majority of a doctoral thesis (a single author task) and also to write and amend a co-authored paper.

The next step, which has just begun, is to gather feedback from use by a wider group of users, some of whom will already be computer literate and some will not. It is expected that this information will address both functionality and interface issues and it will have to be analysed from both perspectives.

What has already become clear is that attention should be turned to development of the Liveware aspect of the system, and provision of more flexible mappings of MILO structures to formatted hard-copy documents. The promise that MILO shows as a useful and usable writing tool will be built upon through these developments.

Steve Jones
Department of Mathematical and Computer Sciences
Dundee Institute of Technology
Bell Street
Dundee DD1 1HG

References

[1] Roger Took, "Surface Interaction: A Paradigm and Model for Separating Application and Interface" in *Human Factors in Computer Systems, CHI '90 Proceedings*, ed. Chew and Whiteside, ACM Press, Seattle, WA. 1990.

[2] Yair Wand, "A Proposal for a Formal Model of Objects" in *Object-Oriented Concepts, Databases and Applications*, ed. Kim and Lochovsky, Addison-Wesley, Reading, MA. 1988.

[3] Oscar Nierstrasz, "A Survey of Object-Oriented Concepts", ibid.

HyperCard: An Object-Oriented Disappointment

Harold Thimbleby*, Andy Cockburn, Steve Jones
Stirling University,
STIRLING, Scotland, FK9 4LA.

October 16, 1991

Abstract

Although HyperCard is claimed to be easy to use it has many limitations and curious features. It is further claimed to be 'object oriented.' This object orientation is also limited and curious. The disappointment is that HyperCard's arbitrariness and limitations are technically unnecessary, indeed result in error prone constructions, slower execution, increased learning effort. Yet HyperCard is successful: we will never know how much *more* successful it might have been had its designers employed any programming language design principles.

1 Introduction

HyperCard is a flexible flat database system with a graphical user interface. It is user programmable in a proprietary language, HyperTalk, and supports some object oriented programming possibilities. The purpose of this paper is to discuss the design of HyperTalk.

For a recent, late 1980s, programming language, HyperTalk is surprisingly arbitrary, and fails to show any obvious benefit from programming language design research (see [8] for a tutorial). We can guess: it appears that HyperTalk was implemented by a group of uncoordinated programmers, each being responsible for his or her own feature, from parsing to implementation. Despite the claims made for HyperCard (in particular in its user manuals which use 'object oriented' as an explanatory hook), technically, there are no obvious design rationales.

Following some introductory background material, this paper is structured first by HyperTalk features, and then it briefly examines HyperTalk's position with respect to object orientation. To have written the paper 'the other way around'—to start from object orientation—would have necessitated too many exceptions and tedious case by case discussion! The lengthy nature of our criticisms reflects the lack of structure in HyperTalk; and our written criticisms are by no means exhaustive of its weak features.

*Author for correspondence.

1.1 Background

When HyperCard was introduced in 1987 there was a great deal of publicity about the multitude of new possibilities that it presented for Macintosh users. It was ensured a large user-base because it was bundled free of charge with all new Macintosh computers, and was provided at a nominal fee to existing users. So HyperCard has become something of a *lingua franca* among Macintosh users.

HyperCard's roots lie with MacPaint, a painting program, and one of the very first Macintosh applications. Both were designed by Bill Atkinson, indeed HyperCard incorporates the graphical facilities provided by MacPaint, and more: HyperCard also includes facilities for information storage, structuring and recall, which is where its power lies.

> "HyperCard is a new kind of application—a unique information environment for your Apple Macintosh computer. Use it to look for and store information—words, charts, pictures, digitized photographs—about any subject that suits you."
>
> *from the HyperCard user's manual*

HyperCard's innovation was claimed to be the manner in which information could be structured associatively, reflecting human thought patterns (well, that was the idea). The information is presented via a user interface which includes graphics, text, and sound. Various actions can be invoked via mouse clicks, pointing at different parts of the pictures or text, or at *buttons* (which are discussed at greater length below). But the most powerful idea behind HyperCard is that it can be programmed in a fairly simple language, HyperTalk, and a lot of this programming is so simple that the user doesn't think of it as programming. Indeed some common operations can be programmed 'by example' (such as generating the code for a button to link one card to another) and from simple menu choices. More details of the language can be found in [4] and [5].

Thus, when the user's manual says

> "Any piece of information in HyperCard can connect to any other piece of information, so you can find out what you want to know in as much or as little detail as you need."
>
> *from the HyperCard user's manual*

... the "connecting to other pieces of information" is really done by simple bits of program. If a user doesn't like the connections, they can change it—or even get HyperCard to play a little tune instead, or do anything else that the programming language allows.

Interactive systems are notoriously difficult to program well. Casual Macintosh users, therefore, were given the ability to write their own Macintosh programs without much of the time-consuming toil. One of the features of HyperTalk is that it is easy to read: this means that an ordinary user can read other people's working HyperCard programs and very easily convert them to their own purposes.

There are two important practical features in HyperCard to encourage new users. First, much 'programming' can be done by pointing and clicking: the user is 'led' through some programming decisions, and does not need to know the vocabularly a. yntax to make simple applications work. More complex features can be provided by coding in HyperCard's language, HyperTalk. Secondly, almost any HyperTalk command can be evaluated interactively (in the *message window*). It's a separate question why *all* commands cannot be executed interactively!

So, creating a graphical interface to demonstrate "what the real system will look like" is as easy in HyperCard as it could be using any system. In addition, interface objects can be made invisible (permanently or under program control), hence letting painted graphics 'show through,' enabling many styles of interaction objects to be simulated if they cannot be programmed directly. We have found that many beginning programmers gain satisfaction from discovering tricks of this sort: in turn, giving HyperCard an enthusiastic following that closer examination of its design does not bear out (see [6] for an example, and [7] for a study).

1.2 HyperCard's interface metaphor

HyperCard is built around a user interface metaphor of a stack of cards (as from a card index). In any stack the cards are the same size, and although real cards are made of stiff paper, it is better to think of HyperCard's as being made from a combination of acetate and paper. A HyperCard card, then, contains picture and text on its transparent 'acetate' layer or *foreground* and picture and text on its opaque 'backing paper' layer, its *background*. The purpose of having two layers is that a background may be shared between several cards, for example, to provide a common graphical theme. The paint graphics in the foreground can be black, white or transparent, so the background can be obscured selectively for particular card designs.

Unlike a real card index, HyperCard's cards may have active objects attached to them, so-called *buttons* and *fields*. In general, an object has associated program, called its *script* which consists of methods, in HyperTalk terms *functions* or *handlers*. In addition to scripts, objects have *properties*, for example, the coordinates of their position on the card.

Mouse actions, such as clicking the mouse button down or up, generate messages that are directed first to the object the mouse is positioned over. This is the primary way in which the user controls a HyperCard system: by moving the mouse over various objects (typically buttons) and clicking.

The card, background and stack itself are also objects. The card is additionally sent messages by HyperCard that are not generated by mouse actions, for example, keystroke messages and general housekeeping messages. Messages pass through the card to its background and then to the stack, and then through a list of stacks, finally to the distinguished stack (*home*) and then to the HyperCard application itself which implements default actions for messages that have not been otherwise processed.

Fields are used for displaying textual information and have several predefined features such as scrolling, opaque, shadow, visible lines. Buttons are more specifically used for carrying out actions when the mouse does something on them (such as being pressed, released, entering their boundaries). As with fields, buttons come from a fixed repertoire of flavours, including rectangular, invisible, iconic (that is, pictorial, and user (but not program!) editable). Fields and buttons can have their size, shape and position manipulated directly with

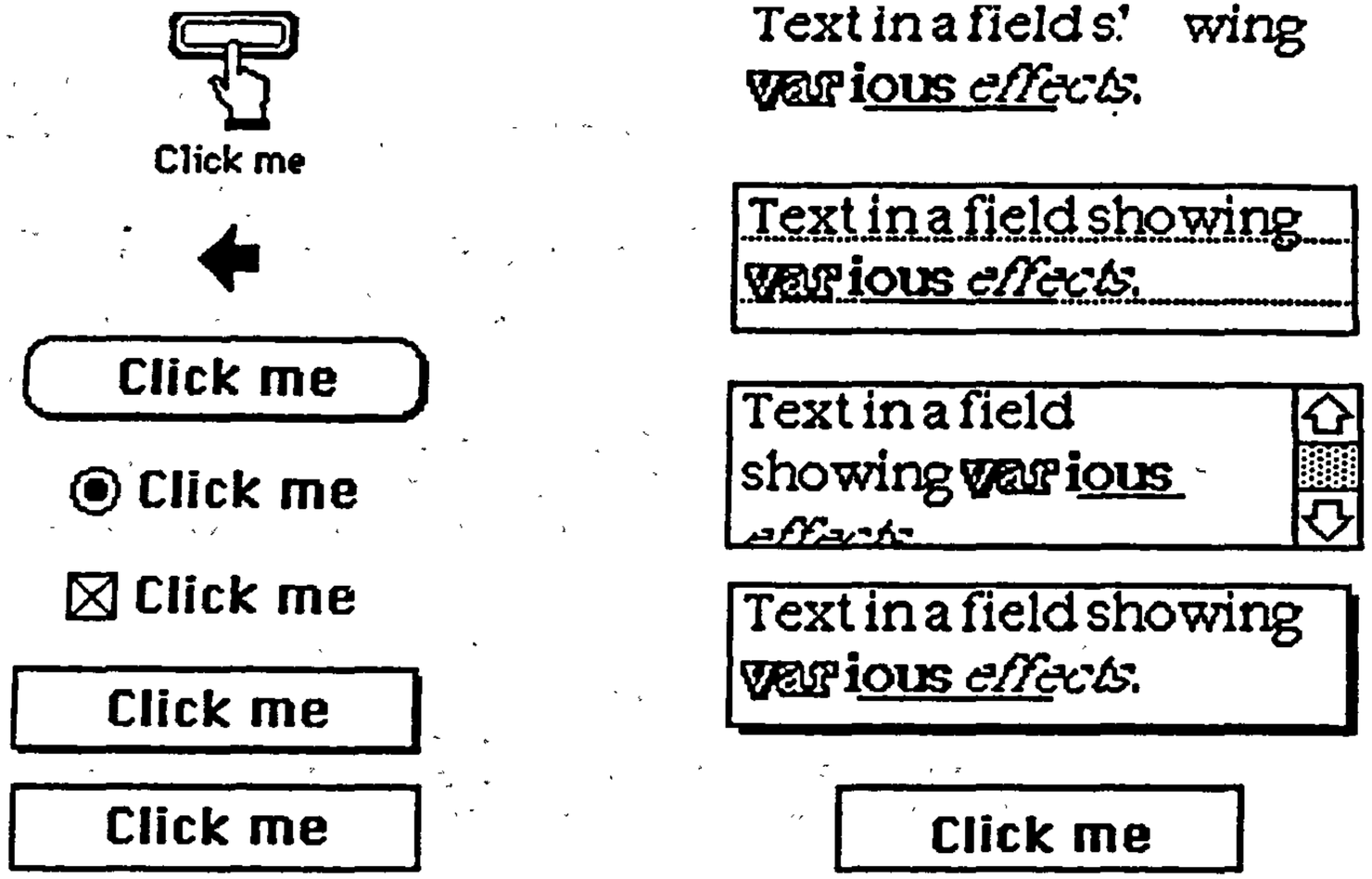


Figure 1: Example styles: button (left) and field (right). Buttons display their name or are entirely graphical (second example from top), fields display arbitary text.

the mouse (in the appropriate editing mode), and if required they can have an associated program, which specifies the actions to do when particular user actions are made.

Note that in the various editing and graphics modes, HyperCard does not send messages, therefore an object cannot know directly that it has been moved or otherwise modified by the user. Of course a development mode is required to debug faulty objects, but the flexibility of direct manipulation for moving objects is intrinsically denied the end user.

Although cards are numbered and arranged sequentially, backgrounds can be shared between non-contiguous cards. The user has various ways to move from card to card, optimised for linear operations (first, previous, next, last) and temporal 'recent' operations, such as 'go back.' Programmers may override this scheme or add alternative methods.

1.3 "Doing things at once"

HyperCard has the advantage that ideas can be added, bit by bit. There is no restrictive idea of 'the' program that has to be got completely right before it can be tried out: the user can change or add bits here-and-there as the fancy takes them, and try them out as they go along. In particular the presentation (graphics, object positions), content (text and/or graphics) and application (programming) can be done in any order, to suit the design process. (Of course this *laisez faire* approach has disadvantages for serious programming.)

*A very nice, a distinctive, feature is that everything can be done "at once"[9] (strictly, interleaved, since the user can only do one thing at a time): there is no set order in which a HyperCard program must be written. In contrast, in a conventional programming system, you first have to get the program right, provide its data, and—often the hardest part—design what it looks like. These are three separate and distinct phases of conventional design. Revisions to the overall design plan, motivated by experience or evaluation at any phase, may have unfortunate repercussions on earlier decisions. The problem with a conventional system is that designing what an interactive program should look like is difficult to do until you have had it working and seen (or got prospective users to see) how well it works: maybe it should be changed here or there. And if you do change the program to improve how it looks, this probably upsets some other part of the program. In a conventional programming language, it's easiest not to bother. The result is that interactive programs are often nowhere near as good as they could be. HyperCard, then, provides the opportunity to design better interfaces in an interactive style. A HyperCard programmer is generally happy to modify graphics and layout to suit customers' requirements as they change with experience of use.

1.4 Interaction objects are editable as objects

The amount of time and coding effort required to produce a complete system is greatly reduced because buttons and fields can be copied (including their associated program) between HyperCard stacks and then modified if and where necessary.

There are some example stacks provided with HyperCard and many sources of free and cheap stacks that are easy to obtain. Useful items (buttons, scripts, pictures, and so on) can be copied from them, and work the same way (bugs notwithstanding) in the user's own programs. In these aspects, HyperCard could have been an ideal vehicle for shareware and cooperative program development generally.

2 Syntax

There are many features of HyperTalk which make it attractive for prototyping, that is, getting experimental programs together quickly and easily. The pseudo-English style of HyperTalk attracts experimentation, which makes it easily and quickly mastered; it is vaguely reminiscent of SQL, though less powerful, less systematic and with imperative semantics. Learning HyperTalk doesn't feel the same as learning one of the 'hard' programming languages like C. A typical HyperTalk line might be

```
put "hello" before the first word of customerName
```

On the one hand, the Englishy feel to HyperTalk makes reading scripts very easy; on the other hand, the inevitable fussiness of the "English" means that you never quite know how to say what you mean—there are many little exceptions and idiosyncracies.

You can say `number of cards in this background` but you can't say `number of cards in this stack` (which makes just as much sense). You

have to say `number of cards`, since (for some reason) the qualifier '`in this stack`' which is allowed in other contexts is not accepted.

2.1 Examples of inconsistent 'English'

- You have to write `item 1 to 5` not `items 1 to 5` as is correct (for English, that is);[1]
- You can refer to the last word of a string `s` with `last word of s`, you can refer to several words with the form `word 2 to 3 of s`, but you cannot refer to `word 4 to last of s`, which instead has to be written as `word 4 to number of words in s of s`.
- One can say `lock screen` and `unlock screen` and equivalently `set the lockscreen to true` (or `false`); but the recent buffer which can also be locked or unlocked is only controlled by the `set` form. The potential form `lock recent` is not permitted. Similarly for the system message sending (`lockmessages`) for which only a `set` form is provided.

2.2 Examples of inconsistent syntax

Regardless of the claimed 'Englishness' of the syntax, many syntactic forms are internally inconsistent.

A principle that would have made sense is for any expression that takes an index (e.g., `card i` or `card field "x" of card 1`) could be preceded by `number of` and the index (here, `i` and `"x"` respectively) omitted to form `number of cards` or `number of card fields of card 1` respectively; but the latter is not permitted. In conformance with the flavour of HyperTalk, note that we would allow an optional English plural (as in `cards` or `fields`) to improve readability.

This is just one example where orthogonality—a basic syntactic principle—has been lost in favour of *ad hoc* rules. This makes HyperTalk much harder to learn than many other languages. There is no way for the user to generalise his knowledge about the language: each construct *for each class of object* has to be learnt individually. One of the advantages of object oriented languages, polymorphism, is lost even at the syntactic level in HyperTalk.

The operator `number` has other meanings, and this perhaps explains its confusing and limited syntax (though it invites other design questions). Thus if `x` is an object (rather than an expression) `the number of x` obtains the number of that object. Hence `number of this card` yields a number, being the number of this card within the stack. Yet `number of this card in this background` gives the *same* value as `number of this card`, say 2—despite `go card 2` and `go card 2 of this background` which generally go to different cards! The issue is that `number of` an object gives the value of that property (the number) of that object, which is the same regardless of the expression evaluating to the object. This example shows there is no way to determine the number of a card (that is, its numerical offset) within a background, since the number property of a card gives its offset in the stack regardless. The opportunity for a conceptually simplifying identity law has been lost on HyperTalk.

Again, the `there is a` predicate should be able to work with any object expression, but it cannot. For example, although `go card 10` and `go card 10 of stack x` are both correct expressions, only `there is a card 10` is acceptable.

[1] The selector `menuitem` selects items from menus (since the context is unambiguous, it could have been `item`): however constructs of the form `menuitem 1 to 5` are not permitted.

Dangling else

HyperTalk has 'dangling elses.' There is no need for a modern language to suffer from this confusing problem! HyperTalk suffers with panache.

The syntax of the HyperTalk `if` command is defined as follows:

```
if condition then statement [else statement]

if condition then statement
[else statement]

if condition then
  statement-list
[else
  statement-list]
end if

if condition then
  statement-list
else statement
```

The reference manual further requires that a nested multiple-line `if` must have its own `end if`. However, this is not a sufficient condition as the following example (which is accepted by the implemented parser) shows:

```
if c1 then
  if c2 then s1
else s2
```

This, containing a nested *single*-line `if`, has two possible parse trees (one of which is incomplete). Since by definition HyperTalk disambiguates with the rule that `else` associates with the nearest preceding `if`, in this case, depending on the intended meaning, one is required to write either:

```
if c1 then
  if c2 then
    s1
  end if
else s2
```

or

```
if c1 then
  if c2 then s1
  else
    s2
  end if
end if
```

Surprisingly the following example is correct despite the `else` association rule: a comment (taken from the symbol '`--`' to the end of the line) contributes a 'statement,' making the `else` associate with the first `if`:

```
if c1 then
  if c2 then s1
  --
else s2
```

The following forms are allowed in practice, as is reasonable, but are not defined in the manual:

```
if condition then statement
else
  statement-list
end if
```

```
if condition
then statement
else
  statement-list
end if
```

2.3 Syntactic sugar

The language definition claims to allow `the f` and `f()` as alternatives, but *only* if `f` is a built-in function (such as `time`). Yet `the long time` has no equivalent using the `()` form! In practice, however, if the user has redefined a built-in function, say `time`, then `time()` obtains the value of the user's function but `the time` still returns the value of the built-in function—so the forms are *not* equivalent. Although the user can override built-in functions like `time` with their own definitions, they cannot simulate modifiers (like `time`'s `long` and `short`).

2.4 Abbreviations and synonyms

Most HyperTalk keywords can be abbreviated. Some synonyms permit US or UK variant spellings, such as `hilited` and `highlighted`; on the other hand, `sharedhilite` has no English variant.

Thus `background` can be written `bkgnd` or `bg`; `card` can be abbreviated `cd`. There are a large number of abbreviations, and there is no systematic rule for their derivation. The shortest abbreviation for `card` is `cd`; the shortest abbreviation for `background` is `bg`; yet the shortest abbreviations for `button` and `field` are `btn` and `fld` respectively. Finally, some words only occur in a (partially) abbreviated form, such as `editBkgnd` (but not `editbg` or `editbackground` as might be expected from the abbreviations for `background`). `Button` is an abbreviation for `card button`, but `field` is an abbreviation for `background field`. Some words are abbreviated one way, others another (the word `abbreviated` itself can be abbreviated as `abbr[ev[iated]]`).

Some words are alternatives, e.g., `in` and `of` are synonyms in some contexts, for example: `word 5 of x` can be written `word 5 in x`. The `of` in function applications, however, such as `number of` has no synonym. Or `is` and `=` are synonyms (`x = 5` and `x is 5` are equivalent); yet though `5 is a number` is correctly interpreted as `true`, `5 = a number` causes an obscure error.

2.5 Object naming

Although fields and buttons are conceptually very similar HyperCard makes a clear distinction between them, and provides different visual styles for them. *Fields* can contain and display) arbitrary text, whereas buttons can only display their name. A problem for buttons is that their representation to the user has to be the same as their name; one might want, for example, a button either showing 'on' or 'off.' Doing this would mean that the name of the button in the program has to be 'on' or 'off' accordingly, hence encouraging the programmer to refer to it by other, less direct, means.

Since names of objects can be changed, there are several alternative reference mechanisms.

A so-called ID (guaranteed to be unique within scope and over time; IDs are not recycled) can be used, or a number. However, both of these may change as the structure of the HyperCard system develops (even as a program runs), so there is in fact no generally reliable way of referring to objects. For example, card numbers depend on the linear position of a card within a stack, and this is readily changed by resorting or other such operations. The ID of a card may change when it is cut-and-pasted (of course, one or other name inevitably changes when it is *copied*-and-pasted, since this action creates a new object, a copy of the card). Although button and field IDs are supposedly unique per card, the object ID will change when it is pasted to a new card. Curiously, the manual claims only that IDs change when an object is copied.

Apart from the expressions `me` and `target`, then, all object references are variable—and unreliable. Quality programming in HyperCard requires an unusual level of self-imposed programming discipline, either not to change names (which restricts what can be done) or to change them in carefully prescribed ways (which anyway won't be very secure)! Where user requirements call for objects with names that do change (such as the button example mentioned above), there may be no alternative to risky coding.

To confuse matters further, there is no syntactic distinction between object names and numbers: the numeric value `3` is indistinguishable from the character literal `"3"`. Hence `card "3"` refers to a card named `3` yet `card 3` generally refers to card number 3. Consequently if the name of a card is a numeral, ambiguity is inevitable. The obvious work-around is to name objects with non-numeric names, but this can be a problem for some applications (e.g., where a program naturally subscripts the objects), but a constructed name might be used (e.g., `X2`, which is not a numeral, but can be constructed from an expression such as `"X"&n`). The problem becomes much more serious when referring to buttons, since the user interface may require a button to be represented to the user as 1, 2, 3: if such names must be visible there is no sensible solution except *ad hoc* simulation using paint graphics.

Clearly both the syntax and semantics of object naming represents a major weakness in HyperCard.

Is there a reliable way to refer to objects? An object can refer to itself using `me`, so long as `me` occurs lexically in the object's script. If the object is a card, background or stack, it can refer to itelf using the form `this card`, `this background` or `this stack`. No other possibilities are reliable.[2]

Further confusion arises since asking for the name of an object that does not have a name initialised instead obtains an expression, and indeed one that cannot be used as a name!

[2] `The target` is supposedly the object to which a message is first sent. If a card, however, receives a message, then sends a message to a button, `the target`—whether evaluated in the button or card—incorrectly yields that button.

- First, consider the normal case, when a card has a na , x, say. The expression `short name of this card` obtains the value x, and this may naturally be used in an expression such as `card short name of this card`. (The alternative form `name of this card` obtains the expression `card "x"`.)

- Now, suppose we have just created a new, as yet unnamed, card. The expression `the short name of this card` obtains a value like 'card ID 4523.' Yet as the expression `card card ID 4523` fails, one asks why the name couldn't have been returned as simply `ID 4523`, or—since there is a method to obtain an ID—that `the short name of this card` should return an empty string: the card supposedly has no name!

In short, the potential identity: `card the short name of this card` ≡ `this card` fails, and, indeed, there is no way to determine whether a card has a name or not. Things are further confused if an object name happens to be, say, `card ID 123`, which would be the outcome of executing `set the name of this card to the name of this card`, or equivalent.[3]

3 Semantics

Such HyperTalk confusions are syntactic; there are plenty of semantic confusions too. For example, and not exhaustively,

- `put x & return after last line of x` loses the return character if x contains just one line.

- Dividing by zero results in the value `INF`. For example, `put 1/0 into x` puts the value `INF` into the variable x. This value may be used further, with the expected arithmetic results: `put x+1 into x` leaves x as `INF`. Yet even though `x = INF` is true directly after executing `put INF into x`, executing `put x+1 into x` results in an error (whereas before it worked)—because the numerical value `INF` generated by arithmetic can be operated on as either a number or string, yet the value denoted by the literal `INF` is not treated as a number. By similar methods we can arrange that `x = y` is true, with `x is a number` unequal to `y is a number`. So much for the equivalence of strings and numerals!

  We might also add that the blank characters `space` and `return` are treated as the numeric value `0` and `is a number` is true of both; yet `tab` is not a number.

- Menus are available as a return separated list of menu items. Hence `line 3 of menu "edit"` yields 'Cut,' since cut is on the third line of this menu. Yet a menu is initialised by a command such as `put "Undo,-,Cut,..." into menu "edit"`—which uses commas, not returns: the entire menu is all, apparently, on line 1!

[3] One would normally `put the name of this card into ...` a variable which is later used to `set the name of the card to ...`; the statement shown above simply shows clearly the obscurity of HyperCard's approach to naming: setting an object's name to be its own name sometimes changes that name!

- When HyperC d was extended into HyperCard2, an additional property was added to cards, namely a Boolean `marked`. New commands were introduced to test and manipulate this property (e.g., `mark all cards` and `print marked cards`). One asks why only a Boolean (why not a standard variable?), and why only cards?

- It is possible to insert stacks into the inheritance path dynamically. However, a stack is not sent an initialise message when it is placed in an inheritance (*used*) path (so it can't initialise itself).[4] All handlers in a used stack come in scope, so there is no encapsulation: a used stack cannot contain hidden auxilliary handlers. Although a used stack cannot initialise itself or anything else (since it does not know when it starts being used) it does not have access to its own objects, so, apart from menu entries, there is very little it could initialise!

- It is possible for any object, including a used stack, to send messages to other handlers (hence programming a limited form of encapsulation). But the `send` primitive can only send messages to command abstractions, it cannot send messages to functions. Hence function calls cannot be encapsulated.

3.1 Data structures

In general it is not possible to construct data structures, though assuming certain restrictions (that data contains no spaces, commas or returns) it is possible to use `lines`, `words` and `items` to simulate up to three dimensional values; four dimensional arrays of bytes are possible, under the same restrictions (using the additional selector, `character`). Neither structures with named fields (as in Pascal) are provided nor are pointers. Items are always comma-separated substrings; even though some HyperTalk commands use returns and slashes for separators, it is not possible to select items separated by alternative characters. A `word` is normally a blank (space, return, tab) separated substring, unless it contains quote symbols.

Unfortunately, the variables provided by HyperTalk are limited in structure and accessibility; it often becomes necessary to store record components (*fields* in Pascal terminology) in hidden fields—involving a sleight of hand you feel should not be really necessary. Also, this trick changes the number of fields and will therefore impact on the meaning of other code. (For example, a script may want to clear all fields, but if some of these are components serving other purposes then they should not be cleared.) In short, there is no encapsulation for data structures.

3.2 Variable names

Unlike programming languages like C and Pascal, HyperTalk does not need you to declare or even initialise variables. The result is that when you accidentally miss-type a variable, everything may initially run correctly, but be incorrect. Indeed, an uninitialised variable has as its value its own name. This is a

[4] In contrast, stacks are sent messages like: `openstack` or `resumestack` when they are opened or resumed.

HyperTalk feature that might well be exploited by a programr. (to get default values),[5] with obvious consequences. This 'feature' results from the design decision that quote marks need not always be placed around string literals (thus "hello" and hello are equivalent, namely the strings hello, *unless* there is an initialised variable called hello). Note that this 'feature' also explains HyperCard's confusion over object names and numbers.

3.3 Generality

Not everything that the user does in HyperCard can be mirrored in HyperTalk. Thus there are occasions when the user will say, "I want to get that programmed" but won't be able to. Consider *dialog boxes*—where the Macintosh seizes up until the user does something (clicks on one of the few dialog box button offered). This user interaction is beyond the scope of HyperTalk even though in principle it could have been handled automatically. The result is that a number of facilities are available only to the interactive user and cannot be controlled from programs.

A specific example of a typical dialog box 'missed' by Hypertalk is the one that allows a user to interactively change the paint patterns: from within HyperTalk changed patterns cannot be detected, default patterns cannot be restored. (A user has to do each interactively.) This oversight means that program control of graphics is always unreliable. Given that a HyperCard card can be made to look exactly like a dialog box, a major simplification could have been achieved if they were treated exactly like cards (perhaps on a 'dialog box' background). This would have meant that any feature that could be controlled by the user from a dialog box could have been controlled from HyperTalk in the usual way (such as typing into fields, clicking buttons).

You can cut-and-paste most things in HyperCard (like pictures, buttons, text), but you can't cut-and-paste everything. You can't cut-and-paste backgrounds, for example, and these often contain lots of details that you therefore have to cut-and-paste one-by-one tediously. It is almost possible to write a script that cuts-and-pastes each object on the background: it fails on the graphics cut-and-paste since an attempt to cut may result in a diagnostic that there is nothing to copy (*sic*) and that one should try the background (the diagnostic obviously assumes you have tried copying (not cutting) from the card (not background) image!)

3.4 Error handling

The most severe problem of all is error handling. With some ingenuity and perseverance, the interpid programmer can overcome most problems. But errors still happen! But in HyperTalk, as soon as an error happens, the HyperTalk programmer has all control taken from him. For example, the HyperTalk program wants to copy a picture the user has drawn: but if the user has actually drawn nothing, this is an error. In HyperTalk, you can neither detect this situation to stop the error arising nor recover from it when it does happen—the *user* is given a dialog box and this takes control away from the program!

[5] An example is a repeat loop that tests the value of the variable flag, say, by the expression flag="flag" which is true by default until flag is assigned another value.

It would have been really easy for the HyperCard designers to arrange for each error to send a message, error "description of problem" to the object where the error occurs, to see if there is a method written to handle the error. With this approach (as used in Smalltalk) only when HyperCard itself catches the message error should it finally complain to the user. But this was not done, and the result is that any error causes catastrophe—and it is not always possible to 'program around' potential error-making traps.

3.5 Second class graphics

HyperCard has a paint model for graphics. But a HyperTalk program cannot 'see' what has been drawn; and generally the limited range of buttons available is a significant drawback (though a user, not a program, can paint new icons). Creating the effect of a controlled graphical object is difficult and requires tricks, for instance, placing the graphics under a button that can be dynamically made transparent or obscure. Using this method allows a 'one off' graphical button representation but if we want to move or copy the 'object,' then of course the separate items fall apart. Allowing graphical buttons or a whole new graphical object type would solve this major problem.

3.6 Automatic saving

A difficulty that a user (and particularly teams) must overcome is that of HyperCard's automatic saving of alterations. This feature is undoubtedly useful for the normal HyperCard user, avoiding the annoyance of an explicit save at the end of each session but it forces the programmer into the added burden of maintaining an arbitrary back-up regime.

3.7 Program structure

The main problem with HyperCard is the very thing which made it so popular with casual users—it is informal. The chatty, natural features of HyperTalk are not conducive to clear, compact, logically separated code sections and the problems with unstructured and invisible data stores aggravate the situation. The lack of rigour and plausible guidelines on where to locate methods (in the stack, on the background, cards, fields, buttons?) causes erratic, hacked solutions. One ends up with lots of global variables, and the associated difficulties of losing track of where they are initialised, and where they are used. The debugging assistance given to the programmer is minimal (and has its own bugs).

3.8 Variables and persistence

Naturally, data stored in HyperCard fields is persistent, in that it persists from one run of HyperCard to the next. Without this, of course, HyperCard's stacks could hardly function as databases! However, since field contents can be made invisible, they are a way for the programmer to implement persistent variables, whose values are maintained from one run of an application to the next.

HyperTalk provides global variables. Global variables persist during a single run of HyperCard. Thus, running a new application (stack) within the same session of HyperCard starts off with whatever global variable bindings were left by the last stack. Since there is no way for a program to determine what the names the current of global variables are, it is impossible to protect against accidental corruption of other stacks' data (stacks can be run concurrently, and therefore share the global name space).

Additionally HyperCard provides a *single* LIFO stack used for implicit transfer of control (we discuss the explicit go command below). The only operations on this shared data structure are push and pop. Since the LIFO stack is shared, one HyperCard stack can pop and go to a card pushed by another stack. It is possible to pop the top of the LIFO stack into a variable (without the side-effect of a go) to check whether it corresponds to a card in the current stack, but if it does not, push is not permitted to push a card from a different stack. Popping another stack's card is an irreversible error. Had HyperTalk provided a top function, to give the top element of the stack, it would have been possible to detect the error before taking irreversible steps.[6] In short, reliable algorithms cannot be written using push and pop: they can only make sense to the user, who like HyperCard, has a single view of all stacks running.

An obvious, conventional, but missed, solution to such extent problems is that HyperCard variables (including LIFO stacks) should have been instance variables of appropriate objects, for example individual stacks. (We describe below a similar situation, with menus, but having a more immediate impact on the user interface.)

4 HyperCard as a programming environment

HyperCard is an integrated programming environment; external interfaces (so-called *XFCNs* and *XCMDs*) notwithstanding, applications in HyperCard can be constructed entirely within the environment. HyperCard provides a text editor for editing object programs and a very simple debugger. It is also possible to monitor message sending and variable values (incidentally, the only way to determine the names of global variables).

4.1 HyperCard as a prototyping tool

It is a major selling point for Apple that all applications for the Macintosh have the same 'look and feel' through a common graphical interface [1]. HyperCard, however, is being used to prototype and create graphical interfaces for the Macintosh, yet does not support the standard objects that make up the Macintosh interface. It does not support a Macintosh-like style of interaction, for example, there is no mouse double-click message.

Experienced HyperCard developers may claim that these problems can easily be solved by using external commands (XCMDs) and external functions (XFCNs). These are pieces of code written outside of the HyperCard environment in another language such as Pascal or C, compiled and then integrated into a HyperCard application (using Resedit). This is the kind of activity that users wanted to get away from in the first place! If you have to be a 'real' programmer then nothing has been gained.

[6] As well as omitting top, HyperTalk does not provide a predicate to test whether the LIFO stack is empty.

4.2 Structuring programs

HyperCard provides no browser; indeed, the basic environment does not provide facilities for finding or printing entire programs. (Utilities can, with some difficulty, be written in HyperTalk.) Since objects can be made invisible (for user interface reasons or to maintain state from one run to another) it is very easy for programmers to loose track of their systems.

When debugging, viewing hidden fields can be complicated, involving HyperTalk commands passed through the message box. If, for example, you can't remember the fields' names (or whether they're on the foreground or background) finding the required field can be very awkward. And hidden fields is only one of many tricks you get led into, trying to circumvent little restrictions that, perhaps, at first seemed to make learning more interesting.

4.3 Compilation

It is clear that many design decisions in HyperTalk mitigate against efficient compilation (e.g., the confusion of literals and names). The very varied syntax and semantics from command to command suggests that there was no systematic attempt at organising HyperTalk. It is no surprise, then, to find that HyperTalk is very slow, even in HyperCard2 which compiles scripts on demand.

4.4 Practical limits

HyperTalk permits recursion, but to a limited depth of only 95 calls (in version B1-2.1). There is no tail recursion elimination. Messages sent from handlers, even when the last thing the handler can do (e.g., using the primitive pass), are also very limited (24).

Suppose we have a group of cards on a common background, and we want all cards to go to the next card (perhaps invoking a visual effect, which we don't discuss here). Since HyperCard sends the message opencard, we can handle this in the background as follows:

```
on opencard
  -- visual effects, etc. omitted
  go to next card
end opencard
```

Such code may be conceptually simple but it is not reliable! One only needs about twenty cards before a limit is reached and the code fails catastrophically. This code, indeed, is tail recursive: there is no technical reason why, regardless of limited recursion or pending message depth, that it should be so restricted. Given that stacks can have thousands of cards, imposing such unjustifiable limits in the programming language implementation is incomprehensible.

It is possible to program around this particular problem. (One approach is to use global variables to communicate with an idle handler in the background.

Since HyperTalk sends the message `idle` when it is doing nothing else, it follows that `idle` will only be invoked when the recursion and message stacks are empty.) But why force programmers to be so sophisticated or devious?

5 Object orientation

We have come so far in our criticisms without mentioning object orientation. The *nice* part of HyperCard's object orientation is that objects (cards, fields, buttons) can be copied, cut and pasted as units. As such the user can operate on them with the standard select/cut/copy/paste/clear operations that are used for text editing. A copied or cut object 'carries' with it its associated program code and other properties (apart from its number and ID).

Pasting an object will generally change the message hierarchy for that object (for example, we might paste a copy of a button onto a different card, or even a different stack), so semantics are rarely preserved following a move. Furthermore, groups of related objects (for example, radio buttons[7]) that refer to each other will be compromised: there is no reliable way to refer to objects that is also unaffected by cut-and-paste operations: IDs and numbers change, and names (even if they are not changed under the normal operation of the objects themselves) may not be unique.

5.1 Graphics

Since HyperCard's graphics model is paint there are no graphical objects as such. Buttons, which are objects, have associated icons (square bitmaps of limited size), and they may be moved under program control to provide very simple animation. However, the graphical design of an icon is *not* under program control (although a program can choose an icon from a given repertoire, it cannot edit one or confirm it has changed the icon to what it expects).

5.2 Assignment

HyperTalk permits components of objects to be either the source or destination of assignments (puts). For example: in the command `put word 3 of x into word 1 of x` the component `word 3 of x` is the source, and the component `word 1 of x` is the destination. This might lead one to think that if you can write `put name of this card into x` that one could equally write `put x into name of this card`. You can't. Instead, a special syntax has to be used for certain properties of objects: as in `set name of this card to x`. In an object oriented language one would have expected operations such as setting the name of an object to have been achieved by sending messages; instead HyperTalk has a special (non-message) primitive (`set`) for this purpose. Users, then, cannot extend or adapt the properties of an object.

[7] A button may be highlighted when clicked. Of a group of radio buttons only one may be highlighted. Hence clicking on any button in a group must refer—by ID, name or number—to the highlighting of all other buttons in that group.

5.3 Classes

There is no concept of class; there are exactly five kinds of object (stack, background, card, field, button), not counting menus (see below). The hierarchical relationship between these objects is fixed.

HyperTalk supports a go operation, with a new twist. It is used to change the current card, but in doing so also changes the inheritance path to be through the new card and background (and possibly stack). However, the current method continues to run—meaning that its name bindings change on each go.

It is not possible to subclass objects. For example the common need for a group of mutually exclusive radio buttons has to be programmed explicitly. It would have been preferable to have a radio button class to instantiate as groups of related buttons.

5.4 Messages

Objects may send messages to other objects, either along the current inheritance path or explicitly to named objects. Two object names (*viz*, the object where the currently executing method resides and the object where the last system-generated message originated) can be obtained from primitives.

Since it is not possible to define one's own classes the utility of messages is greatly reduced, and few HyperTalk programs use messages except as procedure calls. A lost advantage of binding methods with a class of objects is that sending the same message to various objects can have a suitable effect (the classic example is sending a `print` message to, say, an integer or a date). In HyperTalk, if a method to handle a message is positioned in the hierarchy so that different sorts of objects inherit from it, then it will require explicit code to sort out what to do for each sort of object that is the target of the message! In a conventional object oriented language, the method to handle the message could be associated with the class of object concerned.

In HyperTalk, this means that if the programmer wants all buttons (buttons cannot be grouped into subclasses) on a card to respond to a new message `cycle`, say, then the method for that message must be placed in the *card's* code. Hence all fields also inherit the same message, as does the card itself. The `cycle` method must then use code to determine whether it was really invoked by a button.[8] Since this requires explicit programming we can hardly call HyperTalk 'object oriented': we can do similar explicit programming in other languages with no pretentions to being object oriented. And in our description of this example we have not mentioned that there are foreground and background buttons on each card, and the syntax to refer to them differs. Hence it follows that, once the method has determined to its satisfaction that it is indeed called from a button, the action it implements in response will have to be written out *twice*, once for the foreground and once for the background cases.

Such are the more easily explained semantic problems of messages. The syntax, too, of message passing is also peculiar. Although message handlers can take parameters containing commas, there is no simple way to send a message that contains parameters including commas or quotes. Thus, what

[8] We've already noted that the primitive required for this, the `target`, is bugged!

might have been written as `send cycle("a,b,c") to button 2` has to be written: `send "cycle(" & quote & "a,b,c" & quote & ")" to button 2`. It is not possible to send function messages to other objects: `send "put f()" .to ...` evaluates `f` in the current object regardless.[9]

Messages can also be attached to menu items, and the same problem occurs there. We discuss menus next.

5.5 Menus

Conceptually, menus are objects with a carefully prescribed user-behaviour. The user clicks on a menu and it reveals a list of choices, *menuitems*. If the user selections one of the options the corresponding action is invoked.

HyperCard implements this behaviour with *two* alternative mechanisms. The default is that a message `domenu X, Y` is sent to the current card for a menu selection *X* from a menu *Y*. The interpretation of a default menu item is under the control of the current card, background and stack (in that order). This has the advantage, perhaps, that the exact meaning of a menu item can be associated with the current context (i.e., the currently displayed card). On the other hand, if the menu really needs a fixed meaning (and not one that can be implemented by a built-in primitive of HyperCard), there is no way to implement it reliably.

The second method is to bind a *menumessage* to the menu item. Now, when the item is selected, HyperTalk will send this arbitrary message along the same path the `domenu` message would have been sent. Despite its name, the menumessage can be a command, in particular a `send` to send the required message to any desired target object.

For some reason, menumessages are restricted to a single line: unlike the code associated with all other objects. Thus it is not possible for a menumessage to check that the target object of a `send` exists, and it cannot be certain to invoke the right handler to check for it! Given that HyperCard permits many stacks to be run together, each possibly with its own menus, reliable menu programming is extraordinarily difficult.

In terms of inheritance, buttons and fields are associated with cards or backgrounds; cards are associated with backgrounds; and all with stacks. Thus, as the user moves from one card to another, if the background changes, HyperCard (naturally!) changes the current inheritance path through the new background. But menus are different. As it were, HyperCard doesn't know about menus, and the programmer has to control them explicitly. Thus it is not possible to associate a menuitem with a particular background, say, so that when the user moves from that background the menuitem disappears (or, better, is sent a suitable message so that it can decide what to do, perhaps overriding the default action).

The consequence of this is that menus and individual menuitems must be controlled explicitly by the programmer. This is a problem of considerable complexity: other stacks, running concurrently, may also alter menus and they may not adhere to any useful conventions. Menus are an object oriented disappointment.

[9] One problem is that `send` effectively applies the function `value` to its parameters, stopping quoted (verbatim) messages being sent.

5.6 Inheritance

If an object intercepts a message it may still want the default action to occur. HyperTalk provides no way to refer to the immediate inheritor of a message, that is the object that should implement the default behaviour (though one could program it in explicitly—which is the option for languages that are not object oriented!)

The mechanism that HyperTalk provides is `pass` which sends the *original* message on. Thus an object cannot modify the message it handles.

6 HyperCard: an object oriented disappointment

The previous part of this paper described HyperCard from the point of view of features, and served to both introduce HyperCard and to indicate various design criticisms, criticisms with respect to programming language design generally and object orientation in particular. Having provided an overview of HyperCard, then, we now briefly discuss it from the point of view of object oriented design principles.

Uncontentiously following [3], an object oriented system should provide the following functionality. We place a + before features adequately provided in HyperCard and a − before features not provided or badly provided:

− **Encapsulation.** In HyperCard, objects' state are available without restriction. Even the methods of an object may be changed by another object.

+ **Dynamic lifetime.** Objects *can* be created and destroyed as HyperCard executes. (All garbage collection is implicit.)

− **Identity.** Object naming in HyperCard is a mess; names are not unique.

+ **Substitution.** With various provisos, HyperCard objects can be substituted; indeed the cut-and-paste editing takes advantage of this potential.

− **Message.** Messages are provided, with various restrictions, both syntactic and semantic.

+ **Method.** Methods in HyperCard are termed *handlers*.

+ **Receiver and self.** HyperCard permits access to the receiver of a message and the object self, the object whose method is executing. HyperCard uses the expressions `target()` and `me` respectively—note that one is a function, the other a pseudo-variable, and that (as mentioned earlier) `target()` is unreliably implemented.

− **Class and instance.** It is not possible to draw together objects sharing common behaviour.

− **Instance variable.** The only variables provided in HyperCard are local variables (i.e., local to methods), global variables and fields (i.e., database entries). With ingenuity it is possible to use object properties (such as `name`) as an instance variable, but with severe restrictions. Instance variables (e.g., `marked` for cards) are not inherited.

+ **Inheritance.** Inheritance is provided, and is also used as the 'mechanism' to support non-HyperTalk extensions to HyperCard (XFCNs and XCMDs). If a message is not handled by a stack (in the manner described earlier), it is sent to the stack's *resources* which include the XFCNs and XCMDs, then through an ordered list of stacks (which can be dynamically configured) and their resources, then to a final stack ('Home'), to HyperCard's resources, operating system resources, then ultimately to the HyperCard system itself.

– **Multiple inheritance.** Multiple inheritance is simply not provided. A peculiar form of multiple inheritance can be simulated by using go in objects, since this changes the inheritance path, but at the cost of various side effects.

7 Summary

To conclude, HyperCard is very widely used as an application development tool, and this is because of the very real ease with which attractive graphical interfaces can be created using it. Its popularity has proven the need for such a prototying tool.

It is interesting that, even though Apple must be aware of this, they have not developed a tool specifically for this task. Have they missed the chance to exploit a user need, when at the same time they could have made their machine far more accessible as a programming platform? Or, thinking more cynically, do political considerations concerning Apple's investment in Macintosh software development and links with third party developers have a large part to play in this issue?

Apple's own HyperCard stack design guidelines [2] perhaps shows their attitude: excellent as they are for making systems look nice, they say nothing about programming in HyperTalk. And conversely, HyperCard appears designed to break Apple's own user interface design rules (e.g., not handling double mouse clicks). HyperCard systems are doomed, it seems, to look nice but feel terrible when you get down to using them. HyperCard will remain an excellent and inspiring prototyping tool, but a system that just does not go far enough for anything like serious development. Few are the HyperCard programs that you would let anyone else use. *Serious HyperTalk programming requires non-trivial compromises to be made in the user interface.*

HyperTalk is depressing: there are so many missed opportunities that it is impossible to say "HyperTalk fails such-and-such well known principle"—rather, HyperTalk fails principles wholesale. Tony Hoare once said that Algol 60 was so well designed that it was an improvement over many of its successors. Following almost three decades of research and practice, HyperTalk failed to learn anything from the Algol heritage or any other programming language developments. It would not be unfair to say that HyperTalk is a distinct step back over all its predecessors. We dispute the view expressed by Greg Kimberly (of Apple) that "HyperCard's missing features *are* a feature of HyperCard"[7]—unless he means no features are provided except by their absence.

Whatever we think of it, HyperCard *is* a success: it is free and one of the most widely used programmable systems (beating Lotus-123 and BASIC). As Apple's CEO John . ulley has said, "With HyperCard, virtually anyone can become a software author, producing an information-based application that looks like a professionally designed Macintosh application." That is why it is successful. It is a failure because software authors can produce information-based applications that look, and only look, like professionally designed Macintosh applications.

References

[1] Apple Inc., *Human Interface Guidelines: The Apple Desktop Interface*, Addison-Wesley, 1987.

[2] Apple Inc., *HyperCard Stack Design Guidelines*, Addison-Wesley, 1989.

[3] M. C. Atkins & A. W. Brown, "Principles of Object-Oriented Systems," *Software Engineer's Reference Book*, J. A. McDermid, ed., pp39/1–39/13, Butterworth-Heinemann Ltd., 1991.

[4] Claris Corp., *HyperCard Script Language Guide*, 2nd. edition, 1990.

[5] G. F. Coulouris & H. W. Thimbleby, *HyperProgramming*, Addison-Wesley, in press.

[6] J. Gervich, "How I Learned to Stop Worrying and Love HyperCard," in B. K. Laurel, ed., The Art of Human-Computer Interface Design, pp131–133, Addison-Wesley, 1990.

[7] J. Neilsen, I. Frehr & H. O. Nymand, "The Learnability of HyperCard as an Object-Oriented Programming System," *Behaviour and Information Technology*, volume 10, number 2, pp111–120, 1991.

[8] R. D. Tennent, *Principles of Programming Languages*, Prentice-Hall, 1981.

[9] H. W. Thimbleby, *User Interface Design*, Addison-Wesley, 1990.

Reducing Requirements in Computer Supported Writing Tasks

Steve Jones
Dept of Mathematical & Computer Sciences
Dundee Institute of Technology
Bell Street
Dundee DD1 1HG
email : mctsrj@uk.ac.dct

Andy Cockburn
Dept of Computing Science
University of Stirling
Stirling
FK9 4LA
email : agc@uk.ac.stir.cs

March 1993

Abstract

Although many computer systems support single author writing tasks, few address the fact that writing can be a highly collaborative activity. However, the commercial success of computer systems which support group work in general has, as yet, been limited.

In this paper we address a problem that we believe has hindered the acceptance of groupware, and must be avoided in the design of groupware for writers: that of imposing requirements on users.

We accept that to support such complex tasks as group writing, systems require information about the task. However, an all too easy method of gathering this information during operation of the system is to 'ask the user'. This imposes requirements on users for both information and action.

We argue that these requirements are unecessary and that the onus should be shifted on to systems (ie their designers) to collect information from sources other than the user.

We suggest strategies for reducing requirements in groupware and show how they have been successfully implemented in MILO, an asysnchronous collaborative writing tool.

1 Introduction

Computer Supported Cooperative Work (CSCW) (Ellis *et al.*, 1991; Rodden, 1991) is concerned with the development of theories and tools which facilitate, support, and enhance the process of carrying out computer mediated group work. CSCW tools, which are termed *groupware*, can be divided into several categories. Some are concerned with the coordination of group tasks, others with communication between group members. This paper is concerned with a further class of tool, that which addresses the act of collaborative authoring.

The commercial success of groupware has, as yet, been limited, and many prototype systems have failed to progress to the marketplace. This is perhaps surprising, considering the extent to which everyday tasks are performed in collaboration with others. Possible

reasons for this are varied. First, many systems have failed to correctly address the needs of users in appropriate ways. Second, insufficient commercial interest has existed to emphasise the potential of groupware in people's everyday tasks; it is only recently that large developers such as Microsoft and Lotus have produced applications which are specifically concerned with cooperative work (Microsoft Windows For Workgroup and Lotus Notes, for example).

This paper addresses the first reason, and considers techniques for encouraging use of groupware, and making it more productive. In light of the second reason, and growing interest in CSCW systems this is all the more important. Effective theories and principles for the design of groupware must be developed and implemented, to ensure that bad design is eradicated, an good design is encouraged as the groupware market expands.

2 Groupware for writers

How then is this paper concerned with writing?

Document production has long been a major use of computer technology, with develpements in word processors encouraging the widespread use of computers as writing tools. However, almost without exception, this expansion in availability of software tools for writing has failed to address the fact that it can be a collaborative task. In fact, we may view all writing tasks as being, to some extent, collaborative.

Even single author tasks can be affected by comments from others on the product and the constraints that the potential audience introduce. Approximately 20% of the papers presented at this conference are explicitly collaboratively authored, and we would contend that a large proportion of the others have been the subject of review, suggestions, and proof reading by individuals other than the named author.

The current increase in commercial development of groupware has failed to address the collaborative nature of writing. These systems are primarily concerned with coordination and communication (consider the proliferation of elecronic mail systems and the popularity of electronic mail as a communication medium). We suggest that this is because the act of writing collaboratively is highly complex. Systems which support it must be concerned not only with the writing act itself but with the processes involved in group work and are therefore far more difficult to develop.

As a result of this, current groupware, most noticeably that for writers, is too prescriptive. Because of the complexity in supporting group work, systems impose inappropriate *requirements* on users. They take a system centered rather than user centered view for information capture. By reducing requirements in such systems, user effort can be reduced and redirected

to user centered activities.

We go some way towards addressing this problem in the following sections, providing high level design aims for collaborative writing systems and lower level strategies for achieving them. The aims and strategies are part of a larger set described by Cockburn and Jones (Cockburn, 1993; Cockburn & Jones, in press). They will be useful in the development of such systems and help to promote their use and efficacy by easing the author's task. The implementation of these aims and strategies is illustrated by reference to MILO (Jones, 1993b; Jones, 1992c; Jones, 1992b; Jones, 1993a; Jones, 1992a), a system which supports the work of both single and collaborating authors.

3 Requirements imposed by groupware

We consider there to be two types of requirements imposed by computer systems on users.

Explicit requirements are actions which users *must* carry out for the benefit of the *system* rather than for immediate personal or group benefit. Explicit requirements are unavoidable, and must be met for use of the system to be initiated or continued.

Implicit requirements occur when the system's facilities do not conform to users' needs. The user must augment the functionality of the system by adopting his or her own techniques within the contraints of the system.

In the user's view of working with the system these the actions undertaken to meet both types of requirement are unecessary and interfere with prosecution of the task being undertaken. Examples of such requirements are given below.

System dictated actions can be quite easily avoided. For example, the text editor being used to write this paper has a word search facility. However, the user is required to be aware of where possible occurences of the word might be in relation to the current insertion point, and to indicate this to the system. A mistake results unexpected responses, perhaps even that the text is not present, when it is but in the other direction. This required awareness and action is explicit, yet can be avoided through implementation of a more simple searching technique—one direction only with wraparound when the end of the text is reached.

An implicit requirement of the editor under consideration is that work is saved by the user at regular intervals to avoid loss of text in the event of a system failure or suchlike. The editor could easily save the work at regular intervals to a backup file.

To some extent these examples are general user interface issues, applicable beyond writing applications. However, such issues are also important in groupware, and in conjunction with specific groupware oriented requirements (which will be discussed in later sections) can

unecessarily increase user effort.

3.1 The origin of requirements

We see requirements as products of the design and implementation stages of system development, in both single user systems and groupware. In a groupware writing system the designer might suggest a model of group interaction which is to be embedded in the system. In this model, participants are assigned roles which determine their possible activities with the system in the task. It is the job of the implementor to make this model concrete, but how will the system 'know' what the role of each group member actually is? The simple answer is to get the system to 'Ask the user', and so a requirement is imposed. The 'ask the user' syndrome is widespread in computer systems, yet can often be avoided. In later sections we describe how problems such as this one can be overcome without 'asking the user'.

While we accept that requirements will be necessary in some circumstances, we aim to minimize them and to relate them to costs and benefits to the user. Rather than impose requirements (which are system directed) systems should be *guidance-free* (Cockburn & Thimbleby, 1993). Guidance is the information demanded of the user by systems. In a guidance-free system, the provision of information to the system, is initiated by the user because the cost of doing so provides an acceptable level of benefit. Successful use of the system should not be dependent on supplying such guidance.

Historically, groupware has been rife with requirements, perhaps because of the complexity of supporting group work brought about by its diverse and dynamic nature. Coordination systems have required participants to enter diary details, communication systems have required users to classify received messages themselves or provide information within messages so that they can be classified by receiving systems (Malone *et al.*, 1988; Malone *et al.*, 1986). Writing systems have required participants to describe and conform to determined roles.

We contend that even given the complexity of group work, guidance-free systems can, and should, be promoted over those that impose requirements, and in the following sections consider this with respect to writing support systems.

3.2 Requirements in collaborative writing systems

This section presents examples of both explicit and implicit requirements in collaborative writing systems. Requirements of individual systems will vary, but those presented serve to exemplify common ones facing users, and we later show how they can be reduced.

3.2.1 Explicit requirements

Some collaborative systems such as Quilt (Leland *et al.*, 1988), require that participants be registered at the outset of the task. Users must enter their own details or this may be the job of a coordinator, but unless this is done participation is not possible. As participants join the group it is again required that they are registered before carrying out any work, and an implicit requirement is that they are deregistered if they leave. This does not effectively support the dynamic nature of such group work. Why *is* it necessary for all participants to be registered? The answer is that the system needs to have records of members to enforce roles.

This leads to another requirement; each registered member must be allocated a role within the group. The choice of roles may comprise options such as author, commenter, or reader, and the system then ensures that the actions of participants does not extend beyond their assigned roles. But why do users have to supply this information for the benefit of the system? In a non-computer based task such roles would be socially mediated. It may be desirable to restrict the activities of some participants, but the system may, in fact, be able to do this without demanding information from the user.

Access rights to the product of group work are also an issue, both in synchronous and asynchronous environments. They concern the scope of operations that individuals are permitted to carry out on parts of, for example, a document. This is a more complex problem than it first seems. One view might be that once roles have been allocated, they apply to all products of the group's work, so an author would be able to generate new text at any point in the document, and a reader would not be permitted to make any changes. Another view may be that permissible actions are dictated by 'owners' of sections of the document. A member's role may be author or editor, but they cannot amend or insert text into anyone else's work. This complexity has driven systems to require users to provide the information required to impose appropriate access rights; defining the scope of the applicability of each member's role.

Synchronous editing systems typically require explicit transfer of control between participants, even when participants are in visual contact. This is implicit in most of our everyday collaborations or communcations. Explicit transfer is not always necessary, and the frequency of such interruptions into a dialogue can be reduced.

Some systems (including an initial version of MILO) require that ownership or authorship of parts of the product of the group's work (such as document sections) be explicitly specified. This information may be used by the system in presenting information to users or to enforce

the access rights and roles that the user has already be required to describe. This can be tedious for users, and we shall show how the system may gather this information from elsewhere.

3.2.2 Implicit requirements

When systems impose explicit requirements, the act of meeting them is forced by the system. The user has no choice. Implicit requirements are not so immediately obvious to the user and become evident as experience with the system grows. They are actions which the user must carry out to fill gaps in functionality of the system, or to enhance effective completion of subtasks which the system does not support appropriately.

Referring again to the text editor being used to record these ideas; the text on the screen appears to be broken into lines of 80 characters, yet with experience it has become obvious that this is for display purposes only. The stored text only contains line break characters where they have explicitly been inserted. The implicit requirement for our purposes (to submit the text to a text formatting application) is to insert a line break at the end of each line.

In collaborative writing, coauthors require version archiving (the importance of a historical record of the collaboration is stressed by Miles et al (Miles *et al.*, 1993). This may apply to archiving of individual contributions, or to the text as a whole. Few systems provide support for archiving in this way, implicitly forcing users into developing and adopting their own protocols.

A further *user* requirement in group authoring is the merging of the contributions of each participant into a coherent whole. In general this is achieved in one of two ways. In the first, authors contribute to, and amend a single version of the document, stored in a single location and accessible to all group members. In this case integration takes place as the document is developed but limits flexibility of change; there is an emphasis on ensuring that the document is 'correct' before proceeding to the next stage. In the second, more flexible case, members generate and amend their own versions of the document or document sections. At appropriate stages integration of the individual contributions takes place. Few systems provide support for this process, implicitly requiring that users develop their own techniques for carrying it out (Baydere *et al.*, 1993).

During collaborations, group members need to communicate ideas, questions and contributions to colleagues. As stated earlier, groupware tends to address a specific class of task such as communication, coordination or writing. CSCW writing systems tend not to provide

support for the communication inherently necessary in the task, implicitly requiring users to adopt their own strategies for deciding what is to be communicated to others and when, and which tool to use. Of course, systems where a single version of the document is accessed makes information *available* to colleagues, but does not communicate the scale, location, time or intention behind the changes. There is no reason why the support of communication cannot be integrated into writing systems.

3.3 Strategies for reducing explicit and implicit requirements

This section outlines strategies for reducing requirements in collaborative writing systems Ways in which the strategies have been implemented in an asysnchronous collaborative writing system, MILO are described.

It is important to keep in mind the change of emphasis that is required in the relationship between system developers and users. It is all too common that implementors turn to user for information that will facilitate or ease the provision of functionality. It is our belief that developers/systems should *demand* information or action of users only as a last resort. The onus should be on the developer to find ways of gathering the same, or equally useful information from within the system, from the environment in which it being used, or from records of previous task which have been undertaken. However, this should not rule out support for those users who are willing to provide the information. In such cases the user's effort should yield appropriate benefits.

3.3.1 Don't depend on user actions

The system should not be dependent on users carrying out certain actions for it to function correctly and effectively, nor should users be precluded from actually using the system if certain actions are not carried out (such as specifying roles). However, the system should be prepared to accept and make use of guidance, and provide commensurate benefits to users. It is important that users are aware of what guidance can be provided and will be exploited, and that the costs of doing so will provide acceptable benefits.

3.3.2 Use whatever information is available 'for free'

This strategy uses the term 'for free' from the user's perspective; no effort is required of the user to enjoy certain functionality provided by the system. The previous strategy seems to produce conflicting aims; leave users free from requirements yet provide effective functionality.

However, information can be gathered from sources beyond that explicitly provided by users. Information is often accessible through the process of communication. Electronic mail messages, for instance, contain information about who sent them, where from, when, who else they have been sent to, subject matter, and the path they have traced to arrive.

'Knowledge' about the collaborative process can be inferred from such information. For example, the regularity of previous communications can be used to predict arrival new messages and flag absence of expected messages, or overdue repsonses. Reports can be produced on group dynamics; who communicates most with whom, about what and when? Are subgroups emerging for which explicit support can be provided?

The product of a collaborative writing task, the document itself, can be a rich source of information. This can be subjected to analysis to facilitate actions such as invoking spelling correcting options, provide information on style, highlight differences between versions of the document (perhaps to aid in integration of multiple contributions and so on. MAFIA (Lutz *et al.*, 1990) and LSI (Foltz & Dumais, 1993) are systems which analyse text to provide facilities to authors and other users.

The environment in which the system is active can also provide information useful in the task. For example roles in a collaborative writing task might be automatically allocated with reference to each group member's status in the system. Issues include: what access priveliges does each member have throughout the filestore? What protection does each member apply to their own data (are they open with their information perhaps?). What class of user are they in? For example user groups in a university UNIX system such as staff, postgraduate, student, guest might provide a framework for automatic allocation of roles and priveliges. What are the favoured computer tools (predicted by analysis of commands) of each of the members? Will this ease or hamper integration of the document? Can a consensus be suggested?

Historical information about previous collaborations using the system can also be a rich source of guidance for the system. What were the roles and priveliges of members in previous tasks? Were they amended in any way during previous tasks? How much of the product of previous tasks were members responsible for and so should they be promoted/demoted? Which group members communicated most frequently? Which members adhered most reliably to schedules and deadlines? Are there participants in previous collaborations who could have useful input for the current task?

It is obvious that 'asking the user' is unecessarily a first resort in many instances. With effort *on the part of the system (ie its designer)* requiring actions or information from the user can be relegated to a last resort.

4 MILO

MILO is computer based system which supports the work of both single and asynchronously collaborating authors. It has been developed by the first author of this paper. Its purpose is to support and enhance the writing process and support the needs of coauthors.

At the most basic level of use, an individual author can utilise MILO as a text editor to construct a simple document. In such circumstances it provides similar functionality to text editors such as EMACS or *vi*. However, at the other extreme it can meet the needs of geographically distant coauthors who are writing a continually revised document of complex structure containing both text and graphics.

MILO document structure

Documents in MILO are constructed of a hierarchy of logical units called *notes*. This is a similar approach to systems such as Notes (Neuwirth *et al.*, 1988) , NoteCards (Trigg & Irish, 1988), HyperCard (Apple Computer, Inc, 1987) and InterNote (Catlin *et al.*, 1989). Authors are provided, by default, with a graphical representation of this structure, but can also access alternative views of both structure and content. Elements in the hierarchy can be added, deleted, copied and repositioned by direct manipulation of objects on the screen. Notes whose place in the document structure is as yet inclear, can be created and integrated into the structure when appropriate. Users are not required to consider the place of a element of the document before they wish to. In systems such as HyperCard, for example, when an element is created it is immediately assigned a place in the overall structure which the user must alter if necessary. Each note can contain text and/or graphics.

Providing a context history for free

MILO generates information about the authoring process 'for free', drawing on information sources other than the user. For example, a user can view a history of each note, showing who created it and when it was created, and who last amended it and when this took place. As operations are carried out on notes the system collects this information from the environment, that is the author's username and the time. This information also serves to provide alternative views of the document, such as time ordered. This shows where and when in the structure changes have taken place, from most to least recent, and also who made them. This goes some way towards providing the context history of interaction which Miles et al (Miles

et al., 1993) advocate, allowing authors to readily see the most recent additions of colleagues or to remind themselves of their own focus of activity in the document.

Use of heuristics

When the document is saved to backing store, heuristics are used to map the hierarchy to a linear structure, which is in LaTeX format (Lamport, 1986), plain text or a textual representation of notes and their relationships which can be read by MILO to reconstruct the document. The user is not required to consider the mapping of elements to objects in a printed document such as sections, abstract and so on, and can concentrate on the structure and content. To provide alternative linearisation heuristics is almost trivial.

Use information from the environment

Communication with colleagues is supported from within MILO. Authors are not required to remember or structure the electronic mail addresses of colleagues or electronic mail commands. Users are provided with a graphical interface containing the names of colleagues who they regularly email, and the names of other contributors to the current document. Documents, or portions of documents can be electronically mailed by direct manipulation; clicking on the button containing the name of the target and providing an indicator of the information to be sent. The system seeks the first type of information from the environment (specifically the user's mail alias file). It is important to note that the user is not required to have a mail alias file in order to communicate in this manner. Many users do, but if such a file is not present, the cost of creating one (providing guidance) has benefits for *the user*. The information for the buttons containing the names of other contributors to the document is readily available as it has already been collected for other purposes in the system.

Make as much use of free information as possible

This highlights an important point. Effort in collection of information 'for free' by the system can pay off in a variety of areas within the system. Its value can go beyond the purpose for which it was collected, as it can facilitate and enhance other user activities. For example, using the information gathered as notes are created or amended facilitates provision of a filtering facility for users. Authors can selectively view notes which conform to specified attributes. For example, those created by a certain author, those created after a certain date

and so on.

Indeed the small amount of information that MILO holds could be to also generate a report on the collaborative process, showing which elements underwent the most work, the volume and rate of contributions of each author, and providing graphs of activity on the document as a whole or sections of it in relation to time.

Avoiding implicit requirements

Through its note-based metaphor, information collection techniques and merging utility (described later in the section) MILO implicitly supports a variety of collaborative writing strategies without requiring the user to specify which strategy is to be used. Sharples et al (1993) outline two approaches that may be adopted for document creation in the kind of distributed, asynchronous task which MILO supports. Longitudinal partitioning entails dividing it into a sequence of stages, each of which is completed by a different individual or group. MILO supports this approach implicitly through its encouraged division of the document into logical elements, and provision of information about them. In parallel partitioning different individuals or groups produce different sections of the document concurrently. This is also supported by MILO. However, as Sharples et al point out, merging of document sections to form a single cohesive text may be a slow and difficult process with this approach. MILO avoids this implicit requirement by providing a semi-automated merging utitlity.

Liveware (Witten *et al.*, 1991) is a novel approach to sharing data in social networks. It is designed to take maximum advantage of irregular communication for information exchange. By applying heuristics which consider the information 'freely' available about notes such as author, time of amendment, contributions of multiple authors are merged. This can also be used by a single author to combine work from multiple incarnations of the same document. The heuristics used are easily extensible, which opens the opportunity of providing the user with a pallette of approaches to merging contributions.

Users have more control over meeting implicit requirements

MILO does not require users to describe roles or priveliges of participants in a task. It emphasises social imposition of such attributes, yet would be easily extensible to adopt the strategies described in earlier sections in support of the role of social protocols. The requirements of the context in which it was developed did not facilitate such desirable extensions.

This approach may be seen as a technique which avoids explicit requirements yet imposes

an implicit one; the users must manage the collaboration. However, a prescriptive approach by the system in such a case may impose inappropriate practices on users. In this case, the implicit requirement provides participants with greater flexibility and adaptability in roles and actions during the task. Yet guidance may still be provided, such as amending the merging heuristics for the Liveware facility.

5 Summary

We have considered the negative impact that the explicit and implicit requirements that systems impose on users can have. Such requirements may involve user actions or provision of information. This is an unecessarily system centred approach, in which user effort is expended to benefit the system rather than the user.

We promote the reduction of requirements; users can choose to carry out actions or provide information to the system and receive benefits commensurate with the effort involved.

Strategies have been suggested to minimise system requirements and move towards guidance-free systems: don't depend on user actions, and use whatever information is available 'for free'. Examples have shown that the 'ask the user' approach to providing or enhancing system functionality can be demoted to a last resort rather than a first resort as it currently seems to be.

The strategies have been shown to be applicable to a useful and usable writing tool, MILO. It is evident that the gathering of relatively little information from sources other than the user can facilitate the provision of many diverse utilities.

In conclusion then, we stress that minimizing requirements does not necessitate minimizing functionality.

References

Apple Computer, Inc. 1987. *Apple Macintosh HyperCard user's guide.* Apple Computer, Inc.

Baydere, S, Casey, T, Chuang, S, Handley, N, Ismail, N, & Sasse, A. 1993. Multimedia conferencing as a tool for collaborative writing. *In:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Catlin, T, Bush, P, & Yankelovich, N. 1989. InterNote: extending a hypermedia framework to support annotative collaboration. *Pages 365–378 of: Hypertext '89 Proceedings.* Pittsburgh, Pennsylvania November 1989. ACM Press.

Cockburn, AJG. 1993. *Groupware design: principles, prototypes, and systems.* Ph.D. thesis, University of Stirling, Stirling, Scotland.

Cockburn, AJG, & Jones, SRA. in press. Four principles for groupware design: encouraging adoption and easing system use. *In:* Dourish, P (ed), *Implementation perspectives on cscw design.* Springer-Verlag.

Cockburn, AJG, & Thimbleby, HW. 1993. Reducing user effort in collaboration support. *Pages 215–218 of:* Gray, WD, Hefley, WE, & Murray, D (eds), *Proceedings of the 1993 international workshop on intelligent user interfaces, orlando, florida. january 4–7.* New York: ACM Press.

Ellis, CA, Gibbs, SJ, & Rein, GL. 1991. Groupware. some issues and experiences. *Communications of the ACM*, **34**(1), 38–58.

Foltz, PW, & Dumais, ST. 1993. Personalized information delivery: An analysis of information filtering methods. *Communications of the acm*, **35**(12), 51–60.

Jones, S. 1992a. Designing computer systems to support collaborating authors. *Pages 134–141 of:* Rees, MJ, & Iannella, R (eds), *OZCHI'92, Interface Technology: Advancing Human-Computer Communication.* Ergonomics Society of Australia.

Jones, S. 1992b. Milo. *ESRC data archive bulletin*, October, Software Bulletin, S1–S3.

Jones, S. 1992c (August). *A user's guide to MILO: a computer based system to support (co)authors of structured documents.* Technical report. Department of Computing Science and Mathematics, University of Stirling, Stirling, Scotland. T.R. 92.

Jones, S. 1993a. *Easing the writing task: designing computer based systems to help authors.* Ph.D. thesis, University of Stirling, Stirling, Scotland.

Jones, S. 1993b. MILO: a computer based tool for (co)authoring structured documents. *In:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Lamport, L. 1986. LaTeX: *a document preparation system.* Addison-Wesley.

Leland, MDP, Fish, RS, & Kraut, RE. 1988. Collaborative document production using Quilt. *Pages 206–215 of: Proceedings of the 2nd conference on computer supported cooperative work (cscw '88).* portland, oregon. september 1988.

Lutz, E, Kleist-Retzow, H, & Hoernig, K. 1990. MAFIA – an active mail-filter-agent for an intelligent document processing support. *ACM SIGOIS Bulletin*, **11**(4), 16–32.

Malone, TW, Grant, KR, & Turbak, FA. 1986. The Information Lens: and intelligent system for information sharing in organizations. *In: Proceedings of the chi '86 conference on human factors in information systems.* ACM Press New York.

Malone, TW, Grant, KR, Lai, K-Y, Rao, R, & Rosenblatt, D. 1988. Semistructured messages are surprisingly useful for computer-supported coordination. *Pages 311–331 of:* Greif, I (ed), *Computer supported cooperative work: A book of readings.* Morgan Kaufmann.

Miles, VC, McCarthy, JC, Dix, AJ, Harrison, MD, & Monk, AF. 1993. Reviewing designs for a synchronous-asynchronous group editing environment. *In:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Neuwirth, C, Kaufer, D, Chimera, R, & Gillespie, T. 1988 (March). The Notes program: a hypertext application for writing from source texts. *Pages 121–141 of: Hypertext '87, University of North Carolina, Chapel Hill, North Carolina, November 13–15 1987.*

Rodden, T. 1991. Survey of cscw. *Interacting with computers: the interdisciplinary journal of human-computer interaction*, **3**(3), 319–353.

Sharples, M, Goodlet, J, Beck, E, Wood, C, Easterbrook, S, Plowman, L, & Evans, W. 1993. A framework for the study of computer supported collaborative writing. *In:* Sharples, M (ed), *Computer supported collaborative writing.* Springer-Verlag.

Trigg, RH, & Irish, PM. 1988 (March). Hypertext habitats: experiences of writers in NoteCards. *Pages 89–108 of: Hypertext '87, University of North Carolina, Chapel Hill, North Carolina, November 13–15 1987.*

Witten, IH, Thimbleby, HW, Coulouris, G, & Greenberg, S. 1991. Liveware: a new approach to sharing data in social networks. *International journal of man-machine studies*, **34**(3), 337–348.

Trust in CSCW

Chapter for *Computer-Supported Cooperative Work*,
S. A. R. Scrivener (ed) Ashgate Publishing, 1993.

Harold Thimbleby*, Steve Marsh*, Steve Jones†, and Andy Cockburn‡

May 18, 1993

Abstract

CSCW technology is about facilitating, supporting, and enhancing cooperation between people. In this chapter, we focus on the role of trust in cooperation, specifically between people collaborating on shared activities mediated by computers. We emphasise the central role of trust in CSCW systems, suggest a formalism for trust, and discuss the application of trust in CSCW systems. We contend that consideration of trust during the design of computer support for collaborative work provides a unifying platform for the social, technological, and work-task issues that each frequently dominate the design and acceptance of CSCW applications.

1 Introduction

CSCW is taking computer support of people's activities to a new level of complexity. This entails consideration of a much greater range of issues than is required when developing single user systems, but the potential benefits are impressive. CSCW is still an emerging field: until recently, much CSCW work was purely academically driven. However, commercial software developers are now prepared to invest in CSCW. Systems such as Lotus Notes, Windows For Workgroups and others, are attending to the inadequacy of computer support for cooperative work. These commercial systems will start to mould users' behaviour and tasks, in turn to define users' expectations. It is important then, that new generations of CSCW tools are appropriate, acceptable, and truly supportive.

There are several ways to view CSCW. It is often treated as a technological concern: technology can support certain activities, and some system is implemented to exploit this, but then fails to work as hoped in the real and complex social milieu. CSCW may also be treated as a social concern: people have complex relationships within groups, within subgroups and externally with other individuals. This approach fails when the technology

*Computing Science, University of Stirling, Stirling, Scotland.
†Computer Science, Dundee Institute of Technology, Dundee, Scotland.
‡Computer Science, University of Canterbury, Christchurch, New Zealand.

is inadequate and does not support nuances of human communication. Another way of viewing CSCW is as a way of getting a job done, but this fails when inappropriate constraints and requirements are imposed on participants both at a group level and a personal level: the users' preferred work methods may differ from those imposed by the system.

CSCW needs a unifying concept that at once spans technical support, social issues, and work performance. Such a concept may be used for clarifying design, or explicitly supporting cooperative work practice. We suggest *trust* as a suitable concept.

This chapter argues that trust is an important underlying concept in successful CSCW that should be acknowledged and exploited in the new generation of CSCW applications, either in the systems themselves, or in the design and participatory processes that lead to the requirements. As yet little concrete work has been done with trust; this chapter is a first contribution. Our aim is to raise issues, and to formulate them in a way that can be carried forward and evolved.

Introductory comments

Trust. The word 'trust' is widely used in an informal way, even in otherwise formal settings: Thomas claims that whether we should trust computers is one of the most important questions facing the computing industry [Thomas, 1989]. Witten notes, "to trust a computer system as such is meaningless, for machines are neither trustworthy nor untrustworthy" [Witten, 1987]. Whether, how, and to what extent, one can trust users at the other end of a computer network is the issue we address in this chapter.

In the following sections we outline the relevance of trust to CSCW, and review existing definitions and discussions of 'trust.' In providing a formalism of 'trust' we make it—rather, an aspect of it—a precise concept. We argue that the failure of CSCW systems can frequently be interpreted as a poor understanding of the central role of trust (however defined) in collaborative activity. Furthermore, we contend that explicitly accounting for issues of trust in CSCW can be extremely beneficial, both as a design activity, and as an explicit activity of the computer-supported task itself.

Trust is a tool for constraint in activities involving others.

Trust is used and controlled implicitly in many activities, even where the everyday use of the word might seem inappropriate or judgemental, and when no judgement is intended. In everyday activities our actions are dependent on trust in people, objects and organizations. For example, we trust a mechanic to service our cars, but not to diagnose an illness; a mountain climber must trust his rope; we (mostly) trust banks to retain the money that we deposit with them. The level of trust that we adopt changes from situation to situation, and is influenced by a host of judgements.

Trust, therefore, is highly dynamic, and is present in our activities at *appropriate* levels. Hence, describing trust as a tool for constraint indicates that it may, in fact, impose no constraints at all.

Computer-supported cooperative work. CSCW is about cooperation between people who wish to work together by *optimally* constraining everyone's (including their own)

behaviour to bring about beneficial outcomes. Without any constraints, life would be too full of distractions and uncertainties for effective work to progress. But people in a CSCW system are unlikely to completely trust each other (or themselves, perhaps knowing themselves to be forgetful). However, they almost certainly trust each other enough to make some progress.

Since CSCW attempts to allow people to work together, it follows that such collaborative working should be made as easy and painless as possible. In this way, a harmonious relationship between group members can be achieved quickly and easily, and developed to attain a strong working group. However, from the list of issues above, which is far from exhaustive, it is clear that harmony in collaborative work is a tenuous balance between individual, social, technical, and task-dependent factors. Trust, a factor that is tacitly present in groups task without computer support, plays a substantial role in maintaining this balance.

CSCW systems can be viewed as an attempt to foster and increase the trust between members of a working group, and within the members themselves. We may not completely trust ourselves or others to do things properly—on time, in time, the 'right way,' with enough detail, and so forth. This is a major obstacle in attaining any working relationship at all, and more so in attaining our goal of a harmonious relationship between members of the working group, be it localised or geographically displaced.

CSCW should, then, allow members of the group to work together confident in the knowledge that what needs to be done will be done, in time and as agreed between members. A framework is required that can reflect and support this. Trust is such framework, and we propose the application of this framework to CSCW.

2 Trust and CSCW

We concentrate on one central reason for CSCW: the activities of groups are complex, and computers can support complex activities. CSCW, then, helps users manage shared and complex activities. Consider some of the issues present in a generic collaborative task:

Personal and group perspectives of work. Each participant may carry out tasks in an individual manner, yet with the aims of the group in mind. The manner in which the participants do this may or may not be in accord with the other members of the group. The group must trust the individual to satisfy the work requirements, and vice versa. (The motivation and advantages of merging personal and group perspectives on work are discussed in [Thimbleby *et al.*, 1990, Cockburn & Thimbleby, 1991].)

Coordination and work commitment issues. The tasks that members carry out should be coordinated so that the following issues are adequately satisfied: work is most likely completed on-time; the work product is coherent when individual contributions are amalgamated; the tasks are carried out by those most skilled to do them; difficulties with personal relationships are avoided; responsibilities are clearly defined; and so on. Those

people committing to a course of action are trusted to achieve completion; should they fail to do so, they are likely to become less trusted.

Communication requirements. Successful coordination requires communication. Directives and information must be communicated in time to the appropriate people in an appropriate manner. Members may utilise different modes of communication such as facsimile, telephone, electronic mail, hard copy, and what is received from other members must be integrated into the work processes of a participant. Collaborators trust their systems to convey the information that they transmit, and to convey the information that is directed to them.

The roles of group members. The members roles may or may not be clearly delineated. There may be a hierarchical structure to the group, or it may be an egalitarian organisation. Who is responsible to whom? Whose opinion is most respected? Can members stray out of their allocated task? In addition, the roles that are established at the outset of the task may change during its course. People may be promoted or demoted within a hierarchical structure, or a hierarchy may emerge from a structure where all members begin as equals. The tacit assumption that all participants view others as deserving of their allocated roles may not be accurate. One member may be designated a role and be viewed by one colleague as worthy of more responsibility or a different role, and be viewed by another colleague as less able, or trustworthy, in that role. Again these views may change with time, and not necessarily in a global manner. A formalism of trust must therefore account for the highly dynamic processes involved.

Other related issues. There is very considerable work on cooperative problem solving; we just mention [Clarke & Smyth, 1993, Fischer & Reeves, 1992] as pointers to this literature. Social issues, such as etiquette and manners are also covered elsewhere [Brotz, 1983]. Of course, from a technical point of view, it is much easier to design systems that isolate users and do not rely on any level of trust. Cryptographic techniques can be used to ensure, for example, that only known people are in the work group, and to ensure that when they say they have done something, they really have. The practical organisational advantages of one cryptographic approach, one that isolates groups of mutually trusting users, is discussed in [Thimbleby, 1994].

3 Trust as a formal concept

Defining trust is a difficult task: everyone has their own view of it, and any specific definition is unlikely to satisfy everyone's views. Nevertheless, any definition is a useful framework. It raises and clarifies questions, and (for example) greatly helps in analysis of static configurations.

There have been several academic studies of trust, some with a theoretical flavour [Boyle & Bonacich, 1970, Deutsch, 1962], others are more sociological and non-specific

[Boon & Holmes, 1991, Golembiewski & McConkie, 1975]. Although there is no common definition, there are similarities which we use below to provide a classification of definitions. We use this classification to motivate our own definition.

Risk. Trust involves risk [Boon & Holmes, 1991, Deutsch, 1962, Golembiewski & McConkie, 1975 Risk may carry negative connotations, so we define 'risk' as entering into a situation where a possible outcome leaves the participant worse off than if he had not entered into the situation in the first place. This definition is broad. In terms of CSCW, a risk is taken by, say, group members that others will not indulge in malicious behaviour which adversely affects other members. Or perhaps that members have the skills which they claim. Or that blame for mistakes is not wrongly attributed, and so on. Risks such as these are inherent in group work and need to be addressed. It is sufficient to say that, in taking a risk, an element of trust is involved.

Cost and Benefit. Since a decision involving trust necessarily takes risk into account, there will be some form of cost-benefit analysis (most often implicit) in the decision to trust. Costs of broken trust, such as time lost if a task is not completed, money lost, even opportunities lost, should be taken into account, when considering a CSCW application. Benefits may include possible promotions for jobs well done, accounts gained from influential customers, and so forth. Balancing these costs and benefits will help the truster to reach an informed decision, and explicit balancing may be necessary in certain trusting decisions [Marsh, 1992].

Deliberation. In addition to the taking of risks, trust is *deliberative* [Marsh, 1992]. It requires that the truster actually make a decision about the trustee. The decision may be unconscious, and undoubtedly frequently is [Luhmann, 1979, Luhmann, 1990]; it is, however, still made. Decisions involving trust rely on past experiences of 'similarities' [Marsh, 1992]. Such similarities cover situations, trustees, time, place, and so forth. In the case of CSCW, similarities that may be taken into account include the person to be allocated a task, their previous record, either through personal experience on the part of the truster, or through say, a work record, and the task to be allocated—does experience dictate that the task is uncommonly difficult or large? Has a similar task created problems in the past? Experiences of, or knowledge about, any of these similarities would affect a final decision regarding trust.

Experience. A definition of trust provided by Good [Good, 1990] is: trust is confidence in how a person will act in the future, as a function of past and present claims (implicit and explicit) and of experience. The experiences considered when using trust centre on specific situations, the similarities between them, and their costs, benefits, and importance. In humans, experience is used to a large extent to determine familiarities between situations [Dechter & Michie, 1984]. In CSCW, experiences in cooperative work can be recorded (explicitly by individuals, cooperatively, or automatically through inferencing techniques) to

affect future interactions with particular people: information filtering systems, such as the INFORMATION LENS [Malone *et al.*, 1988], allow individuals to explicitly record their experiences (or trust) with people (*social filtering* [Malone *et al.*, 1987]); SYNVIEW [Lowe, 1985] supports collaborators in cooperatively recording their experiences with people.

Prediction. An agent may be able to construct a model of another agent's behaviour, and use it to try to predict what the other agent is likely to do. The relation between prediction and trust is complex; for example, if you can predict what someone will do, you can have blind trust in them. For our purposes, a problem with predictive modelling is its complexity and sensitivity to assumptions about the possible behaviours of other agents. In a practical CSCW context, we want to use trust, but we do not want to depend on details of, say, parsing natural language and knowledge representation for open-ended human tasks. (Instead, our approach is a drastic simplification compared to a prediction-based method.)

Agents. Muir's paper [Muir, 1987] provided a major advance on the notion of trust between humans and machines. Using a Human-Computer Interaction perspective, it provided a formal description of trust using variable values about trusting decisions. The description of trust presented below extends this work by providing a generalised framework for trust between 'agents.' In this description, an agent is a general entity existing in the world, and can thus be a computer program or a user. The resultant formalism simplifies Muir's agent-based description of trust.

A formalism for trust

The formalism presented here is concerned with cooperation between two (or more) agents, from the point of view of one of those agents, x, with reference to the other, y.

We notate "the amount x trusts y" by $T_x(y)$.

$T_x(y)$ has a value in the open interval $(-1, 1)$ (i.e., $-1 < T_x(y) < +1$); 0 means no trust; -1 would represent total distrust (these two are not the same).

If we simplistically assume that agents' behaviour (over a fixed set of outcomes under consideration) is *totally* determined by trust, then we can derive a definition of T from probabilities.

There are various definitions of probabilities; the simplest is the frequentist definition. The number of times a particular event occurs is its frequency, and the proportion of times, out of all events, is its relative frequency. As the number of events increases, the relative frequency tends to converge to a constant value, the probability of that event.

Alternatively, probability is the degree of belief that someone has in the event occurring, normalised in the foregoing sense. Thus, we may view a probabilistic basis for trust as either an empirical interpretation, or as based in something as subjective as belief. We might ask the user to lay bets, this being a standard way of obtaining subjective estimates

of probability; or we might fit the times of sending or receiving messages to a Poisson distribution; and so on.

Particular situations have particular levels of importance to agents, dependent on various circumstances, all, or mostly, purely subjective. We represent x's view of the importance of a task a by $I_x(a)$. In general we would consider **I** to be a vector, assigning importance to each possible outcome in a; simplistically, we will treat I as a scalar (i.e., all outcomes in a given situation have equal weight for a given agent). In an artificial life environment, the set of possible outcomes are likely known a priori, in which case **I** is sensibly a vector; in real life, we are unlikely to know the space of outcomes, and a scalar is a sensible choice.

$I_x(a)$ has a value in $(-1, 1)$. Whilst this is a subjective measure, in a human system an estimate of the importance of a particular situation may be relatively easy to ascertain. For a computer-based agent, there are many different ways of determining the importance of a situation, such as payoff functions (cf [Rosenschein, 1985]).

We represent the utility (cost/benefit) of situation a for x with $U_x(a)$, with value in $[-1, 1]$ —we normalise utility to be in this range. It might be conventional to use 'cost' as the weight, but it is more convenient to take a value (utility) that correlates with trust. Utilities can be negative, since they must be to cater for utilities involving a conserved benefit. For example, if utilities are monetary, then y obtaining a benefit of $+£1$ from an outcome affecting only x and y implies x has cost $-£1$ for the same outcome. (Usually, money is conserved!) Again, we take utility as a scalar, rather than a more general vector. In CSCW, computers can overcome preservation of psychological benefits, such as 'effort.' By processing information, extracting relevant facts, and presenting them in a suitable manner, significant benefits can even be provided 'for free' [Cockburn & Thimbleby, 1992] to some, or all users, and at no cost (negative utility) to other users.

The difference between importance and utility is subtle. Utility can often be measured, whereas importance (as we use it) is typically a personal judgement. (A user's employer may impose that judgement!) Consider entering a national lottery as the task. The utility of winning is enormous; however, the importance of winning (unless one is in debt) is quite small, since it is not rational to depend on a win. On the other hand, if the odds of winning are known to be high enough to suggest relying on winning, one might then organise one's life so that the importance of winning increases—though the utility, in this case, need not have changed.

We informally define trust of an agent x of y (in some given meeting) as the probability weighted by UI that x acts to achieve any outcome *as if* it trusts y. In other words, trust is the degree of certainty that people act to increase one's utility. "I don't know what y will do, but I trust him just so-much to have my best interests at heart in his actions," is the notion captured by the more formal expression.

Defining trust

We now formally define trust as an abstraction.

For x and y in a situation a, from x's point of view, we represent the amount of trust x

has in y with the notation $T_x(y|a)$. This is to be read as, "the trust of x in y assuming the situation (meeting or task) a." This takes into account the fact that different situations require different levels of trust, even in the same person [Marsh, 1992].

We now come to formulae for determining trust in a cooperative situation. The values expressed within the formulae below are naturally subjective. In the case of humans, values for initial trust, importance of situation, and so forth could be supplied by the human, and would most likely be different in each case, for each team member. Alternatively, we (or the computer) might estimate them by parameter fitting.

To determine situational trust, notated $T_x(y|a)$:

$$T_x(y|a) = \widehat{T}_x(y)U_x(a)I_x(a)$$

where $\widehat{T}_x(y)$ is an estimate x has of how much he can trust y. For example, $\widehat{T}_x(y)$ may be an average over (a sample of) tasks, $\widehat{T}_x(y) = \frac{1}{|A|}\sum_{a \in A} T_x(y|a)$. Other statistical tendencies may be used; the mode is also a sensible measure. If x is an optimist, then the estimate is more likely to be the maximum rather than the average; if x is a pessimist, the estimate may be the minimum. If x has a poor memory, the sample size of tasks will be small. And so on.

Since this holds for all x, we can write it succinctly,

$$T(y|a) = \widehat{T}(y)U(a)I(a)$$

Given an initial value of $\widehat{T}_x(y)$, a computer could estimate values of trust in a given situation. We do not have space here to discuss how an agent, such as x, revises and improves his estimates as collaboration proceeds. The values are a dynamic reference to relationships, and in the instance of CSCW, to particular forms of relationship.

If a knowledge of trust was embedded in a system, the system and agents within it would constrain activities in a manner that was always as *appropriate* as possible.

This section has defined trust, and follows Steve Marsh's work [Marsh, 1992]. Of course, what we have defined is not, and cannot be, the wide-ranging, vague notion of trust that we had to start with; we have defined something precise, and which may or may not correspond closely to particular everyday uses of the vague concept. What has been defined, being precise, being implementable, promises to be useful in CSCW design and application.

In the very worst case, we may have defined trust so that somehow it can be shown to be inappropriate to CSCW requirements—but, then, we have better focused what the appropriate sort of trust is, for it could be no more than what else remains. Moreover, the sort of reasoning required to be developed to show this negative result would itself have contributed to the proper use of trust in CSCW.

Generalising to wider issues

Trust becomes much more interesting when more than two agents are cooperating. First, note that trust is not transitive: although $T_x(u) > T_x(v) \wedge T_x(v) > T_x(w) \Rightarrow T_x(u) > T_x(w)$,

because we assume x is rational [Cherniak, 1986, Moser, 1990], it is *not* the case that the following generally holds, for any relation $>$: $T_u(v) > T_v(w) \Rightarrow T_u(v) > T_u(w)$; put in words, if u trusts v (so much), and v trusts w, this says little about how much u trusts w.

All things being equal, of a set of users U, a rational user x will choose to assign a task a to some $u \in U$ to maximise $T_x(u|a)$. It is quite possible that x will prefer $u = x$, that is, that x does the task himself. More generally, x may have to partition a composite task A into component-user pairs, a_u, where $\bigcup a_u = A$, possibly overlapping, and maximise $\sum_{u \in U} T_x(u|a_u)$. This optimisation problem will usually be intractable, even when the task A is easy to decompose. Typically, x (either through pressure of time, lack of knowledge, or knowledge that trust estimates are imprecise) uses a greedy algorithm: find some $a_0 \subseteq A$, then find u to maximise $T_x(u|a_0)$, then solve the simpler problem of assigning $a_1 \subseteq A \setminus a_0$. Note that $T_x(u|a_1)$ may even be larger than $T_x(u|a_0)$: there is no reason why two components, a_0, a_1 cannot be assigned to the same user, u, nor that doing one should reduce x's trust in u being able to do the other.

A problem is that as an agent shows himself trustworthy to undertake a_0, he may then be allocated a_1 and then $a_2, \ldots$, for the same reason; ultimately the agent runs out of capacity to perform the tasks. This is Peter's Law, that people rise until they hit their level of incompetence. However, our model does not consider resource allocation, which is clearly crucial in a world of finite resources and limited skills. Thus, we assume that the trust invested in an agent to perform tasks is independent, but, nevertheless, it may vary in a complex way.

The issue for CSCW becomes more interesting again when we realise that a user u may object to some 'cooperative' decomposition, if, for example:

- If $T_u(u|a_v) > T_u(v|a_v)$: a user u considers himself more trustworthy at the task assigned to v than he considers v.
- If $T_u(u|a_v) > T_u(u|a_u)$: a user u considers himself more trustworthy doing the task assigned to v than to the task (a_u) already assigned to u.

A pair of users can constructively object if they are prepared to swap tasks, for example when $T_u(u|a_v) > T_v(v|a_v) \wedge T_u(u|a_u) < T_v(v|a_u)$; this is then an interesting case of the stable marriage problem, with changing preferences. Note that the stable marriage problem is readily solved by algorithms or heuristics; when users object (or wish to negotiate), then, the computer can come into its own in helping the users find an optimal (re-) allocation of tasks amongst the workgroup.

Finally, in a CSCW context, there is perhaps no reason for a user to be able to obtain T_x for any $x \neq u$ (other than himself). This greatly simplifies the social issues (!), and makes x's control simpler (but perhaps not as efficient if completion of tasks take appreciably longer than their allocation).

4 Trust as a CSCW design consideration

Although trust, in itself, is not sufficient to engender cooperation, an absence of trust (or distrust) is likely to render cooperation impossible: "the initiation of cooperation requires trust whenever the individual, by his choice to cooperate, places his fate partly in the hands of others" [Deutsch, 1962, pp302]. If we could make trust an explicit part of the system, then the system could provide a backup for trusting behaviour and trust knowledge. We believe that this is one of the key problems in CSCW at present. To date, little has been done with a view to providing such assistance.

Although several CSCW systems have implemented schemes that address some of the trust issues discussed in this paper, few (if any), have explicitly incorporated a notion of trust in their support. Our argument is that trust is a fundamental issue in cooperation; its importance in collaborative work is too great for it to remain a 'floating' design issue that is sometimes glossed, but more usually omitted due to its complexity. Trust needs to be an explicit part of the system strategy and design.

The range and potential uses of explicit trust in CSCW are as diverse as the collaborative activities that CSCW supports. In the most generic case, a system would support a 'network of trust', dynamically built and maintained to allow team members to work together in the knowledge that each of them can be trusted—and to what extent.

In the remainder of this section, we focus on the CSCW application COORDINATOR [Winograd, 1987, Flores *et al.*, 1988], and use it to described issues of trust in CSCW. COORDINATOR is chosen primarily because it is well known, but also because it has been widely studied and evaluated [Carasik & Grantham, 1988, Perin, 1991].

Discussion

Some co-workers are reliable, and need little prompting or cajoling for them to get the job done. Others, however, are notoriously unreliable. Incorporating trust into the design of a system requires us to take this into account. Systems like COORDINATOR are useful for one class of people—those who may be unreliable and in need of prompting. They are, however, too restricted for many people, causing ill-will towards the system, and perhaps co-workers. Whilst the principle is valid, the design is questionable, as it provides too rigid a framework for effective teamwork to exist.

We may view such co-working systems as a kind of spectrum. At one end, we have totally constrained systems that provide a rigid framework with no way out of that framework. Speech Acts [Searle, 1979], as used in COORDINATOR, are rigid in that sense (but only since COORDINATOR provides a simple Finite State Machine (FSM) to use the speech acts literally). At the other end of the spectrum are free speech systems—do what you want when you want, and how you want to. In this case the onus is on users to develop and apply any constraints that may be required, and avoid those that are not.

One approach is to extend the COORDINATOR (FSM) end of the spectrum and stretch it towards the free speech end. There are ways and means of accomplishing this. An obvious one is to completely redesign the system, taking this into account. The design

may incorporate more states for the finite state machine, allowing more complex actions, commitments, and so forth. Another way is to incorporate some flexibility into existing systems. Trust, due to its flexible nature, is suited to both of these approaches—we can design a system, that attempts to reason using trust as a central design parameter. Alternatively, we can use a knowledge of trust to extend what is already a working system, such as COORDINATOR. Consider such a system using a simple FSM. The states provide a rigid framework that, alone, is too rigid and inflexible. Enhancing the states by adding extra considerations, regarding trust, provides a way of extending the functionality of each state without actually changing the machine. Each state then becomes a separate agent that can reason using this new design constraint (trust), and can act accordingly, providing flexibility to, say, trustworthy team members, whilst optimally constraining the less trustworthy into performing work that is necessary at the time it is necessary. (For more information on the use of trust in modifying system imposed constraints, see [Jones, 1993].)

Uses of trust go beyond this static extension of system states. Trust is a dynamic concept that changes according to many factors: the experiences gained from interactions, the environment the agents exist within, situations, tasks, and similarities that are encountered, and so on. Consequently, the flexibility that enhanced each state in our previously inflexible FSM is considerably greater than we would at first expect, since trust values for each member of the team change from task to task, and from state to state. Different actions will be taken by each state in the machine according to its own knowledge of the person or people concerned. For example, consider two states, one dealing with people writing up part of a document, and another considering team members writing bits of code. It is likely that a team member who 'cannot be trusted' to write pieces of prose on time will be an excellent coder, and always gets the code in well before time. The different states have a different view regarding trust of the person concerned; the states consider different situations that may be *within* the same task.

With such a system, it seems likely that a large amount of information is needed in order to allow each state to reason properly with complete information. This, however is not the case. The additional overhead required to store the values for trust (importance, utility, and so forth), is trivial, even where each state has to consider people differently to each other state. Each person can be represented by a vector of trust values, and these values may even be different when different people are actually doing the trusting. States can communicate trust values autonomously between themselves if they perceive similarities among themselves.

This discussion of trust in commitment and project management support, as provided by COORDINATOR, is a single example of the application of trust. Some other areas for trust, both speculative and those already implemented (but not under the banner of 'trust') are briefly reviewed below.

Support for explicit roles in collaborative work. Systems such as the co-authoring application QUILT [Leland *et al.*, 1988] enforce explicit 'permission hierarchies' that control the rights of access to documents. Certain colleagues are constrained to read-only access

or annotate-only, and others will be allowed full permissions to update. Embedding an explicit notion of trust into such mechanisms would enable the system to (autonomously) adapt to particular work-group and work-task scenarios, and optimally constrain those involved. For a further discussion of accommodating and adapting systems to particular work-group roles, see [Greenberg, 1990].

Trust in social presence and directed encounter assistance. When an urgent solution to a problem is required, the optimal solution is to direct a single enquiry to the most appropriate advisor. Social awareness systems (such as CRUISER [Root, 1988, Fish *et al.*, 1993] or PORTHOLES [Dourish & Bly, 1992]) facilitate the decision on who to direct these enquiries to by promoting awareness of who is available. Trust however increases the system's ability to advise on suitable addressees by accounting for past experiences, the context of the task, and so on. The ANSWER GARDEN [Ackerman & Malone, 1990] uses AI techniques to provide similar intelligent routing of requests for assistance.

Trust in communication routing and underlying functionality. An important related issue to the incorporation of trust in CSCW applications is the users' perception of trust in the support it provides. For instance, frequently on failing to receive a reply to email messages a decision is made that the message must have been lost, and so it is resent, or an alternative communication channel is used. Agents, such as intelligent information filtering agents [Malone *et al.*, 1988], could do much to improve the users' perception of trust in the system by, for example, providing 'return-receipt acknowledge' functionality on request.

5 Conclusions

There exists a spectrum of cooperative work situations. At one end of the spectrum, we may be constrained in such a way that we can take very few actions at all beyond the premise of the system, leading to frustration and inflexible solutions to problems. At the other end, we may do and say as we please. This has its own problems concerned with lack of any real direction. We would wish to constrain ourselves and others to perform desired tasks in the future, since we may not trust ourselves or others to carry out those tasks when needed. In a cooperative situation, trust is of importance, and incorporating the concept into a CSCW system allows us to constrain those whom we need to constrain, since we trust them less, and to give free rein to those who can work well without constraint. Whilst this is possible for humans, such a consideration may well prevent them from actually getting anything done themselves, and they may also wish to constrain themselves into doing some!

The addition of trust to an already existing system, or incorporating trust into the design of a new system, allows users to leave such considerations to the machine, with a minimum of effort. Hopefully such a system will be used! In fact, the system could itself assign initial trust, importance, and so forth, values to team members and situations, and eventually reach an equilibrium after several tasks have been performed. The explicit role

that trust can play in CSCW could provide a computer-supported society built upon trust, not unlike the idea of Yamamoto's *Wa* [Yamamoto, 1990].

One of the chief flaws of CSCW systems is that they have failed to correctly account for the inherently social nature of human interactions, even in the workplace. Cooperation is a social phenomenon which directly involves consideration of trust [Golembiewski & McConkie, 1975]. If trust is not considered, systems become too inflexible to incorporate social working practices, with the result that either the systems are not used, or perhaps worse, are used incorrectly. Trust allows the system to be flexible whilst still constraining those users into carrying out allotted tasks, with very little overhead in terms of space and time. As a result, we believe that the incorporation of trust into CSCW, at all stages of design and use, will allow more flexible systems, leading to improved support for working practices.

References

[Ackerman & Malone, 1990] Ackerman, M. S. & Malone, T. W. 1990. Answer Garden: A tool for growing organizational memory. *pp31–39 of:* Lochovsky, F. H., & Allen, R. B. (eds), *Proceedings of the Conference on Office Information Systems.* Cambridge, Mass.

[Boon & Holmes, 1991] Boon, S. D. & Holmes, J. G. 1991. The dynamics of interpersonal trust: resolving uncertainty in the face of risk. *pp190–211 of:* Hinde, R. A., & Groebel, J. (eds), *Cooperation and Prosocial Behaviour.* Cambridge University Press.

[Boyle & Bonacich, 1970] Boyle, R. & Bonacich, P. 1970. The Development of Trust and Mistrust in Mixed-Motive games. *Sociometry*, **33**, 123–139.

[Brotz, 1983] Brotz, D. K. 1983. Message System Mores: Etiquette in Laurel. *ACM Transactions on Office Information Systems.* **1**(2), 179–192.

[Carasik & Grantham, 1988] Carasik, R. P. & Grantham, C. E. 1988. A Case Study of CSCW in a Dispersed Organisation. *pp61–65 of: Proceedings of CHI'88 Conference on Human Factors in Computing Systems.*

[Cherniak, 1986] Cherniak, C. 1986. *Minimal Rationality.* MIT Press.

[Clarke & Smyth, 1993] Clarke, A. A. & Smyth, M. G. G. 1993. A Co-operative Computer Based on the Principles of Human Co-operation. *International Journal of Man-Machine Studies.* **38**(1), 3-22.

[Cockburn & Thimbleby, 1991] Cockburn, A. J. G. & Thimbleby, H. W. 1991. A Reflexive Perspective of CSCW. *ACM SIGCHI Bulletin*, **23**(3), 63–68.

[Cockburn & Thimbleby, 1992] Cockburn, A. J. G. & Thimbleby, H. W. 1992. *Reducing user effort in collaboration support.* Tech. Rep. 93. University of Stirling.

[Dechter & Michie, 1984] Dechter, R. & Michie, D. December, 1984. *Induction of plans.* Tech. Rep. TIRM-84-006. The Turing Institute.

[Deutsch, 1949] Deutsch, M. 1949. A theory of Cooperation and Competition. *Human Relations*, **2**(2), 129–152.

[Deutsch, 1962] Deutsch, M. 1962. Cooperation and Trust: Some Theoretical Notes, in Jones, M. R. (ed.) *Nebraska Symposium on Motivation.* Nebraska University Press.

[Dourish & Bly, 1992] Dourish, P. & Bly, S. 1992. Portholes: Supporting awareness in a distributed work group. *pp541–547 of: Proceedings of CHI'92 Conference on Human Factors in Computing Systems.* Addison-Wesley.

[Fischer & Reeves, 1992] Fischer, G. & Reeves, B. 1992. Beyond Intelligent Interfaces: Exploring, Analysing, and Creating Success Models of Cooperative Problem Solving. *Journal of Applied Intelligence.* **1**(4), 311–332.

[Fish *et al.*, 1993] Fish, R. S., Kraut, R. E., Root, R. W. & Rice, R. E. 1993. Video as a technology for informal communication. *Communications of the ACM*, **36**(1), 48–61.

[Flores *et al.*, 1988] Flores, F., Graves, M., Hartfield, B. & Winograd, T. 1988. Computer Systems and the Design of Organisational Interaction. *ACM Transactions on Office Information Systems*, 6(2), 153–172.

[Golembiewski & McConkie, 1975] Golembiewski, R. T. & McConkie, M. 1975. The Centrality of Interpersonal Trust in Group Processes. *Ch 7, pp131–185 of:* Cooper, C. L. (ed), *Theories of Group Processes.* John Wiley.

[Good, 1990] Good, D. 1990. Individuals, Interpersonal Relations, and Trust. *Ch 3, pp31–48 of:* Gambetta, D. (ed), *Trust.* Blackwell.

[Greenberg, 1990] Greenberg, S. 1990. *Personalizable Groupware: Accommodating Individual Roles and Group Differences.* Tech. Rep. 90/404/28. Alberta Research Council.

[Jones, 1993] Jones, S. 1993. *Easing the writing task: designing computer based systems to help authors.* PhD thesis, Department of Computing Science and Mathematics, University of Stirling.

[Leland *et al.*, 1988] Leland, M. D. P., Fish, R. S. & Kraut, R. E. 1988. Collaborative Document Production Using Quilt. *pp206–215 of: Proceedings of the Second Conference on Computer Supported Cooperative Work.* ACM Press.

[Lowe, 1985] Lowe, D. G. 1985. Co-operative structuring of information: the representation of reasoning and debate. *International Journal of Man-Machine Studies*, **23**(9), 97–111.

[Luhmann, 1979] Luhmann, N. 1979 *Trust and Power*, John Wiley.

[Luhmann, 1990] Luhmann, N. 1990. Familiarity, Confidence, Trust: Problems and Alternatives. *Ch 6, pp94–107 of:* Gambetta, D. (ed), *Trust.* Blackwell.

[Malone *et al.*, 1987] Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A. & Cohen, M. D. 1987. Intelligent Information-Sharing Systems. *Communications of the ACM*, **30**(5), 390–402.

[Malone *et al.*, 1988] Malone, T. W., Grant, K. R., Lai, K-Y., Rao, R. & Rosenblitt, D. 1988. Semi-structured messages are surprisingly useful for computer-supported coordination. *pp311–331 of:* Greif, I. (ed), *Computer Supported Cooperative Work: A Book of Readings.* Morgan Kaufmann.

[Marsh, 1992] Marsh, S. 1992. Trust and Reliance in Multi-Agent Systems: A Preliminary Report. *In: MAAMAW'92, 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Rome.

[Moser, 1990] Moser, P. K. (ed) 1990. *Rationality in Action.* Cambridge University Press.

[Muir, 1987] Muir, B. M. 1987. Trust between humans and machines. *International Journal of Man Machine Studies*, **27**(5 & 6), 527–539.

[Perin, 1991] Perin, C. 1991. Electronic social fields in bureaucracies. *Communications of the ACM*, **34**(12), 75–82.

[Root, 1988] Root, R. W. 1988. Design of a Multi-Media Vehicle for Social Browsing. *pp25–38 of: Proceedings of the Second Conference on Computer Supported Cooperative Work.*

[Rosenschein, 1985] Rosenschein, J. S. October, 1985. *Rational Interaction: Cooperation among Intelligent Agents.* PhD thesis, Stanford University.

[Searle, 1979] Searle, J. R. 1979. *Expression and meaning: Studies in the theory of speech acts.* Cambridge University Press.

[Thimbleby *et al.*, 1990] Thimbleby, H. W., Anderson, S. & Witten, I. 1990. Reflexive CSCW: Supporting long-term personal work. *Interacting with Computers: the Interdisciplinary Journal of Human-Computer Interaction*, **2**(3), 330–336.

[Thimbleby, 1994] Thimbleby, H. W. 1994. An Organisational Solution to Piracy and Viruses. *The Journal of Systems and Software.* (In press.)

[Thomas, 1989] Thomas, M. 1989. Development methods for trusted computer systems. *Formal Aspects of Computing*, **1**(1), 5–18.

[Winograd, 1987] Winograd, T. 1987. A Language/Action Perspective on the Design of Cooperative Work. *Human-Computer Interaction*, **3**(1), 3–30.

[Witten, 1987] Witten, I. H. 1987. Computer (In)security: Infiltrating open systems. *Abacus*, 4(4), 7–25.

[Yamamoto, 1990] Yamamoto, Y. 1990. A Morality Based on Trust: Some Reflections on Japanese Morality. *Philosophy East and West*, **XL**(4), 451–469.

Four principles for groupware design: encouraging and easing system use

Andy Cockburn
Department of Computing Science
University of Stirling
Stirling, FK9 4LA
email : agc@uk.ac.stir.cs
Telephone : (0786) 67444

Steve Jones
Department Mathematics and Computer Science
Dundee Institute of Technology
Bell Street
Dundee DD1 1HG
email : mctsrj@uk.ac.dct.cc.vaxb
Telephone : (0382) 308619

October 1992

Abstract

Groupware fails when it enforces an imbalance between costs and benefits onto users. The costs of system use include additional work explicitly required by groupware and the effort required to overcome boundaries between work support tools. The cost/benefit imbalance is particularly significant during initial system use because groupware benefits are typically dependent on a "critical mass" of users.

We introduce four groupware design principles, together with strategies for achieving their aims. The principles address the imbalance between cost and benefit, and provide practical guidance for designers. Though adherence to the principles requires high levels of design effort, we contend that this is not a consequence of the principles, but rather it is a necessary requirement to avoid the failings of previous groupware systems.

1 Introduction

Attempts to support and enhance group work through the use of computers have been largely unsuccessful. The *personal* computer, despite research efforts, has remained exactly that, and attempts to increase the efficiency of organisations through computer supported collaboration have failed.

Many computer supported cooperative work (CSCW) researchers have modified their aim from the production of deliverable systems to gaining an understanding of the subtle issues of group work. CSCW research, then, focuses on a range of goals—from providing an understanding of social factors involved in support for group work, to the development of systems demonstrating the potential of new and innovative technologies. Evaluation and explanation of systems developed is frequently a casualty of "more interesting" research, this is partially explained by the exceptional difficulty of evaluating groupware. Mistakes and misguided decisions made by system developers during their experiments are frequently unreported, resulting in a lack of guidance for the next generation of system developers. Consequently the same, or similar, design errors are replicated and rediscovered.

In this paper we examine the major causes of groupware failure, focusing on the different forms of additional effort required in their use. User effort and its relationship with issues of integration and critical mass are also investigated. These observations are used to develop four design principles for groupware: maximise personal acceptance; minimise requirements imposed on users; minimise constraints imposed on users; and provide for integration with external resources.

The structure of the paper is as follows: section 2 introduces the four principles; section 3 examines their motivation—the causes of groupware failure. Sections 4, 5, 6, and 7 detail each principle, provide practical strategies for achieving its aims, and briefly discuss its impact on designers. Section 8 summarises the impact on designers, and section 9 concludes.

2 The Principles

The social implications of groupware are often cited as the fundamental barrier to its success (Sproull & Kiesler, 1991). We do not contend with this claim; rather, we note that a society's rejection of groupware is driven by an accumulation of individuals rejection. If principles for groupware design make systems more acceptable to individuals, without hindering their value in group support, they are likely to be more acceptable to the user society.

The three major issues hindering the success of groupware (detailed in section 3) are:

- the lack of integration — this applies to several aspects of computer systems including interface differences, incompatibility caused by information format requirements, and the absence of heterogeneous access to various communication/work support facilities;
- the additional effort required to use systems — largely due to a lack of system integration, but effort is also imposed on users by explicit system requirements for information, and the enforcement of particular usage styles;
- the vicious circle of groupware adoption (section 3.5) — certain benefits from groupware can only be realised once a critical mass of users has been established. If all system borne benefits are dependent on critical mass there will be little to encourage adoption until critical mass has been achieved.

Each of the four principles for groupware design addresses a combination of these problems.

Maximise the likelihood of personal system acceptance — aiming to increase the perceived and immediate benefit. By making systems appealing to individuals, regardless of the number of other users, systems become less dependent on critical mass and provide the "kick-start" necessary to overcome the vicious circle of adoption.

Minimise the requirements imposed on users — explicit system requirements (actions that must be executed by users) increase system dependence on structured information. This dependence imposes an additional work burden on users (someone must carry out actions for benefits to be realised), reduces system compatibility, and reinforces the vicious circle of adoption.

Minimise the constraints imposed on users — constraints imposed on users restrict their styles of working, and limit their flexibility in customising information input/output formats.

Maximise the potential for external system integration — the principles above reduce system dependence on user actions, structured information, and styles of working; they therefore increase the potential for system integration. This principle examines integration with facilities and resources beyond the control of system designers, including the different hardware platforms supported by organisations, and how to draw together disparate resources contributing to efficient cooperative work.

This description of the principles concentrates on their effect on system-use; the end-users' perspective. From the designers' perspective, the relationship between the principles is illustrated in figure 1. In this figure, each work/communication resource is depicted by a set of three concentric circles representing the system's user interface, information requirements, and user-constraints — these issues are respectively targeted by the first three principles. The dotted box of figure 1, enclosing the separate systems, represents an external integration environment: the aim of the fourth principle. From the designers' perspective the personal acceptance principle attends directly to interface details. Minimising requirements concentrates on implementation details, while minimising constraints is concerned with the abstract model of the design (minimising requirements focuses the implementation of this model). The final principle, from the designer's perspective, examines platforms which can draw communication facilities and work support resources together.

The causes of groupware failure, motivating these principles, are examined in greater detail in the following section.

3 Causes of Groupware Failure

Forming and maintaining collaborative relationships is difficult. Even in ideal work environments continual trade-offs, give and take, between collaboration participants is required. The inclusion of computer support in this complex balance is frequently counter-productive. Rather than enhancing group efficiency and cohesion, they hinder it. Typifying the extreme level of discontent with groupware, users called the *Coordinator* "fascist" (Erickson, 1989) and "worse than a lobotomised file clerk" (Carasik & Grantham, 1988), and reactions like the following about the Colab meeting support system are not uncommon:

> "...they found it so frustrating that they put their heads in their hands, raised their voices, and ultimately threatened to walk out. They expressed astonishment that anyone would build such a tool", (Tatar *et al.*, 1991) page–190.

What can be done to improve the poor performance of groupware? Selecting an appropriate task within collaboration is of primary importance; Egido (1988) provides some guidance to this end, recommending that groupware should answer two questions: what does it do that couldn't be done otherwise?; how to enhance communication intense functions that are already in place?

The focus of this paper, however, is on principles for groupware designers who (we assume) have previously assessed the viability of the support they aim to provide. The issues of interest to these groupware developers are "what can be done to avoid the failure of previous systems?", and "why have previous applications failed?".

Grudin's widely cited paper (1988) identifies three major causes of failure in CSCW applications: the disparity between who does the work and who gets the benefit; the breakdown of intuitive decision-making in design; the underestimated difficulty of evaluating CSCW applications. These three points can be generalised into three levels of failure: system-use, system-design, and system-evaluation.

In the following sections we examine the causes of groupware failure at the system-use level, identifying the components of Grudin's disparity—work (or effort) and benefit—and analysing the relationship between them, particularly during initial system use. The observations made motivate the principles which assist designers in overcoming failure at the system-design level.

3.1 Effort in Collaboration

The costs or undesired aspects of collaboration are the overheads of *effort* beyond that required to execute personal work tasks.

Naturally, there are a plethora of social factors which can inhibit and discourage collaborative work, regardless of whether it is supported by computers. Many of these factors can be considered to increase "effort"—certainly personality clashes make collaboration burdensome. Social complications in collaboration are, however, beyond the scope of this investigation.

The importance of *effort* in the formation and maintenance of collaborative relationships is widely recognised (Kraut *et al.*, 1988; Ishii & Miyake, 1991), and several systems have attempted to imitate the minimal effort provided by face-to-face interaction. When collaboration participants are physically remote, communication mechanisms must be adopted to mediate interaction. All non face-to-face interaction mechanisms are limited by their band-

width which reduces the richness of interaction[1]. The reduction in communication richness necessitates greater *effort* in completely and accurately transferring information (Hollan & Stornetta, 1992).

These issues are inherent in collaboration and its mediation. When computers are used to support group work there is a further imposition of effort due to explicitly required user-actions, constraints on flexibility, and transitions between methods of task accomplishment.

3.2 Effort imposed by system requirements

Many systems require and depend on additional effort from users to support their functionality. This dependence is best exemplified by enhanced asynchronous messaging systems which improve management of communications (filtering messages, assigning priorities, and so on), provide conversational representation of message relationship (allowing the review of past decisions, or a basis for making decisions), and allow computers to take an active role in managing/maintaining collaborative commitments. Almost exclusively, these systems depend on message senders supplying additional information, or *guidance* (Cockburn & Thimbleby, 1992). If guidance is not supplied then, for the benefits to be realised, some other user must provide it on the sender's behalf. Failure to capture guidance detrimentally effects *all* users when systems attempt to maintain an active role in collaboration; the knowledge base on which active assistance is based becomes corrupt causing problems such as redundant or mis-timed reminders.

3.3 Effort imposed by lack of flexibility

Several CSCW applications have been based on rigid theories of cooperative tasks—examples include speech-act theory (Flores *et al.*, 1988) and IBIS (Conklin & Begeman, 1988). Systems based on rigid theories are likely to be inflexible and impose specific styles of use which may conflict with those preferred by users.

Inflexible systems will, then, be unpopular; they necessarily enforce a form of "work to rule", a phrase synonymous with inefficient, restricted, and *inflexible* working practices. Requirements for specific styles of use, and the failure to account for sub-group or personal preferences/interpretations discourage system use. Users may find ways of working around restrictions, perhaps using alternative mechanisms to record personal views (a paper note pad for instance), but such work-around strategies illuminate system inadequacies, and require additional effort.

3.4 Effort imposed by lack of integration

Sources of additional effort in computer supported cooperative work go beyond those explicitly imposed by systems. Computer supported workers are required to make transitions—changes to their styles and methods of working—between various tools used in their *personal* everyday work. Effort in such transitions is derived from both the cognitive burden of remembering separate interfaces and the burden of manipulating data into compatible formats. Interfaces maintaining a consistent "look and feel" such as those employed by the Apple Macintosh (Apple Computer, 1988) or Windows III can ease these transitions; methods and techniques

[1]Whether this will always be the case is briefly discussed in (Hollan & Stornetta, 1992), and is the subject of much futuristic virtual reality research

used in one interface can be transferred to others, and consistent utilities such as cut, copy, and paste ease data manipulation.

When computer support is provided for *group* work, additional transitions between personal work support tools and group work tools are required. Typically, in focusing on the work products of groups, CSCW applications have ignored the individual's work environment under which much group work is initially executed. If a collaborative system is required infrequently, the overhead of re-learning the interface may be sufficient to discourage participation in group work altogether.

3.5 Adoption and Critical Mass

The additional effort required by a system will discourage users from adopting it. Goodman and Abel (1987) state that "People will use new communication systems to the extent that they require no more effort than existing ones" page–144. Users will similarly be discouraged by the imposition of inflexible working methods. While Goodman and Abel acknowledge that the enhancement of work resulting from system use could offset this discouragement, groupware tools are prone to a vicious circle restricting the initial realisation of system borne work enhancements (figure 2).

Groupware's chances for successful integration into the work environment are reduced as a consequence of this vicious circle, it inhibits the establishment of a "critical mass" (Ehrlich, 1987) of users necessary for all cooperation support tools and communication mechanisms. The factors in this chain of dependencies are the system's perceived *benefits*, its *achievement of critical mass*, and its *adoption by individuals*. The key determinant in the realisation of each of these components is the level of *effort* involved in system use.

Benefit and benefit-lag — willingness to adopt a system is dependent on the benefits derived from its use, and during adoption this is determined by *immediate* gains—users have no other experience on which to gauge a new system's value. All computer systems, however, suffer from "benefit-lag", the period during which the effort put into mastering a system out-weighs the benefit received. Benefit-lag can lead to users adopting "satisficing strategies" (Thimbleby, 1990); rather than devoting time and effort to learning new and (probably) more efficient ways of doing things, people continue to use methods which get the current task done *now*. Mantei (1989) notes that "... a high learning threshold would cause meeting participants to reject the technology".

Overcoming benefit lag is a complex problem for groupware. Not only must each individual undergo the learning process necessary to master new mechanisms, but the benefits encouraging this learning burden may not be available until critical mass has been established.

Achievement of critical mass — obviously, achieving critical mass depends on adoption by a *sufficient* group of individuals. Sufficiency in this context is contingent on the group, individual, and task requirements—in one group-task the main factor for overcoming critical mass might be the number of collaborators, while in another, the involvement of particular individuals might be the main determinant.

Adoption by individuals — individuals will be encouraged to adopt a system if there is an established base of regular users; information about the system and how to use it will

be readily available, easing the learning process and helping to break the inertia driven maintenance of current working practices. Personal adoption will be similarly encouraged if the benefits for doing so are clearly apparent—*personal* adoption is most likely to be stimulated by *personal* benefit.

Effort and the vicious circle of adoption — figure 2 illustrates the importance of effort in the initial acceptance of groupware. A system's prospects for adoption by individuals, and thus its likelihood of achieving critical mass, is dependent on the level of effort involved in employing the application. Many systems also depend on additional work in order to provide their benefits, should this work be omitted the system will fail to provide the benefit motivating each individual's adoption.

The vicious circle relating benefit, critical mass, and personal encouragement requires that all these properties are simultaneously available *before* systems can become successful—critical mass depends on adoption by individuals which is encouraged by benefits, but the benefits are, in turn, contingent on a critical mass of users. This situation appears to foretell a gloomy future for goupware!

What is required is some sort of kick-start, a break in the vicious circle allowing, for instance, benefits without the achievement of critical mass. The dominant and discouraging role of effort throughout system adoption and subsequent use must also be minimised.

4 Maximise personal acceptance

Maximising personal acceptance is concerned with encouraging individual users to incorporate new systems into their work routines. While promoting user acceptance may be sufficient, the provision of benefit is a more ambitious aim, emphasising the positive aspects of system use. Groupware success is dependent on each user's willingness and ability to incorporate the system into personal working methods; all classes of users must be content with their role in the computer's support. Each user must therefore receive a satisfactory balance between the amount of effort required to use the system, the benefit derived from it, the flexibility it supports, and the encouragement provided.

There is a similarity in how users view systems for personal and group work (for example, a word processor and a collaborative writing system). A common question users ask about both types of tool is "what can it do for me?"; during the initial system use, this question will carry an additional component, "*now*". Naturally, users will be more willing to devote time and effort to learning a new system if the benefits they receive for doing so are immediately available and clearly apparent.

In general terms, the personal acceptance principle argues for greater consideration of interface issues in groupware. If a task is better supported by a personal work tool than its collaborative equivalent users will continue to work primarily in the personal environment, overcoming transitions to the collaborative tool only when necessary. In addition to improving groupware user-interfaces, implementation strategies for encouraging personal acceptance, particularly during early stages of system use, include "feature ticking", and the use of "champions". The "reflexive perspective" argues more generally for increased design attention on the single user, basing this argument on the group-like behaviour of individuals.

4.1 Catchpenny systems

Feature ticking (Thimbleby, 1990) is a sales ploy used to add instant appeal to a wide range of modern products. Attractive features and additional facilities supplement the core functionality, turning attention away from the key task and onto fancy bells and whistles. While not condoning the design of poor (but feature rich) systems, a form of feature ticking can be used to supply instant user-appeal. Subtle forms of feature ticking can be used to encourage and reward exploration of system facilities, for example, the sequence of commands necessary to laser-print a document might be executed by an item in a sub-menu.

While personal benefits can encourage system use, acceptance of groupware is dependent on the costs incurred through its use. Feature ticking system attributes, specifically designed to attract individuals, will be worthless in the collaborative context if they fail to forward group work. To this end, the "reflexive perspective" of CSCW reduces the distinction between personal and group support. It aims, as far as possible, to generalise the mechanisms, tools, and techniques used for personal work to the collaborative environment.

4.2 The "Reflexive Perspective" of CSCW

CSCW concentrates on providing computerised support for people (*more* than one) who's joint work is distributed through both time and place. Reflexive CSCW (Thimbleby *et al.*, 1990; Cockburn & Thimbleby, 1991) is motivated by the observation that personal work is similarly distributed through time and space. Single users may work on several machines, one at the office, one at home, a lap-top, and a secretary's machine, and several projects may be pursued at different times. The group-like properties of the single user are further illustrated when the separate roles undertaken in personal work are examined. These roles include:

the management role — before starting work, the individual must carry out a series of management functions such as deciding which project or task is to be undertaken, coordinating the necessary resources, establishing appropriate reminders to prompt further work on tasks (for instance, when a colleague sends some necessary information), and so on;

the worker role — in which the actions necessary to advance or complete the work are executed;

the meta-management role — when the individual has an assistant in coordinating work (human or computerised), the assistant requires instruction on the style of support required: for example, what tasks are to be carried out on the individual's behalf, and whether he/she should be notified.

With multiple tasks, roles, and work places, the individual's coordination requirements are similar to those of asynchronously collaborating co-workers; the restriction to asynchronous work applies because of the inability to be in more than one place at a time!

The aim of the reflexive perspective, then, is to use similarities in personal and group work to blur the distinction between support mechanisms. By doing so, the user benefits from familiarity and predictability arising from a consistent interface to the personal and collaborative work environments. Skills transfer from one environment to the other, and the effort of learning and remembering separate interfaces are shared over a wider range of tasks. Reducing the disparity between personal and group support can bring further benefits by increasing awareness of involvement in a chain of collaborative commitments.

4.3 Champions and encouragement

The "personal acceptance" principle is primarily concerned with ensuring designers attend to the individual's requirements and preferences during the development of groupware. The principle can, however, be maintained during system installation/use to encourage adoption. Studies of CSCW adoption (Ehrlich, 1987; Francik *et al.*, 1991) have shown that enthusiasm for new systems is greatly enhanced by "champions" or "evangelists" who promote the use of the technology, raise awareness of what it can achieve, and generally encourage system use. Fafchamps *et al* (1991) noted this in their study of decision making through email conferences, "...the single most important factor for a successful computer conference is the activity level of the organiser of the conference.", page–220.

4.4 Implementing personal acceptance

Designing for personal acceptance is not the same as designing single user interfaces. Although the reflexive perspective observes similarities between personal and group work, it argues that these similarities should be used to reduce boundaries between personal and collaborative support, rather than basing collaboration support around the similarities.[2]

Borenstein and Tyberg (1991) note that a "highly polished and usable interface" is fundamental necessity of groupware, they also state that the traditional design trade off between power and usability is inappropriate and damaging in group work support—to cater for diverse expertise levels among users, power *and* ease of use are mutual necessities.

The personal acceptance principle therefore offers little repreive for designers. Not only must they provide the relevant functionality for the group task, they must do so in a highly usable manner, and without compromising experts' or novices' requirements.

5 Minimise requirements

The primary aim of the personal acceptance principle is to promote direct benefit, independent of other users, in order to overcome the vicious circle of adoption. Reducing the level of user effort *required* by systems is the primary objective of the minimise requirements principle.

User effort plays a pivotal role in system adoption (see section 3.5) but a system's *dependence* on user effort has detrimental effects beyond issues of system adoption. These include the imposition of a cost/benefit disparity between those carrying out actions and those receiving the benefit, and an increased likelihood of system incompatibility due dependence on particular information structure/format.

The minimise requirements principle promotes development of systems reducing the disparity between cost and benefit to user-acceptable levels. Strategies for achieving this goal, detailed below, include avoiding dependence on additional work, utilising information inherently available through communication, and shifting benefits onto those people undergoing the cost (alternatively, shifting cost onto those requiring benefit).

---

[2]Greenberg (1990) discusses ways and advantages of "personalizing" groupware; Patterson (1991) provides a comparison between the requirements of personal and group applications.

5.1 Avoid dependence on user actions

Systems depending on the provision of user-supplied information, or guidance, impose a cost/benefit disparity. In section 3.2 enhanced messaging systems were used to exemplify explicit requirements for user actions. These systems impose a cost/benefit disparity at one of three levels:

- Message senders have a cognitive burden in selecting appropriate templates, this process can be non-trivial when messages discuss several or inchoate ideas. If no suitable message types are available the user must define a new one. This effort, and that of filling in the relevant fields, is executed by the message sender for the eventual receivers benefit.

- If the sender fails to carry out these actions, for the benefits to be realised, some other user must execute the actions on the senders behalf. While the sender may deliberately omit the actions (due to work pressure, laziness, or whatever) its omission may be due to lack of access to the correct and compatible messaging facilities (Lee & Malone, 1990). Missing guidance has serious consequences for systems attempting to maintain an active role in work coordination—if the information is not supplied the systems knowledge of the status of commitments will become non-current, causing problems such as redundant and mis-timed reminders.

- The sender provides guidance, but the receiver fails to gain benefits; possible if the message structures supported by the senders and receivers systems are incompatible, or the knowledge of commitments maintained by an active coordination system is corrupt.

Rather than *requiring* guidance a more acceptable approach is to provide benefit when guidance is present, while not requiring it for system operation. It has however been argued that a relaxed approach of this nature is impractical due to the inter-relations and dependencies inherent in collaborative work:

> "Can a CSCW application succeed if doing the extra work is left to individual discretion? Unfortunately, *probably* not." Grudin (1988) page–86 (my emphasis).

While Grudin's observation is *probably* correct, the apparent corollary that systems should require users to provide the additional effort, is not. Depending on and requiring actions from users is as likely to cause system rejection as leaving the work to individual discretion. The necessity of structured information is further noted by Rodden & Sommerville (1991) ,

> "An underlying requirement within cooperative working support systems is the need for some structuring facility upon which to construct information handling systems", page–161.

The problem for system designers in supporting structured information is twofold. First, how to avoid incompatibilities with the information structures of other systems (standards such as X.400 (CCITT, 1987) may assist the designer). Second, where to retrieve this information from; certainly the user is the most obvious source, but experience has shown that systems might profitably look elsewhere (Cockburn & Thimbleby, 1992).

5.2 Use what's available "for free"

This strategy addresses the conflicting aims resulting from the previous strategy—how to leave users free from requirements, and yet still provide enhanced facilities and system borne benefits.

While guidance information is required to provide certain types of benefit, information sources go beyond that explicitly provided by users. Information is often accessible to computers through the process of communication—for instance, email messages contain header information revealing at least the, who, when, and where information about a communication, and can also detail the subject matter, the direct relationship previous messages, and so on. *Mona* (Cockburn & Thimbleby, 1992), uses standard email header information and inferencing heuristics to provide an interpretation of conversational context. Another source of "free" information in text-based communication is the text itself which can be scanned by natural language parsers, perhaps simply searching for keywords—*MAFIA* (Lutz *et al.*, 1990) demonstrates such schemes.

For non-text based communication, statistical information can be used to infer knowledge about collaborative work—for instance, the regularity of previous communications could be used to predict the arrival of new messages, notify the absence of expected messages, and prompt overdue responses. The use of interaction pace in CSCW is discussed in Dix (1992) and Gordon *et al* (1985) , while the use of prediction based on the statistics of previous user behaviour is examined in Darragh and Witten (1992) .

5.3 Enable shifts of cost and benefit

The cost/benefit disparity in some groupware systems is attributed to designers and managers striving for increased efficiency in the work place (Nagasundaram, 1990)—their assumption is that message senders will be willing and able to categorise their messages. In focusing on enhanced efficiency these designers ignore social effects, including the users' reluctance (or inability) to carry out actions which provide no *personal* benefit.

By shifting the provision of guidance (the cost) onto users gaining the benefit the cost/benefit disparity is reduced—users execute additional actions as and when they require the benefits without the system *requiring* actions from others. Thus, if a system utilising "free" information (following the previous strategy) fails to correctly or adequately support the individuals requirements, the user can execute additional actions to gain the desired result.

The applicability of this strategy depends on the politics and hierarchical structure of the organisation in which it is implemented (Erickson, 1989). While it may be reasonable to expect subordinates to work on behalf of a manager, often the converse will not be true. This strategy recommends that social protocols should be allowed to resolve conflicts between expectations of actions and execution of actions. Enforcing rigid dependencies on the work of others will, for many groups, encourage system rejection. Supporting both the ability to work on behalf of others *and* flexibility available through self motivated work improves groupware prospects.

5.4 Strategies for minimal requirements in conjunction

These strategies for minimal requirements are not intended to replace guidance dependent schemes, rather they should be used to supplement them: freeing users from a required cost/benefit disparity; increasing potential for system compatability; enhancing flexibility in

working for personal benefit. When used in conjunction the strategies provide alternative sources of the guidance information necessary to provide some enhanced collaborative work facilities. For example, while the strategy for "avoiding dependency on user supplied guidance" frees message senders from restricted (and burdensome) working practices, it is only when this strategy is combined with "utilise free information" that receivers stand to benefit regardless of the sender's actions.

The *potential* of modern technology rather than its social implications appears to have motivated much CSCW research (Sproull & Kiesler, 1991; Nagasundaram, 1990). While attempting to enhance work efficiency, groupware developers cannot ignore the pragmatic issues of how it will be used. By reducing dependency on user actions systems become tools which users manipulate and adapt, rather than the tools forcing users to adapt.

Combining guidance-dependent and guidance-free approaches may provide the optimum balance between cost and benefit. Through combined approaches systems do not depend on or require guidance, they adopt "free" mechanisms offering benefits when guidance is absent, but use it when available.

Minimised requirements, and guidance free schemes, reduce system dependence on critical mass. Systems depending on specific formats of user-explicit information can only provide their benefits when messages are both sent and received by systems supporting the same structures. "Free" benefits, independent of an established critical mass, are immediately available and assist in overcoming a system's "benefit lag".

5.5 Implementing minimise requirements

To provide systems merging guidance-dependent and guidance-free techniques, designers must in effect develop two (or more) integrated systems. Guidance-free schemes are likely to draw on heuristic methods which can not ensure the validity of inferences made, they must therefore allow modification and correction. The flexibility to correct, modify, and personalise should also be available to supplement information derived from guidance-dependent schemes.

In similarity with the personal acceptance principle, minimising requirements demands additional work from designers. While this may discourage designers from using the principles, the additional work is not a consequence of the principles, but rather it is a necessity to avoid the failures of previous application. This issue will be discussed further in the concluding parts of the paper.

6 Minimise Constraints

Minimising requirements is concerned with the implementation stage of system development, primarily aiming to avoid dependence on user actions. "Minimising constraints", however, attends to problems arising at an earlier and more abstract level of system development, focusing on models and theories underlying the support. The aim is to avoid models which enforce detrimental effects such as inflexible and constraining styles of use, and user dependent mechanisms.

Constraining users to particular styles of use necessarily inhibits their flexibility. While rigid working practices can in principle support highly efficient organisations, in reality few organisations work according to such deterministic methods, and they can't be made to do so (Nagasundaram, 1990). The minimise constraints principle concurs with the aims of Dykstra and Carasik (1991) who, during development of the *Amsterdam Conversation Environment*,

considered "...what it was that we really wanted to support: processes or people?", page–420. Their definition of support for *people* as "non-dependency-creating enablement" argues for groupware which leaves users free to develop protocols governing collaborative work as *they*, rather than their systems, see fit.

Strategies for satisfying minimal constraints, detailed below, primarily aim to increase designers' awareness of problems arising from inflexible and rigid systems; detailed and globally applicable strategies are likely to be inappropriate due to the diversity of groupware tasks.

6.1 Be aware of the two level perspective of technology

Sproull and Kiesler's (1991) two level perspective of technology examines conflicts between increased efficiency available through computer support and its negative social implications. The first level addresses the increased efficiency enabled by particular styles and uses of technology. The second level is concerned with social effects, raising issues such as user acceptance, personalised views of information, and individual preferences. The distinction between these levels can be expressed by the contrasting questions "what is *possible* with technology?" at the first level, and "how will it be used?" at the second.

Groupware designers, and all those involved in systems development, must be aware of the social implications inherent in group work support. Technology capable of enhancing organisational efficiency will fail if relevant social factors are ignored. Design alterations based on projections of a system's social implications may temper the efficiency improvements achievable, but it is better to provide acceptable mechanisms providing some benefit than unacceptable ones which, despite great potential, fulfill none.

6.2 Beware of rigid models and theories

Models and theories explicitly embedded into systems should be capable of supporting flexibility, completeness, and dynamic adaptability in a manner acceptable to all users. Developing such models/theories is likely to require more design effort than an equivalent system developed in an open, non-dependency-creating, manner. This difficulty has been illustrated by the failure of many systems adopting rigid models and theories, most notably the *Coordinator* (Carasik & Grantham, 1988), and *Cognoter* (Tatar *et al.*, 1991). In both these systems, restrictions imposed by underlying theories of coordination and communication substantially contributed to system failure.

Accurate and flexible models do promise improvements in computer support for group work, but the youth of research into models of collaborative activity limits their value in *current* groupware products.[3].

6.3 Open, unconstrained enhancement

While adequate and complete models/theories are under development open, systems allow users to develop protocols as they see fit. User and task models might be used to supplement an open system (for example, recording and installing user preferences) but they should not be allowed to impose constraints.

Existing systems exemplifying the open approach support a variety of collaborative activities: several social browsing and virtual presence systems allow social protocols to prevail

[3]Dourish (1991) examines computational reflection as a method enabling flexibility

(Fish *et al.*, 1992; Dourish & Bly, 1992); the *Object Lens* (Malone & Lai, 1988) and other toolkits for enhanced asynchronous communication allow users to develop the support they require through semi-structured applications without enforcing particular styles of use [4]. In support for collaborative writing, *Milo* (Jones, in press) avoids modeling writing styles/roles in order to free users from constraints and allow flexibility.

6.4 Implementing minimal constraints

Much of the development effort for a system based on an explicit model or theory will be devoted to establishing a suitable model or theoretical platform. An alternative approach is to use a range of models and theories, adaptable depending on properties displayed by users, but this requires similar levels of design effort to provide a sufficient range of models, and methods of incorporating them into the system.

Theories of communication and models of user behaviour promise to support successful systems in the future (Keeler & Denning, 1991), but until fully developed they are unlikely to afford adequate platforms for cooperative work. Designers aiming to produce working and workable groupware (rather than research systems) should therefore beware of the constraints imposed by embedding explicit theories/models into systems.

7 External Integration

Providing external integration requires designers to consider their system's role within, and relationship to, the entire work environment. In this extended collaborative context group members use competing systems to execute similar tasks, and a variety of tools (computer and non-computer based) are drawn upon to support and assist collaboration.

The three principles above are primarily concerned with design and use of cooperative work support systems *in isolation*: how to make a particular system acceptable; how to limit the level to which it imposes actions and work styles on the user. They also increase system integration: personal acceptance addresses boundaries between personal and group work; the requirements and constraints principles reduce system dependencies enhancing their openness.

To limit the complexity of developing *personal* computing systems each software tool tackles a specific task such as word-processing or drawing. Integration between tools is facilitated by consistent user interfaces, complying to sets of guidelines laid down for particular hardware platforms. In collaboration support, following consistent user-interface guidelines is hindered by the use of differing hardware platforms—while individuals benefit from intergation across applications on a single hardware platform, they must still overcome interface (and other) barriers when transferring their work between hardware platforms. Consequently, computer supported collaborative workers, and individuals using more than one hardware platform, need to master a range of interfaces. Not only does this impose a learning and remembering burden, but overcoming incompatibilities in the format of information requires effort and perhaps the use of translation tools. Figure 1 provides a summary of some of the transitions (changes to the users favoured styles and methods of working) required.

Increasing external integration, then, aims to reduce the number and magnitude of transitions between tools supporting collaborative work. The benefits of external integration are

[4] However, their benefits are to a large extent dependent on others using the same semi-structured systems (Cockburn & Thimbleby, 1992)

not only curative (in reducing seams brought about by computer support), it also argues for integrated and improved access to resources serving communication and collaboration requirements. Information such as who's available for interaction, and how to contact people (telephone numbers, email and surface addresses, video connection dial-up sequences, and so on) can be pooled with access to interaction media—thus, while the mechanisms are not integrated as such, access to them is presented in an integrated manner.

The strategies for achieving external integration, detailed below, both exemplify systems that have reduced boundaries in CSCW (namely video-fusion techniques and work towards heterogeneous collaboration environments), and guide designers on avoiding the implementation of non-integrable schemes.

7.1 Video fusion

Video fusion techniques, best exemplified by the *TeamWorkStation* (Ishii & Miyake, 1991), reduce the seams or barriers in synchronous computer supported work. These seams include the lack of compatability between personally favoured tools which force at least one collaborator to adopt "foreign" working methods, and the loss of important communicative cues such as gesture. Video-fusion enables collaborators to maintain personally favoured tools by overlaying the video images of separate computer screens, like layers of transparent acetate. In this way two otherwise incompatible applications can be used together.

Video-fusion techniques are not limited to computer output. Using video cameras, physical workspaces can merged with each other and/or computer applications: for example, fusing the images of a computer drawing application with a paper pad allows annotation of a computerised picture with a pencil.

While extremely appealing video-fusion techniques have limitations, the most serious of which is storing the results of collaborative sessions. Integration between applications is purely visual, so the visual result can only be sustained while the video connection is maintained. Once the real-time collaboration is terminated, if the result of the session is to be maintained for future reference, the constituent parts of the collaborative product must somehow be re-integrated. In such cases, the problems of integration due to system differences must still be overcome.

7.2 Heterogeneous environments

Video-fusion techniques are primarily curative, overcoming problems brought about by the lack of integration between existing work support tools. In contrast, heterogeneous platforms augment collaboration by drawing together access to, and information about, collaboration resources in a way impossible without computer support. They work towards an "integrated portfolio of media" (Bair, 1989), represented by the "tools environment" in figure 1.

The *MOCCA* project (Benford *et al.*, 1992), examined the requirements for a conceptual "heterogeneous collection of applications, paradigms and models...", and collaboration requirements, particularly the maintenance of social presence knowledge, have been investigated by several systems including *Cruiser* (Fish *et al.*, 1992) and *Portholes* (Dourish & Bly, 1992). It is interesting to note that a primary use of *Cruiser*'s high-bandwidth video links was in establishing whether colleagues were present, and if not, when they arrived; they used the term "ambush" to describe this redundant use of an open video connection to an empty office.

Telefreek (Cockburn, 1992) implements a less ambitious, but working and extensible CSCW environment based on, but not limited to, standard networked computers. By drawing together computer supported information sources (such as address files, user whereabouts, "ambush" style notification of arrival, and so on), with various communication mechanisms and collaboration applications, *telefreek* users are provided with a platform for communication and collaboration.

7.3 Minimise dependence on structure and format

Dependence on system specific information formats and structures not only likely to impose additional user-effort, reduce flexibility, and enforce constraints (section 3), it is also likely to reduce the potential for system integration.

Structured information can, however, be incorporated into systems without causing incompatibilities: for instance, the X.400 (CCITT, 1987) email standard allows additional fields to transfer structured and semi-structured information. The International Standards Organization, Open Distributed Processing (ISO ODP) recommendations for information exchange (van Griethuysen, 1989) promote the use of compatible information formats. Groupware products must follow the relevant standards for the communication media on which they are based.

Systems built on existing interaction media should do so monotonically—new facilities and enhanced features should not affect users maintaining the original methods.

7.4 Implementation platforms

*** ALSO MARK AND SAUL's GroupKit... a bolt on platform for synch collab and Obj Lens a platform for asynch work. *** Groupware's prospects for overcoming critical mass are affected by the number of potential users. Any restriction on the range of potential users is therefore counter-productive, especially during early stages of system use. Developing systems on specific hardware platforms with propriety user-interface toolkits restricts the domain of potential users to those supporting that particular type of hardware.

To minimise the impact of incompatible hardware and interface platforms designers must either replicate some of the implementation work, enabling their application to run on a variety of support mechanisms, or choose a suitable hardware independent development system, for example, the X Window system (Scheifler & Gettys, 1986).

7.5 Implementing external integration

External integration is concerned with issues external to, and beyond the control of, systems under development: integration with existing systems which *do* impose specific information format requirements; integration across hardware platforms; merged access to differing communication mechanisms, resources, and requirements.

Strategies for this principle address various types of external integration, the impact of the principle on designers therefore depends on the strategies used—for instance, the heterogeneous environment strategy argues for the development of entire systems, and the hardware independence strategy influences the choice of user-interface implementation software. To achieve hardware independence, designers must either use hardware independent user-interface software (such as the X Window system) or supply several versions of the system, each capable of running on different hardware platforms. The increasing availability of

portable graphical user interface packages such as $OpenInterface^{TM}$ (Neuron Data, 1992) will reduce design effort required to achieve user-interface hardware independence.

8 The principles effect on designers

Guidance provided to designers by these principles is relatively abstract. The potential range of group tasks undertaken with computerised support is too diverse for globally applicable strategies to be proposed. Though some strategies are not directly relevant to specific systems, they illuminate common problems and increase designers' awareness; the central aim of each principle remains applicable to all collaboration support tools, particularly during the establishment of user critical mass.

The discussions on implementation issues (sections 4.4, 5.5, 6.4, 7.5) show that following the principles' recommendations will be a major undertaking—they offer no repreive for designers. The question asked by designers, however, should not be "which approach is the easiest?", but rather "which potentially successful approach is the easiest?".

Research and experience has shown that technology is able to offer novel, efficient, and work-enhancing facilities. However, providing systems *capable* of enhancing group work is insufficient, they must also (in many ways, primarily) be acceptable. Surface system issues such as interface quirks, or failure to provide adequate appeal to new users are likely to prompt system rejection. Designers may argue that such superficial problems will be overcome once enhancements in work efficiency are recognised, but rejection at an early stage means that the benefits will never be attained.

Designers adopting these principles are, then, required to maintain the highest levels of motivation and professionalism. We contend that these high standards are not a consequence of the principles, rather they are qualities required for the development of successful and (necessarily) acceptable group work applications.

9 Summary

Groupware failure is caused by an accumulation of rejection by individual users which is, in turn, largely due to an imbalance between the costs and benefits of system use. In this paper we examined these costs in terms of user-effort, and identified its sources—explicit system requirements, and a lack of integration between systems and between working methods.

Groupware, naturally, aims to enhance group work, but the definition of *group* as "two or more people working together to a common goal" excludes the single user. Individuals are therefore uninspired, they execute personal work on single-user applications, and overcome transitional barriers to groupware only when transferring work is necessary. This lack of design attention on end-users would be of limited importance if group benefits were readily available, but groupware suffers from a vicious-circle inhibiting the attainment of group benefits. Until a "critical mass" of users has been established new users have little to attract them, especially if their individual requirements are ignored.

The principles described in this paper assist groupware designers in avoiding the pitfalls discovered by previous groupware systems. They aim to reduce the inhibiting role of user-effort in groupware, particularly concentrating on avoiding system dependence on user-effort and the consequences of the vicious-cirle in groupware adoption.

The kick-start of encouragement necessary to overcome groupware's vicious-circle of adoption is enabled by "free", or non-effort-dependent facilities, and by improving the personal appeal of groupware. Having established critical mass, system use is still enhanced by the principles which increase integration between tools and minimise the consequences of absent structured information. Finally, by merging access to communication and collaboration facilities, the effort of accessing useful support mechanisms is minimised.

Three systems specifically adhering to the principles have been developed by the authors. *Mona* (Cockburn & Thimbleby, 1992) provides a conversation-based email platform without requiring explicit user guidance, *Milo* (Jones, in press) is a minimally-constraining collaborative writing tool, and *telefreek* (Cockburn, 1992) supports an extensible and customisable heterogeneous platform for communication. These systems are available from the authors, they are written in C and run under Unix and X Windows.

Acknowledgements Many thanks to our friends and colleagues who commented on early versions of this paper. This research was partially funded by the Isle of Man Department of Education.

References

Apple Computer, Inc. 1988. *Human Interface Guideleines: The Apple Desktop Interface.* Addison-Wesley.

Bair, JH. 1989. Supporting Cooperative Work Teams with Computers: Addressing Meeting Mania. *Pages 208-217 of: Proceedings of the 34th IEEE Computer Society International Conference - CompCon Spring. San Francisco, CA. February 27 - March 3. 1989.*

Benford, S, Prinz, W, Mariani, J, Navarro, L, Bignoli, E, Brown, CG, & Naslund, T. 1992. *MOCCA - A CSCW environment.* Part of the European CO-TECH programme.

Borenstein, NS, & Thyberg, CA. 1991. Power, ease of use and cooperative work in a practical multimedia message system. *International Journal of Man-Machine Studies*, **32**(2), 229–259.

Carasik, RP, & Grantham, CE. 1988. A Case Study of CSCW in a Dispersed Organisation. *Pages 61-65 of: Proceedings of CHI'88 Conference on Human Factors in Computing Systems.*

CCITT. 1987. *Draft recommendation on message handling systems, X.400, Version 5.*

Cockburn, AJG. 1992. *Telefreek: an extensible heterogeneous platform for communication requirements.* Chapter 6 of incomplete PhD thesis.

Cockburn, AJG, & Thimbleby, HW. 1991. A Reflexive Perspective of CSCW. *ACM SIGCHI Bulletin*, **23**(3), 63–68.

Cockburn, AJG, & Thimbleby, HW. 1992. *Reducing user effort in collaboration support.* Tech. rept. 93. University of Stirling, Stirling, Scotland, FK9 4LA. To appear in Intelligent Interfaces Workshop, Florida, 1993.

Conklin, J, & Begeman, ML. 1988. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6(4), 303–331.

Darragh, J, & Witten, I. 1992. *The Reactive Keyboard.* Cambridge University Press.

Dix, AJ. 1992. *Pace and interaction.* Submitted to HCI '92.

Dourish, P. 1992. *Computational Reflection and CSCW Design.* Rank Xerox EuroPARC, Cambridge, UK.

Dourish, P, & Bly, S. 1992. Portholes: Supporting awareness in a distributed work group. *Pages 541–547 of: Proceedings of CHI'92 Conference on Human Factors in Computing Systems* Monterey, May 3–7 1992.

Dykstra, EA, & Carasik, RP. 1991. Structure and Support in Cooperative Environments: the Amsterdam Conversation Environment. *International Journal of Man-Machine Studies*, 34, 419–434.

Egido, C. 1988. Videoconferencing as a technology to support groupwork: A review of its failure. *Pages 13–24 of: Proceedings of the Second Conference on Computer Supported Cooperative Work* September 26–28 1988. Portland, Oregon.

Ehrlich, SF. 1987. Stratagies for Encouraging Successful Adoption of Office Communication Systems. *ACM Transactions on Office Information Systems*, 5(4).

Erickson, TD. 1989. Interfaces for Cooperative Work: An Eclectic Look at CSCW 88. *SIGCHI bulletin*, 21(1), 56–64.

Fafchamps, D, Reynolds, D, & Kuchinsky, A. 1991. The dynamics of small group decision-making using electronic mail. *Pages 211–224 of:* Bowers, JM, & Benford, SD (eds), *Studies in Computer Supported Cooperative Work: Theory, Practice and Design.* North-Holland.

Fish, RS, Kraut, RE, Root, RW, & Rice, RE. 1992. Evaluating Video as a Technology for Informal Communication. *Pages 37–48 of: Proceedings of CHI'92 Conference on Human Factors in Computing Systems* Monterey, May 3–7 1992.

Flores, F, Graves, M, Hartfield, B, & Winograd, T. 1988. Computer Systems and the Design of Organisational Interaction. *ACM Transactions on Office Information Systems*, 6(2), 153–172.

Francik, E, Rudman, SE, Cooper, D, & Levine, S. 1991. Putting Innovation to Work: Adoption Strategies for Multimedia Communication Systems. *Communications of the ACM*, 34(12), 53–63.

Goodman, GO, & Abel, MJ. 1987. Communication and Collaboration: Facilitating Cooperative Work Through Communication. *Office:Technology and People*, 3(2), 129–146.

Gordon, M, Belew, R, Kochen, M, & Lindsay, R. 1985. Managing Computerised Conferences. *Pages 179–184 of: Proceedings of the National Online Meeting.*

Greenberg, S. 1990. Sharing Views and Interactions with Single-User Applications. *Pages 227–237 of:* Lochovsky, FH, & Allen, RB (eds), *Proceedings of the Conference on Office Information Systems.* April 25-27 1990. Cambridge Mass.

Grudin, J. 1988. Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces. *Pages 85–93 of: Proceedings of the Second Conference on Computer Supported Cooperative Work* September 26–28 1988. Portland, Oregon.

Hollan, J, & Stornetta, S. 1992. Beyond being there. *Pages 119–125 of: Proceedings of CHI'92 Conference on Human Factors in Computing Systems* Monterey, May 3–7 1992.

Ishii, H, & Miyake, N. 1991. Toward an Open Shared Workspace: Computer and Video Fusion Approach of TeamWorkStation. *Communications of the ACM*, **34**(12), 37–50.

Jones, S. in press. MILO: a computer based tool for (co)authoring structured documents. *In:* Sharples, M (ed), *Computer Supported Collaborative Writing.* Springer-Verlag.

Keeler, MA, & Denning, SM. 1991. The Challenge of Interface Design for Communication Theory: from interaction metaphor to contexts of discovery. *Interacting with Computers: the interdisciplinary journal of Human-Computer Interaction*, **3**(3), 283–301.

Kraut, R, Egido, C, & Galegher, J. 1988. Patterns of Contact and Communication in Scientific Research and Collaboration. *Pages 1–12 of: Proceedings of the Second Conference on Computer Supported Cooperative Work* September 26–28 1988. Portland, Oregon.

Lee, J, & Malone, TM. 1990. Partially Shared Views: A scheme for Communicating among Groups that Use Different Type Hierarchies. *ACM Transactions on Office Information Systems*, 8(1), 1–26.

Lutz, E, Kleist-Retzow, H, & Hoernig, K. 1990. MAFIA – An active mail-filter-agent for an intelligent document processing support. *ACM SIGOIS Bulletin*, **11**(4), 16–32.

Malone, TW, & Lai, KY. 1988. Object Lens: A "Spreadsheet" for Cooperative Work. *Pages 115–124 of: Proceedings of the Second Conference on Computer Supported Cooperative Work* September 26–28 1988. Portland, Oregon.

Mantei, M. 1989. Observation of Executives Using a Computer Supported Meeting Environment. *Decision Support Systems 5*, 153–166.

Nagasundaram, M. 1990. Style and Substance in communication: Implications for message structuring systems. *ACM SIGOIS Bulletin*, **11**(4), 33–41.

Neuron Data, Inc. 1992 (July). *Portable graphical user interfaces across all windowing standards: Product Information on the* Open Interface *system.* 488 NEURON DATA, 156 University Avenue, Palo Alto, CA 94301.

Patterson, JF. 1991. Comparing the demands of single-user and multi-user applications. *Pages 87–94 of: Proceedings of the ACM Symposium on User Interface Software and Technology.* Hilton Head, South Carolina, USA. November 11–13, 1991. ACM Press.

Rodden, T, & Sommerville, I. 1991. Building Conversations Using Mailtrays. *Pages 159–172 of:* Bowers, JM, & Benford, SD (eds), *Studies in Computer Supported Cooperative Work: Theory, Practice and Design.* North-Holland.

Scheifler, RW, & Gettys, J. 1986. The X Window System. *ACM Transactions on Graphics*, 5(2), 79–109.

Sproull, L, & Kiesler, S. 1991. *Connections: New ways of working in the networked organization.* The MIT Press.

Tatar, DG, Foster, G, & Bobrow, DG. 1991. Designing for Conversation: Lessons from Cognoter. *International Journal of Man-Machine Studies*, **34**, 185–209.

Thimbleby, H. 1990. *User Interface Design.* ACM Press.

Thimbleby, HW, Anderson, S, & Witten, I. 1990. Reflexive CSCW: Supporting long-term personal work. *Interacting with Computers: the interdisciplinary journal of Human-Computer Interaction*, **2**(3), 330–336.

van Griethuysen, JJ. 1989. Open Distributed Processing (ODP). *Ninth IFIP WG 6.1 International Symposium on Protocol Specification, Testing, and Verification (6th - 9th June 1989)*, June.

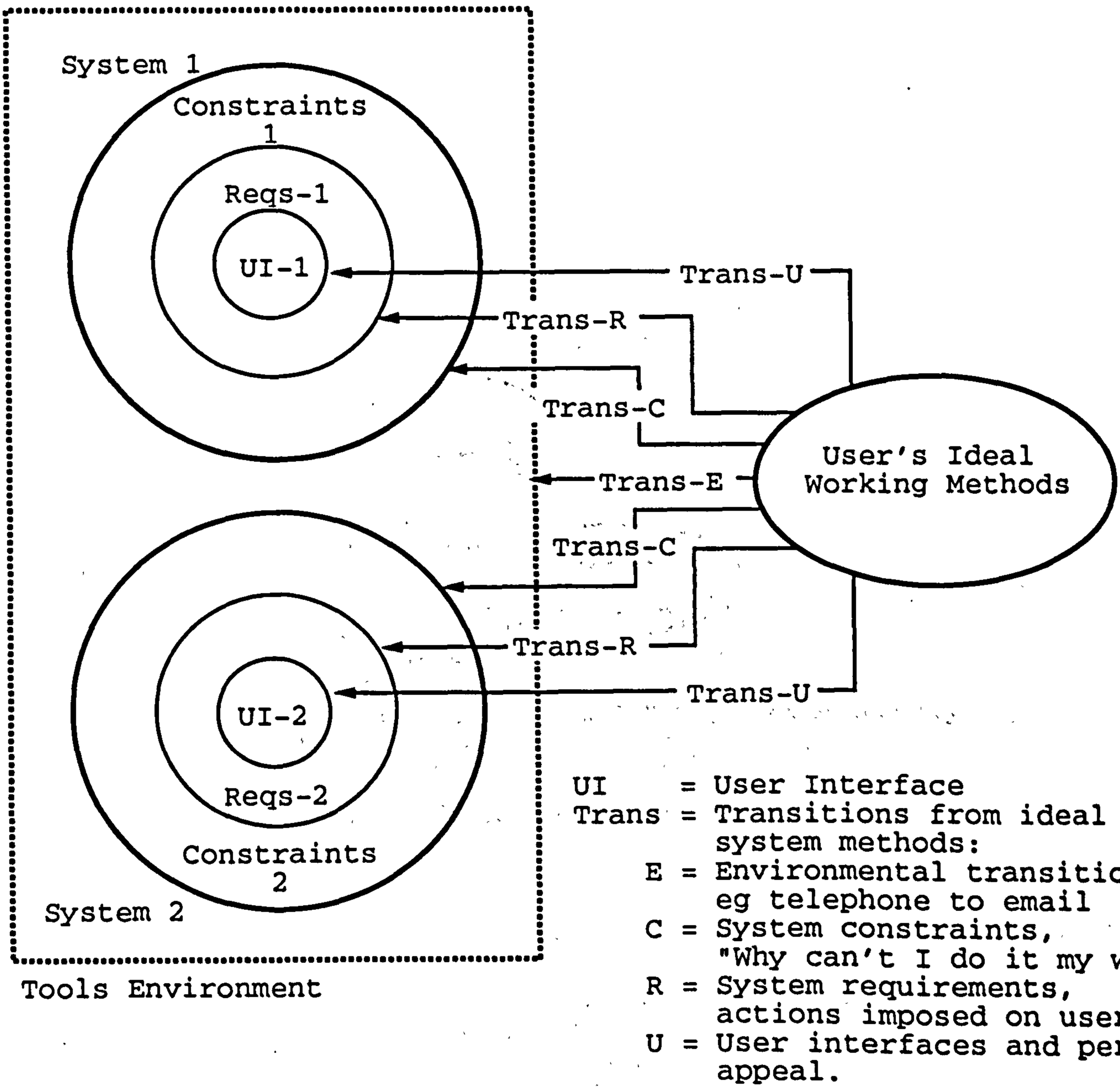


Figure 1: Transitions between the user's ideal working methods and those supported by systems

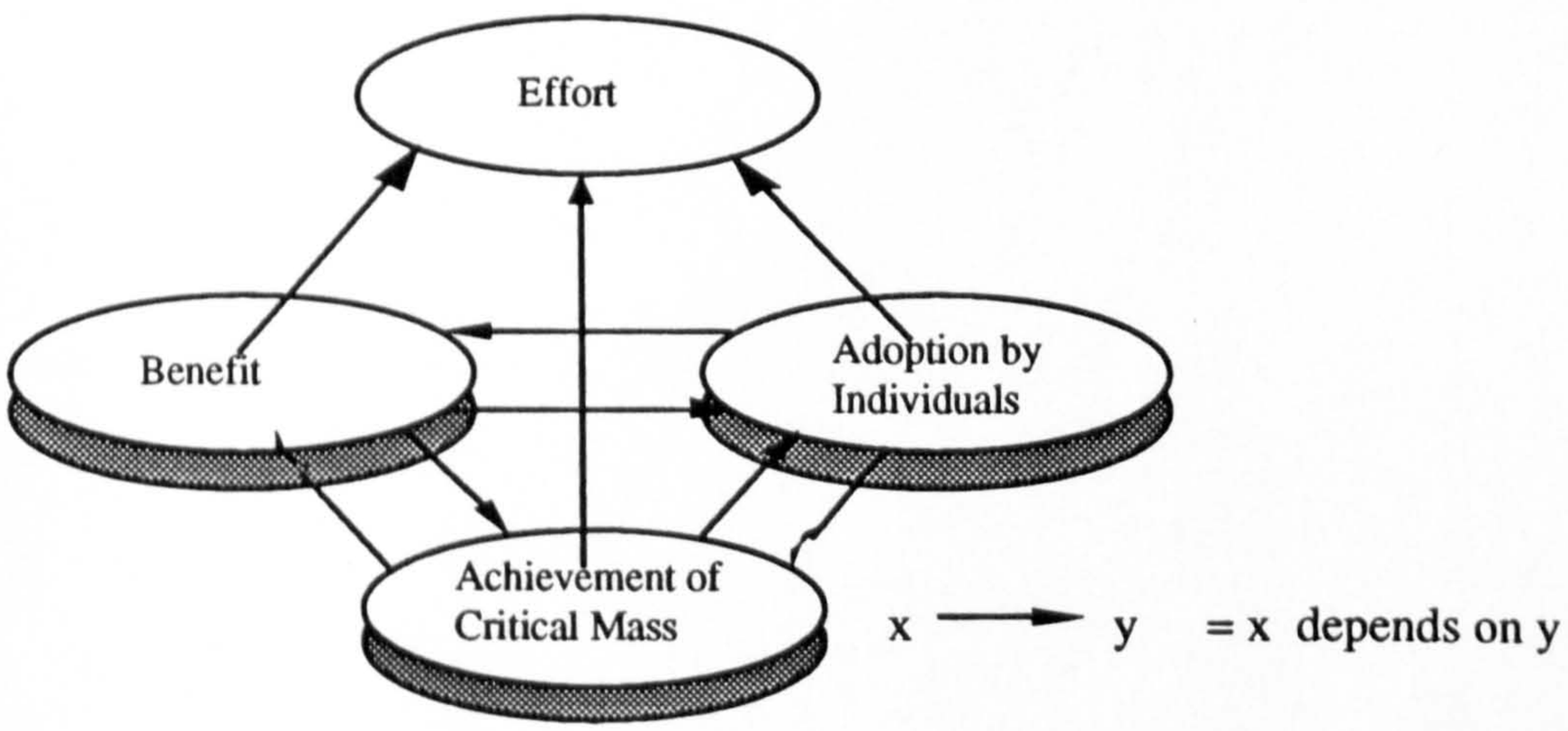


Figure 2: The "vicious circle" of dependencies in groupware adoption