Un-Constraining the Medium: Designing Software Systems to Support Situated Action

by

Ben Anderson

A Doctoral Thesis

Submitted in partial fulfilment of the requirements
for the award of

Doctor of Philosophy of Loughborough University

August 6, 1997

Abstract

This dissertation is concerned with Computer Supported Cooperative Work (CSCW) and, in particular, with ways in which insights from ethnomethodology can be melded into the design of CSCW systems - a relationship that has been labelled *technomethodology*. The dissertation outlines a number of possible ways in which system design can learn from ethnomethodology and concentrates on one particular aspect - namely that CSCW should look closely at its foundational assumptions and, if necessary, respecify any concepts which appear problematic in their formulation.

The dissertation provides an example of exactly this sort of respecification by examining CSCW's assumptions about the nature and use of rules in everyday life. It provides a case study of the design, implementation and use of a functional CSCW system - *TelePort*- that takes account of an alternative view on rules and rule use grounded in studies of everyday work and social activities.

The CSCW system that is described is a prototype tool to support group awareness between users of a test-bed packet switched multimedia telecommunications network - the Internet MBONE. Whilst the design, implementation and usage of this prototype provide a research vehicle for investigating requirements for group awareness protocols and tools in the Internet context, it also serves as a case study of *technomethodology*.

The case study offers the conceptual framework and design methods developed as the building blocks for the technomethodology research program. Reflections on the design, implementation and usage of the prototype provide testament to the utility of respecifying assumptions whilst discussions of the viability of a number of core concepts in HCI shows that ethnomethodological recommendations could, potentially, radically alter some of the foundational concepts of Computer Science.

To Claire, for keeping me sane.

Acknowledgements

The ideas reported by this dissertation have grown from, and been improved by, innumerable conversations, arguments, arm-waving and beer-drinking with friends and colleagues at the LUTCHI Research Centre: Tim Appleyard, Adam Bridgen, Kelvin Clibbon, Claudia Eckhert, Roger Knott, Tunu Miah, Michael Smyth and Dave Williams have all contributed in various ways to the ideas. Jim Alty has provided much support in his roles as Supervisor, equipment provider, ideas supplier, and general sounding-board. Chris Hinde, as Research Director, has smoothed the often bumpy path of post-graduate study and has been crucial to the production of this dissertation by forcing me to do what I said I would do, when I said I would do it. Jon Knight's attempts to get me interested in the MBONE and in his multicast extensions to Tcl-DP started me on the path which this dissertation concludes whilst he, Martin Hamilton and the other 'MBONEers' at LUT deserve immense credit for helping to debug the prototype and for their patient explanations of how IP multicast works and why it is *a good thing*.

This dissertation would undoubtably not have been completed but for the support of a Research Scholarship funded by the Human Factors group at BT Labs. Dave Linton, Charles Brennan, Nigel Cliffe, Mark Courteny and other members of the Advanced Applications and Technology Group at BT have all contributed significantly to this work at both the conceptual and practical levels by being willing to discuss ideas, help solve problems and cope good-naturedly with bug-ridden software. I trust that there is something in here for them.

As with any research effort, contact and discussion with peers has variously provoked, goaded and convinced me that I should be thinking about, and drawing upon, the variety of sources that have affected the outcome of this work: Bob Anderson, Sara Bly, Graham Button, Jon Crowcroft, Paul Dourish, Mark Handley, Christian Heath, Paul Luff and the participants of the CHI 96 Doctoral Consortium have all influenced my thinking (whether they knew it or not) - the blame for the results rests, of course, entirely with me.

As will become clear in the dissertation, I am indebted to researchers at the Lawrence Berkeley Laboratory for their foresight in openly distributing their IP multicast audio, video and shared whiteboard tools. In combination with the University of California at Berkeley's, and latterly Sun MicroSystems's, free distribution of Jon Ousterhout's Tcl/Tk, and Brian Smith and Lawrence Rowe's distributed programming extensions (Tcl-DP), these tools have provided the possibility to build multiple media telecommunications systems without a disproportionately large investment of time and effort in building the individual media applications themselves. Unknowingly, these researchers enabled this work to take the shape that it has.

Final words of gratitude should go to my family without whose patience, support and confidence this work would have been given up long ago. I am indebted to my parents for their active encouragement and support irrespective of the impecunious directions I have chosen to take.

More latterly, my wife Claire's faith that I knew what I was doing, that it was worth the time and effort, and that, in the great scheme of things, it was of some importance carried me through the times when I wasn't convinced on all or any of these things.

It can't repay you but to all of you, thank you.

Preface (or "How to approach this dissertation")

This thesis describes an example of applying Ethnomethodology's study policy of re-specifying its foundational assumptions in the light of studies of the phenomena on which it focuses to the field of Human Computer Interaction (HCI).

As such it attempts to show how concepts that underpin system development can be re-specified to provide alternatives that are a viable resource for design. In so doing, the dissertation proposes a design framework that could support a respecification, a requirements capture method that the framework requires and a case study of the construction of a functional prototype based upon principles and resources resulting from this respecification.

These issues provide the conceptual backbone of the thesis and also therefore its structure.

Specifically, Chapter 1 describes the attention that is being given in HCI and Computer Supported Cooperative Work (CSCW) to ethnomethodology, a branch of sociology. The chapter charts the emergence of a relationship between ethnomethodology and CSCW in particular and outlines ways in which such a relationship might work. It identifies and articulates a research program that could be used to reflect on the value of learning from ethnomethodology in various ways. In particular it focuses on ethnomethodology's recommendation to respecify foundational assumptions and base the reformulation on studies of, rather than assumptions about, the phenomena in question.

Taking its lead from Part 1, Chapter 2 describes historically pervasive assumptions about the nature of rules and the way in which they are used by human actors in natural situations. It shows the effect that these assumptions have had on the nature of the systems that are implemented and the way in which they can be used. Drawing on recent critiques, it argues that these assumptions are ill-founded and describes an alternative view of rules based on ethnomethodological studies of rule use. It describes the implications that this view has for the design of CSCW systems. Chapter 3 proposes that systems can be characterised in terms of cues, actions and the mappings between the two. It suggests that designing from this conceptualisation will result in systems that take account of the view of rules developed in the previous chapter. It uses this conceptualisation to describe a number of existent systems.

Part 3, the core practical work, describes a case study that used the conceptualisation of cues and actions to support the design of a prototype group awareness tool and multimedia conferencing call manager, *TelePort*. Chapter 4 introduces the case study and describes an exercise in requirements elicitation that was informed by the conceptualisation developed in the previous section. Chapter 5 describes other resources used to develop system specifications and the implementation of the prototype's user interface. This prototype is related in concept to much of the Media Space work, especially RAVE, CAVECAT and MMConf, and also draws on recent work in the IETF's MMUSIC working group which is working on conference control issues in wide area internetworks. The prototype has been implemented in Tcl/Tk using a version of Tcl-DP with extensions for network programming in a multicast unix environment and uses readily available audio and video software codecs in order to create packet-switched multiple media connections over IP networks. Chapter 6 describes the architecture of this prototype, its functionality and the communication protocols on which it depends. Throughout each of these chapters, issues raised by the design conceptualisation are flagged for later comment. Chapter 7 describes the trial usage of the system and provides a discussion of issues arising from the user interface and of the utility of the system architecture. It provides an analysis of the protocols developed to support the prototype and illustrates how considering these issues from the point of view developed in Chapter 3 can generate novel ideas for future investigation.

Part 4, the last major section of the dissertation, looks back over the case study and provides a detailed analysis and discussion of the utility of the framework and principles proposed, discusses the benefits of this particular respecification, and outlines areas for further work.

Part 5 is a reference section providing the dissertation bibliography and four appendices. The first details the data from the requirements exercise, the second is an Internet Draft describing the group awareness protocol developed for use by the prototype, and for which BT is currently seeking patent protection, the third is the prototype code whilst the fourth is a version of Chapter 4.

This description of the work suggests that the dissertation only makes sense if read in a linear fashion. Whilst this is certainly the intention, readers familiar with ethnomethodology, HCI and CSCW may wish to skim Chapter 1 and the less technically minded may find it best to skim Chapters 5 and 6. For those short of time and/or attention span, the crucial ideas are to be found in Chapters 2 and 3 whilst the discussion and analysis are to be found in Chapters 7, 8 and 9.

Contents

List of Figures

List of Tables

Part I

Setting the Scene

Chapter 1

Introduction

1.1 Framing what is to come

This dissertation is concerned with Human Compyter Interaction (HCI) and more specifically Computer Supported Cooperative Work (CSCW). In particular it is concerned with ways in which insights from ethnomethodology can inform the design of CSCW systems. The dissertation outlines a number of possible ways in which system design can learn from ethnomethodology and concentrates on one particular aspect - namely that system designers should look closely at the foundational assumptions of their discipline and practices and, if necessary, should re-specify any concepts which appear problematic in formulation.

In essence this approach seeks to address the primary question:

- Does system design that takes account of a respecification of particular foundational views in HCI produce demonstrably *better* computer systems?

However, in order to achieve this a number of other questions need to be addressed, namely:

- Just *how can* practical system design learn from ethnomethodology? What concepts can be re-specified and how?
- What sort of *design frameworks* might support this respecification? How does this influence how practitioners think about design?
- What sort of *requirements* capture methods (in the loosest sense) might such frameworks demand?
- How can the output of such methods be *incorporated* into the design and implementation of a system?
- Can such a combination of respecified concepts, theoretical insights, design frameworks and methods result in the implementation of *functional, usable systems*?

The contribution of this dissertation is to articulate this program of work and to provide a case-study of it in action by examining problematic assumptions about the nature and use of rules in everyday life. The dissertation provides a case study of the design, implementation and use of a functional CSCW system - *TelePort* - that takes account of an alternative view on rules and rule use grounded in studies of everyday work and social activities. In order to do so, the dissertation introduces a practical design framework based on the notion of characterising systems in terms of *cues*, *actions* and the *mappings* between the two, combined with a method derived from cognitive anthropology which can be used to elicit such cues and actions from a user population. The dissertation demonstrates the utility of these *design tools* in enabling the implementation of a CSCW system to take account of the re-conceptualisation of rules and rule-use. The dissertation also contributes to current research on technologies to support group awareness and communication in the Multimedia Internet context at the user interface, system architecture and the network protocol levels.

The dissertation therefore articulates an emerging research program linking CSCW system design and ethnomethodology, and demonstrates how the work reported in this dissertation can serve as a crucial enabler for that program.

1.2 A brief tour of intellectual parasitism in HCI and CSCW

In attempting to discover ways in which human-computer interfaces can be intentionally engineered to improve user interaction, the field of HCI has looked to learn as much as it can from any discipline that would appear to offer useful methods or relevant guidance. In each case it is hoped that the discipline can provide access to a set of phenomena that affect the use of computer systems that hitherto may have been ignored, or perhaps not fully appreciated; and that the discipline's methodologies and theories may be of value in discovering how these influences affect design and use.

HCI has predominantly drawn upon psychology, and in particular the information processing paradigm exemplified in Card, Moran and Newell's work that called for the use of an applied psychology in the design, rather than just the evaluation, of human-computer interfaces [42]. Other aspects of psychology such as research on human perceptual abilities and on motor behaviour for the ergonomic design of interaction devices have also influenced the theoretical and practical development of user interfaces [80].

However, as it has become apparent that this focus on basic perceptual and cognitive processes misses many equally important influences on usability, there have been significant attempts to extend the scope of HCI from simply the user interacting with the computer, to the user interacting with the computer in a particular environment and as part of work with other people [80, 14]. The growing interest in computer systems as tools for communication and interaction with others which is represented by the Groupware and Computer Supported Cooperative Work (CSCW) communities has provided additional impetus to this extension. In recognising that

> Computer systems within real world organisations are set within a complex environment of cooperating users, complex and interrelated tasks and organisational prejudices and practices [30, pp 255]

HCI research has turned to disciplines which can provide access to influences on social behaviour. Thus HCI (and hence CSCW) have looked to organisation theory, management science and the social sciences for help in designing interfaces to computer systems that must exist and be used in the context of organisations and groups [13, 134]. Social psychology, for example, has heavily influenced many investigations of group decision support systems (cf. [113] for an overview) and the effectiveness of electronic communication. Development techniques such as *contextual design*, where prospective users are interviewed in their work setting [96], and *participatory design*, where groups of users are drawn into the design team itself [130], have evolved as ways of doing design which attempt to build organisational and social factors into the system from the outset.

Other research, which has been driven by the perceived need to improve the requirements capture and specification methods for CSCW systems, has turned its attention to the development of fieldwork techniques that borrow heavily from the participant-observer paradigm of some sociological and anthropological research in order to access relevant aspects of group settings [98, 140, 103].

This kind of fieldwork, which is often labelled 'ethnography'[1], originally came to the attention of the HCI community through Suchman and Wynn's work on the elicitation of office procedures and practices [146, 148] in preparation for the subsequent installation of an office information system. The idea of using such naturalistic studies to analyse use and inform design was further developed in Suchman's analysis of the problems users had with the interface to a complex photocopier [147]. It has subsequently received much attention in the CSCW community which claims that naturalistic studies can provide detailed descriptions of naturally occurring work settings and the activities and knowledge of which they consist. In this instance, the key benefit of doing fieldwork in this way is claimed to be the commitment to describing the work context and knowledge that is significant to the participants, and doing so using the categories and language which make sense to them unlike other techniques such as task analysis and questionnaire based interviews which are said to be unable to provide this ability [25, 98, 97].

It is important to note that these reports on 'ethnography for design' draw their inspiration from a particular kind of fieldwork - namely *ethnomethodological studies*. Suchman's work on office procedures [146], for example, was heavily influenced by Garfinkel's studies of how work was achieved [69], whilst her subsequent work on the nature of planning drew quite specifically on two aspects of ethnomethodology - a fieldwork technique for recording people's talk, and a theoretical position on the nature of procedures and plans [147]. Recent critiques of the use of 'ethnography' in CSCW have noted that many of the studies that followed Suchman's lead have, whether implicitly or explicitly, been influenced by this ethnomethodological orientation - they have been doing, to borrow a phrase from Shapiro, 'ethnomethodological ethnography' [134].

However, it is becoming clear that using ethnomethodological field studies of workplaces to inform design is but one of the ways in which design and the ethnomethodology can inter-relate. The next section provides an overview of ethnomethodology and briefly describes why it has been attractive to HCI in general, and CSCW in particular.

1.3 CSCW and Ethnomethodology: How did we get here?

Ethnomethodology is a branch of sociology derived primarily from the work of Harold Garfinkel [69, 70] which seeks to provide sociology with an alternative research base. Rather than assuming that people share a common culture (ie. rules and definitions) whose elements can then be specified Garfinkel, as a result of attempting, and failing, to do precisely that, suggested that the apparent organisation, rationality and understandability of actions is constructed by the participants as they go along [135]. As a result, Garfinkel recommended that sociology must re-specify many of its foundational assumptions because, in his characterisation, sociological research proceeded by theorising upon the basis of what were assumed to be cultural givens and concepts. Garfinkel's empirical attempts to explore these theories consistently failed because these givens, categories and concepts are not at all stable but turn out to be re-worked and

[1] see [12] and [134] for discussions of why this term may not be entirely appropriate.

redefined by the participants in the social order themselves [69, 134]. A classic example of this problem is the supposed differentiation between formal and informal working practices which much organisational theorising takes for granted. Bittner, in a study based on detailed field studies, showed that this distinction consistently fails to be useful in categorising practices because what workers classify as formal or informal varies from situation to situation [24]. For Garfinkel then, these findings suggested that sociology could only proceed by throwing away the idea that theorising should come before detailed studies of the phenomena itself. According to this approach, such studies must come first, and the theorising should come second making use of the results of the studies in order to generate theories that were adequate descriptors of, and demonstrably grounded in, the real phenomena rather than in assumptions about the nature of those phenomena.

Crucial to this approach is the use of detailed observational (viz. 'ethnographic') studies that sought to systematically observe and describe the phenomena of interest. In following this approach, ethnomethodologists evolved a set of research methods that were a mix of the anthropological participant/observer together with an interest in the minutely fine-grained detail of the activities under study. Whilst Garfinkel's studies of work [70] are good examples of this kind of research in action, perhaps the most clearly developed exposition is in Sacks and colleagues's ethnomethodological investigations into turn taking in conversation [124]. In these studies, segments of talk are recorded before being transcribed and coded in minute detail in order to provide a detailed description of what was said and in what way.

As many recent authors have noted, these studies have been extremely attractive to HCI and CSCW as both an alternative method of doing requirements elicitation and also as a way of providing a view of human activities, and of work in particular, which is based on detailed studies of what people are actually doing. Thus, the ethnomethodological ethnographies to which Shapiro refers [134] have proceeded in much the same way as Garfinkel's studies of work practices, and have used similar fieldwork techniques. Suchman's studies of work practices [146] are in a similar vein, but her later work on the interfaces to 'intelligent' systems introduced the use of Sacks' conversation analysis techniques as a means to record and describe the ways in which people interacted with and through computational artefacts [147]. Her subsequent development of these techniques into interaction analysis [145] has been influential in CSCW and studies such as those by Heath and Luff [89, 90, 91] have continued this development.

However, as the next section will demonstrate, it is not just from ethnomethodology's methods and techniques that HCI, and CSCW, can learn. Whilst this is perhaps the most developed relationship, others are beginning to emerge. For example, ethnomethodology is a discipline that recommends looking again at foundational concepts from which current theorising proceeds. If system designers do as Garfinkel did for sociology and look again at some of their foundational assumption, it may be that respecifying them in the light of studies of the phenomena themselves can raise important issues for the design of interactive systems. The next sections address in more detail the ways in which ethnomethodology has influenced system design, particularly the design of CSCW systems, and addresses the question of learning from ethnomethodology's recommendation to examine and, if necessary, re-specify foundational assumptions.

1.4 Contributions from Ethnomethdology

Section 1.3 has described the emerging interworking between system designers and ethnomethodologically oriented sociologists. Whilst there have been a number of recent critiques of the mutual impacts of social science and CSCW on each other from the point of view of sociologists interested in what a practical sociology for design might look like, and also in demarcating their territory and clarifying possibly misappropriated terms (cf. [12, 134]), there have been no attempts to map out exactly how this relationship is proceeding, whether it has been successful, nor indeed of what it, as a research program, actually consists. Given the relative recency of its emergence, this is hardly surprising.

However, one exception has been Button and Dourish's attempt to articulate this interworking in terms of their research program of 'technomethodology' [33]. Their articulation of what is going on is built around the suggestion that ethnomethodologically oriented studies tend to fall into one of two types:

Studies of Everyday Action: Such studies have focused on describing everyday social activities at an extremely fine grained level. In particular, the use of communication technology has been extensively studied from the perspective of conversation analysis and has provided some detailed insights into the ways in which human communication as mediated by Media Space technology differs from 'normal' communication (cf. [89, 91]). Other studies, with a similar analytic bent, have examined the way in which activities and collaboration progress in a variety of situations such as control rooms, surgeries and architect's offices (cf. [90, 76, 93, 92]).

Studies of Work: Although similar in nature to the first type, these studies concentrate more on the way in which work activities and practices are organised. Suchman's work on office procedures [146] and more recent studies on Air Traffic Control (eg. [143]) are prime examples, as are those that describe work practices in various industries such as manufacturing [34] and software design [35] as well as in government offices [28].

In discussing the ways in which such ethnomethodological studies can influence or feed into design, Button and Dourish contend that 3 forms of 'technomethodology' can be identified and that, whilst this characterisation is perhaps difficult to sustain in reality, they claim that it at least serves to frame discussion:

Learning from the Ethnomethodologist: Here, an ethnomethodologist is used as a field worker who can study the work domain for which a system is to be designed, or into which the system has already been deployed.

Learning from Ethnomethodological Accounts: In this case design makes use of the specifically ethnomethodological account of a particular work domain or human activity as a resource for design.

Learning from Ethnomethodology: In this case the suggestion is that system design can take heed of the way in which ethnomethodology has sought to recast ('re-specify' in the jargon) foundational concepts within sociology. The suggestion is that the examination and respecification of foundational concepts in HCI could have considerable consequences for the nature of systems and the way in which they are built.

A great deal of effort in the CSCW community has been devoted to the first two of these relationships - well articulated research programs have investigated the utility of ethnomethodological (1) field work and (2) accounts in the design process. As this chapter will briefly describe these programs are now relatively well-populated by publications which demonstrate the utility of an ethnomethodological orientation.

But what, then, of the third relationship? In calling for more attention to be paid to a foundational relationship between ethnomethodology and design a research program has been articulated which is currently poorly defined.

This chapter triangulates the work described in the rest of the dissertation by providing a brief overview of the first two relationships listed above. It illustrates that these relationships are currently maturing as research programs and that the utility of an ethnomethodological orientation is relatively well demonstrated. The chapter then focuses on the idea that system design can *learn from ethnomethodology* and describes an outline of a research program to explore this idea which would require the construction of a number of conceptual and practical bridges.

In essence, this dissertation seeks to provide the first attempt to build these bridges in order to enable the 'technomethodology' research program to move to a stage where the utility of the initial ethnomethodological insights can be evaluated. In order to do this, the dissertation describes a case study which considers the implications of respecifying implicit assumptions within HCI and CSCW concerning the nature of rules and their use in everyday work, and applies this respecification to the design and implementation of an interface to a multimedia telecommunications call manager. It therefore describes an example of design *learning from ethnomethodology.*

1.4.1 Learning from Ethnomethodologists

As was noted in Section 1.3, perhaps the most widely practised relationship between ethnomethodology and design is the leveraging of the field work skills of an ethnomethodologist. Section 1.3 described how ethnomethodology can be characterised in part by a focus on the importance of fine-grained studies, 'in the field', of what people actually do. In this relationship then, ethnomethodologists are valued as members of the design team who are able to study the activities of (for example) potential user groups and can use the knowledge thus gleaned to inform design (see Figure 1.1).

Perhaps one of the best documented design projects that illustrates this use of ethnomethodology has been the work carried out by a team of sociologists and computer scientists at Lancaster University. In order to inform the design of a prototyping toolkit [22], and eventual prototype interfaces for a computer based air traffic control system, the sociologists carried out a number of detailed workplace analyses at air traffic control centres [87, 99]. As is described in the computer scientists' concluding publication [140, p361], the design process involved periods of sociological fieldwork, followed by 'debriefing meetings' at which the fieldworker(s) and system designers attempted to reach some common understandings about significant characteristics of the work domain. Once the system designers had some notion of what activities were significant, they could then question the field worker further in order to gain a deeper understanding of what kind of technological support might be required, and what form it

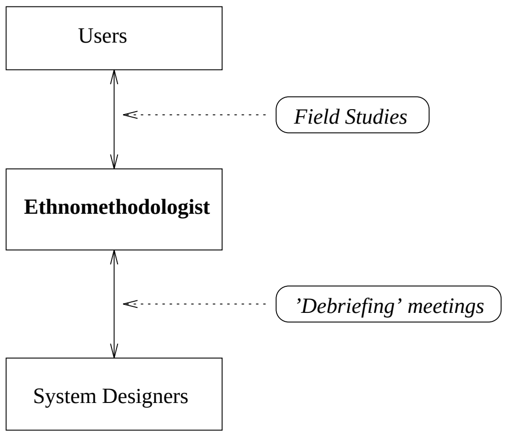

Figure 1.1: Learning from ethnomethodologists

should take. These meetings were then followed by more focused fieldwork that attempted to answer some of the designer's more detailed questions so that an iterative process of debriefing meetings followed by concurrent fieldwork and design was established.

Other examples of this kind of relationship include Heath and Luff's extensive and detailed analysis of the way in which communication technologies, such as Media Spaces, are used [89, 91]. In particular, these studies have been of use in highlighting the deficiencies of such technology when compared to everyday interaction and communication patterns. Whilst these studies, in contrast to the Lancaster work, do not explicitly describe the mechanisms by which these insights have been recycled into design, it is clear that subsequent development of the Media Space technologies has drawn heavily from them (cf. [73]).

More recent examples of these kinds of studies include fieldwork that has been devoted to uncovering aspects of work in customer service industries [123] and in governmental offices [27] both of which are presumably intended to allow the fieldworkers to act as a resource for or otherwise influence the design or redesign of computer systems in those particular work domains.

As can be seen, the primary resource for design, or re-design in this instance, is the knowledge that the ethnomethodologically oriented fieldworker has about the particular domain under study. As such the ethnomethodologist can be seen as, and in some of the research described was used as, a surrogate or proxy for the real user community themselves. The fieldworkers, in some senses, represent the interests of the users during the design process so that

> Design ideas can be "bounced off" the ethnomethodologist, who draws on field observations ... to contribute to aspects of design. [33, p22]

Further the fieldworkers could act as preliminary evaluators of the system by helping

> ... to find gross design errors which can irritate end-users and cause them to reject a system without a thorough evaluation. [140, p362]

Button and Dourish suggest that the locus of the ethnomethodology in this instance is

> primarily ... in the ethnomethodologist's head. [33, p22]

As they point out, the fact that the fieldworker has an ethnomethodological orientation is not of direct concern to the system designers, nor to the way in which knowledge is being transferred from the work domain to the design process. One might therefore ask, 'What is the value of an ethnomethodological orientation in this instance?' and answers to this are less well articulated than is the fact that some sort of ethnographic fieldwork can be of use in system design. In partial answer, Button and Dourish suggest that:

> presumably [ethnomethodologists] will use their analytical perspective in shaping a story about the field setting, and in evaluating and contributing to the design. [33, p22]

What then is the value of learning from an ethnomethodologist *in particular*? The answer derives for the most part from the emphasis that ethnomethodology places on detailed studies that aim to uncover the activities in which people engage and so, it is claimed, these studies produce rich descriptions of work practices. For the study of air traffic controllers, this emphasis leads to the kinds of detailed descriptions of practices that are found in, for example, [87] and [98] where the implicit work activities are recognised as highly significant in the way they contribute to the successful management of the flight strips (and hence the aircraft). Thus, the value of the ethnomethodologist can be assessed by the degree to which these insights prove useful in design. Whilst Hughes and colleagues note:

> While we are confident that the ethnographic studies have been valuable in the context of system design, we must admit that this is as much a matter of faith as it is backed up by the evidence. [98, p250]

their colleagues are, perhaps, more convinced that such studies can be 'surprisingly useful' [141] and, indeed, subsequent publications have set out a number of features of ethnomethodological ethnography that are claimed to be of value [97].

In the case of Heath and Luff's analysis of Media Space technologies, the value of the studies is reflected firstly by their ongoing work on the redesign of such technology to alleviate the problems of flexible access and of asymmetry that they uncovered [73, 93]. Secondly, their studies are generally recognised within the field (see eg. [117]) as providing insights that had not, in general, been apparent from earlier studies of Media Space technologies, such as those described in Fish et al [64], that used evaluation techniques derived from other perspectives. This then is evidence of the value of an ethnomethodological orientation.

Of the other studies less can, as yet, be said about either their intrinsic worth or the value of an ethnomethodological orientation in particular because the projects to which they contribute have not yet moved from the fieldwork to the system design and implementation stages. As these research programs reach maturity, it is likely that they will provide the kind of reflective analysis that concludes the Lancaster based project and so will help to answer such questions.

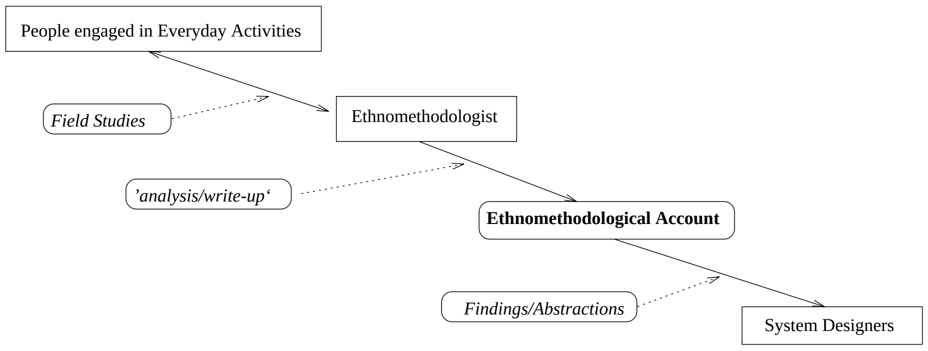

Figure 1.2: Learning from the ethnomethodologist's account

Overall however, it is clear that a definite research program exists - the basic outline presented in Figure 1.1 is being followed by many of the studies referenced or described. Thus, as Button and Dourish make clear, the work here is in following the program in order to assess the value of an ethnomethodological orientation in this kind of relationship, and to document methods of practice than can be used to realise this value in the system design process.

1.4.2 Learning from the Ethnomethodological Account

In this relationship, the design or redesign of a system is based upon an explicitly ethnomethodological account of a domain or activity. As is shown in Figure 1.2 this introduces a second mediating artefact between the users and the designer - the ethnomethodologist's account. Such an account is essentially the 'write-up' or 'report' that is a result of an ethnomethodological study and which generally takes a particular form. As evidenced by recent collections of ethnomethodological accounts that have technology as their orientation [32] and their increasing appearance in the CSCW and HCI literature, this form consists of an introduction to the setting for the study (the *where*); detailed descriptions of work practices together with snippets of data, such as videotaped conversations or activities, office plans and schematic representations of work flow (the *what*); one or more sections highlighting phenomena that the ethnomethodologists take to be particularly significant and, in the case of studies motivated by the problems of system design at least, a general discussion of the implications for system design (in many senses, the *why*).

Whilst there are clearly many overlaps between this relationship and the previous one, Button and Dourish note that in learning from such an account, the ethnomethodological orientation makes itself far more explicit than when it merely resides 'in the head' of the fieldworker. In contrast ethnomethodology is now

> ... an explicit part of the communication between field and design ... which proceeds from an understanding that ethnomethodology is an analytic perspective, a form of 'writing up' rather than a form of data collection. [33, pp 22]

It is clear that the work of Heath and Luff also fits into this form because the analyses they provide can also be considered as explicitly ethnomethodological accounts (notably [89, 91]). So too with a number of the studies that derive from Lancaster's Air Traffic Control work such as the accounts reported by Harper and Hughes in [87], and the studies reported in recent HCI and CSCW conference proceedings which take a wide range of work domains as their subject matter (eg. [34, 35, 28, 123]). However, there are relatively few documented descriptions of exactly how such accounts can be drawn into the design process although Suchman's accounts of work practices [146] and of sense-making in human-computer communication breakdowns [147] are, perhaps, examples of studies whose implications for design have been quite thoroughly worked through.

However one particular example of the use of ethnomethodological accounts deserves attention and this is the attempt to leverage the models of turn-taking and conversational activity provided by Ethnomethodological Conversational Analysis in the design and implementation of 'conversational interfaces'. Specifically, this research program, which is reported in the collection edited by Luff et al [106], took as its resource the 'findings' (in the guise of conversational rules and structures derived from [124]) of conversational analysis and used them to implement a natural language user interface to an public welfare rights advisory system for the UK Department of Health and Social Security. Whilst discussion continues as to whether or not the system and user can truly be said to be conversing (see for example the exchanges in [106], and [151]) this is quite clearly an instance of system design *learning from an ethnomethodological account* of turn-taking in conversation. That a system could be built incorporating the product of conversational analysis is clear and is documented in [68], whilst user evaluation of the system has demonstrated the value of this particular orientation in design.

As with the previous relationship, it is clear that ongoing research has and is mapping out the processes by which system design can learn from ethnomethodological accounts. Whilst the use of conversation analysis is a clear cut example, the implicit and occasionally explicit grounding of a number of recent CSCW systems in ethnomethodological accounts of work practice illustrates the point that here too, a research program has been articulated and it is becoming well-represented in the literature (eg. [53, 23]). As with the previous relationship, the work is in establishing the value of particular ethnomethodological accounts, and the processes by which such accounts can inform design in specific or even more general design cases.

1.4.3 Learning from Ethnomethodology

Perhaps the most radical of the relationships that could exist between ethnomethodology and design is the one Button and Dourish characterise as *learning from ethnomethodology.* Here the suggestion is that system design needs to pay attention to fundamental axioms and theoretical stances that are central to the ethnomethodological position. As they explain

> we consider [that] the implications of foundational ethnomethodological principles - those insights and perspectives which characterise the discipline - hold for both the artifacts and process of design. In this approach, design does not take on board ethnomethodological analysis and insights, but takes on board the very study policy

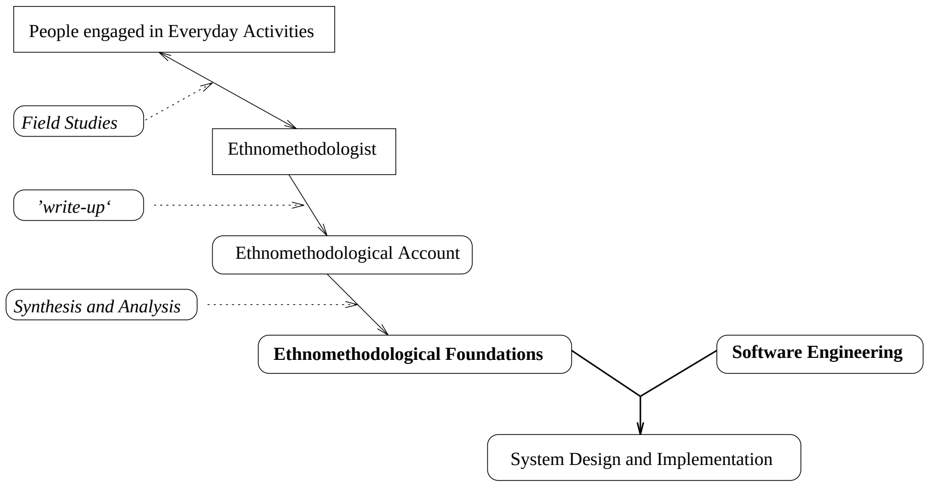
People engaged in Everyday Activities
Field Studies
Ethnomethodologist
'write-up'
Ethnomethodological Account
Synthesis and Analysis
Ethnomethodological Foundations
Software Engineering
System Design and Implementation

Figure 1.3: Learning from ethnomethodology

> of ethnomethodology. [33, p22]

One such study policy is ethnomethodology's recommendation that the concepts and assumptions from which sociology had traditionally proceeded should be examined and re-specified. As has been mentioned, one such example is that the traditionally held sociological assumptions of commonly held meanings (and hence culture) and of rational action are not borne out by empirical investigation. For ethnomethodology the key proposal was therefore: if these assumptions have to be suspended, just how do people achieve agreement on meaning such that this commonality appears to be true, and how do they go about 'sensible' action [135]? Such a re-examination of foundational assumptions is exactly what this aspect of technomethodology recommends for Computer Science, and for HCI in particular.

In contrast to the first two relationships then, this is viewed as primarily a theoretical connection between the foundations of system design and of ethnomethodology. Rather than looking to learn from the practical endeavours of ethnomethodology in particular system design situations, this relationship looks to draw lessons from views or stances that are inherent in the ethnomethodological orientation itself (see Figure 1.3).This opens the way for design to consider such issues as the situated nature of everyday activities,

> ... practical action and representation, achievement and mechanism, phenomenon of order and accountability [33, p22]

as well as

> ... generalisation and abstraction, configuration, data and process, fixedness and mutability. [33, p 22]

in a more general manner, and from a different perspective than has previously been the case. In essence then, the suggestion is that by emulating the way in which ethnomethodology has sought to examine and then re-specify basic assumptions in sociology, 'technomethodology' can re-examine any of the key

HCI and Computer Science concepts and in re-specifying them, can provide instructive and novel insights into the nature of software systems, and into the assumptions that are embedded within them.

By way of an example of this relationship, Button and Dourish describe current efforts to design flexible systems to support group work which have been based on the ethnomethodological notion of 'accountability'. One result of the rejection of the idea of commonly held meaning is that for ethnomethodology all social actions must be observable and reportable by and to the participants in those actions [135] - they are therefore *accountable* and the participants explore these accounts in order to work out what is going on. In recent work, Dourish describes the application of this idea to the design of computer systems [51]. Specifically, he suggests that computer systems can also be considered as accountable - that is their actions are observable and reportable by their human users. A natural result of this accountability is that humans attempt to find out what is really going on through the account that the computer presents of itself - in other words its surface representation. The crucial point for Dourish is that the traditional software engineering approach of abstracting the underlying system activity away from the interface breaks the link between the surface representation (the user interface metaphor for example) and *what is really going on inside*. As a result users who attempt to 'work out what is going on' when something goes wrong are consistently frustrated because the account that the system presents of itself is virtually never an accurate or causal 'account'.

The result of applying these ideas to software engineering forces, it is claimed, a fundamental rethink of how to build systems in general, and systems to support group work in particular. It leads Dourish away from the traditional notion of functional abstraction towards that of open systems and computational reflection [51, 52]. Whilst a full description is outwith the scope of this chapter, the idea is that a system provides mechanisms by which the internal functionality of all of its constituent parts is open to inspection and alteration. Dourish's claim is that in the first instance, the inspection allows the surface representation (interface) to forge causal connections with (and so generate 'accounts' of) what is actually going on, and to actively maintain the accuracy of those accounts. In the second instance the ability to alter the internals of the system has implications for configurability and hence for the flexibility of the system itself. This program of work is reported in full in [53]. What is important for the present discussion is that it is clearly an attempt to learn from ethnomethodology at a fundamental level. As such the work fits into the research program sketched in Figure 1.4 which provides a diagrammatic description of how this third form of technomethodology might work, and what needs to be done in order to concretise a research program to explore this relationship - a research program anticipated, but perhaps not fully articulated, in Button and Dourish's paper.

In essence, the research program to which they look can be characterised as aiming to answer the primary question:

- Does system design that takes account of a respecification of particular foundational views in HCI produce demonstrably *better* computer systems?

That Dourish's work is contributing to such a program is clear when it is considered in the context of Figure 1.4. This sketchy outline of the project

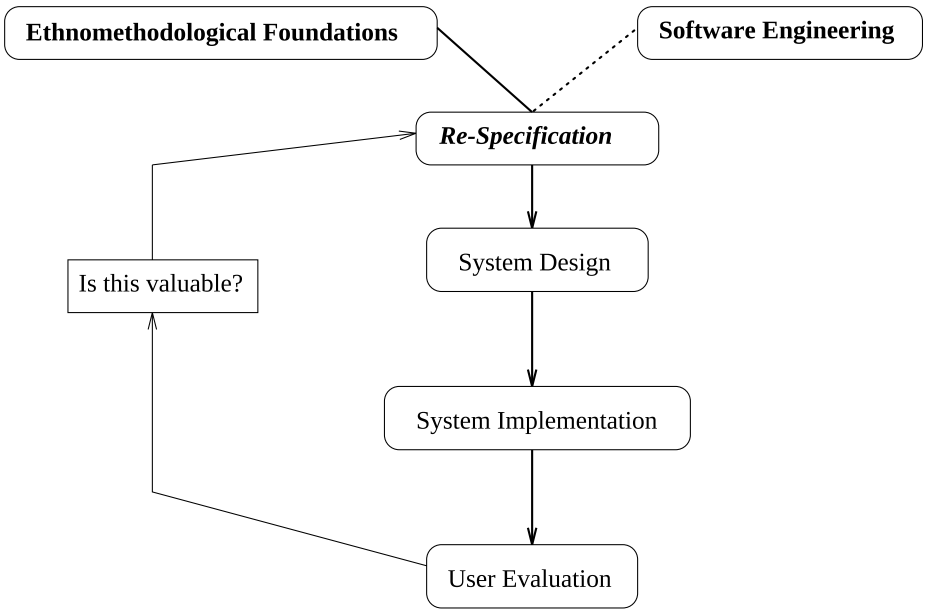

Figure 1.4: Technomethodology: A Sketch of a Research Program

demonstrates that in order to be able to begin to answer the primary question, a number of conceptual and practical bridges must be constructed before the general question of whether this form of technomethodology is valuable can be answered. Thus technomethodology cannot proceed until the stages that are described by Figure 1.4 have been fully worked through. Only when it is possible to evaluate functional systems which can be shown to incorporate reformulated HCI concepts, will it then be possible to state whether or not doing design in this way produces systems that are, in some sense, 'better'.

As this chapter has suggested, there is currently far less research attention being given to this aspect of technomethodology than to the other forms that have been discussed in previous sections. Indeed it is currently extremely unclear how technomethodology can move toward answering its prime motivating interest in this instance because there is little, if any, attention being given to building the required enabling bridges.

It has been claimed elsewhere that learning from ethnomethodology could be of value to system design and that recent work has provided some illustrative examples of how this relationship might proceed [33]. What is missing however, is firstly a clear articulation of a research program to explore this relationship, and secondly a set of conceptual and practical design processes or frameworks that can be used to enable the research community to assess the value of this form of technomethodology. Mapping such a path, and evolving such frameworks and practices is critical because it is only by encouraging other researchers and developers to follow the path, use the frameworks, and evaluate their systems, that the CSCW research community will be able to build a consensus on the value of *learning from ethnomethodology*.

The remainder of this dissertation documents a case study in system design that sought to build such bridges. More specifically the case study reports on the design and implementation of a functional CSCW system that explicitly takes into consideration a respecification of rules and rule-use, turning to an account that is based on detailed field studies of work practices, rules and cultural norms.

Part II

Respecifying Rules

Chapter 2

Rules, and rule-use in CSCW Systems

2.1 Introduction

Much of the literature in Human-Computer Interaction (HCI), Computer Supported Co-operative Work (CSCW) and increasingly, Telecommunications, reports work aimed at constructing systems to support human to human communication and interaction. Such systems include experimentation with computer controlled audio and video to provide 'Media Spaces' and the interaction and collaboration of distributed workers through shared information systems, shared applications and other group-oriented tools such as shared editors and multi-user whiteboards. Other systems, mainly originating from work on office automation, have provided work and workflow management systems, whilst others have sought to support decision making in both distributed and face-to-face groups.

Inherent in much of this development has been the use of everyday norms, rules and practices to provide users with a familiar conceptual structure for the system's functionality and their interaction with and through it. Examples of this strategy include systems that make use of practices from familiar activities such as meetings, lectures, or walking down a corridor. Further, the majority of these systems have attempted to leverage everyday rules and practices by embedding some form of model of them in the computer system. The nature of these rules, how they are used and how they manifest themselves in everyday work is (or should) therefore be of considerable interest to designers of such systems [121]. Until relatively recently the design of such systems has been based on a view of rules and rule-use that draws its intellectual heritage from the bureaucratic view of work. As this chapter describes recent work, predominantly in the European CSCW community, has demonstrated that such a conception of rules has resulted in the construction of inflexible systems which have often subsequently failed because they are incapable of supporting the capricious nature of everyday work. This chapter recasts this critique in terms of a respecification of the traditional view of rules. The chapter draws on these recent critiques and on empirical studies of work and rule-use that emphasise the situated, flexible nature of work in order to describe the result of this respecification.

Thus as technomethodology recommends, this chapter examines and respecifies a core foundational concept - the nature of rules and their use. It then describes an alternative view that has drawn on ethnomethodological studies of rules and rule use. This chapter describes some of the studies that have led to this reformulation, and discusses the implications that it has for design and for the nature of the systems to be designed. Throughout the chapter, comparison will be made with the traditional view of rules which has often, if implicitly rather than 'on purpose', been embedded in system design. In conclusion, it is suggested that the construction of systems flexible enough to support everyday work and which take account of the notion of work as situated action and of rule-use as interpreted, negotiated and dynamic will require a practical design framework. The remainder of this dissertation is the development and use in a case study of just such a framework.

2.2 Rules, Rule-use and System Design

Many current approaches to the design of CSCW systems tend to constrain users with behaviour options based on models of work or social practices that have been embedded within the system itself. Thus systems are built which have rule-based *mechanisms* embedded within them based, in turn, on models of group work and behaviour derived either from the literature, or from studies of the work domain [77, 121, 132]. This section describes a number of exemplars of this approach from a range of domains and in each case illustrates that many of their problems are due not just to over-simplified models, but to the assumption that implementing rule-based models *in this way* is an appropriate solution in the first place [121].

Perhaps the most widely discussed examples of such systems are structured messaging tools such as The Coordinator [1] [158, 65] and COSMOS [29]. In the case of The Coordinator, the system provides users with ways to create, send, archive and review records of 'turns' in conversations. In particular it provides two models of 'efficient' conversation which users are encouraged to follow - a request or an offer. Which type a user selects then affects the subsequent allowable types of responses with the intention of forcing users to recognise the sequences of stylised conversations in order to more efficiently bring them to conclusion. Thus requests can be answered by 'accept' or 'decline' whilst offers of action are followed by 'report' of completion or 'revocation' of the promise. To say the least, the success of The Coordinator is still much debated both in terms of its utility and the design assumptions that underly it (cf. [144] and subsequent replies [159] and comments). Field studies of the system in use have highlighted the problem of building communication systems which explicitly enforce particular communication structures. Bullen and Bennett for example noticed that users tended to send request messages irrespective of their content - in many cases circumventing the structure and simply using The Coordinator as a free-form email system [31].

Multi-user editing and drawing tools such as QUILT [63] GROVE [62, 61] and ShrEdit [105, 54] are also good illustrative examples of this point because many such systems use models of organisational roles and rules to determine what editing functionality is available to the users. In the case of QUILT, users may be readers, annotators or co-authors but precisely which is determined by a combination of the user's social role, the nature of the information being manipulated and the current phase of the writing activity. As a result, changes in roles, working styles and activity phases must involve reconfiguration of the system itself in order to maintain the accuracy of these rules. It is interesting to note therefore that whilst GROVE had similar role-based permissions for editing functions, its creators report that they were very rarely used precisely because of their inherent inflexibility. Instead GROVE's default mode of allowing all users to see and edit everything turned out to be

> surprisingly useful, because social protocol mediates.[62, p47]

Recent studies of the actual process of co-authoring such as that reported by Beck and Bellotti [18] have revealed that significant activities do not revolve around the carrying out of pre hoc agreements but instead emphasise the

> great flexibility and context sensitivity with which co-authors interpret information and situations and come to decisions about appropriate courses of action, even to the extent of unilaterally **contradicting** agreements. [18, p235, emphasis in original]

[1] 'The Coordinator' is a registered trademark of Action Technologies.

Thus, even where initial roles or strategies had been agreed in order to ease the problem of managing access to the documents, this structure gradually and unproblematically broke down as the writing progressed. Other studies of co-authoring have also illustrated that access privileges in practice tend to be far removed from initially defined roles (cf. [54],[121]). Clearly then, implementing systems based upon assumptions of co-authoring, or indeed upon rule-based models of practice is problematic. Instead, Beck and Bellotti call for the design of co-authoring systems that can support the flexible nature of work that they have observed. As will be discussed, this call is common to many of the analyses of the systems described in this section and, it will be argued, has it's roots in a growing appreciation of the nature of rules and processes in everyday work, an appreciation that can trace it's conceptual roots to ethnomethodology.

Other obvious examples are systems which use rule-based models to regulate access to shared information spaces [79, 136, 138]. Here, for example, rules relating roles and permissible actions are used to restrict access to particular data, or to provide particular users with specific views on that data. Clearly an access model such as the familiar UNIX shared file system is also an example of this type with read, write and execute permissions being set for each file at the owner, group and public levels. In this instance the rules for who can access which files, and with what privileges, are applied by the system manager and, to a certain extent, the owner of a file. As Greif and Sarin have noted, following experiences with a shared calendar tool [79], such models require refinement so that operations can be performed 'on behalf' of another user, given appropriate authorisation; permissions can be negotiable; and restrictions can be over-ridden in situations where there is no other course of action. It is interesting to note that these refinements are not only required if such models are to support everyday work, but are necessitated by the very fact that the rules or models are being inflexibly embedded in the system in the first place [121].

Less obvious examples of the embedding of social rules in CSCW systems are provided by workflow and office automation systems. In this case some sort of abstract model of the 'right way' in which the work is to be done is encoded into the system. Whether this model is derived from documented codes of practice [102] or from studies of the work domains themselves [155] is, in many ways, an irrelevance. Whatever the derivation, these systems assume that routine work and the flow of information, tasks and constitution of goals are part of an external order that is a 'given' and which people 'enact'. The consequence of this assumption is that such systems embody

> the use of explicit rules and procedures to coordinate the internal operations of the organisation.[155, p122]

Unfortunately for workflow systems this assumption turns out to be far removed from the truth. Gasser, for example, has shown that 'routine' work is anything but routine because offices are fundamentally open systems where the 'exceptions' (that are the bane of rule and process oriented workflow systems) are in fact the norm [71]. It turns out that there is no sensible demarcation between routine and exception - and attempts to build systems based on this mythical contrast (ie. implementations based on models of an organisation's 'routine' work procedures) have had a notable lack of success because, as Wastell and White have noted, following Suchman [146],

> such models fail to capture what is most essential about office work, namely its contingent and problem solving character...[so that the models]...become reified in inflexible and obstructive office systems. [155, p125]

As Wastell and White discovered, the introduction of such systems into a workplace can meet with fierce opposition - in their case simply because the system was too prescriptive.

Given the continuing business interest in workflow technology, these problems are currently receiving renewed attention, and reviews of experiences with such systems have discussed some of the issues involved in reconciling the design of workflow systems with the 'real' nature of work [1]. Indeed it is becoming clear that such troubles are not just due to the use of sparsely detailed models of activity since an obvious solution to that is to improve the detail in the model [21]. Rather, it is because such systems are attempts to structure user interaction through the inflexible, mechanistic encoding of social rules [121, 132].

A final illustrative example of the implementation of social rules is the use of access controls in Media Space technologies. In EuroPARC's RAVE system the controlling software, Godard, uses user-specified access control lists to determine whether a particular connection of a given type ('glance','vphone','office share' etc) should be created [72, 50]. Thus each user maintains a set of service specific lists detailing who can connect and in what way. This idea has been extended by the University of Toronto's CAVECAT system which has used an iconic depiction of an office door to represent the 'availability state' of the user to whom that door 'belongs' [109]. Here, the state of the office door explicitly determines which of a set of connection services (similar to those of RAVE) can be created and the rules that govern these permissions are embedded within the system itself. The problem then is how to manage and maintain these rule-sets, and how to cope with the flexibility of access that is required since today's interruption could be tomorrow's emergency.... As Dourish notes in a review of this and other systems [50], this formalisation of social conventions serves merely to replace the social with the technical and when this transformation occurs the resulting systems tend to be fundamentally less flexible. In this respect it is interesting to note that Bellcore's CRUISER system [122, 64] embodied no explicit access control models but instead used a combination of the principle of reciprocity[2], and a user-set lock (such that all connections are refused) in order to provide a basis on which to build a privacy culture. Experience with this system suggested that

> people are every bit as sensitive to the possibility of committing a socially offensive act - of intruding - as they are of being intruded upon.([41], p30)

and that

> within a single work group there is often a common group norm about privacy settings and expected availability. [ibid]

That is, users of CRUISER, were using socially constructed methods of control even though the system itself did not necessarily explicitly support them.

[2]If I can see you then you can see me.

Perhaps the most fully explored use of embedded access rules for Media Spaces is that provided by Anderson et al's Doors system [11]. Using the same representation of availability state as CAVECAT, the Doors system explicitly altered the available communication functionality as and when users altered their door state. Thus when a door was set to ajar, users could 'glance' or 'knock' but not 'enter', similarly when a door was set to closed, only a 'knock' was allowed by the system. These access rules had been previously elicited from a representative user population in an attempt to generate a model of 'how to enter an office' that could be of use in the design of such an interface (cf. [10]). However, as Anderson and Alty discuss, because this approach assumes that social models of 'what to do when a door is in position X' can be capturable in some rule-based formalism, and that this formalism can then be translated into a rule-based interface, the resulting system is extremely inflexible [10]. It does not, in essence, deal with the exceptions to 'the rule' that are a standard part of normal behaviour. In fact who can do what, when and to whom, is fundamentally context sensitive. As Anderson and Alty discovered, it is simply not possible to generate a rule-based model that captures, even remotely, a flavour of these rules because deciding what is and is not appropriate behaviour depends on open-ended and unforeseeable factors such as urgency, subject, context and social status. The more rules that are elicited, the more become necessary to define additional scenarios...[3] Thus the Doors system is as guilty of misconstruing the way in which rules are used, and of ignoring the consequences of implementing normative models of those rules, as are the other systems described in this section.

It appears then that embedding social rules into interactive systems is prevalent in system design and implementation. It appears also that in doing so, system designers are implicitly or explicitly drawing upon a view of work, indeed of human behaviour, that has become widely accepted [114]. This view conceives human behaviour as fundamentally rational, goal or plan oriented and rule governed so that orderly activities progress from the (assumed) fact that human actors are equipped with a set of rules that they follow. Thus whenever a human actor encounters a particular situation, one or more of the rules they possess will be applied (see [39] for a detailed exposition of these ideas). In the context of work and organisation, this view is often termed the bureaucratic model and traced to Taylor's work on productivity management [150]. As Morgan suggests, this view has become pervasive not only in organisational research but also, perhaps due to it's firm rooting in the cognitive sciences, in many areas of modern society [114] and so, by extension, in many areas of creative endeavour - such as the development of interactive systems as CSCW researchers have noted [132, 15, 120].

It is perhaps inappropriate at this point to digress into the metaphysics (and psychology) of software engineering but the fact that this 'traditional' bureaucratic view of procedures and rules has become the 'accepted truth' and led to so much 'assumption implementation' [57] is an interesting manifestation of software engineering's tendency to modularise, rationalise and to decompose

[3]Interestingly this was exactly the same phenomenon experienced by Garfinkel's students when asked to list the implicit meanings embedded in a very short snippet of conversation. In the end they could not complete the task because it was impossible to do so - defining any one meaning always lead to the need to define yet more ([69, p24-25] and see also [135, p30]).

implementation problems[4].Whatever the root cause, this conception of rules and the way in which they are used has been foundational to the design of many interactive systems. More recently a critique of this conception has been developing which is based on its fundamental conceptual flaws and on evidence of its failure (cf. [132, 120]). This critique has asked whether this particular model of rules (and of human action in general) is usefully accurate when used to implement interactive systems, and if not, whether or not an alternative model is available. This section has described this critique and the next describes just such an alternative.

2.3 Respecifying 'Rules-Use': Ethnomethodology, Rules and Everyday Work

The previous section described the way in which the design of CSCW systems has frequently conceptualised rules as governors of behaviour. Put simply, this model is that people act according to procedures which can be specified as rules and this assumption (consciously or not) has underpinned the implementation of many interactive systems [121, 127, 132]. But what if this were not the case? What if, as the descriptions in the previous section have hinted, this conception does not seem to work? What happens if this conception of rules and rule use is re-specified in a way that draws upon empirical studies of how people actually do use rules. This respecification is the next move in this critique and it is to ethnomethodological studies of rule-use in everyday life that it turns.

As was discussed in Section 1.3, the ethnomethodological stance emphasises the study of how everyday orderliness is produced, recognised and described by the people engaged in that order. By concentrating on everyday practices in this way, Garfinkel and his followers have chosen to base their conceptual work on finely grained field studies of how the practices of interest are carried out [135]. In so doing, those ethnomethodologists who have chosen to study organisations have developed a view of work that is considerably at odds with the traditionally accepted 'bureaucratic view' of work that was briefly described in the previous section.

Based on their detailed studies of how work really does get done, ethnomethodologists propose a view of work that stresses the interpretive nature of rule use in each individual's situation. Empirical studies of the actual use of rules, such as those reported in Garfinkel's collections [69, 70], in Zimmerman's study of case allocation in a Health Care Centre [160], Wieder's discussion of meaning by rules in structural semantics [156] and in Button's recent collection [32] have all concluded

> that persons continually discover the scope and applicability of rules in the developing occasions in which they use them. [156, p109]

According to this view, there is more to behaviour than mere rule or procedure following because being able to apply a rule requires much more than just knowing about the rule itself. Since rules are necessarily incomplete, it is also necessary to be able to judge the relevance of a rule in a given situation [114].

[4]For illuminating discussions of these and related issues the reader is directed to two recent articles by Philip Agre [7, 6].

Social rules then are not stable in meaning, nor can a finite set of rules be determined that can be invoked in any or all situations. Thus, as Hughes and Harper describe:

> ... rules have to be applied within a setting such that what a rule or procedure means, what actions fall under it, is a matter which has to be decided, judged, determined on occasions of its application. Social actors, that is, have to make judgements as to whether *this* rule applies *here* and *now* in respect of *these* circumstances. [87, p128, emphasis in original]

The ethnomethodological stance on the nature of social rules and behaviour can be summarised as:

Context Sensitive: Each member of a culture is able to make their own choice about what is appropriate in a given situation. An excellent example is given by Morgan:

> our understanding of the nature of the [drinks party] situation will lead us to invoke certain rules (eg., that it is OK to go to the refrigerator to fetch another beer, or to search for a corkscrew in the kitchen drawers), even though these rules might be considered quite inappropriate on other occasions. The point is that the norms operating in different situations have to be invoked and defined in the light of our understanding of the context.[114, p130]

As a result, it is possible for two members of a culture to act differently in ostensibly the same context; and conversely to act the same in ostensibly different contexts.

Indefinable: That is, it is simply not possible to define a set of rules that can specify all the possible courses of action in a given situation. An excellent example of this is provided by Heritage who shows that the rule-governed model cannot even cope with as supposedly simple a situation as a greetings exchange because it is impossible to specify a complete set of contingencies over which the rules will operate [94, p104].

Meaning is Constructed in situ: As with the notion that the meaning of language cannot be determined outside of the context of its use, so the meaning and importance of particular norms or rules is constructed as they are used. Thus, rather than seen as governing behaviour, rules are seen as resources used in the achievement of that behaviour. In a study that concentrates on the idea of 'organisational practices as rules', Zimmerman concludes

> that the notion of action-in-accordance-with-a-rule is a matter not of compliance or noncompliance per se but of the various ways in which persons *satisfy* themselves and others concerning what is or is not 'reasonable' compliance in particular situations. [160, p233, emphasis in original]

So, in order to be able to justify their actions, workers are continually exploring the meaning of the rules of practice or accepted procedure so that their actions in doing the work can be said to be in accordance with those rules. Rather than having rules cause actions, workers actions are arranged such that they appear to be in accordance with what the rule 'would really mean' in that situation.

Variable: The same set of criteria can have different implications for different people. For example Hartland, reporting on the use of 'intelligent filters' on an electronic cardiograph machine, describes that

> disagreement about the criteria for an abnormal ECG is widespread. Similarly, what constitutes a normal ECG is a source of debate amongst medical practitioners. As one cardiologist put it: 'There are as many definitions of what's normal as there are cardiologists'.[88, p62]

The point is clear: there can be no common definition of what 'normality' is since each cardiologist differs in their view. Given the view that such definitions are, in any case, constructed by the participants over time it is clear at once that a group's common definitions can and do change over time. Thus the ethnomethodological view emphasises that rules and definitions (such as they are) are not only situated in context but are also situated in time.

Rule-informed: Any activity, in this view, is fundamentally not seen as rule-governed but as *rule-informed*, so that *rules are seen as resources to be used in deciding what action to perform* since, as Goffman has noted,

> we deal not so much with a network of rules that must be followed as with rules that must be taken into consideration, whether as something to follow or carefully to circumvent. [74, p 42]

So, what is seen as rule or procedure following behaviour in the classic bureaucratic image of work is inverted - what looks like rule following in fact turns out to be the reconstruction of order so that the work can satisfactorily be seen as fitting the accepted pattern or procedure. An acute example of this is provided in Suchman's work on procedures in an accounts office where she finds that

> Standard procedure is constituted by the generation of orderly records. *This does not necessarily mean, however, that orderly records are the result, or outcome, of some prescribed sequence of steps.* Workers in the Accounting Office are concerned that (1) money due should be paid, and (2) that the record should make available both the warrant for payment and the orderly process by which it was made. In this case, once the legitimate history of the past due invoice is established, payment is made by *acting as though the record were complete* and then filling in the documentation where necessary. The practice of completing a record or pieces of it after the fact of actions

> taken is central to the work of record-keeping. *Standard procedures are formulated in the interest of what things should come to, and not necessarily how they should arrive there.* It is the assembly of orderly records out of the practical contingencies of the actual cases that produces evidence of action in accordance with routine procedure. [147, p326, emphasis added]

Here then is a view of rule-use, and of human social behaviour in general which is radically different from the mechanistic view of behaviour as enacting some externally defined set of rules. Further it casts real doubt on the idea that such rules could be determined in anything other than a partial manner and finally, it emphasises that appropriate behaviour is determined by persons in particular contexts with reference to features of the situation at that time. As will be described in the next section, this re-specification of the conception of rules and rule-use has had fundamental implications for the design of CSCW systems.

2.4 Implications for the Design of CSCW Systems

The view described in the previous section, which has come to be known to the HCI and CSCW communities as 'situated action' initially through Suchman's work on office procedures [146] and interfaces to 'intelligent machines' [147], acknowledges that human interaction and collaboration take place in the context of richly varying cultural and organisational norms of behaviour. Section 2.2 described the various ways in which the accepted conception of rules and rule-use had been incorporated into system design. By re-examining some of these systems in the light of the re-specification described in the previous section, it is possible to draw out a number of implications for design.

For example, it is not at all surprising that the GROVE editor seemed to be most successful in 'free-for-all' mode because it was only in this mode that participants were able to decide the roles and access privileges themselves and to flexibly re-arrange those roles and privileges as appropriate to the course of their work [132]. Now too, it can be seen that Beck and Bellotti's observations on the flexibility of collaborative writing are revealing many of the features that would be expected from an ethnomethodological perspective. Thus it is not at all surprising that co-authors are highly flexible and sensitive to contextual influences on their work activities, nor that systems to support them must therefore cope with this flexibility. Thus systems such as GROVE (in 'free-for-all mode') ShrEdit and MESSIE [125] have, perhaps unintentionally, shown the way in which the ethnomethodological stance can direct system design. That is, the system needs to provide the users with the means to be flexible whilst also providing the objects of work. As Robinson suggests, this requires systems to provide two levels of interaction - the level of 'doing the work', and the level of 'talking about the work' [120]. In this way, users can rapidly rearrange their roles and access to work objects through social protocols and it is exactly this sort of behaviour that is found in synchronous work experiments where a shared workspace is augmented by an audio channel. The former is the level at which the work is done, whilst the second, the audio channel, is the level at which the organisation of the work is done. If these levels can be incorporated into systems, and whilst the technical implementations will vary, it may be that system imposed roles, protocols (such as floor control) and access controls may be unnecessary. A number of recent research systems are either explicitly exploring this view or are can be seen as doing so (eg. [17, 153, 101, 78]) whilst Beck and Bellotti's paper provides a number of important design recommendations in this context [18].

In the context of rules and roles as access controllers for shared information or file systems, the ethnomethodological position raises some serious concern as to whether this approach is feasible for flexible, dynamic work groups. If the goal is to build information spaces that have some element of privacy and protection, it seems clear that this cannot be done by simply trying to implement a more complex 'privacy management' model - this would be analogous to trying to get out of a hole by digging ever deeper into it. With the increasing emphasis on short-term work groups that are brought together to complete specific tasks[5] the work-roles, and therefore the information access requirements of members of an organisation are likely to by highly dynamic. It may, for example, be perfectly fine for a worker to rummage through another's desk in search of a particular document whilst they are working on the same task, but not so a few minutes later when that brief passage of work has been concluded. Drawing this example into a shared information system, the ethnomethodological stance suggests that system designers may well have to totally re-consider their approach to the management of privacy and access privileges. It may be that an approach based on the explicit support of social and organisational protocols can provide such flexibility. The trade-offs between the flexibility of access provided by social controls and the protection afforded by system imposed control is a research area that is currently little explored and that would seem to merit attention. In particular it is interesting to note that much of the current access control in an organisation is based upon accountability, the fact that members can be held accountable for their actions - and hence be asked to explain them; upon organisational practices which are readily learnt; and upon effort, that is the effort it would require to behave inappropriately - breaking open a filing cabinet for example. It may be that shared information systems built around these concepts can usefully combine elements of system imposed control (through 'effort') and social and organisational control. The architectures described by Trevor and colleagues [152] and Smith and Rodden [138] can be seen as initial explorations of some of these ideas.

In the context of workflow and office automation, this re-specification is currently receiving considerable attention stemming, in part, from Suchman's early articulation of the ideas in her work on 'office procedures'. The key insight here is that treating work as programmable is not necessarily appropriate where the work itself is anything other than rigidly repetitive. The ethnomethodological stance suggests that it might be far better to focus workflow on the provision of adequate work objects and representations of possible work paths, but to enable the users to control which specific paths particular work objects follow. In addition it may be that enabling the system to continually re-present the activities of the users will allow them to re-engineer the process representations themselves so that the representation of work activity becomes (again) part of

[5]So-called 'virtual organisations'.

the activity of doing the work. Recent reviews of workflow research such as that of Abbot and Sarin have raised these issues [1], whilst systems such as ConversationBuilder [104], Freeflow [56] and those based on the Milano Conversation Model [46] are directly feeding these ideas into system design.

Finally, in the context of the design of user interfaces to Media Space systems, this re-specification of the notion of rules and rule-use suggests that it could be a serious mistake to embed rules of access (based on roles or status) into the system. In contrast, the need to cope with the flexible nature of communication situations suggests that it may be preferable to design systems which provide a range of possible actions, the system functionality, together with a set of information that enables users to decide for themselves what the appropriate behaviour would be in a given situation. Thus, rather than enforcing the rules that link door state to available functionality as was the case with the CAVECAT [109] and Doors [11] systems, it may be preferable to provide contextual awareness information (who the user is and what they're doing) which can be used to make a decision over which of the various communication options it is appropriate to use at that particular time. If this approach is to be followed, then it is clear that what the potential actions are, and what information people need in order to decide on appropriate courses of action are going to be critical resources for design. Thus, if the goal is to redesign an interface to a Media Space system in a such a way as to incorporate this re-specification of rules and rule use, it is clear that some way of framing this design approach, and of generating the necessary resources will be critical.

The next chapter provides just such a framework in the context of redesigning a user interface to a Media Space system in order to take account of this re-specification. As such, the chapter introduces a framework that can be seen as providing one of the enabling bridges described in Section 1.4.3 which will be needed in order to further the research programme of technomethodology.

Chapter 3

Options for Action and Cues for Behaviour: A Framework for Design

3.1 Introduction

The previous chapters have suggested that one aspect of learning from ethnomethodology has been the idea that foundational concepts can be re-examined, indeed respecified, and the implications of such a re-specification analysed in terms of its potential impact on design. In re-specifying the concept of rules and rule-use, it has been suggested that rather than embedding rules for appropriate behaviour into a system, an implementation must be able to support users in selecting and carrying out particular activities. This re-formulation suggests that instead of focusing solely on descriptions of work processes or practices, system designers need to be able to provide a rich set of information which *the users* can use in deciding what appropriate behaviour might be [132, 133]. So, if CSCW practitioners are to follow this reformulation through into the design process, it is imperative that a design framework is evolved that can be used in other situations. This chapter proposes such a framework, outlines it's basic concepts and structure, provides an analysis of its generality and describes the key requirements that it implies.

3.2 Conceptual Basis

In his book 'The Psychology of Everyday Things', Don Norman describes four classes of constraints that effect the outcome of possible actions: *physical*, based upon physical properties of the world; *semantic*, based on knowledge of the world or situation; *cultural*, based on cultural conventions; and *logical*, based on natural mappings [115]. Similarly, this chapter suggests that, in a literal sense, there are no constraints on behaviour, other than those imposed by the *physical* world in which we live. It is entirely possible for example, to burst through your boss' closed door without knocking first. It is possible, but as has been shown, it is socially acceptable in some situations, although not in others [10]. The everyday world, then, consists of 'cues for behaviour' that allow human actors to choose the most appropriate from a range of 'options for action', any one of which is physically, although not socially, possible. To continue the example, human actors decide whether or not to burst through their boss' office door based upon cues of context, urgency, role and previous experience.

When considering the design of systems to support social interaction, this view suggests that user interfaces to CSCW systems can be conceptualised as providing a set of possible actions - in other words the functionality of the system, together with a set of cues that can be used in deciding what to do. As Figure 3.1 shows, these cues may be either detected and interpreted by the system, using whatever model that the designer has implemented, or perceived and acted upon by the user. In the former case, the system is responsible for mapping the cues to the actions using the rules embedded within it. In the latter case the user perceives these cues and decides what to do on the basis of these and other cues that may be outside of the scope of the system itself.

The critical point here, and it is this characteristic that has been informed most directly by the re-specification of rule-use, is that the implementors of the system must consider very carefully how much of the mapping between the cues and the actions is left to the system, and how much is the province of the user. In Figure 3.2 for example, which characterises many of the systems

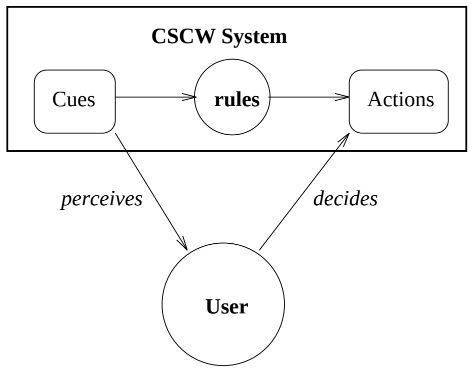

Figure 3.1: Cues, Rules and Actions: A Conceptualisation

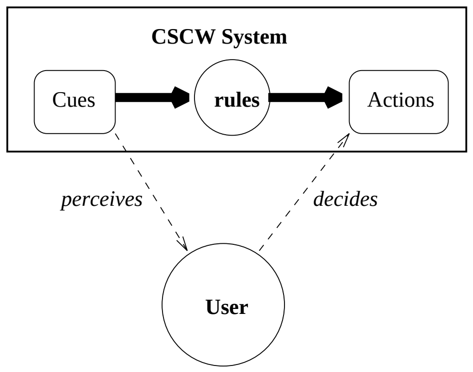

Figure 3.2: Cues, Rules and Actions: The System Decides

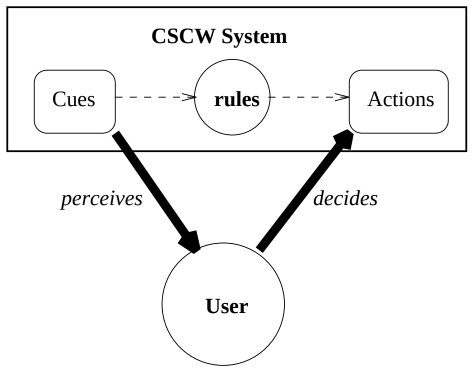

Figure 3.3: Cues, Rules and Actions: The User Decides

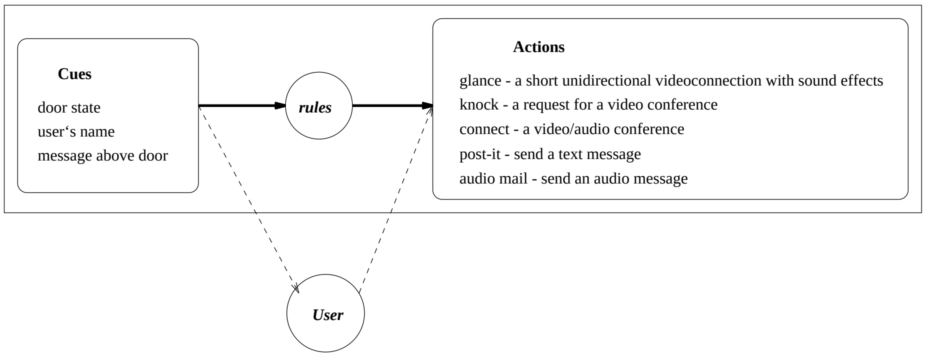
Cues
door state
user's name
message above door
rules
Actions
glance - a short unidirectional videoconnection with sound effects
knock - a request for a video conference
connect - a video/audio conference
post-it - send a text message
audio mail - send an audio message
User

Figure 3.4: Doors: Cues and Actions

described in Section 2.2, it is clear that much of the mapping between what has been termed 'cues' and the possible actions is determined by the system via its explicit model of the situation - the rules that are embedded within it. If designers consider how they might present users with cues and actions, but not enforce a mapping between the two, then a strategy emerges that may counter the problem of the complexity and dynamism of social relations by avoiding the embodiment of a set of social rules as *physical* constraints in the system. Such a system need embody no model of the users except those external cues that are used to guide behaviour, and the full range of potential actions. As is shown by Figure 3.3, the mapping from one to the other, that is determining which actions are acceptable and when, is no longer a concern of the system. Instead, the system remains relatively neutral with respect to action, it merely supplies the cues that the user needs, so that the problem of coping with the dynamism and complexity of *cultural* constraints remains firmly in the realm of the user, rather than the system.

3.3 Cues and Actions: Intimations of Generality

In order to clarify the framework, this section provides characterisations of a number of CSCW systems in terms of the options for action and cues for behaviour that they provide for the user. The examples include a Media Space system, a group editor, a group messaging system and an integrated CSCW workspace. Whilst not intended to be exhaustive, it is suggested that the ability of the framework to provide useful characterisations of a representative range of CSCW systems provides intimations of its generality. In addition the examples will show how representation of these systems using the concepts of cues and actions highlights the issues of embedded rules and flexibility.

3.3.1 Doors: An Interface to a Media Space

Doors was a user interface to a multimedia teleconferencing application, or Media Space [11, 10] which was developed as a front-end to the Cambridge Rank Xerox Research Centre's (formerly EuroPARC) audio visual infrastructure (cf. [72]). Doors was a client-server system that provided an interface to a centralised database of information about each user of the audio/video infrastructure. The

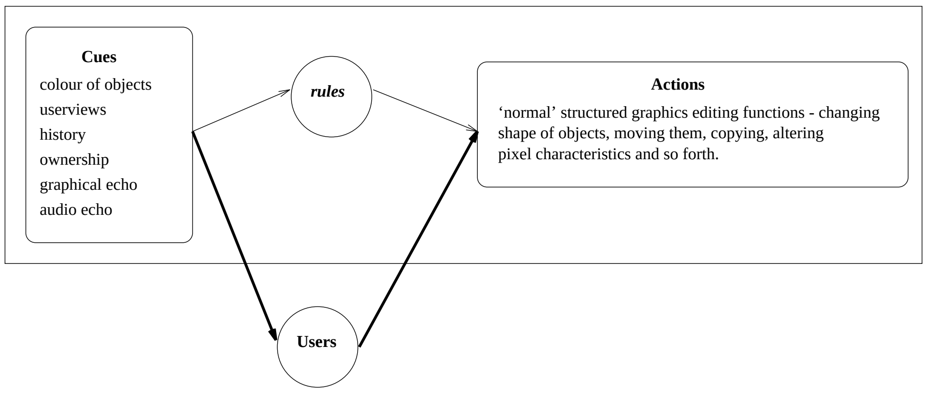
Cues
colour of objects
userviews
history
ownership
graphical echo
audio echo
rules
Actions
'normal' structured graphics editing functions - changing shape of objects, moving them, copying, altering pixel characteristics and so forth.
Users

Figure 3.5: GroupDesign: Cues and Actions

user interface was based on the concept of representing the availability states of users by an iconic office door. Thus the door could be set to 'closed', 'ajar' or 'open' and each state corresponded to a set of 'allowable' actions. As Figure 3.4 shows, a range of other cues were provided by the interface. For example, a participants list kept a record of all those who were currently running a Doors client, each door was explicitly associated with a particular user who can define the name that is displayed above each door. As was mentioned in Section 2.2, the Doors system used a model of what kind of actions should be available based on different door states in order to determine which of the potential actions should be possible. Thus when a door was set to ajar, all but the 'connect' action were enabled whilst when the door was shut both the 'connect' and 'glance' actions were disabled. Interestingly, Figure 3.4 shows that even though the Doors system actually provides additional cues other than just the door state, these cues cannot be used by the system (or the user) in modifying what actions are available and when.

3.3.2 GroupDesign: A Structured Drawing Tool

GroupDesign is a multiuser editor developed by Beaudouin-Lafon and Karsenty [17] that is similar in intention to GROVE [62] but which focuses on enabling users to manipulate structured graphics rather than text and outlines. GroupDesign consists of a number of pages each of which contains any number of editable structured objects. As Figure 3.5 summarises, users are able to edit these objects in a rich variety of ways. In order to provide users with information on who is doing (or did) what, GroupDesign implements cues for history, age and identification. Thus users can see who created which objects, how long the objects have been there and what changes have been made to them. In order to provide real-time cues for what is going on during synchronous editing, GroupDesign implements graphic (a user's actions are represented to others via background animation) and audio (a user's action on part of the page which is off-screen for another user is represented by sound) echo as well as enabling each user to see which parts of a page other users are currently viewing. In addition colour is used to denote which objects have been created and edited by which users - who, in a sense, is the 'owner' of each object. When a user starts to act on an object,

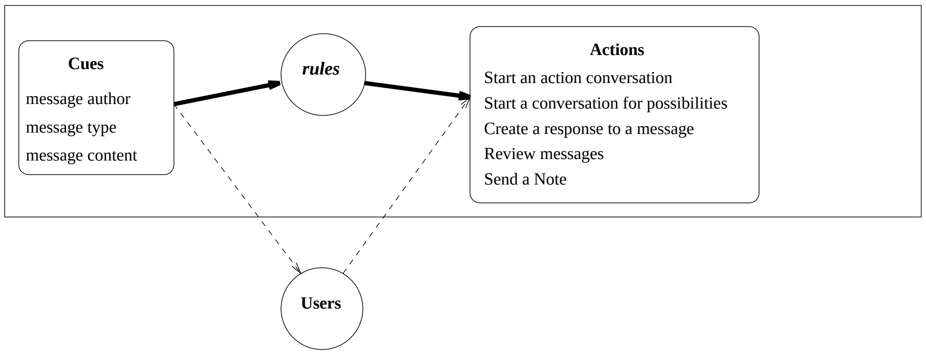
Cues
message author
message type
message content
rules
Actions
Start an action conversation
Start a conversation for possibilities
Create a response to a message
Review messages
Send a Note
Users

Figure 3.6: The Coordinator: Cues and Actions

it is represented in that user's 'colour' and an icon within the object shows what that action is. Taken together, these cues provide users of GroupDesign with a rich set of information that they can use in deciding what to do - whether or not to edit particular objects, with whom they should discuss particular edits, and who has access to which of the objects displayed - as Beaudouin-Lafon and Karsenty put it, these cues provide the users with answers to the questions "where are we, how did we get here and what can we do now?".

It is obvious from Figure 3.5 that the GroupDesign system leaves much of the mapping between cues and actions to the users - Beaudoin-Lafon and Karsenty explicitly state that they are interested in supporting social protocols rather than system imposed access controls and the representation in terms of cues and actions clearly illustrates this. As a result, GroupDesign is a good example of the kind of system for which the previous chapters has called although it is interesting to note that the system does impose *some* controls, for example it stops a user moving an object which is also currently being moved by someone else. GroupDesign is therefore a working example of the trade-offs between system and user control, a point which Beaudoin-Lafon and Karsenty note but do not expand.

3.3.3 The Coordinator: A Structured Messaging System

The Coordinator[1] is a structured messaging system that attempts to enhance workgroup productivity in organisations by providing users with sets of possible message types and by enforcing particular responses to particular types [65]. The Coordinator draws on the idea that action can be seen as constituted through language, hence a structured messaging system can provide 'action through language' by enabling, and enforcing, certain sorts of conversations. In the context of an organisational system, The Coordinator provides different message types that reflect the author's intention to try to improve organisational productivity. For example, The Coordinator provides users with the ability to start a 'conversation for action' or a 'conversation for possibilities'. On receiving such a message, a user must select from a small number of acceptable responses (including free-form) in order to reply. As Figure 3.6 shows, the system uses it's model of how a particular kind of conversation should proceed to provide users

[1]'The Coordinator' is a registered trademark of Action Technologies.

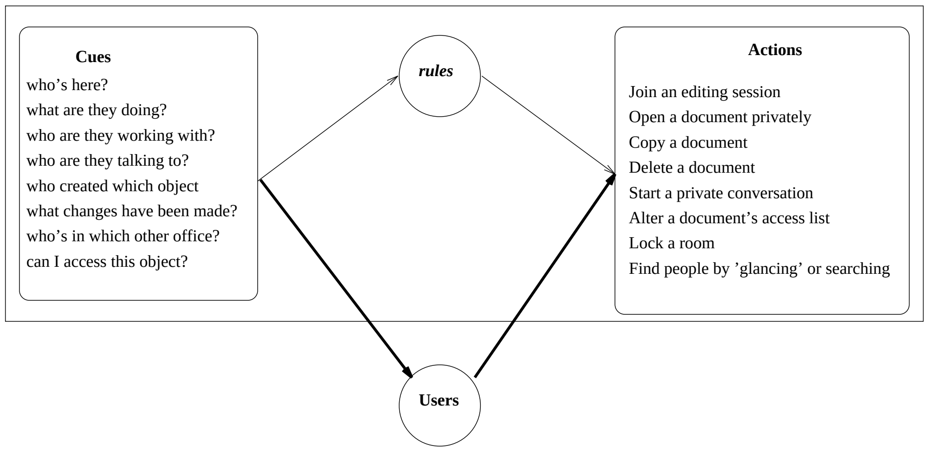
Cues
who's here?
what are they doing?
who are they working with?
who are they talking to?
who created which object
what changes have been made?
who's in which other office?
can I access this object?
rules
Actions
Join an editing session
Open a document privately
Copy a document
Delete a document
Start a private conversation
Alter a document's access list
Lock a room
Find people by 'glancing' or searching
Users

Figure 3.7: DIVA: Cues and Actions

with alternatives. Thus when a user chooses to 'Answer' a message, the system determines what

actions could sensibly be taken next. [65, p162]

The problem is that the designer, not the user, has decided what that 'sensible' next action should be. Users are actively discouraged from producing responses outside of these limited alternatives - it is considered unhelpful to the maintenance of the structured conversation if they step outside this embedded model of process. Unfortunately for The Coordinator, the result is that the 'next available action' is often not sensible to the user at all...

As this characterisation makes clear, the user's experience of such a system is likely to be very different from those which do not enforce such procedures and is as open to criticisms of inflexibility as were the workflow systems described in earlier sections.

3.3.4 DIVA: A Networked Work 'Place'

DIVA is an example of a complex distributed CSCW system which consists of integrated groupware tools arranged using the metaphors of 'rooms' and 'places' [139]. Users navigate between rooms, users in the same room can automatically see and hear each other through audio/video connections and they can also create a private conversation with another individual in the room. Each 'room' may contain any number of objects such as written documents, spreadsheets or drawings which may be manipulated by multiple users using integrated groupware tools. Leaving aside the mechanics of the various tools (shared editing and drawing tools such as those already discussed), DIVA provides a rich array of cues for users in deciding what to do (see Figure 3.7). For example, DIVA uses similar representations to CAVECAT in indicating the availability of a user - the door to a user's room may be locked, shuttered or open - and enforces particular access policies based on these states. It is unclear whether these policies can be overridden if required, nor if such policies can be altered if desired. Access to particular documents is managed using a simple access list method which is configurable by anyone on that list. DIVA uses this list to provide users with cues as to which documents they have access to (and also who to contact in order to alter this...), which have been changed, when and by whom.

Apart from this simple access control mechanism however, DIVA provides little in the way of system imposed policies via rules, leaving most of the control to social protocols. Cues such as who is in the room, who they're talking to and about which document combine to provide users with information they can use in deciding whether or not to join editing sessions or conversations, whether to work independently on a shared document or to work closely.

As DIVA's authors note, it is clear that many of the cues users expect to find in the real world can be directly transferred to the system by using interface objects that mimic those of a real office. By using rooms, desks and briefcases to organise access control, cues and actions, DIVA leverages many familiar concepts and implicit rules. It is worth noting then, that designing systems that explicitly present such cues in order to allow users to select appropriate actions might well turn to studies of real world activities to furnish a set of cues that can be a starting point for an implementation.

3.4 Implications for the Design Process

The previous section has briefly illustrated the use of the framework of cues and actions to characterise a number of CSCW systems. In each case it has been shown that recasting the systems in this way highlights the extent to which the system imposes control over action through an embedded model or policy. In some cases this turned out to be very little whilst in others it appeared to be central to the design. Further, those systems which provided little in the way of system mappings between cues and actions appeared to be those which provide for more flexible use.

It seems that characterising systems in this way opens up for inspection design decisions that may otherwise pass unnoticed. For example, designers who characterise their systems in this way can reflect on the degree to which the system under construction implements models and assumptions of social rules. In forcing designers to consider this, the trade off between system and user control can be identified and discussed during design. If the goal is to build systems that are intended to enforce certain social or work procedures, then it is possible to identify this within the framework and develop appropriate models as needed. However, if this is not the intention, then providing this characterisation forces designers to think about other ways of providing users with the options for actions that correspond to the system's intended functionality. Thus the key recommendation made by this framework is that, as design progresses, practitioners must repeatedly ask themselves whether or not the system behaviour they are currently encoding involves the implementation of social rules. If so, then they must be aware of the dangers of doing so. If they wish to avoid these dangers then it is recommended that they should focus on providing users with cues for behaviour and options for action, but not implementing a mapping between the two.

It is at this point that the framework serves a second purpose because it suggests that designing systems to take account of the flexible nature of rules and the interpretive nature of their use can usefully focus on providing users with cues and actions. As a procedure of design then, the framework calls in the first instance for the articulation of the actions that the designers intend the users be able to do, and secondly the elicitation of the cues that the users may be expected to use in deciding between appropriate courses of action. As the discussion of the DIVA system demonstrated, studying the real world of work and of the potential mappings between real world objects, activities and cues may provide a rich resource for designing in this way.

3.5 Summary

This chapter introduced a design framework based on the idea of characterising CSCW systems in terms of the actions they make available to their users, and the cues they provide for users to decide which of the actions to choose. Further, in each case the framework characterises the degree to which the system maps cues to actions, and the degree to which the user is left to decide what actions are appropriate. What is more, this characterisation can be used to describe a number of CSCW systems and, in each case, usefully highlight important design issues. Finally, the framework recommends that the design of systems that do not want to impose system constraints on action might usefully focus on providing cues and actions in the user interface itself. It therefore recommends that the framework can be used as a guide for doing design. As a consequence of this recommendation, it is clear that the design of a system in this way will require the elicitation of actions and cues that make sense to the users so that they may leverage their own knowledge of the situation in order to decide what is appropriate. The next chapter provides an example of exactly how cues and actions may be elicited in order to form a foundation for design. The general motivation in this case was to redesign and re-implement the Doors system described earlier using the framework introduced by this chapter.

Part III

A Case Study in Design

Chapter 4

Actions and Cues as Design Resources

4.1 Introduction

The previous chapters of this dissertation have developed an argument that recommends re-examining taken-for-granted concepts in HCI and CSCW in order to explore the implications of their re-specification. In particular, this dissertation has focused on apparently pervasive assumptions about rules and the nature of their use by human actors. After a consideration of the implications that this re-formulation has for the design of CSCW systems, the previous chapter recommends a framework and an approach to design that presents users with options for action and cues for behaviour but which does not necessarily enforce the mappings between the two.

This and subsequent chapters describe a case study of the use of this framework in the redesign and implementation of the Doors system which was briefly described in Section 3.3.1. The goals of this redesign are to:

Explore the Framework: By using the framework in a system development situation it will be possible to reflect on the utility of the framework during design.

Provide methods: It might be expected that concentrating on cues and actions would require particular design methods. If this dissertation is to recommend that other systems of this kind be built from this framework, the development, use and reflection on such methods is of great importance.

Build a Working System: By building a system using the framework as guidance, it will be possible to reflect on the utility of the framework of cues and actions in producing a *functional* CSCW system.

In addition, the prototype implemented as part of the case study will act as a vehicle to investigate:

General support for awareness: By extending the scope of awareness information made available and by developing a scalable group awareness protocol to pass this information between clients, it will be possible to develop technological support for awareness in the general Internet context.

Packet audio and video: Doors made use of RXRC's analogue audio and video infrastructure, as have many of the other experimental Media Space systems. In order to widen the applicability and increase the flexibility of the system, packet audio and video over digital networks will be supported.

The utility of IETF draft protocols The prototype will provide the opportunity to explore the use of evolving IETF protocols designed to support a multimedia conferencing architecture over the Internet. By using these IP based protocols, the possibilities for global awareness and user location services through public-access packet switched networks can be explored.

Incorporate other CSCW tools: how the incorporation of shared text editors and whiteboard tools can improve the support for work.

It should be clear then that this case study operates at three levels. Firstly it serves as an example of design that is based upon providing users with ‘cues for behaviour' and 'options for action'. Secondly, it is an example of how a respecification of widely held foundational assumptions can be incorporated into the design of a functional CSCW system. Finally, it is a redesign and re-implementation of the Doors system as a vehicle for exploring the technical issues of scalable support for group awareness and group work in the Internet context. This chapter describes the general background to the prototype and reports a study designed to elicit 'cues' and 'actions' as a resource for interface design. Chapter 5 describes the resultant interface design and implementation whilst Chapter 6 describes the underlying architecture developed to support group awareness in general and this prototype in particular. The remaining chapters in this part of the dissertation provide an analysis of the implementation and use of the prototype in the light of the aims outlined above.

4.2 TelePort: Redesigning Doors

TelePort is a prototype system designed to mediate communication and interaction between users in a distributed broadband office environment. It seems clear that the extension from POTS[1] to broadband telecommunication, and the consequent enrichment of functionality and increased importance of issues of privacy and control (cf. [50, 20]) requires a considerable re-think of the user interface to communications devices. The *TelePort* system addresses this problem by displaying the availability state of the owner of an audio-visual node using a graphical representation of different states of an office door, and by providing socially grounded mechanisms for communication that are consistent with this representation.

Thus, the *TelePort* prototype focuses on the problem of controlling point to point multimedia conferencing calls over local and wide area packet-switched digital networks. Specifically, the prototype enables users to request a number of different user-oriented telecommunications services using real-time packet switched audio and video conferencing tools. These services include short, video only glances as well as full two-way audio, video and data conferencing. The prototype provides the user interface to this system and implements mechanisms for geographically dispersed users to be 'aware' of one another through the provision of regularly updated awareness information; and thence to communicate and interact. The prototype may be considered a direct derivative of the Doors system [11] and as being conceptually related to CAVECAT [109] and Montage [149].

The use of the office door as a representation of availability clearly provides users with cues from which they can make certain sorts of predictions about the availability state of a person based upon the state of their office door. For example it may be acceptable to knock on a closed door, but not to enter without invitation; whereas in the case of a fully open door, a knock-and-enter action may be socially acceptable. If the interface is to provide the cues associated with an office door as a representation of availability in a telecommunications system, it is clearly imperative that an attempt is made to find out what those cues are for the user group or culture for whom the system is intended.

[1]Plain Old Telephone System

This chapter[2] describes one way in which designers might elicit options for action and cues for behaviour from potential users. As such it describes one of the practical bridges that will be needed to develop technomethodology into a research programme. The chapter reports an exercise in deriving actions from a potential user group which could map onto given system functionality, and cues that might make sense to the potential users. In particular, it describes the use of a method derived from Cognitive Anthropology, namely frame analysis, in the elicitation of cues that office workers use in deciding when to communicate with colleagues during the course of their normal work. These cues, and the actions that are associated with them, are central to the subsequent design of the system, *TelePort* .

4.3 Getting at 'cues' and 'actions'

Previous chapters have developed the idea that human behaviour can be thought of as the selection of appropriate actions from a range of options in a particular situation. Further it has been suggested that interface design can proceed by implementing ranges of actions and providing users with cues which help them to decide what to do. To be successful, it has been suggested that transferring cues and actions from user's everyday world to the user interface can provide a 'bootstrapping' effect because users can immediately apply the social mores with which they are familiar. Such a transfer would therefore provide users with a ready made and understandable social context within which they could act - many elements of the culture of usage would effectively be known in advance and thus immediately applicable.

Clearly then, a designer who intends to build a user interface in this way needs to know what cues and actions are relevant to the users in question - it would be pointless, if following this strategy, to implement a set of cues and actions that do not make sense to the users since this 'bootstrapping' effect would be lost. It seems logical therefore, that design in this way should make a commitment to eliciting and utilising the *user's* conceptions of cues and actions. If they do not, any system will inevitably incorporate the *designer's* intuitive assumptions about what actions and cues are significant, rather than the user's. Here then is a re-iteration of the principle of user-centred design [116] except that in this context it is a recommendation to focus on 'cues' and 'actions' as they appear in the user's experiences of everyday work. What is needed therefore, is an elicitation method that can encourage users to describe what actions they might find appropriate in a communication situation, and what cues they might use in deciding amongst these actions. This section introduces one such method, frame elicitation, which is derived from fieldwork methods in cognitive anthropology, and demonstrates its use in eliciting 'cues' and 'actions' from a potential user group.

4.3.1 Cognitive Anthropology

In essence, the focus of cognitive anthropology has been to map out what an individual needs to know in order to generate culturally acceptable acts in a

[2]This chapter is an extensively revised version of [10]. A copy of this paper is included in Appendix D for reference.

given social context [75]. It is claimed that the techniques used to do this can generate, as far as is practically possible, a cultural description that is phrased in the conceptual terms of that culture and which, crucially, would make sense to a 'native' informant if re-presented to them [142]. In the terms of the current discussion, cognitive anthropology attempts to describe the 'cues', 'actions', and the rules that map the one to the other using the conceptual categories and terms of the informants themselves. As was discussed in the previous section, this is almost exactly is required by the design strategy proposed in this dissertation. Therefore it is likely that a number of research methods found in cognitive anthropology will be of use in generating descriptions of cues and actions from the user's point of view.

4.3.2 Frame Elicitation

One such data collection method, frame elicitation, seems particularly relevant. Frame elicitation attempts to elicit conceptual schema or scripts [2, 4] from everyday talk or from 'elicitation sessions'. A schema or script can be considered to be a high level description of a group of related inferences which holds generally true in a number of decision making situations. In Agar and Hobbs's study of events in the lives of inner city drug addicts for example, an arrest schema is described which is derived from analysis of a number of interviews describing particular arrests, and this schema can then be applied to other similar situations [4].

Such schema are elicited by means of specifically designed questionnaires or frames. Frames can be thought of as

> simply a statement with a hole in it that can be filled in a variety of ways. [2, p99]

such as:

> If I wanted ______________ I would ask my secretary to arrange it.

A selection of such frames can be presented to informants who are asked to supply appropriate words or phrases to complete the statement. By varying the wording of the frames, an investigator can assess the effects of such variations on the phrases used to complete the frame. In the example provided for instance, changing the words 'my secretary' to 'the Company Director' is likely to produce different responses if the informants were in an organisational environment. The crucial point here is that the frames enable the informants to construct their own context to the enquiry by using phrases that make sense to them, rather than by selecting from amongst a range proffered by the investigator. More importantly, from the point of view of a design process that focuses on and seeks to elicit 'cues' and 'actions', frames provide a relatively simple way of generating the range of options for action and cues for behaviour in a particular context or domain. With respect to the case study then, a frame such as

> Ben's door was ____________________ so I ____________________ .

can be used to generate a list of actions that people would expect to be able to take, together with cues related to the office door which they would use in deciding what to do.

4.3.3 Method

In order to elicit such information, a study was carried out that utilised the schema or script elicitation techniques described above. 17 business personnel (15 male, 2 female), who were attending a week long residential course contributing towards a part-time Master of Business Administration (MBA) qualification at the Loughborough University of Technology Business School, were asked to act as informants. In order to provide some background information and to set the results of the frame completion in context, the informants were asked to specify their occupation and provide a short job description. They were then presented with a frame completion exercise designed to elicit their probable responses to different states of a person's office door. The frames to be completed took the form of two partial statements where each variable was to be completed by the informants.

The first statement was:

> As I walked towards _*person*_'s office door, I saw that it was _*state*_, so I _*action*_.

This frame was used to generate as many different combinations of *person/state/action* as possible. Note that the wording of the partial statement in this case was such that informants were not restricted to any particular person, state or action. Following Agar ([2, p142]), the resulting phrases were grouped firstly into similar door states, and secondly into similar actions within each state.

The second statement was derived from the results of the first exercise:

> As I walked towards _*person*_'s office door, I saw that it was **open/ajar/closed**, so I _*action*_.

This frame was intended to examine how actions varied with the door owner's status given a particular door state. In this case informants were presented with 3 different frames, one with **open** as the state, one with **ajar** as the state and one with **closed** as the state. In this case the informants filled in the person and action slots as appropriate.

In each case, the informants were asked to provide as many completed frames as they could so that a range of *person/state/actions* triads could be examined. The frame-completion exercise generally lasted for around 20 minutes.

Further grouping was then carried out on the words used to describe the person whose office was being approached, so that the effect of status on acceptable actions could be analysed. This grouping was carried out by a researcher not involved in this investigation whose cultural and working background was similar to that of the earlier informants. It is acknowledged that these grouping tasks should, ideally, have been carried out by members of the original group of informants. Unfortunately this was not possible due to the timing and limited scope of the investigation with respect to the informants course attendance.

4.3.4 Results

The full results of this investigation, which produced a large number (127) of completed frames, are provided in Appendix A and are summarised below.

As would be expected from the participants of an MBA (Master of Business Administration) course, the informants consistently referred to their jobs as

Door State	**Inferred Implications**	**Acceptable Options**
Closed	busy - not disturbable not in office	walk straight in (W) knock and wait for a reply (Kw) leave a message (M) check with secretary (S) go away and try again later (La)
Partially Open	busy but can be interrupted	walk straight in (W) knock and wait for invitation (Kw) take a quick glance in (G) go away and try again later (La)
Fully Open	available for communication	Walk straight in (W) Knock and wait for invitation (Kw) knock and walk in (Ke) take a quick glance in (G)

Table 4.1: Glosses of informants 'action' phrases for particular states, together with implications for communication.

being junior or lower-management. As such they represent a horizontal slice through a number of organisations each of which may differ markedly in the way in which communication or interaction is socially mediated and regulated. Given that the informants could be seen to represent a diversity of different business cultures, it is interesting to note that the range of responses is relatively narrow. The phrases used by the informants to fill the *state* frame were: 'open', 'closed', 'shut', 'ajar', 'partially open', 'half open', 'closed with do not disturb sign' and 'hanging off its hinges'. This last was paired with the *action* phrase 'went to tell a policeman'[3]. By grouping these phrases it was possible to reduce the states of the office door to a set of three - 'closed', 'ajar' and 'open', although there was a subtle distinction between 'closed' and 'closed with do not disturb sign' because in the latter case the cue for non-availability is that much more forceful and the question of whether the person is present is, at least partially, resolved.

From the phrases used to complete the *action* section of frames, it is clear that there are different options that are acceptable in certain situations. By grouping these options, the responses indicate that there are seven different actions that have been identified by this study (see Table 4.1). The actions listed in the third column are paraphrases or 'glosses' [67] covering the meanings of the actual phrases used by the informants, the set of symbols will be used in subsequent figures to aid legibility.

Figure 4.1 shows the percentage frequency of each of the *actions* for a particular *state*. This provides an indication of what people are likely to do (or want to do) given a particular door state.

Even at this gross level of analysis, it is clear that there are different options that are acceptable in certain situations. If a door is open for example, the acceptable actions tend to be 'Walk in' or 'Knock and enter', whilst in the case of the door being ajar the majority of responses were 'Knocking and entering',

[3]The importance of humour and irony when used by an informant to reflect upon cultural norms is one that is often discussed in the literature - e.g. [67]. It is obvious perhaps that the recognition of these 'manners of speaking' is easier if the informant and investigator share a common language and culture, as was the case with the study reported here.

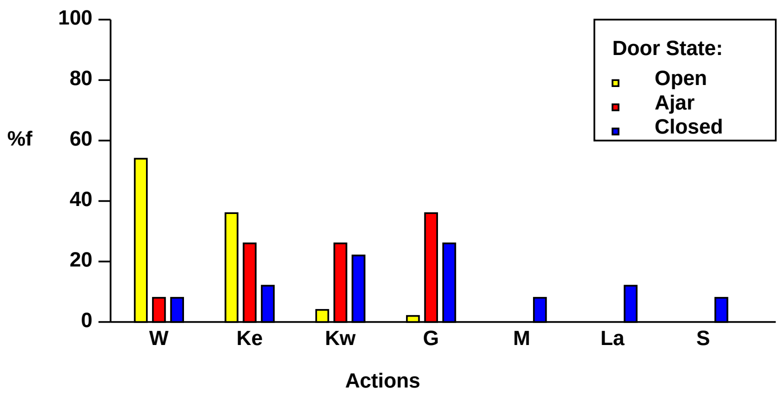

Figure 4.1: Percentage frequency of *actions* for each *state*

Gloss	**Elicited phrases**
friend	friend, colleague, manager I knew well, neighbour
boss	boss, senior manager, immediate manager,superior
boss' boss	boss' boss, superior

Table 4.2: Glosses of *person* phrases

'Knocking and waiting' or 'Check status'. As Table 4.1 shows different inferences were made about what the door's owner would be doing depending on this state. Further, different ranges of actions were suggested as being appropriate. This is important because it suggests that by providing the state of the office door as a cue, the system might indeed support the social self-regulation of appropriate behaviour. In other words the users can quite easily map the cues to the actions as they see fit.

The frame analysis also addressed the issue of social status in order to provide a more detailed analysis of one of the factors that might determine which actions are more acceptable in certain situations. Figures 4.2 to 4.4 show the percentage frequency of phrases used to fill the frames for each door state, subdivided into the different categories of 'person' to whom the door belongs.

The categories of person that resulted from the grouping exercise are shown in Table 4.2.

Figure 4.2, which shows the responses to the open door frame, demonstrates the effect of social status quite clearly; people will 'walk straight in' to the office of a colleague or their immediate boss if the door is open, but they will not do this to their director or their boss boss. Similarly they are much more likely to 'knock and wait', or to 'check their availability', if the person in the office is of considerably higher status.

Figure 4.3 shows the percentage frequency of responses when the door state was ajar. As in the previous case, there are differences in behaviour depending on the relative status of the people involved. Here, far fewer would 'walk straight in' and only then if the person whose office they were entering was a colleague or friend. Rather, people would prefer to 'check the status' of the person (usually in an unobtrusive manner), although it is noticeable that this option was not

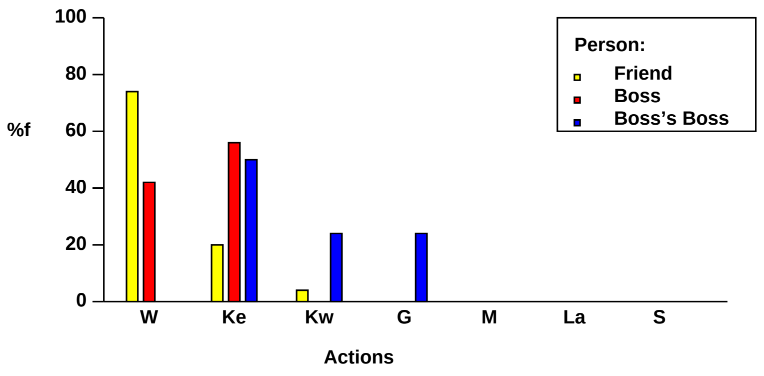

Figure 4.2: Percentage frequency of phrases used to fill *action* slot for **open** *state*, grouped by status of door 'owner'

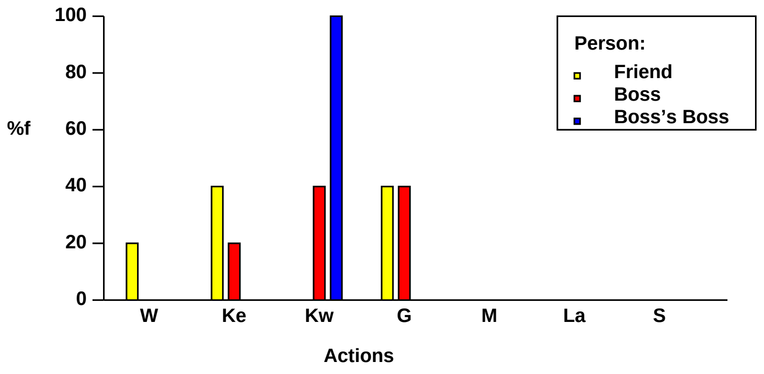

Figure 4.3: Percentage frequency of phrases used to fill *action* slot for **ajar** *state*, grouped by status of door 'owner'

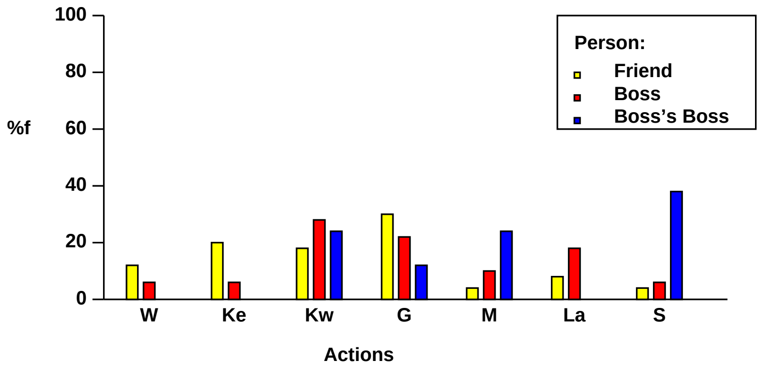

Figure 4.4: Percentage frequency of phrases used to fill *action* frame for **closed** *state*, grouped by status of door 'owner'

suggested in the case of the person being of considerably higher status, where 'knocking and waiting' is the sole response.

Figure 4.4 shows the percentage frequency of responses when the door state is closed and where the range of behaviours is greatest. In this case the options of 'walking in' or 'knocking and walking in' are both much less frequent. Instead there is much more emphasis on 'knocking and waiting', on 'checking status' and on 'leaving a note'. In the case of the person being of much higher status, the importance of a surrogate in the form of a secretary is noticeable.

4.3.5 Analysis: Generating Cues and Actions

It is clear that the frame elicitation exercise above can provide information on both the cues that are used, and the actions that are selected from the available options. In the first instance, it has suggested that three significantly different door states **open**, **ajar**, and **closed** may be enough to cover most situations. In the second, it has generated a set of seven different actions that are described in Table 4.1. Finally, the analysis suggests that informant's actions are influenced by the state of the office door, and by their status or role with respect to the door's owner. It has therefore shown that these are two of the cues that might be required.

These findings suggest that the *TelePort* system could use three different iconic representations of an office door to represent users of an office-based broadband telecommunications system. Further, the set of actions can form the basis for communication functionality that makes sense to the user and which is grounded in the user's everyday experience of communication at work. At the simplest level, the actions listed in Table 4.1 can be translated into a set of menu options that are available to users. Thus 'knock and enter' can create a bi-directional audio and video connection whilst playing a suitable sound effect to warn the recipient that the connection is being made; 'knock and wait' can invoke the sound effect alone together with a dialogue box requesting a connection that can then be acknowledged or ignored by the recipient; 'leave a message' invokes an email or voice mail tool; whilst 'check status' can invoke a short unidirectional video-only connection to the recipient's office (cf. glance service in [49]).

It can be seen that the perceived social status (i.e. role relative to the informant) of the person has an effect on the options that are deemed to be acceptable. Few people for example, are prepared to walk straight in to the office of their Director if the door is open, but will do so to a colleague or to their immediate superior. Similarly, in the case of a closed door, people will attempt to attract their superior's attention by knocking and waiting for a reply; under no circumstances would they initiate communication without acknowledgement from the other that such communication would be acceptable.

This finding is important because it implies that if the final system provides methods of initiating communication based on the phrases elicited (e.g. knock and wait, knock and enter etc.) and uses the office door to represent availability, it is possible that the system will not need to explicitly implement particular access rules. This is therefore suggestive evidence that the approach outlined in previous chapters may be successful. In the case of the *TelePort* system the social rules that are apparent from the preceding results and discussion, and which are explicitly instantiated in systems that do embed social models, are implicitly invoked by the use of the office door as a representation.

In this situation, it may be unnecessary to explicitly define access privileges because users will be able to select appropriate behaviour using the same social rules that govern the interactions involving real world office doors. Further, the social mechanisms that prevent the breaking of those rules in their everyday world may well act to prevent transgression in a computer-supported audio-visual environment. If this is the case, then the system will have succeeded in lifting elements of control from the technical to the social level (cf. [50]) because the social rules are no longer embedded within the system, but rather the user is supported in making appropriate choices about what to do in a given situation. This hypothesis can only be confirmed by examining the patterns of behaviour over an extended period of system usage and, as such, is an area for future work.

4.4 Limitations of the Method

Whilst this chapter has demonstrated the utility of one specific data collection method in the elicitation of 'cues' and 'actions', the method is certainly no panacea. Proponents of the method from within cognitive anthropology itself have often noted that such language oriented methods can only access information that a person *can actually articulate* [67]. Thus whilst it may be a useful method of getting at what people think they know, and can talk about, it may not be a sufficient method for eliciting what people will actually do in given situations - rather it captures *what they say they will do*, which may not be the same thing at all. Therefore, as has been discussed elsewhere [10], 'formal' elicitation methods such as frame analysis should not completely supplant observational field methods which, it is claimed, are more likely to uncover the subtle details of interaction.

In addition, it is openly acknowledged that the use of glosses and grouping of phrases can effectively 'drown out' subtle details and differences between responses [3]. The trade-off between the detail provided by observational methods and the kind of broad 'design-ready' results of the frame elicitation method described, is one that is currently attracting much attention elsewhere in CSCW and HCI (cf. [121, 12, 134, 6].

It is an open question, and therefore a possible area for further work, whether or not an observational study of people's behaviour with respect to their office doors and their availability would have added significantly to the design resources generated by the frame analysis reported in this chapter. One obvious limitation was that the frame analysis clearly focused on the state of the door, and the identity of the door's owner as useful cues. This precluded the elicitation of other potential cues which may be just as significant and which may have been uncovered in previous experiments or by other observational studies. As the next chapter will describe, a number of other resources have been used to develop the user interface for the *TelePort* prototype which draw, in turn, on both observational studies and on experimental experiences with this kind of system. As many authors have noted, users often make use of information that is unintentionally provided by the system and use the system in un-anticipated ways [120]. Thus, another open research issue and one which can also only be addressed through a long term user study, is whether *TelePort* users make use of cues other than those intentionally provided by the system, and through their use, can enhance the awareness information it provides. Such information can then be fed into any further design cycles the system may undergo.

4.5 Summary

This chapter has introduced the case study of design that constitutes the core practical work reported by this dissertation. The case study involved the re-design and re-implementation of the Doors Media Space system to take account of the conception of rules and rule use developed in Chapter 2 and the use of the design framework developed in Chapter 3.

This chapter then reported a study that used methods derived from Cognitive Anthropology to elicit the cues that workers use in deciding how to communicate with colleagues in an office-based environment. Further, it has used these methods to determine what actions users might expect to be supported by a system that provides management of multimedia conferencing calls. In generating such resources for design, these methods have demonstrated their value to a design process that seeks to concentrate on options for action and cues for behaviour. It has been possible to derive resources that suggest ways in which the interface can enable users to decide for themselves what actions are appropriate in a given context. In particular it has demonstrated how grounding the functionality in common everyday activities, such as entering an office, can not only provide a rich resource for design but can also provide users with ready-made and familiar cues. If systems are to support the kind of social control that the idea of 'options' and 'actions' recommends, then such familiarity is likely to be an important factor in the initial usability of the system.

The next chapters describe the user interface and system architecture of the *TelePort* prototype in some detail, showing how the implementation draws on the resources described in this chapter, and how the design focus of supporting 'cues' and 'actions' impacts on both user interface and architectural issues in a variety of ways.

Chapter 5

TelePort: User Interface

5.1 Introduction

This chapter describes the specification and implementation of the *TelePort* user interface. It therefore focuses directly on how the cues and actions generated in the previous chapter, and the general design framework developed in Chapter 3 impact upon the user interface and the design decisions that must be taken during its implementation. As was mentioned in the previous chapter, the first section draws in other resources that have identified potential cues and likely actions from other experimental systems or from observational studies. These are used to enrich the resources generated by the frame elicitation. The chapter then demonstrates how these resources can be used to generate user interface designs and how these specifications can then be implemented[1]. At each stage, the impact of the focus on 'cues' and 'actions' is discussed as it affects the design process.

5.2 Awareness cues and communication actions: Other sources

Thus far, this dissertation has described the elicitation of the actions listed in Table 4.1, and the cues based on the state of an office door, together with the name and status of the door's owner. These cues appear to provide relevant information to workers on the availability of another for communication and also, in the case where the other is not known, the person's relative status through name, title and position since all of these affect appropriate behaviour.

However, as was discussed in Section 4.4, the frame elicitation has focused solely on the cues of door state and identity. Other recently reported work aimed at supporting intra-group informal awareness has suggested a number of other possible cues, actions and/or useful information which a system such as *TelePort* may need to support [108, 149, 78, 60]:

- **Time and place:** In a distributed group information on the local time and the current geographical location of a user (through an active badge system for example) can be important.
- **Fine grained activity:** Activity cues such as time since last keyboard action, current login status, whether or not currently in a video conference, making a telephone call and if so with whom.
- **Contact Information:** Email address, World Wide Web Home Page, Telephone number.
- **Software in use:** To enable fluid transitions from single-user to multi-user work, cues such as what document or object is currently being used or edited may be important.
- **Coarse grained activity:** Enabling users to transmit a short text message indicating current activity.

[1]It should be noted that attention was not given to the design of the user interface in terms of metaphors used, GUI layout and other 'usability' issues - the use of the 'door' is well-known in awareness systems. The system reported here is a re-implementation of a previous system, 'Doors' [11], in the context of the arguments being made by this dissertation.

- **Capability:** In a heterogeneous environment it is unlikely that all users will have access to all the media tools through, for example, lack of appropriate hardware. Enabling users to be aware of the constraints on potential callees before initiating a communications action avoids the failure of calls for these reasons.

In addition, the media space work described in previous chapters has highlighted the need for other communications functionality:

- **Email:** If, for some reason, real time communication is not appropriate or cannot be achieved, other methods should be provided. Electronic mail is one such method.

- **Shared work tools:** Much communication is about some topic or artefact rather than for its own sake. In order to increase the utility of a call manager such as *TelePort* the integration of shared work tools such as whiteboards and text editors is essential.

Clearly then, the development of a usefully functional system needs to incorporate many of these features. This impacts in two ways - firstly at the user interface and secondly at the architectural levels. The next section shows how the user interface elements may be combined in such a way as to remain faithful to the principle of enabling users to exercise social control by mapping the cues to the actions themselves. The architectural issues this raises are discussed in detail in Chapter 6.

5.3 *TelePort* user interface: Specifications

Thus, in re-implementing Doors to provide an integration of these cues and actions, *TelePort* draws its design from both the literature and also from a study of potential users' likely communication behaviour. This section outlines the user interface and the describes the communications functionality (actions) provided at a relatively high level.

TelePort is designed to provide the user with a range of telecommunications actions:

- **Knock:** Make a request for a video conference to a particular user.

- **Knock and Enter:** Initiate a video conference without a request.

- **Glance:** Make a short one way video only connection to a particular user.

- **Workspace:** Initiate a shared whiteboard session with a particular user

- **Email:** Email the user.

- **View WWW Home Page:** View the user's world wide web home page.

In order to enable users to locate others, and then to initiate these actions, *TelePort* must provide awareness information about each user. Once a user has started *TelePort* the client needs to receive information from other clients in the same 'awareness group'. This information can then be used used to build a participants list of who is also in the group which would provide users with a general idea of who is around. The availability state of each user must also be represented and the other cues derived from the information received from that user's client must be displayed in the interface so the users can decide which of the actions is appropriate at a given time. The cues provided by *TelePort* are:

- User's name.
- Local time at user's location.
- User's iconic door state - set by each user.
- Current status of user in *TelePort* application - in conference, glancing etc. Includes names of others involved so that users can see who is talking to whom, and the nature of that communication. Note that this status information is not linked to the state of the door - a person may be involved in a video conference but may still have their door 'open'.
- A short textual note that users can edit.
- Audio capability: whether user can send/receive packet audio.
- Video capability: whether user can send/receive packet video.

The user interface for the *TelePort* application can integrate the functionality and cues by providing three main user interface components as shown in Figure 5.1:

- **Main Window:** Should enable user to set and configure information being sent such as their name, the state of their door, the textual note and whether or not they can send or receive the various media.
- **Participants List:** Frequently updated list of who else has joined the 'awareness group'.
- **User Door:** Individual user doors each representing one of the members of the group. This component can also display the other awareness information or cues and can provide the actions via standard interface widgets such as menus.

If the design is to take account of the view of rules and rule use developed in previous chapters, it is vital that the user interface, and indeed the system in general, should avoid implementing a set of social rules that map certain configurations of cues onto particular 'permissible' actions. Thus, *TelePort* should not map awareness cues to telecommunications actions so that, unlike the CAVECAT or Doors systems, *TelePort* should not prohibit certain actions when the door is in a certain state, nor should it prevent a user from making a 'knock' or 'glance' (or any other) request even if the recipient is currently in a conference with someone else. Thus, *all* potential actions should be available *all the time* - there should be no disabling of particular menu items when the user's door is in a particular state, or when a user is in a conference. As argued in previous chapters, *TelePort* should seek only to provide the user with sufficient cues with which to decide upon appropriate courses of action. It should therefore be the user who decides whether or not to knock, to glance to to 'walk in' depending on their current knowledge of the person in question, their situation and the context. The next sections describes how this is realised in the implementation of these specifications.

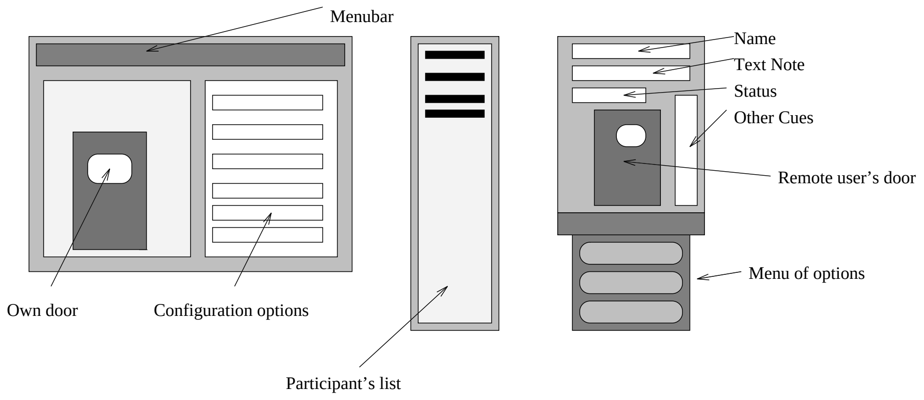

Figure 5.1: *TelePort* user interface sketch

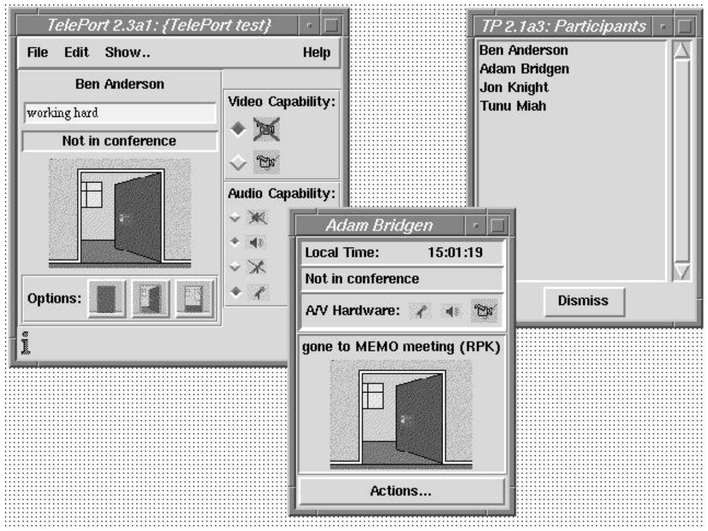

Figure 5.2: *TelePort* User Interface (x 0.5).

5.4 *TelePort* user interface: Implementation

As will be described in more detail in Chapter 6, the *TelePort* system is implemented in Tk, the interface building portion of Tcl/Tk a freely available interpreted scripting language [118]. Tk is based on the Xlib toolkit and provides standard interface widgets that are compliant with the Open Software Foundation's Motif user interface recommendations on any X Windows platform. *TelePort*'s user interface is implemented in approximately 1442 lines of Tcl/Tk code which is included in this dissertation as Appendix C.

Figures 5.2, 5.3 and 5.4 show screenshots of the *TelePort* interface in use. Figure 5.2 shows the main window on the top left, with the participants list on the top right. The main window enables the user to alter the state of their door using the three buttons at the bottom. The large image is updated as soon as the user changes the state of the door and this change in state is immediately sent to the other members of the awareness group. Users can alter the information that is sent about their media capabilities using the radio buttons on the right. The field immediately above the large door image indicates whether or not the user is currently in a conference that *TelePort* knows about. This is exactly the same text string that will be displayed to other users. The field at the bottom of the main window, marked with an 'i', provides the user with feedback about what *TelePort* is currently doing. The participants list is a simple scrollable list box and a particular user's door can be viewed by clicking on their name. Feedback within the list box indicates which user's doors can be viewed - *TelePort* does not draw a user's own door as it would be identical to that displayed in the main window itself. However, all members of the awareness group, including the user, are shown in the participants list.

The *Edit* menu provides access to the user's preferences which consist of the information depicted in the main window, together with:

- **Users name:** As it appears in the participant's list and in the title bar of 'their door'.
- **Telephone Number:** For the case where reliable audio or more forced contact is required.
- **Email Address:** To allow users to email a participant who is not responding or is unavailable.
- **World Wide Web Home page:** To allow users to point other participants to information about themselves or their group.
- **Sound effects volume:** To allow users to set the volume at which sound effects are played on their workstation.

Each of these preferences is editable, any changes made are saved to a preferences file and are sent to the other *TelePort* clients. The *Show* menu enables the user to view the participants list window. It also gives access to a panel of network information. The *Help* menu item provides access to built-in help and to further information via URLs.

In the lower middle of Figure 5.2 is a smaller window representing 'Adam Bridgen'. Adam has set his door to ajar, is not currently in a conference that *TelePort* knows about and can send and receive audio and video and his local time is 15:01. He has edited his text message to indicate that he has gone to a MEMO project meeting in the office of one of the researchers (RPK). The menubutton at the bottom of this window activates a pull-down menu which provides access to the communication actions.

Figure 5.3 shows Ben's door and the set of options for action that are available via a pull-down menu. As previously mentioned, these actions are available at any time irrespective of the state of the user's door or the other cues provided by the user or the system. It is therefore possible, at any time, to 'glance' at Ben or to 'knock and walk in' - it is entirely up to the user to decide whether such actions are appropriate at a given time[2].

Figure 5.4 provides a view of the user 'glancing' at Ben. The window on the left shows Ben's door, the window in the lower centre is the video stream

[2]Note that this policy means that the locus of control is very much with the initiator of the action rather than with the recipient who is unable to 'physically' prevent unwanted access. This point is examined in more detail in later sections.

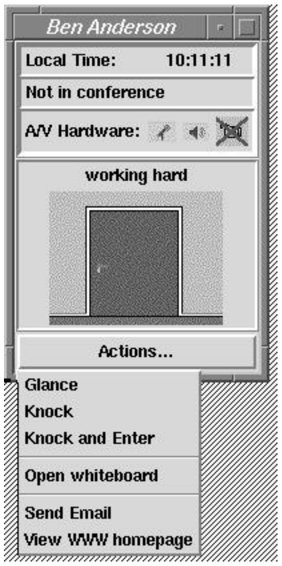

Figure 5.3: Menu of options for action on a user's 'door' (x 0.5).

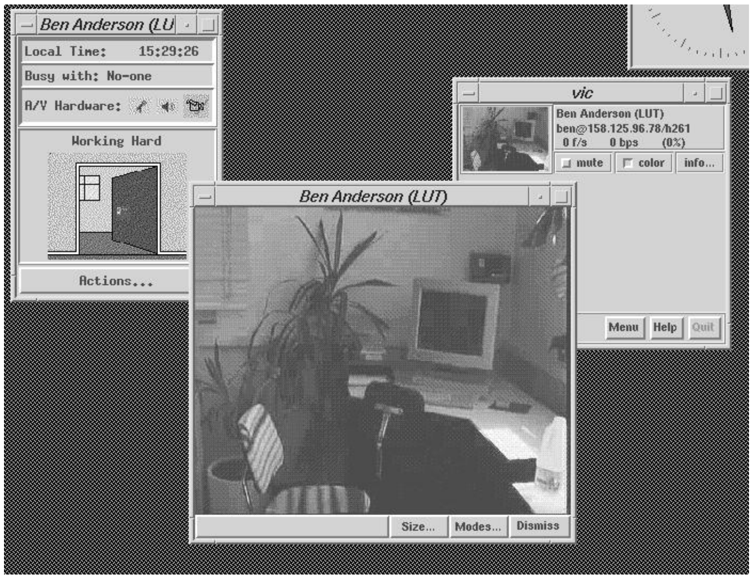

Figure 5.4: Using *TelePort* to glance at a colleague (x 0.5).

from Ben's camera and the window on the right is the main window of the vic video tool that is used to send, receive and display video (see Chapter 6). Ben is obviously not in view of the camera and so the glancer may resort to email or may 'knock' hoping that Ben was simply in a part of the room that was out of camera view ....

5.5 Summary

This chapter has described the user interface of the *TelePort* prototype. It has described the major features of the user interface has discussed the manner in which the design principles outlined in earlier chapters have been incorporated into the user interface.

The implementation of the user interface to *TelePort* had three core aims:

- The provision of a usable interface to the multimedia user services.
- Incorporate the design principles outlined in Chapter 2.
- Use the design framework outlined in Chapter 3

In order to satisfy the first aim, the user interface design drew on a number of similar systems which have been shown to be effective [49, 11]. In particular, it uses the metaphor of an 'office door' to represent the availability state of a particular user, together with related metaphors of 'knocking' and 'glancing' to refer to particular services as is shown in Figure 5.3. The major part of the interface therefore consists of a set of user's doors, the state of which is set by the door's 'owner' as shown in Figure 5.4. Each of these doors also provides the menu of service options that may be requested so that it is clear to a user which member of the group they are attempting to communicate with.

In order to satisfy the second and third aims, the user interface design has been constructed using the framework of providing *options for action* and *cues for behaviour*.

The next chapter shows how the functional requirements generated in this and the previous chapters intermesh with the demands of the design framework outlined in Chapter 3, and how this impacts upon the system architecture that underlies *TelePort*'s user interface.

Chapter 6

TelePort: System Architecture

6.1 Introduction

This chapter documents the design and implementation of the *TelePort* prototype multimedia conference control system. After providing an overview of the system, the prototype's architectural model and communication protocols are described with particular attention given to showing how the system implementation uses the framework of *options for action* and *cues for behaviour*.

6.2 Definitions

Given that the exact meanings of many of the terms used in this chapter vary in the current CSCW and Telecommunications literature, the following definitions are made:

Media Service: - low level media transmission. Examples include audio, video, shared workspace data.

User Service: - any combination of a range of Media Services defined from the point of view of the user. Examples include a 'glance', a 'conference' or an 'office share'. Note that different user services may consist of identical combinations of Media Services, differing only in their purpose (cf. [49]).

Client: - a *TelePort* application running on a particular workstation. A particular client is usually associated with a single user or location.

Awareness Session: - a group of *TelePort* clients which are sending awareness information to each other.

6.3 Platform

As mentioned in Chapter 5, *TelePort* has been implemented using the Tcl-DP [137] extensions to Tcl/Tk [118][1] on a multicast enabled unix workstation running SunOS 4.1.4 and X11R5.

Tcl is a freely distributed interpreted language that, combined with the Tk toolkit, enables rapid prototyping on a variety of platforms including X, MS Windows and Macintosh. The Tcl-DP extensions provide TCP/IP networking functionality that are ideal for developing Internet Protocol (IP) based applications such as *TelePort*.

Tcl/Tk was chosen as a development environment for the following reasons:

- Tcl/Tk provides a wide range of interface widgets that can be easily adapted for most purposes.

- Tcl/Tk's interpreted nature means that the coding to testing cycle is foreshortened compared with equivalent development in a compiled language such as C. However tcl code can also be compiled so that a binary distribution can be made available whilst protecting intellectual and commercial rights to the code. As a result, it has been possible to release pre-compiled binaries of *TelePort* to the Internet's multicast backbone (MBONE) research community for trial use.

[1]Specifically Tcl-DP3.3b and Tcl 7.4 / Tk 4.0

- Tcl/Tk is a widely used prototyping toolkit with significant user support through newsgroups, mailing lists and other online and easily accessible resources.
- the Tcl-DP extensions provide sufficient IP functionality for *TelePort*'s requirements. It was therefore unnecessary to re-implement network code, saving considerable development effort.
- the video and audio tools used by *TelePort* are built from a a mixture of C++ (for the data processing and display) and Tcl/Tk (for the interface). As will be discussed later this allows any other Tcl/Tk application to remotely control these media tools using Tcl/Tk's built-in interprocess communication mechanism.
- Sun Microsystem's current effort to port Tcl/Tk to the Windows and Macintosh platforms mean that any Tcl/Tk application will be able to run on any of these platforms without alteration. Near future releases of Tcl/Tk from Sun will incorporate the IP functionality provided by Tcl-DP. In this event, a small amount of recoding effort should allow *TelePort* to be ported to these platforms providing wider scope for future development and deployment.
- Tcl/Tk is an easily extensible language - new tcl commands can be implemented in C/C++ and added to those available in the core Tcl/Tk distribution. Thus, Tcl/Tk applications can easily be extended to control new devices (such as audio and video codecs for example) provided only that the drivers for such devices can be controlled by C/C++ commands. Thus applications such as *TelePort* that are built in Tcl/Tk can easily be extended to remotely control an immense range of telecommunications hardware across a variety of platforms.
- The combination of Tcl/Tk's cross-platform portability and the use of IP multicast and unicast as the network communication protocols means that the potential user group for *TelePort* is far larger than would have been the case had a platform dependent toolkit and proprietary or untested protocols been used [2]. In particular, this has enabled *TelePort* to be ported to the Sun's Solaris, and Silicon Graphic's IRIX operating systems with very little effort.

TelePort is implemented completely in Tcl-DP except for one additional C function used to return the current time in seconds which is unavailable through either the Tcl-DP extensions or Tcl 7.4/Tk 4.0. The Tcl/Tk and C source code for *TelePort* is included in this dissertation as Appendix C.

6.4 Software Architecture Overview

TelePort provides management of audio, video and data (media) conferencing tools in order to provide a range of different kinds of user service. In essence,

[2]In essence *TelePort* will run on any machine that supports IP multicast and on which Tcl-DP3.3b, and Tcl/Tk can be compiled. At the time of writing this means that *TelePort* can be compiled for virtually any X11 unix system. Although Macintosh and Windows95 versions of Tcl/Tk do exist there is currently no built-in support for unicast UDP or IP multicast since a port of Tcl-DP to these platforms is not yet complete.

the prototype provides users with the ability to set up, use and subsequently destroy, communication connections. In order to do this, each *TelePort* client communicates with its peers by means of a scalable group awareness protocol (GAP), which has been developed specifically for this prototype. *TelePort* uses the GAP protocol to transmit cues about the activity of each user to their colleagues in order to facilitate the use of social cues in access control and privacy management. These cues are then displayed by the user interface. *TelePort* uses the currently evolving session invitation protocol (SIP) [85] to initiate point-to-point, and potentially multipoint, user services. It communicates with local media tools using the Tcl/Tk 'send' command [119]. Conference management, that is exchange of control information once a user service has been initiated is supported by an extremely simple user service control protocol (USCP) that has also been developed specifically for the *TelePort* prototype.

A *TelePort* awareness session is fully distributed; each user runs a client on their workstation, with inter-process communication via unreliable IP multicast ensuring that the system should be globally scalable. Each client maintains its own database of the latest information sent by other members of the group so that the awareness information is entirely replicated within each client - there is no centralised source of data as was the case with the Doors system for example.

Figure 6.1 describes *TelePort*'s general communications architecture. It shows how *TelePort* clients communicate using the GAP, SIP and USCP protocols, and how each client controls its media tools (or other applications) using a 'local conference bus' mechanism. In turn, each of these media tools use RTP (Real Time Transport, [131]), SRM (Scalable Reliable Multicast, [66]) or plain UDP (User Datagram) protocols to transmit their data as appropriate.

Note that as currently implemented, a user is able to run any number of *TelePort* clients at a given time by using different multicast addresses (usually allocated via a multicast session directory tool, such as sdr [82]). It is unclear at present whether this ability is beneficial - users may wish to be members of a number of different 'awareness groups' concurrently (if they are simultaneously members of a number of different work groups for example), but this may lead to media and communication conflicts.

6.5 Media Services

The user services provided by *TelePort* make use of a number of 'low-level' media tools that are responsible for the actual transmission and reception of data, be it audio, video or some other form. In it's current implementation *TelePort* uses three freely available applications that form part of the MBONE tool set.

Video Services are provided by *vic* [110], a software video codec developed by Steve McCanne and Van Jacobson at Lawrence Berkeley Labs, UCB as an instantiation of the RTP protocol [131]. This tool, which is built as a hybrid Tcl/Tk (for the user interface) and C++ (for the video coding/decoding) application, is a highly flexible tool that can make use of a range of hardware devices to capture and display video. It can also use a variety of network transport protocols (eg. IP multicast, ATM, RTIP) and supports a number of video encoding formats (eg. MPEG, H261,

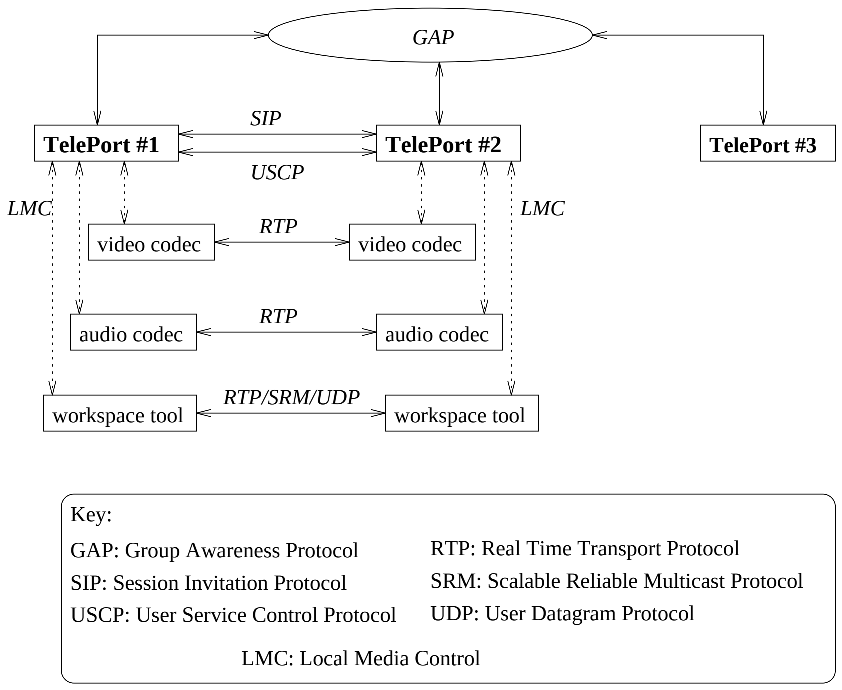
GAP
SIP
USCP
TelePort #1
TelePort #2
TelePort #3
LMC
LMC
RTP
video codec
video codec
RTP
audio codec
audio codec
RTP/SRM/UDP
workspace tool
workspace tool
Key:
GAP: Group Awareness Protocol
SIP: Session Invitation Protocol
USCP: User Service Control Protocol
RTP: Real Time Transport Protocol
SRM: Scalable Reliable Multicast Protocol
UDP: User Datagram Protocol
LMC: Local Media Control

Figure 6.1: *TelePort*Architecture

JPEG). Whilst vic is intended to support multiparty videoconferencing, it can also be used as a point-to-point video conference application. Further, the fact that it has a Tcl/Tk interpreter embedded within it, means that it is highly user configurable and may be remotely controlled by other Tcl/Tk applications. The source for vic has been made freely available as are pre-compiled binaries for a number of systems.

Audio Services are provided by *vat*, a software audio codec developed by Steve McCanne and Van Jacobson at Lawrence Berkeley Labs, UCB as an instantiation of the RTP protocol [131]. The tool uses a similar architecture to vic and supports a number of audio encoding formats (eg. DVI, GSM). As with vic, it has a Tcl/Tk interpreter embedded within it and so is ideal as a remotely controllable audio tool. The source for vat has been made freely available as are pre-compiled binaries for a number of systems.

Workspace services are provided by *wb*, a software whiteboard tool also developed at Lawrence Berkeley Labs, UCB which uses reliable IP multicast to enable distributed users to write/draw on, and import to, a common work space [66].

It is intended that future releases of the *TelePort* tool will enable users to add other media services to those described above by extending the user interface to include other media tools. Examples of these include the reliable audio tool, RAT [154], the shared text editor, NTE [153], and other proprietary tools that are, as yet, undeveloped.

6.6 Local Media Control

Local control of the audio and video tools is realised by the use of the Tcl/Tk *send* command[3] to pass procedure calls from *TelePort* to the local media tool. These commands are interpreted by the tool and the result (if appropriate) is returned to the *TelePort* application. Although vic and vat do not necessarily have hooks that *TelePort* needs in order to provide the user services described in the previous section, it is possible to add functionality to these tools because they automatically read in and execute a Tcl/Tk script file from the user's HOME directory on start-up (usually .vic.tcl or .vat.tcl as appropriate - see [111, 112]). *TelePort* relies on this mechanism to add new commands to the tool's interpreter which are then used as needed during the awareness session. As an example, when *TelePort* sends a 'tp-new-video' command to vic, a command defined as *tp-new-video* in .vic.tcl will be executed - in this case, it forces vic to create and use a new network video object. Other examples include the ability to mute/unmute the microphone, close an audio connection, and pause/unpause video transmission without the user needing to directly interact with the vic/vat interfaces. Since it is important that these additional commands are not called by the media tool in other contexts, and so do not unexpectedly alter the tool's standard behaviour, they are enclosed by procedures whose names are carefully chosen to be specific to *TelePort*.

This mechanism is currently supported by the vic and vat tools, both of which have an embedded Tcl/Tk interpreter. As is discussed below, the whiteboard tool, wb, does not have an embedded Tcl/Tk interpreter and so cannot be controlled in this manner.

6.7 Communication Protocols

As has been briefly mentioned, *TelePort* uses three distinct communication protocols: GAP to distribute awareness information between *TelePort* clients, SIP to initiate user services between those clients as requested by users and USCP to control the user service once it has been initiated.

6.7.1 GAP: The Group Awareness Protocol

The Group Awareness Protocol (GAP) is a scalable text-based protocol designed primarily for the *TelePort* application, but it is also intended as a flexible protocol that could be used and extended in future, or sibling systems[4]. In order to act as a general support for group awareness, GAP has the following requirements:

- It needs to support many to many transmission.
- It must be scalable so that the protocol can support groups of varying sizes without unnecessary and excessive usage of network resources.

[3]Due to changes in Tcl/Tk's send protocol, this means that the media tools must be using Tk 4.0. In addition, the *send* command will not work unless the X server is using Xauth or a similar security mechanism - a built in security feature of Tcl/Tk

[4]BT is currently seeking patent protection for some of the concepts and techniques reported in this section.

- It needs to be more or less reliable, that is users need to know that the awareness information about another user is accurate or, if this is not the case, how likely it is to be inaccurate.
- It needs to be efficient. That is, it needs to transfer the maximum amount of information using the minimum amount of data.

GAP draws heavily on a number of protocols that have previously been developed for multimedia conference control such as Schooler's CCP [128] and Wakeman et al's CCCP [86] both of which have been suggested under the auspices of the IETF's multimedia session control (MMUSIC) working group. It also draws inspiration from work being carried out in the IETF's Audio-Video transport group, most notably the Real Time Protocol (RTP) specification [131].

In order to fulfil the general requirements listed, GAP is a fundamentally lightweight protocol that uses IP multicast [47] to communicate awareness information between *TelePort* clients. Each client is associated with a particular user on a particular workstation and periodically multicasts awareness information about that user to the specified IP multicast address on the specified port. The full protocol description is provided in Appendix B, this section provides a summary of the important features whilst an example of a GAP packet is provided in Table 6.1.

In common with other scalable IP multicast based protocols, GAP avoids the overhead of maintaining a reliable list of members to whom information is explicitly sent. Rather, it utilises the simple best-effort delivery of the basic IP multicast service model. In this model, data is sent to a particular IP multicast address from a given port, and any application that is listening on this address/port combination can receive the data - there is no way for the sender to tell which, if any, of the applications listening on the address/port actually received data, nor if there are any such listening applications at all. As a result, network usage increases as a linear factor of the number of group members who are actually sending data - the network usage for a group of 10 members with one sender will be exactly the same as for 1000 members with one sender because those members do not explicitly exchange any group membership data[5]. As a result, GAP cannot ensure reliable delivery of a given unit of information so there can be no guarantee that members of an awareness session will have consistent records of each other's state. However GAP needs to ensure some level of reliability of information about members of the group and to do this it uses the same mechanism as other scalable multicast protocols such as RTCP [131], namely the regular re-sending of current state information. Since each GAP packet carries a 'sent at' time-stamp, each receiver can calculate the time elapsed since it last received any information from that client, and hence the likelihood that the information it has is inaccurate. Other user interface strategies outside the scope of GAP that can also be used to offset this unreliability are discussed below.

The use of IP multicast has the secondary advantage that it enables the straight forward capture of network activity since logging scripts can parse and

[5]For a detailed explanation of the implications this model has for scalability with respect to multiple data senders and listeners the reader is directed to Steve Deering's PhD dissertation [47]. A consideration of a scalable, reliable multicast protocol for use where guaranteed delivery is essential is provided by Floyd et al's discussion of their whiteboard tool, wb [66].

Item	Example	Definition
0	v=GAP/1.0 INFO	Version Type
1	id=8975664	ID
2	ntp=457386765	NTP Time Stamp
3	cn=ben@128.125.110.123	Canonical name
4	seq=1	Sequence number
5	n=Ben Anderson (LUT,UK)	Name
6	t=14:22	Local Time
7	u=http://www-rs.cs.lut.ac.uk/ben	URI for more info
8	e=B.Anderson@lut.ac.uk	Email
9	p=+44 (0)1509 222689	Telephone
10	ca=158.125.5.11/5536747:Glancing at Jon	Current activity
11	ca=158.125.5.11/4345667:Whiteboard with Adam	Current activity
12	i=doorstate:open	State of user's 'door'
13	i=note:Gone to lunch	Short text message
14	i=havevideo	If user can send video
15	i=havespeaker	If user has speaker
16	i=havemic	If user has microphone

Table 6.1: Example GAP 'INFO' packet

record any GAP packets received without impacting network usage since, as mentioned, members of IP multicast groups can be receivers without being senders [47].

GAP packets consist of a standard set of header fields followed by any number of further text fields. All of the fields are separated by the newline character. The newline character is therefore not allowed as data in any of the fields. Each field has the format 'x=y' where x is a unique identifier and where y is a string whose format is determined by the identifier.

GAP's textual nature reflects it's initial implementation in Tcl/Tk. In common with the IETF MMUSIC Working Group's Session Description Protocol [84], GAP is not a high throughput protocol requiring many hundreds of packets to be parsed every second, thus it's textual format (and hence human-readable nature) is not expected to reduce computational efficiency.

GAP uses 2 different packet types:

INFO Packet This packet contains awareness information about the particular *TelePort* client that sent it. This information consists of the cues generated in Chapters 4 and 5 (see Table 6.1).

BYE Packet A packet multicast when a member leaves the session by quiting the *TelePort* application.

As mentioned, the INFO packet is repeatedly multicast after a given random time period. This time period is recalculated at each send and the value of the time period depends on the number of members in a particular session so that no *TelePort* session, whatever the scale, should use more than a set bandwidth for inter-peer communication purposes. The algorithm used by a GAP client is more or less identical to that specified for RTCP and the tcl code used to implement it is provided in Appendix C. Briefly, a client divides the allocated bandwidth (bytes per second) by the product of the number of GAP clients it knows about and the mean GAP packet size (in bytes). It then multiplies the result by a random factor in the range 0.5 to 1.5. If the resulting periodicity is less than a 5 second time interval, then the interval is set to 5, otherwise the calculated interval is used.

In common with its use in RTCP [131], this algorithm provides:

- protection against bursts of GAP packets that might exceed the allocated bandwidth when group membership is small and when the traffic is not smoothed by random factors.
- protection against unintended synchronisation. In addition the first packet sent is delayed by a random variation of half the minimum time interval to prevent synchronisation between clients joining a session simultaneously.
- a dynamic calculation of the mean packet size to allow automatic adaptation as the amount of awareness information sent varies.
- protection against extremely small resend intervals if a session has few members.

If a member of an awareness session is not heard from for a given time period, they are kept as possible members of the session but the user interface reflects the fact that they may be uncontactable and that their state information may be inaccurate. Experience with the RTP tools vic (cf. [110]) and vat suggests that this can be an appropriate way to offset the inherently unreliable nature of the lightweight group membership model adopted by RTCP and so it might be expected to be equally appropriate for GAP clients. The RTP draft recommends that sites (clients) that have not been heard from for 5 report intervals may be marked as inactive but that sites should not be discounted from the total number of participants for a further 30 minutes to span typical network partitions [131]. In order to investigate the utility of these recommendations for the distribution of awareness information rather than real-time control data, *TelePort* adopts the same timeout algorithms.

Note that by varying the bandwidth allocated to the GAP client, the rate of update of awareness information within the group can be altered. It is expected that different kinds of GAP clients will have different requirements for the rate of propagation of such information. In addition, it may be prudent to reduce the allocated bandwidth as the ttl (or physical scope) of the awareness session increases in order to avoid overloading lower bandwidth wide area networks. This is common practice in the use of other experimental IP multicast applications at the present time.

As mentioned, the unreliable nature of basic IP multicast means that the information held by a particular client about another member of the session may be inaccurate due to network or application errors. This problem is resolved using an implicit synchronisation mechanism which causes peers to correct the information they hold about other members of the awareness session such that they are more likely to be consistent without having to take any explicit action to do so. This is supported in *TelePort* by the frequent multicast of state and by a two-stage check when a service is requested.

Frequent multicasting of state: Even in a relatively high-loss network the frequent multicast of state should provide sufficiently up to date status information to automatically correct most state errors. As mentioned, this approach has been experimentally illustrated elsewhere by IP multicast based RTP [131] and SRM [66] tools. If the bandwidth allocated to a GAP client is high then correction of inconsistencies will occur much more rapidly.

Two stage check: When a service is requested, *TelePort* checks it's local state table to determine whether or not the other client is currently in a SIP initiated session before sending a service request. If the client is in such a session, the user is presented with the option to force a service request to be sent. If the recipient is indeed busy, this request may generate a 'busy' reply thus confirming the validity of the original local state table. On the other hand, if this 'forced' request is accepted and a service is created then the state table will be synchronised automatically by the service set-up procedures.

Taken individually, neither of the above could ensure a high degree of synchronisation. When combined however, it seems likely that the two implicit mechanisms will be effective given the relatively low frequency with which users will interact with the system. If this is the case it may be that similar future systems will not need relatively expensive explicit synchronisation mechanisms provided that the possibility of such inconsistency is made clear to the user through the user interface. Note also that in the case of the two-stage check, *TelePort* allows a user to send a request to another user who appears to be already busy. Such a policy is a result of considering this design decision in the light of the framework described in Chapter 3. This will be discussed in more detail in Chapter 8.

6.7.2 SIP: Session Invitation Protocol

TelePort uses the SIP [85] protocol to initiate user services. SIP is a relatively simple protocol that is being evolved by the IETF MMUSIC Working Group to address the problem of inviting users to take part in multimedia conferencing sessions. SIP uses the SDP [84] syntax to describe invitations and *TelePort* leverages this by using it's own specific session attribute. This is described in some detail in Section 6.8.

6.7.3 USCP: User Service Control Protocol

TelePort uses a simple session control protocol to manage the user services following initiation. In it's current instantiation, this is used to inform a *TelePort* client when one member of a one-to-one user service no longer wishes to participate by sending a 'disconnect' message.

USCP adopts the same service model as SIP, that is it relies on best effort UDP delivery of datagrams and uses a re-send mechanism with a linearly increasing time interval to provide a level of reliability. *TelePort*'s current restriction to point-to-point services means that the use of unicast UDP is sufficient. In the case where a multipoint control protocol is required, if *TelePort* supported many to many conferences for example, a protocol such as CCCP [86] or SCCP [26] would be needed. This is left to future work.

Item	Example	Definition
0	v=USCP/1.0 DISCONNECT	Version Type
1	ntp=888173030	NTP timestamp
2	cn=ben@128.125.110.123	Canonical name
3	id=128.125.110.113/237488	SIP id of conference being left

Table 6.2: Example USCP 'DISCONNECT' packet

An example of a USCP DISCONNECT packet is provided in Table 6.2. Lines 0 and 1 form a standard USCP header; line 0 provides a definition of the protocol version and the USCP message type whilst line 1 is an ntp timestamp which gives receivers information on whether or not this packet has been delayed. Subsequent lines determine the canonical name of the client that sent the packet and the SIP id of the conference concerned.

6.8 *TelePort* User Services

The user services presented as options in the user interface are created by combining the media tools in a number of different ways:

Glance: The glancee's video tool is configured to receive video with the appropriate parameters. The glancee's *TelePort* client plays a short 'door creaking' sound effect and the video tool sends a short low bandwidth video stream to the glancer. A dialog box indicates to the glancee who is glancing.

Knock: Request a connection - the recipient's *TelePort* client plays a 'knock' sound effect and displays a dialog box requesting a connection. If the user accepts the request, *TelePort* launches vic and vat with appropriate parameters and sends a suitable response to the sending client. This client then launches its media tools. Both clients force their media tools to start sending video (in the case of vic) and unmute the mic (in the case of vat).

Knock and Enter: Set up a connection without request. The recipient plays a 'knock and door clicks open' sound effect and then launches vic and vat with appropriate parameters. On receiving a suitable response, the client that sent the original request launches its media tools. As above, both clients force their media tools to start sending video and unmute the mic respectively.

Workspace Launch the whiteboard tool. The *TelePort* client that received the request displays a dialog box to the user. If the user accepts the request, each client launches the whiteboard tool with the appropriate parameters.

TelePort also provides access to two other user services via external applications:

Email: Electronic Mail to contact a member of a session.

URI Request: View a member's world wide web (WWW) home page.

Users initiate a particular user service by selecting from the menu of actions on a particular user's door (see Figure 5.3). *TelePort* sends this service request to the client as a SIP request and adds this service to the list of conferences in which the user is currently engaged. If this service should fail or when the service is completed, it is then removed from the list of current conferences. Similarly, when a *TelePort* client receives a SIP request it adds the service to its own list of current conferences. This list is then suitably updated if the service fails, or when it is concluded.

TelePort uses the SDP session attribute *tpservice:* to define its user services and to use these definitions to differentiate between kinds of invitation. This is not the intended use of SDP's session attribute description mechanism[6] and current discussion within the MMUSIC working group is focusing on whether SIP needs to support 'invitation attributes' as distinct from 'session' (ie. user service) attributes. As will be made clear below, such a distinction allows SIP clients to differentiate between invitation types whose media service descriptions are identical - 'knock' vs 'knock and enter' for example. Experience with experimental media space systems such as RAVE have suggested that this can be an important way to support user's requirements for different kinds of communication actions even though the underlying technology requirements might be the same [50].

It is important to note that in it's current implementation, *TelePort* clients will process user service requests from other members of the session *without regard to the current activity of the client* (or user). It is therefore possible for a user to interrupt an on-going conference by sending a 'glance' or 'knock' or even 'walk in' request to either of the two conferees. Whilst this may appear to compromise privacy, such an implementation policy follows the principle of not embedding rules of access into the system itself. Instead, the fact that two members are engaged in a conference is available from GAP and so can be displayed in a GAP client's user interface. If that third user wishes to interrupt an ongoing conference then the system, by the arguments outlined in earlier chapters, should not refuse access although it may, as is the case with *TelePort* provide a user interface dialog warning the user(s) that an interruption is about to take place.

6.8.1 Glance

A 'glance' consists of a 5 second one-way video only connection analogous to looking briefly through an office door. The state transitions for the 'glance' user service are shown in Figure 6.2 whilst the format of the SIP request is given in Table 6.3.

At start-up, *TelePort* launches one instantiation of the video tool which runs permanently in the background. This particular video tool is used exclusively by the 'glance' user service in order to provide a sufficiently rapid response to a 'glance' request by avoiding the delay needed to launch the tool before video can be sent.

As Figure 6.2 shows, *TelePort* #1 (glancer) sends a glance request and, if successful, adds the service to its 'currently active' list and configures its glance video tool to receive and display video from *TelePort* #2 (glancee). If able to send video, *TelePort* #2 adds the service to its 'currently active' list and configures its glance video tool to send video to the glancer. It then sends video

[6]Mark Handley, personal communication. See also [84].

Item	Example	Definition
0	SIP/1.0 REQ	Version Type
1	PA=158.125.5.11	Path
2	AU=none	Authority
3	ID=158.125.5.11/5536747	Invitation ID
4	FR=B.Anderson@lut.ac.uk	From
5	TO=J.P.Knight@lut.ac.uk	To
6	v=0	SDP version
7	o=ben 5536747 0 IN IP4 158.125.5.11	Origin
9	s=Glancing at Jon	Name
10	i=left blank	Information
11	e=B.Anderson@lut.ac.uk	Email
12	c=IN IP4 158.125.5.11	Connection Details
13	t= 3452334100 3452334150	Timing Information
14	a=tpservice:GLANCE	Attribute field
15	m=video 5646 RTP H261	Media field

Table 6.3: SIP packet for ‘Glance’ user-service

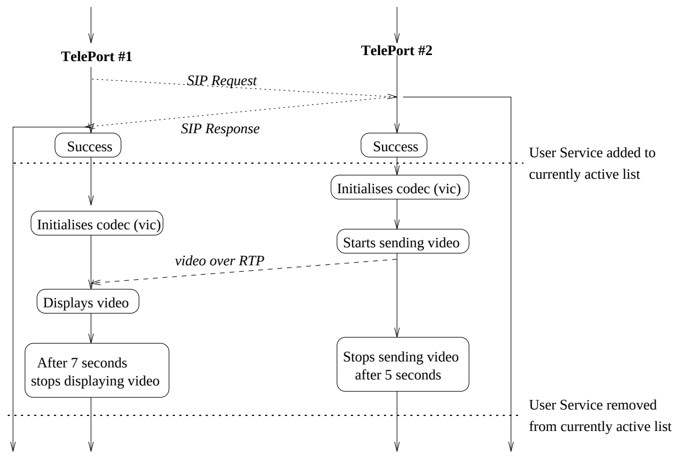
TelePort #1
TelePort #2
SIP Request
SIP Response
Success
Success
User Service added to currently active list
Initialises codec (vic)
Initialises codec (vic)
Starts sending video
video over RTP
Displays video
After 7 seconds stops displaying video
Stops sending video after 5 seconds
User Service removed from currently active list

Figure 6.2: System States for ‘Glance’ Service

Item	Definition	Example
0	SIP/1.0 REQ	Version Type
1	PA=158.125.5.11	Path
2	AU=none	Authority
3	ID=158.125.5.11/5536747	Invitation ID
4	FR=B.Anderson@lut.ac.uk	From
5	TO=J.P.Knight@lut.ac.uk	To
6	v=0	SDP version
7	o=ben 5536747 1 IN IP4 158.125.5.11	Origin
9	s=Knocking at Jon	Name
10	i=left blank	Information
11	e=B.Anderson@lut.ac.uk	Email
12	c=IN IP4 158.125.5.11	Connection Details
13	t= 0 0	Timing Information
14	a=tpservice:KNOCK	Attribute field
15	m=video 52345 RTP H261	Media field
15	m=audio 20101 RTP PCMU	Media field

Table 6.4: SIP packet for 'Knock' user-service

for a period of 5 seconds. If *TelePort* #2 is unable to send video for any reason, the request fails and both clients remove the service from their 'currently active' lists. This also occurs when the glance is complete - ie. after 5 seconds of video. No USCP messages are sent.

6.8.2 Knock

The 'Knock' user service is a request for a two way audio and video conference. The successful initiation of the conference requires explicit acceptance by the receiver of the request. The state transitions for the 'knock' user service are shown in Figure 6.3 whilst the format of the SIP request is given in Table 6.4.

As Figure 6.3 shows, *TelePort* #1 sends a knock request to *TelePort* #2 which plays a 'knocking' sound effect and draws a dialog box requesting user #2 to accept, reject or ignore the request. If the user does not respond, a BUSY SIP response is sent after a timeout period. If user #2 accepted the request, *TelePort* #1 adds the service to its 'currently active' list and launches a new video and audio tool configured with the connection information returned in *TelePort* #2's response. If a success reply was sent, *TelePort* #2 adds the service to its 'currently active' list and launches a new video and audio tool configured with the connection information it just sent back to *TelePort* #1. Both *TelePort* clients force the video tool to start sending video and force the audio tool to unmute the microphone so that audio and video connectivity is as rapid as possible. If the processing of the request fails at any point a FAILED SIP response is sent and both clients remove the service from their 'currently active' lists.

Either of the two participants can terminate the subsequent conference by sending a USCP DISCONNECT message which causes each *TelePort* client to kill the relevant media tools. At this point the two *TelePort* clients remove the service from their 'currently active' lists.

Note that since this request is for a potentially unbounded conference, the

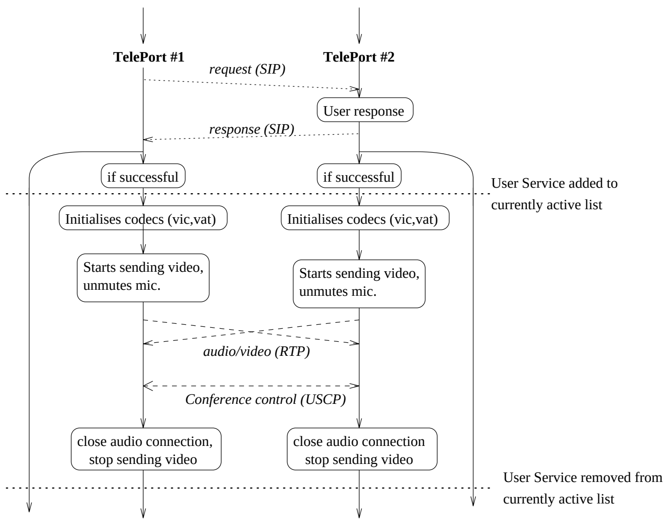
TelePort #1
TelePort #2
request (SIP)
User response
response (SIP)
if successful
if successful
User Service added to currently active list
Initialises codecs (vic,vat)
Initialises codecs (vic,vat)
Starts sending video, unmutes mic.
Starts sending video, unmutes mic.
audio/video (RTP)
Conference control (USCP)
close audio connection, stop sending video
close audio connection stop sending video
User Service removed from currently active list

Figure 6.3: System States for 'Knock' Service

time values are set to zero (cf. [84]).

6.8.3 Knock and Enter

The 'knock and enter' user service is a two way audio and video conference created without requesting permission from the other participant, analogous to walking in to someone's office and announcing your presence by tapping on the door as you do so. The state transitions for the 'knock and enter' user service are shown in Figure 6.4 whilst the format of the SIP request is given in Table 6.5.

As Figure 6.4 shows, there is very little difference between this user service and a 'knock' with the notable exception that the conference is initiated without requiring permission from the receiver of the request.

Note that there is no difference between the SIP requests made for a 'Knock' user service and a 'Knock and Enter' user service other than the SDP session attribute line. This is a pertinent example of how the media service description (ie. the SDP description) of two user services can be fundamentally the same but the invitation type, and therefore the way in which the application's user interface presents it, is distinctly different.

6.8.4 Workspace

The 'whiteboard' user service provides a persistent shared whiteboard connection between the participants. Normally such a whiteboard service would be requested during an ongoing conference but this is not necessarily the case and *TelePort* does not enforce this. Here again is an example of how *TelePort* avoids embedding any assumptions about what users may want to do and when - instead users can request a whiteboard service at any time, even if not currently involved in a conference with the other user. The state transitions of the service are shown in Figure 6.5. An example of the format of the SIP request sent is given in Table 6.6.

Item	Definition	Example
0	SIP/1.0 REQ	Version Type
1	PA=158.125.5.11	Path
2	AU=none	Authority
3	ID=158.125.5.11/5536747	Invitation ID
4	FR=B.Anderson@lut.ac.uk	From
5	TO=J.P.Knight@lut.ac.uk	To
6	v=0	SDP version
7	o=ben 5536747 1 IN IP4 158.125.5.11	Origin
9	s=Connecting to Jon	Name
10	i=left blank	Information
11	e=B.Anderson@lut.ac.uk	Email
12	c=IN IP4 158.125.5.11	Connection Details
13	t= 0 0	Timing Information
14	a=tpservice:CONNECT	Attribute field
15	m=video 52345 RTP H261	Media field
15	m=audio 20101 RTP PCMU	Media field

Table 6.5: SIP packet for 'Knock and Enter' user-service

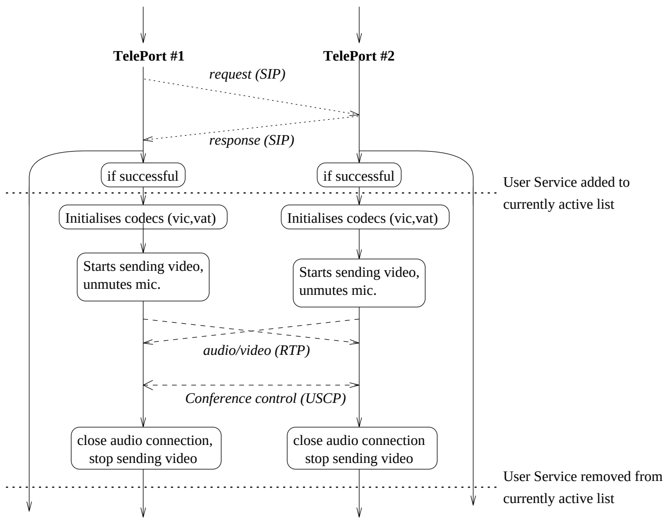
TelePort #1
TelePort #2
request (SIP)
response (SIP)
if successful
if successful
User Service added to currently active list
Initialises codecs (vic,vat)
Initialises codecs (vic,vat)
Starts sending video, unmutes mic.
Starts sending video, unmutes mic.
audio/video (RTP)
Conference control (USCP)
close audio connection, stop sending video
close audio connection stop sending video
User Service removed from currently active list

Figure 6.4: System States for 'Knock and Enter' Service

Item	Definition	Example
0	SIP/1.0 REQ	Version Type
1	PA=158.125.5.11	Path
2	AU=none	Authority
3	ID=158.125.5.11/5536747	Invitation ID
4	FR=B.Anderson@lut.ac.uk	From
5	TO=J.P.Knight@lut.ac.uk	To
6	v=0	SDP version
7	o=ben 5536747 1 IN IP4 158.125.5.11	Origin
9	s=Whiteboard session with Jon	Name
10	i=left blank	Information
11	e=B.Anderson@lut.ac.uk	Email
12	c=IN IP4 158.125.5.11	Connection Details
13	t= 0 0	Timing Information
14	a=tpservice:WHITEBOARD	Attribute field
15	m=whiteboard 34858 UDP WB	Media field

Table 6.6: SIP packet for 'Workspace' user-service

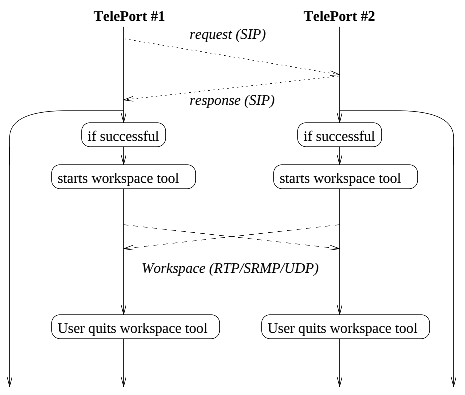

Figure 6.5: System States for 'Workspace' Service

Note that in it's current implementation, the whiteboard tool, *wb*, does not contain an embedded Tcl/Tk interpreter. This means that it cannot be remotely controlled via *TelePort*'s media control mechanism. As a result, a *TelePort* client does not update its 'current activity' list if a whiteboard request is received because it cannot force the whiteboard application to quit (or know when the user has done so), and therefore could not know when to remove the record from the list.

6.8.5 Email

The current version of *TelePort* does not implement an integrated email tool. Thus this service is no more than a dialog box displaying the particular member's email address. The user can use the X-selection mechanism to copy this to their normal email tool.

6.8.6 URI Request

As with Email, the current version of *TelePort* does not implement an integrated WWW browser. Thus this service is no more than a dialog box displaying the particular member's home URI. The user can use the X-selection mechanism to copy this to their normal WWW browser.

6.9 Summary

This chapter has described the system architecture of the *TelePort* prototype. It has described the architectural model upon which the implementation is based and given an overview of the communication protocol by which peer *TelePort* applications exchange session information. It has also described the manner in which the design principles outlined in earlier chapters have been incorporated into the *TelePort* system.

Chapter 7

TelePort In Use: Analysis and Experiences

7.1 Introduction

TelePort was used as a research prototype between May and July 1996 as part of ongoing experimentation with multicast IP tools at Loughborough University of Technology. In particular it has provided support for a loosely associated group of 4 to 6 graduate students and researchers based in the Departments of Computer Studies and Electrical Engineering. *TelePort* has also been used experimentally within BT Labs both as a demonstration of current IETF activity and as part of a suite of evolving tools for integrated services on digital networks. In addition, *TelePort* has been made available in binary form to the IETF's multiparty multimedia session control working group (MMUSIC) as one of only two current implementations of the group's SIP protocol. Experience gained during the implementation and use of *TelePort* have been fed back into this working group. Compiled alpha level binaries for a number of unix platforms are freely available from

`http://pipkin.lut.ac.uk/~ben/PHD/alpha/`

for evaluation purposes[1]. These binaries have been compiled using the freely available Embedded Tk [95] processing macros which process Tcl/Tk scripts into strings that are embedded within compilable C code. Compiling the resultant C code against the Tcl/Tk libraries produces an executable binary which can then be distributed whilst maintaining protection of copyright or patented material.

This chapter provides an analysis of *TelePort* as a group awareness tool - it reports user experiences with the *TelePort* prototype and the issues that usage has raised, and it provides an analysis of technical issues such as the scalability of GAP and the utility of different invitation types in SIP.

7.2 User Interface Issues

Whilst formal or systematic usability trials were never intended to form part of the work reported in this dissertation, the usage of *TelePort* over a period of time has achieved two aims. Firstly, it has demonstrated that the prototype is a functional, usable CSCW system and secondly it has raised a number of interesting issues related to this particular user interface and to awareness tools in general.

7.2.1 General Comments

During trials at LUT, *TelePort* has been used as a way of discovering the whereabouts and availability of researchers in preparation for more focused meetings. One particular example involved an awareness session that was run to 1) test an alpha version of the *TelePort* client and 2) to support awareness between the small group who were helping to debug that version. On a number of occasions, members of the group glanced at each other to see who was around and currently in which office so that they could initiate debugging conversations. Since most of the members of this group were working on different research projects, they were often not all available at the same time due to meetings and off-site visits.

[1]Note that BT retains all rights to these binaries and is seeking legal protection for the ideas and concepts contained within them

As a result of being aware of this, *TelePort* users would circulate bug reports and possible solutions by email. They would often then refer to this permanent resource when, for example, a serendipitous meeting in one researcher's office was discovered by a remote member, via a 'glance'. This remote member would then join in the discussion and often start a shared whiteboard so that they could look, together, at the code itself.

It was noticeable that earlier versions of the *TelePort* tool which did not enable users to start the whiteboard would often force the group to physically meet in an office in order to have the code in front of them. Once *TelePort* enabled them to examine the code together without having to co-locate, this practice generally stopped. Whilst, again, this is an informal observation, it appears that even though a system that provides glances and video conferences can be a good way to coordinate work, it does not actually support the work itself when that work is artefact based. Similar findings have been reported in the Media Space literature and it implies that systems such as *TelePort* must be able to smoothly integrate *work tools* as well as audio or video based *coordination tools*. Further, it must be possible to move smoothly from single-user situation to a multi-user situation with the minimum of effort [93].

In general there have been few comments on the ease of use, or otherwise, of the user interface itself. This is probably due to the use of the familiar Motif-like widgets provided by Tcl/Tk. In addition the user groups have been, for the most part, interested researchers who tend to focus attention more on system and network functionality than on the interface itself.

A number of users have noted that the ability to reduce the volume of the auditory icons is particularly important when *TelePort* is used in an open or shared office situation. In addition there was often some confusion as to the origin of a knock - some users reported checking their workstation screen for a *TelePort* dialog box when in fact someone was outside their physical door and, vice versa, checking to see if someone was outside their door when in fact it was a *TelePort* request.

7.2.2 Tailorability

Users quite quickly discovered that they could edit the graphical icons used in *TelePort*. The icons used to represent door states are read in from GIF files by *TelePort* at start-up time and it was suggested that a simple way to enable users to tailor their representations would be for *TelePort* to distribute these images in real time. Thus if a user wanted to 'change their door' in any way, all they would need to do is create their own GIF files and have their *TelePort* client display and hence circulate them to the other members of the group. A number of mechanisms for achieving this could be considered. It would, for example, be trivially easy for *TelePort* to read in all the GIF files with a certain name (such as door-state-x.gif) and enable the users to select from amongst these using buttons or a menu. These images could then be multicast in real time to the other members of the session using an implementation of Floyd et al's scalable reliable multicast framework [66].

Some users discovered that *TelePort* also used external audio files to provide auditory icons and that by replacing these files with sounds of their own making, they could avoid the "where did that knock come from?" problem mentioned above. Again, it would be trivially easy to implement a mechanism for users to select from amongst a range of sounds for any particular event as has been implemented in the RAVE system [72]. It should be noted however that this could result in users incorrectly expecting certain sounds to be played at another user's workstation which may cause a certain amount of confusion.

7.2.3 Privacy and Control

The ability to immediately see from the user interface who was involved in 'conferences' with others provoked considerable debate. Some users suggested that this was an invasion of privacy whilst others mentioned particular instances when it had been useful such as interrupting an ongoing conference to announce the start of a meeting that both participants should have been attending.

As was mentioned in Section 5.4 the policy of not embedding social rules into the system has resulted in the locus of control being passed to the initiator rather than the recipient. In the light of the various studies of MediaSpace technologies mentioned previously, it might be expected that in this situation users may choose to disconnect their cameras and microphones or even quit the application in cases where they feel social constraints or the cues provided by the system are insufficient. The point at which users feel social control must be augmented by embedded constraints indicates the limits of the approach proposed by this dissertation. The location of this limit for particular situations or systems is currently unclear. What does seem clear is that a system such as *TelePort* needs to be able to provide a range of privacy measures that are under user control. One of the key issues appears to be knowing who had access to 'your' awareness information - in larger, more public groups, it was clear that users would not want details of activities to be broadcast whilst in smaller groups where the members knew each other relatively well, this was less important.

A number of users remarked that they might be interested in being members of multiple awareness sessions at any one time which raises a number of interface, and also architectural issues. It is not at all clear how much awareness information should be shared between different awareness sessions in order to enable users to regulate their access through social controls based on cues derived from that information.

7.3 Technical Issues

7.3.1 Behaviour of the Group Awareness Protocol

The behaviour of the group awareness protocol was an important consideration in the design of the *TelePort* prototype. Figures 7.1 to 7.5 show the results of a number of simulations of the behaviour of GAP under different conditions. As previously described, GAP uses the same algorithm as RTCP to calculate the interval between multicasts of the INFO packet. This time interval is therefore dependent upon the bandwidth that is reserved for the awareness session, the number of members of the session and the mean INFO packet size. Informal observations have suggested that the mean packet size tends to vary between 230 and 280 bytes with most of the within-session variation due to changes in member's current activity lists. Each of the simulations reported in this section assumed a mean packet size of 240 bytes, maintained a constant bandwidth value, and varied the group membership randomly to provide 100 data points representing the calculated time interval. The authors of the RTP protocol suggest that the algorithm provides fast response for small sessions where identification, and in the case of GAP, 'freshness' of information might be critical but also provides automatic adaptation for large sessions. These figures illustrate precisely this principle.

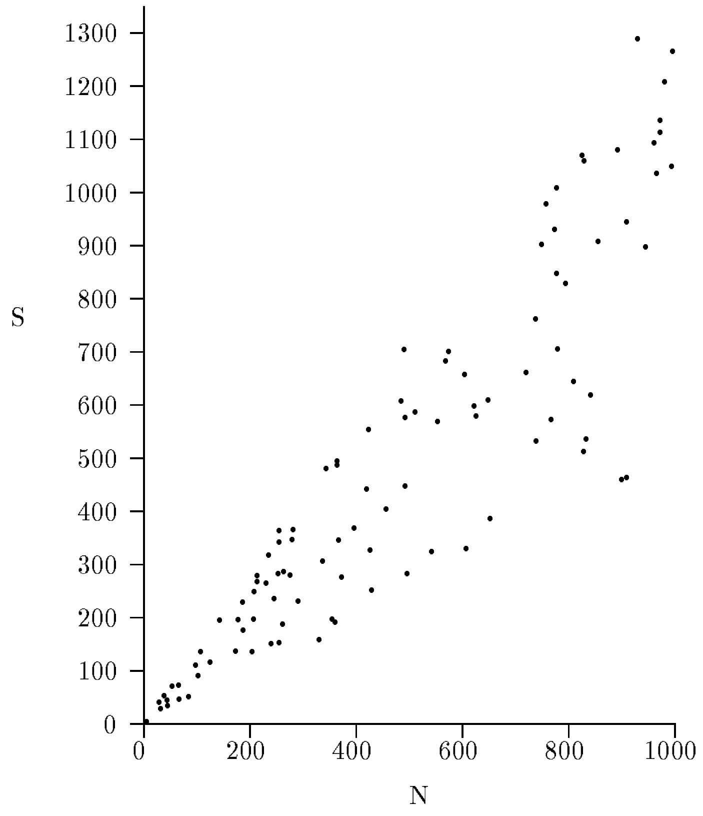

Figure 7.1: GAP resend interval (S seconds) against number of session members (N) where $0 < N < 1000$, bandwidth = 0.25 bytes/s and mean GAP packet size = 240 bytes

Figure 7.1 shows a simulation of the behaviour of GAP given a set bandwidth of 0.25 bytes/s as group membership varies between 1 and 1000. As would be expected from the algorithm, there is clear linear relationship between group membership and the length of the time interval whilst the interval varies randomly between an upper and lower bound. The lower bound is set by the bandwidth limitation and ensures scalability through self-regulation of network usage whilst the upper bound ensures that the time interval is as reasonably small as possible to ensure that information is sent in a timely manner. It is interesting to note that a GAP session with a set bandwidth of 0.25 bytes/s rapidly forces intervals in excess of 300 seconds (5 minutes) once group membership exceeds about 200. At present there appears to be no empirical evidence to determine whether or not this is an adequate rate at which to circulate awareness information. Further, there appears to be no evidence suggesting how large awareness groups are likely to get nor what fraction of an organisation's network resources may be usefully dedicated to sessions of varying size and purpose. It may be that large sessions such as 'everyone in the organisation' are appropriate for low bandwidth, and hence low periodicity and low reliability of information whilst more focused sessions such as for a project group may need to be more reliably and rapidly updated, requiring a higher set bandwidth. It is for this reason that GAP does not define a particular bandwidth, and hence periodicity, but assumes that this decision is made on a per-session basis and handed to the GAP client by an appropriate session management application. Indeed it may be appropriate for GAP clients to allow users to increase the bandwidth devoted to the session in which they are involved, in which case GAP must provide a means for clients to exchange this information in order that they can continue to calculate appropriate time intervals. The authors of RTP suggest the the RTCP report interval is only expected to usefully scale to 2-5 minutes, but it is unclear whether this also holds for awareness information and hence for GAP.

Figures 7.2 and 7.3 show the effect of increasing the available bandwidth for a GAP session. Clearly, as bandwidth increases, the calculated time interval for a given membership does not need to be as great. This confirms the suggestion made in Chapter 6 that GAP clients requiring a greater frequency of information update need only increase the bandwidth devoted to the session. Thus a session with a set bandwidth of only 1.0 bytes/s ensures that the time interval when membership approaches 1000 is about 300 seconds (5 minutes). This membership size is approaching that which would be expected for a campus-sized organisation and, as previously mentioned, it may be that a 300 second time interval for such a general awareness session may be sufficient. Again, only longer term empirical studies of usage and user feedback can confirm this.

As a further example of GAP's scalability, Figure 7.4 shows the effect of increasing the available bandwidth to 128 bytes/sec whilst allowing membership to reach 10000. Such a membership size would be appropriate for a campus or enterprise-wide awareness session, perhaps where each workstation runs a GAP client. It is clear from the graph that, given this bandwidth, even a group size as large as 10000 does not force a time interval greater than 30 seconds thus ensuring that awareness information is updated relatively frequently. Given that even an ethernet-based enterprise network could provide a shareable bandwidth resource of up to 10Mb/s, the reservation of a continuous 128 bytes/s stream for a campus-wide GAP session may not be problematic. As above, only further implementation and trial usage in various network contexts can confirm this.

The final simulation, shown in Figure 7.5, shows the behaviour of GAP where membership is small and bandwidth is maintained at 0.5 bytes/s. Here, the time interval can be seen to correspond to the minimum resend interval (set to 5 seconds) until the membership approaches 14 or 15. The time interval then increases to about 70 seconds when membership approaches 100. As could be seen from Figure 7.4, increasing the allocated bandwidth would have the primary effect of allowing a larger membership before the time interval starts to be significantly larger than the minimum. It also reduces the rate at which the interval subsequently increases.

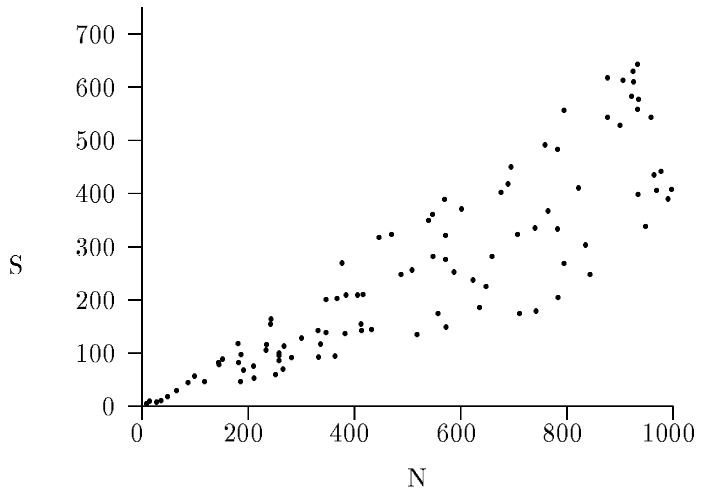

Figure 7.2: GAP resend interval (S seconds) against number of session members (N) where $0 < N < 1000$, bandwidth = 0.5 bytes/s and mean GAP packet size = 240 bytes

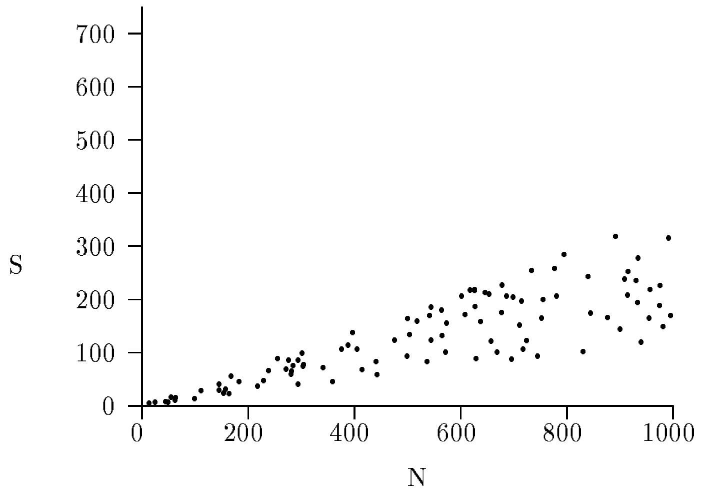

Figure 7.3: GAP resend interval (S seconds) against number of session members (N) where $1 < N < 1000$, bandwidth = 1.0 bytes/s and mean GAP packet size = 240 bytes

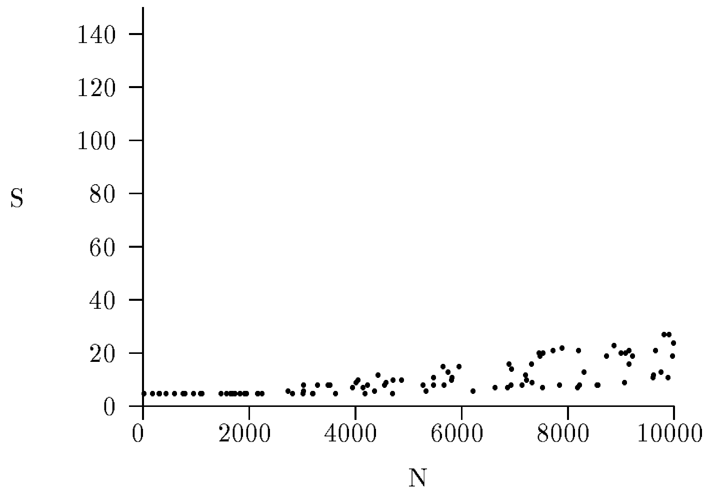

Figure 7.4: GAP resend interval (S seconds) against number of session members (N) where $1 < N < 10000$, bandwidth = 128 bytes/s and mean GAP packet size = 240 bytes

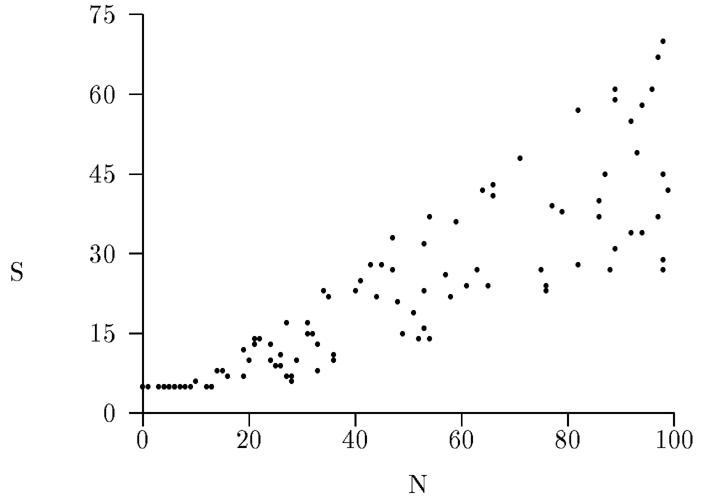

Figure 7.5: GAP resend interval (S seconds) against number of session members (N) where $1 < N < 100$, bandwidth = 0.5 bytes/s and mean GAP packet size = 240 bytes

7.3.2 Predefined User Services

It has been noted elsewhere that it can be useful to predefine a number of user services - examples of this include 'office share', 'glance' and 'video conference' [49]. In this context, *TelePort* provides two predefined user services - a 'glance' and a video conference which can be initiated in one of two ways ('knock' or 'knock and enter'). By providing such predefined user services, it is possible to hide much of the service description from the user, and to remove the need for extensive parsing of that description. In the case of *TelePort*'s 'glance' service for example, the media description is largely redundant because the SDP session attribute field defines the request (see Section 6.8). However such an approach would prevent a SIP client that did not understand the service definition from parsing a 'glance' request and it is for this reason that the SDP authors recommend using implicit mechanisms for defining user services[2]. Currently, the title and session description fields in SDP are used by session creators to provide information on what the session might be expected to be - a meeting, a lecture or a conference - which, together with the other fields, enable an extremely rich yet compact description of a user service. As an example, a glance service can be described as a low bandwidth, send-only video session with a timespan of a few seconds all of which can easily be encoded using SDP.

However, experience with *TelePort* coupled with that from other trials of Media Space systems, suggests that extending the use of SDP from announced sessions to descriptions of personal calls (as is envisaged with SIP) might necessitate the provision of a number of predefined user services, such as glance, videophone or office share, in order to 'bootstrap' the user by providing familiar communications actions. However, once users become familiar with these predefined user services, the ability to manufacture and define their own from the basic media services available is likely to support flexible usage over time. Which predefined services will be necessary in particular contexts and for particular SIP clients such as *TelePort* is an interesting area for future work.

7.3.3 Distinguishing Invitation Types

In providing two ways of initiating an audio/video conference - 'knock' and 'knock and enter' - *TelePort* is an illustration of the advantage of distinguishing between different kinds of invitation. In the first instance, users request participation whilst in the second it is forced. That these two methods of starting an interaction are distinctly different was demonstrated in Chapter 4 and is confirmed by informal observation - users make specific choices about which to use in particular circumstances. It therefore seems likely that an invitation protocol, such as SIP, needs to provide a mechanism by which different kinds of invitation can be defined. Indeed the implementation in *TelePort* of a range of potential actions that are initiated using SIP has necessitated it. Since SIP does not currently support this, *TelePort* uses the SDP session attribute to define invitation types in addition to user services. A cleaner solution would be to redefine SIP to allow an invitation attribute field so that clients that understand particular attributes can act accordingly. As with the SDP session attribute, it is likely that SIP clients will need to be configurable in their interpretation and presen-

[2]M. Handley, email to MMUSIC Working Group mailing list. Archived at ftp://ftp.isi.edu/confctrl/confcrtl.mail.

tation of different attributes. It is currently unclear if it is possible, or even desirable, to attempt to define a set of invitation types. Whilst those defined by *TelePort* namely KNOCK and CONNECT, may be a sufficient basis, different application developers and user groups will have different requirements. It may be that, as with user service descriptions, SIP needs to provide some way in which users can define their own invitation types. Quite how SIP clients should deal with unknown invitation types is, however, an open question.

7.3.4 Awareness Information: 'Pull' vs 'Push'

Approaches to the provision of awareness information can be categorised in two ways:

- Information Pull
- Information Push

In the case of *information pull*, a user who wants information about someone else explicitly requests it from some sort of awareness server or daemon process. Examples of this include the unix **finger**, **ruser** and **who** commands as well as more recently developed examples that use http servers as front ends to 'directory' or location services (cf. [78, 81]). In contrast, *information push* categorises systems such as *TelePort* which do not require explicit action by the receiver of the information because a process associated with each user pushes out information on a regular basis.

Which of these methods is the most efficient largely depends on the context of use, in particular the size of the group involved and the likelihood of any one member of the group needing to see information about any other member. Thus, whilst it would be possible to build a global user location service on a push system such as *TelePort* a more efficient architecture may be a mix of the two so that a person's location and current activity can be *pulled* from a server in much the same way as querying a remote finger daemon. The results of this query could be returned to the enquirer in a form suitable for passing to a SIP client such as *TelePort* or sdr [82]. Such a server might, in turn, maintain up to date information about all the users on that campus by listening on a locally scoped low bandwidth GAP session. A similar architecture has been proposed as part of the MMUSIC User Location service [129] and it may be that a composite of these proposals and GAP can provide an infrastructure for both user location and group awareness on a global scale.

7.4 Summary

In Chapter 4 it was stated that this case study was to act as a vehicle for the investigation of:

- general support for workplace awareness
- issues arising from the use of IP multicast tools and evolving IETF protocols to implement Media Spaces
- the smooth integration of tools other than audio and video into a Media Space

This chapter has described how the usage of the *TelePort* prototype has enabled reflection on these issues.

User feedback from trials of the *TelePort* prototype have raised general issues such as problems that arise when real world cues that might be expected to naturally occur in the user's physical workspace are used in a Media Space system. Thus there can be confusion about the source of the knock sound effect if it is too realistic a noise. Future Media Space systems may therefore need to use sound effects that, whilst evoking concepts from the real world, are different enough to prevent confusion. It is interesting to note that this reflects the finding that highly realistic metaphor based interfaces can cause users severe usability problems because the interface objects are assumed to behave as they would in the real world, when in fact they do not [11].

At the technical level, *TelePort* has provided an excellent vehicle for the implementation and review of a number of communication protocols that are crucial to the IETF's efforts to define a multimedia conferencing architecture for the Internet. In particular *TelePort* is one of only two existent implementations of the SIP protocol and, as this chapter has described, it has provided considerable feedback on the draft proposal. In particular, it has highlighted the need to distinguish between different invitation types and to provide predefined user services. *TelePort* also serves as an example of information push and this chapter has briefly discussed this in the context of current 'awareness servers' and of draft plans for a global user location service for the Internet. The *TelePort* prototype provides an excellent research vehicle for the further investigation of these issues.

Finally, the chapter has provided an analysis of the group awareness protocol used by *TelePort*. It has shown, through the use of simulation, how GAP scales as group membership increases from a few users to thousands. It has also shown that the mechanisms for distributing awareness information provided by GAP can be effective in at least the local area network environment. Future trials of *TelePort* in the wide area are needed to confirm that these simulations are reliable. Such trials, and the implementation of other GAP clients are also essential to determine whether or not GAP as defined in this dissertation can effectively support a range of awareness groups each of which may have different information dissemination requirements.

Chapter 8

Case Study Analysis: An Exercise in Respecification

8.1 Introduction

Chapters 4 to 7 have described the design, implementation and use of the *TelePort* prototype as a case study in design that has been based on the framework developed in Chapters 2 and 3. The introduction to this case study in Chapter 4 set out its primary aims which were to:

- explore the framework through a practical example
- explore methods required by the framework
- show that design using the framework could produce a working system

The discussions in the previous chapter have shown that the *TelePort* prototype is a working, functional system that can be, and has been, used to support awareness between groups of workers who share a common interest. It should therefore be apparent that the case study has achieved its aim of building a functional CSCW system. The remainder of this chapter provides an analysis of, and reflection upon the case study in order to demonstrate the practical implications of the framework for the design process. It therefore shows how the case study explored the framework and demonstrates its utility.

Chapter 2 described a conception of rules and rule-use that has far-reaching implications for the nature of interactive systems, and for the way in which they are implemented. In order to provide a framework that practitioners can use to take account of this alternative view during design, Chapter 3 proposed a characterisation of interactive systems, and of CSCW systems in particular, as the mapping of cues to actions. It was shown that in some systems, this mapping is enforced by an embedded rule-based model, whilst in others the mapping has been left to the users. The chapter showed that it is possible to characterise a range of systems in this way and provided examples such as The Coordinator, where the mapping from cues to actions is enforced by the system; and GroupDesign, where the mappings are left to the users. As Chapter 3 made clear, characterising the systems in this way emphasises the differences between those which enable flexible use by not enforcing the mappings, and those which do not provide flexible use because the mappings are under system control. The discussion of the range of examples provided, coupled with the discussion of the use of these systems in Chapter 2, suggests that these distinctions are reflected in practice and therefore that this conceptualisation is valid.

Previous chapters have suggested that conceptualising the system in this way has proved useful in flagging design issues and subsequently informing implementation decisions at both the user interface and the architectural levels. Whilst recognising that interface and architectural issues are often intimately linked, this chapter discusses each of these in turn.

8.2 Cues and Actions: Characterising Systems

The conceptualisation of systems in terms of cues and actions has shown that any system will have *some* system imposed constraints or rules because the practical nature of design is an articulation of *for whom* and *to do what* [127].As a result systems are designed and built to do certain things and generally for particular groups of users. This design base provides the constraints within which any practitioner works and which, therefore, determine the scope of the system - as users, we would not necessarily expect a system such as *TelePort* to provide word processing functions - such functionality lies outside the basic design constraints. However, within this design 'space' it is also clear that constraints can be made physical or can be left to cultural regulation. Systems can therefore be seen to exist on a continuum from an extreme position where all constraints are encoded, to the other where virtually none are encoded. Examples of the former include The Coordinator [158] and many experimental workflow systems (cf. [1]), whilst examples of the latter include a number of recent realtime collaborative work systems (eg. [105, 62, 153]) and, in fact, most email systems. If systems are considered to lie on this continuum, then a key design question becomes "how much of the mappings should be encoded, and how much should be left to the users?" This is quite clearly a decision on implementation *policy* and as such is highly dependent on the intent behind design. As was argued in previous chapters, designers who want to build flexible systems need to be made aware of when their implementation policy is leading them to encode social rules. Thus designers need to keep asking themselves "is encoding this model/rule going to constrain users? Do I want to do that?"

Thus far, this dissertation has argued that characterising systems in terms of the cues they provide, the actions they afford and the mappings between the two can be used to aid the design of flexible systems. If systems can be characterised in this way, and the dissertation suggests that they can, designers can proceed by making decisions about which of the mappings should be enforced and which should not. However, it is currently extremely unclear what kinds of rules should and should not be embedded in systems for particular contexts and activities although this dissertation argues that systems to support predominantly social behaviour and to support work activities do need to be highly flexible.

Taken to an extreme the principle of not embedding the mappings within the computational system might suggest that no rules or constraints should be built into the system at all. In the context of an awareness system such as that developed as part of this dissertation this may not be problematic although, as was mentioned earlier, users may wish to be able to set system imposed constraints on behaviour in order to ensure privacy. However, there are clearly types of system where embedding rules could be an advantage. An obvious example of such a system is that of safety-critical process control systems where certain rules regarding permissable actions may be required to prevent disasterous operator error. What does seem clear is that the continuum of flexibility can be articulated in terms of the degree to which the mappings between cues and actions are encoded or embedded within the system. Mapping this continuum in these terms may therefore be a fruitful area for future work.

The appraisal, in terms of cues and actions, of the range of systems described in Chapter 3 has also demonstrated that many applications can be characterised *as if* they had been implemented using this framework. In many cases it can be seen that the mappings between the two have become physical constraints on action so that even where users can perceive cues, they are subject to the system (ie. the designer's) interpretation of the meaning of those cues with respect to the model of interaction that is embedded within it. It is quite clear that users faced with these kinds of constraints become extremely frustrated - the computer will not let them do things that they feel are appropriate because their intended actions don't fit the system's model of what they should be doing.

Recent studies of user's experiences with nascent Virtual Reality based CSCW systems have demonstrated precisely this effect [40]. Recent research in many groups interested in such systems has focused on developing more detailed and fine-grained models of interaction in order that they may form the basis for more flexible implementations (cf. [21]). However, the arguments put forward in this dissertation suggest that *this is entirely the wrong approach.* Rather than trying to develop an effective model of the mappings between cues (context) and actions, far more effective and flexible support for multiuser interactions in such systems might be provided by developing more effective models of what cues people use in deciding what to do. These models can then be used to inform the design of VR CSCW systems which support *social* rather than technical regulation of behaviour.

The characterisation of systems in terms of cues and actions also suggests that if a system *is* to enforce the mappings between cues and actions, it needs to access those cues in order to determine the context and so decide what the user is allowed to do. As Suchman made abundantly clear, the vast majority of the cues that humans use to work out the significance of prior responses or of the current situation are completely inaccessible to current computational systems [147, pages 119-132]. Therefore, systems that are intended to provide flexibility and context sensitivity through the application of models based on the interpretation of environmental cues, whatever their nature, are *bound to fail.* This is an extremely strong argument against the idea that 'intelligent' systems of any sort can be built on the basis of these concepts, and it is this which froms the core content of Suchman's book. In the context of this dissertation, these arguments suggest that systems which enforce mappings between cues and behaviour are likely to fail not just because their models are inflexible or partial, but simply *because they are embedded within computational devices.*

8.3 Cues and Actions: Impacting Design

Chapter 3 recommends using the conceptualisation of systems in terms of cues and actions to actually drive design. The premise is that by doing so, the design process will take into account the view of rules described in Chapter 2.

If design is to progress from these concepts then it is clear that some way of generating or eliciting the cues that users might require, and the actions they might deem appropriate is needed. Chapter 4 describes the elicitation of cues and actions for the *TelePort* prototype using fieldwork methods derived from Cognitive Anthropology. Specifically, it proposed that frame analysis, where informants are asked to complete phrases or partial sentences, can produce cues and actions for particular circumstances. The chapter then reported an exercise in the use of frames to generate cues for accessibility based on the state of a person's office door, and their relative social status within the organisation. The chapter therefore serves as an illustration of how design might begin to proceed on the basis of cues and actions.

This exercise not only produced a tangible resource for design in terms of lists of cues that needed to be supported by the system, and actions users might expect, but also provided strong evidence that the approach of enabling users to make the mappings between cues and actions was likely to succeed. For example, it is clear from the results reported in Chapter 4 that people will adjust their behaviour depending on the state of the door and who it belonged to. Furthermore, the elicitation of phrases such as "it was hanging off its hinges" coupled with "so I went to tell a policeman" demonstrates the fundamentally social nature of this regulation. Real transgressions are acted upon in situationally appropriate ways so it may be expected that transferring the ability for users to act in this way to a telecommunication system would, similarly, support social regulation of 'electronic transgressions'.

Whilst the frame analysis approach appears to have derived cues and actions with some success, it is clear that, as a method, it is not comprehensive. Future design and implementation projects that use the conceptualisation of cues and actions might assess the utility of other, more naturalistic, fieldwork methods in the elicitation of cues and actions. It is interesting to note that many of the naturalistic studies of work cited in earlier chapters have tended to concentrate on work processes, presumably because the motivation is often an attempt to automate or 'otherwise improve' them (eg. [34, 87, 28]). It would be interesting to investigate the utility of these approaches given the refocusing from processes onto cues and actions that this dissertation recommends.

8.4 Cues and Actions: Impacting the User Interface

Chapter 5 describes the way in which the results of the frame elicitation served as a resource for the user interface specification and implementation. It makes clear that such elicitation, on its own, is not likely to be sufficient and may be supplemented by resources derived from other studies or from prior experience.

By premising the *TelePort* system on the need to provide cues and actions via the user interface, Chapter 5 shows how a range of system functionality can be specified that is consistent with what the user might expect to be able to do, and which is presented in a familiar way. Thus, the options for action are labelled in ways that are directly derived from the frame elicitation - 'glance', 'knock', 'knock and enter' and so forth. Furthermore, the system functionality defined by these phrases is designed to correspond to the cues that the users might expect to initiate interaction. Thus a 'glance' is associated with a door creaking open and then shut, whilst a 'knock' generates a knocking sound. This functionality is then made available using a menu (in the case of *TelePort*) of possible actions that are *always available for use*. Whilst this point has been made before, it is worth reiterating: Chapters 2 and 3 suggest that the implementation of a flexible system must not embed social rules that might constrain use. Therefore, unlike the CAVECAT and Doors systems, *TelePort* does not disable any menu items - all potential actions are available all the time so that the users can decide what it is appropriate to do. Instead of explicit control *TelePort*, informed by the arguments of previous chapters, implements an implicit warning mechanism so that it warns users when they are about to do something that might effect others. Thus, if a user 'knocks' on the door of someone who is currently in a conference of some sort, it does not stop them from doing so, it merely warns them that this is about to happen. This then is another cue that supports social control - users must decide whether or not it is appropriate to continue with the interruption. A similar mechanism for reminding users that they are about to do something that may go outside the system's model of what is appropriate is described in Dourish et al's report of experience with the constraint based Freeflow workflow tool [53].

Similarly, the cues that are presented in the interface are based on the cues derived from the frame analysis - the state of the door, and the ability to make clear whose door it is - as well as from other sources. In implementing the user interface to display these cues, it became apparent that design choices on how to display the information were almost as influential on the use of the system as the presence or absence of the cues themselves. Thus it is extremely important to explore the designer's assumptions about what cues are best displayed in which ways. Users are well known to attend to system features in many unanticipated ways in order to develop some understanding of what is going on [120]. It therefore seems vital that interfaces built around the provision of cues need to be, in some sense, transparent as well as configurable so that users can access cues that system designers had not anticipated they would require. This suggestion clearly parallels Dourish's work on reflective systems that can present accounts of themselves [51] in order to support tailorability and flexible use. It may be that open systems that are deeply configurable via computational reflection can provide an ideal basis for design based on the concepts of cues and actions since it might allow the user to configure the cues and actions themselves. This is an intriguing area for future work.

8.5 Cues and Actions: Impacting the System Architecture

It became clear at a number of points in the design process that design assumptions were leading to the implementation of social rules as physical constraints. For example, early versions of the system used the information disseminated by the GAP protocol to prevent users from making SIP requests to other members who were currently involved in a conference. At this time, *TelePort* was implemented as a state-based system which could be either BUSY or IDLE and any *TelePort* client that was in the BUSY state refused to accept any SIP connections. On reflection, it was apparent that this was a clear instance of a simplified social rule - that interruptions should not be allowed - being embedded as a physical constraint in the system. There is clearly no justification for implementing this rule (ie. policy) in the *TelePort* system - interruptions are very often necessary and it is not for the system to make decisions in this respect. As a result, the system architecture was radically altered to one that was stateless. Now, a *TelePort* client would not refuse a SIP request, relying instead on the user knowing that they were about to interrupt something, and having decided that this was warranted. As a result, *TelePort* needed to be able to display who was currently in a conference with whom, rather than merely that the client was 'busy'. This could only be achieved by extending the GAP protocol to enable it to send reports of all the conferences in which it was currently active. As a result, the ca= fields described in Appendix B were added to the GAP specification. Thus, as a result of considering a design decision in the light of the arguments presented in earlier chapters of this dissertation, the architecture of the *TelePort* prototype, and the requirements for the GAP protocol were radically altered. Furthermore, the change in the *TelePort* architecture made the system far easier to implement because it no longer had to behave in accordance with specific system states, and did not require that its record of the state of another client be necessarily accurate.

The GAP protocol was also influenced by the likelihood that future designers and users would want to be able to add additional cues (and actions) to their group awareness tools. Thus, GAP is designed to be openly extensible through the use of as many i= fields as necessary, and by the stipulation that GAP clients should ignore any fields whose contents they cannot parse. So, whilst GAP defines an initial core set of cues that might be useful for providing awareness, it makes no claims that these are sufficient or comprehensive. Further, GAP makes no stipulations about what information should be included (ie. what cues should be sent) since this is a policy decision that can be made during design, or by the users. In the latter case, the arguments put forward in earlier chapters suggest that users should be able to modify and hence control what information is sent. It is worth noting that users who decide not to send particular information do not necessarily impair the utility of the system because even the non-dissemination is a cue in itself....

In addition, the use of the RTCP algorithm means that even if a GAP tool requires extra fields, the awareness session will continue to scale its network usage because the algorithm takes account of the size of the packets being sent. Finally, since the bandwidth allocated to a GAP session can be defined on a per-session basis, and the value of that allocation automatically alters the rate at which GAP information is disseminated, a GAP session that requires information 'freshness' can achieve this by requesting higher bandwidth. These then are excellent examples of how designing on the basis of cues and actions can influence a system at all levels, not just at the user interface.

Other design decisions that were influenced by the concepts of cues and actions, and of not enforcing the mappings between the two were:

- Connection termination: In the Doors system, the model had been one of 'whoever initiated the connection should close it' which was based on an analogy of walking into and out of offices via the door. However, usage of the Doors system suggested that this caused considerable confusion about who could and could not terminate the connection [11]. As a result, *TelePort* makes no assumptions about who should terminate the call and allows each user to do so if they wish.

- Whiteboard requests: *TelePort* allows a whiteboard service to be initiated at any point, irrespective of whether or not the participants are already in a conference. Thus, *TelePort* makes no assumptions about when a whiteboard can be used, relying on the users to have created the context for its use for themselves.

Currently, if a user who is running two *TelePort* clients, and so is a member of 2 sessions, requests a conference with a person in group #2, members of group #1 would not become aware of this. A group #1 member could initiate a 'walk in' service with the member in both sessions who is not, apparently, in conference only to find that this action creates an interruption. Thus it would be important for users to be able to see with whom another session member is currently interacting, and what the nature of that interaction is, appears crucial in enabling them to decide what appropriate actions might be. One possible solution to this problem would be for multiple *TelePort* clients to adopt a local 'conference bus' mechanism similar to that used by a number of the media tools mentioned earlier to coordinate access to hardware resources [110]. In this model, *TelePort* clients on the same workstation would share a 'currently active' list so that each of these clients will be sending a list of all the current activity on that workstation. However, this also raises difficult privacy issues because a member of group #1 (Alice) may request a conference with another member of group #1 (Bob) and the existence of that conference may then be announced to all the awareness sessions of which Alice is a member, without Bob's knowledge. Therefore, it is extremely unclear what should be displayed to group #1 about the activity of members of both group #1 and group #2. This is an interesting avenue of research because it is a problem that needs to be overcome if either GAP-based or awareness tools in general are to realise their potential [38].

Conceiving of, designing, and implementing the prototype in terms of cues and actions has prompted the inspection of *TelePort*'s architecture for assumptions that *have* been embedded, and the provision of a rationale where this is the case. One such is the decision to force the video tools to start sending video and to force the audio tools to unmute their microphone as soon as a connection has been made. The decision to implement the assumption that users would prefer immediate audio and video connectivity was based on literature suggesting that asymmetric audio and video connectivity can cause users to experience serious problems in coordinating the start of their interaction [91]. It may be that recipients of connection requests do not wish their audio channel to be automatically opened (for privacy reasons) in which case this implementation may need to be rethought. On the other hand, the warning that *TelePort* provides by way of sound effects may mean that this is not necessary because the imminence of an interruption is apparent and so users can adjust their behaviour accordingly. Only lengthy user evaluation could provide evidence for this. Other embedded assumptions include the nature of the telecommunications services available from the actions menu. For example, it is assumed that a glance should be unidirectional even though the occupier of an office often sees a person who peeks in; and that audio-only calls are not necessary when a combined video and audio call is available.

Perhaps the most important embedded constraint is that requests for video connections force the abortion of any current use of the video frame-grabber. This is not a design 'feature' made out of choice by *TelePort* but is forced by the inability of most current video capture cards to support more than one software codec at a time. *TelePort*'s policy of not preventing interruptions ought to mean that a particular user (Alice, say) can glance at another (Bob) who is currently in a videoconference with someone else (Charles) without affecting Bob and Charles' ability to see each other. However, this is impossible to achieve with current video capture hardware and software codecs which insist on exclusive access by a particular capture process. As a result, Bob and Charles's video is interrupted whilst Alice is glancing because the video output cannot be replicated. Two solutions to this are possible - one is to enable a single codec to replicate its output, the second is to allow two or more codecs to share simultaneous access. It appears that current hardware and software codec architectures assume that access is required by only one process at a time and it seems equally clear that, in the case of multimedia telecommunications systems at least, this is not necessarily true.

8.6 Cues and Actions: Tailorability and Unanticipated Use

It was noted in Chapter 7 that users quite quickly noticed ways in which they could tailor *TelePort* to their own taste. In particular, some wanted to be able to introduce new cues into the system by altering the way in which their door was represented. Given the effect that personal status has on what is deemed to be appropriate behaviour (cf. Chapter 4), it might be hypothesised that supporting such personalisation could make the social control of access even more effective.

In addition, the ability to add new and different cues to the system is vital if it is to support the ever changing nature of social circumstances and cultural norms. As Chapter 2 emphasised, cultural norms are fundamentally situated in time - they will change as members of a culture change their views on what is or is not appropriate behaviour. Such changes may be imposed from outside the group, such as by intentional organisational change, or be emergent through users finding unanticipated uses for the system [120], or simply through cultural evolution. Whatever the cause of the changes, it should be clear that any system which encodes the mappings between cues and actions cannot allow for cultural change without significant re-engineering.

In contrast, a system that does not attempt to map cues to actions is neutral with respect to cultural change. Provided that the users continue to attend to the same cues, and require the same actions, the *significance* of each does not matter to the system. Thus users may completely reverse their view of what is and is not appropriate behaviour or they may attach new significances to the cues. The point is that it is the users who are attaching those significances and so it is they, not the system, which copes with the complexity of this change.

In the case where users do want to add new cues and new actions, it is clear that systems need to be able to support them in doing so. As was made clear in previous sections of this chapter, *TelePort* provides a limited degree of tailorability - it supports the addition of new cues through the extendability of the group awareness protocol but does not, as yet, provide any way for these additional cues to be displayed in the user interface. Currently *TelePort* does not enable users to add new user services in order to define new actions. Whilst providing some sort of application programming interface (API) to systems of this sort may enable sufficient scope for tailorability, recent work suggests that an API may not suffice because it constrains the potential configurability according to the designer's assumptions about what may be needed [53]. The whole area of tailorable systems based on the concepts of cues and actions is therefore an obvious avenue for further research.

These aspects of the design recommendations made by this thesis have far-reaching implications. For example, systems built in this way may be easily adaptable to a range of customer requirements through the re-configuration of cues and actions. Systems can be designed to suit a particular culture and can then, if necessary be carefully redesigned if new cues and actions become necessary. Further, systems built in this way may turn out to be flexible enough to avoid becoming legacy systems when an organisation's practices change, since the practices of use are not rigidly enforced.

8.7 Cues and Actions: Privacy, Social Control and Accountability

In current design practice the electronic 'world' is commonly treated as if it were distinct from the physical 'world'. By this logic, the electronic world therefore needs to have its own regulatory policies to ensure that users do not abuse it (or each other) and it has been argued in previous chapters that this leads to the implementation of social rules. However, it seems that this distinction between 'real' and 'electronic' is not made by users in practice - a number of previous studies have demonstrated that people's behaviour in and through a media space is as available for social control as their real world behaviour (eg. [64, 50]). People feel as acutely aware of the danger of invading another's privacy in a media space as they would in a physical space. As a result, cultures of use spontaneously develop around communication systems. Nowhere is this more apparent than in the online communities who use bulletin boards, usenet news and MUDs. These systems, which have virtually no imposed controls on behaviour, have been extremely successful and one of the primary reasons for this has been shown to be the way in which their users spontaneously develop a culture of accepted usage around the cues that the technology provides [126].

This dissertation suggests that system design can use this phenomenon to design more flexible, and more useful systems through intentional support for social self-regulation. Thus systems that provide users with cues and actions encourage them to see actions through or in the system as influenced by *fundamentally the same* factors as those in the physical world so that there is no difference between 'actions in the system' and 'actions in the world' in terms of their social significance. As a result, transgression in the electronic medium becomes a real social issue which is subject to real social or organisational regulatory pressures. Once this is achieved, the requirement for technical regulation recedes because social, organisational or legal pressures take over.

TelePort has also, not surprisingly, raised questions about privacy and the control of awareness information. Bellotti and Sellen have suggested a framework for use by designers in this context which emphasises that in order to feel comfortable with Media Space systems, users need to be able to control what information is made available and to whom [20][1]. As discussed, *TelePort* provides a suitable vehicle for developing user interfaces to explore this framework. One such strategy, which was suggested in Section 2.4, is an implementation based on *effort* and *accountability*. Thus, users may choose to make it harder for others to access particular information about them and, if a user does make the effort to do so, that user can be held accountable for his/her actions at a social and, if necessary, legal level. Repeated 'electronic' transgressions of socially accepted behaviour are therefore punishable in exactly the same ways as would physical transgressions. This strategy suggests that access to information should not be anonymous since anonymity confounds accountability - users should always know, or be able to find out, who accessed information about them and when.

[1]See also [38] for a wide-ranging review of privacy issues in media space systems

8.8 Summary

This chapter has discussed the way in which the case study has explored the design framework outlined in Chapter 3. It has reflected on the design methods that were developed as a result of the recommendation to view systems in terms of 'cues' and 'actions' and, together with Chapter 7, has demonstrated that this conceptualisation can produce functional, working CSCW systems. It has therefore moved this exercise in technomethodology a step closer to its goal since it has shown that taking account of the ethnomethodological view of rules, and rule use, during design can be a successful implementation strategy. It is therefore an affirmation that 'doing technomethodology' can work and, in a sense, provides the beginnings of a technomethodological 'design cookbook' of conceptual tools and practical methods.

The chapter has also shown that this view can have a fundamental influence on the design of an interactive system because it makes recommendations about all aspects of an implementation from the network protocol level to that of the user interface. In common with other recent research on configurable systems, this view suggests that the traditional separation of user interface and system architecture into independent units cannot suffice because architectural design decisions can have just as great an impact on the flexibility of the system.

However, this chapter has also shown that viewing systems in terms of cues, actions, and the mappings between the two can provide novel approaches to a number of current research issues including support for system tailorability, support for cultural change over time and privacy and control in information spaces.

Part IV

Conclusions

Chapter 9

In Summary

9.1 Learning from Ethnomethodology

This dissertation began by outlining a number of possible relationships between system design and the sociological subdiscipline of ethnomethodology. In particular it focused on the potential for HCI (and CSCW) to learn from one of the key methodological recommendations of ethnomethodology, namely for a discipline to re-examine the taken-for-granted assumptions and concepts upon which its theory and practice are based. If these assumptions turn out to be problematic, then ethnomethodology recommends respecifying them in the light of detailed studies of the phenomena in question.

In the case of HCI and CSCW, this leads to the re-examination of foundational concepts such as system transparency, process, representation, abstraction, accountability and generalisation. Chapter 1 noted that, seen as a research program, the goal of this form of *technomethodology* is to ask if certain basic assumptions in HCI are misconceived and whether or not developing alternative views based on studies of what is really going on can lead to the implementation of demonstrably better systems. The chapter noted that a number of conceptual and practical bridges need to be in place before technomethodology can achieve this goal:

- **Just *how can* practical system design learn from ethnomethodology?**

  Chapter 2 introduced an examination of the nature of rules and rule use in the design of systems that are too inflexible to support real use. An alternative view was developed which drew on detailed studies of how people interact and achieve apparently rule-based action, and from recent critiques of inflexible systems in the CSCW literature.

- **What sort of *design frameworks* might support this respecification?**

  Chapter 3 builds on this re-specification by outlining an approach to design which characterises systems in terms of cues, actions and the mappings between the two. In essence, it was suggested that users are able to perceive information about other participants, and about the system, by means of *cues*, whilst the system provides a range of *actions* - the system functionality. The mapping between the two, that is determining what users may do next can then be viewed as under a mixture of user and system control - in some cases the user decides, in others it is the system.

- **What sort of *requirements* capture methods might such frameworks demand?**

  Chapter 4 reported a study that used methods derived from Cognitive Anthropology to elicit the cues that workers use in deciding how to communicate with colleagues in an office-based environment. Further, it has used these methods to determine what actions users might expect to be supported by a system that provides management of multimedia conferencing calls.

- **How can the output of such methods be *incorporated* into the design and implementation of a functional system?**

Chapters 5 and 6 demonstrated how the cues and actions derived using these methods when combined with the principle of not embedding models of the mappings between cues and actions can influence all levels of system design and implementation. Finally Chapter 8 demonstrates that the result is a functional, usable system.

Thus, as an exercise in technomethodology, this dissertation has achieved its goals. It has

- identified a foundational concept,
- re-specified that concept,
- examined the implications of this respecification for design,
- proposed a design framework which can enable practitioners to build systems that take account of this respecification,
- described methods that this design framework requires,
- explored these methods, and the design framework, through the implementation of a proof-of-concept prototype.

As a result, the conceptual bridges that were noted as necessary in Chapter 1 are now in place. It has been shown that technomethodological respecification is possible, can lead to new design insights and can impact design practice in useful ways. Real, functional interactive systems can be built with these respecifications in mind.

What remains to be seen is whether or not such systems are, in some sense, *better*. There is powerful evidence that systems built in this way will prove to be better able to support real work than current systems that were not. However, only long term user evaluation of 'technomethodological systems' can provide indicators of their true value. The project of technomethodology is now in a position to begin this last stage - it is hoped that the path mapped out by this dissertation can encourage others to follow, re-examining and re-using the design framework as they do so.

9.2 Future Work

This dissertation has highlighted a number of potentially interesting avenues of further work, not only from the respecification of rules, but also from experiences with the group awareness prototype.

To concentrate firstly on the implications of the respecification of rules and rule use, the dissertation suggests that attention should be given to:

Work Process Tools - whether the work involved is the asynchronous editing of a particular document or the achievement of day-to-day tasks in general, it seems clear that supporting flexible, situated action is vital. It would therefore be extremely interesting to pursue Beck and Bellotti's design recommendations for collaborative writing and to implement systems that can provide *resources for work* rather than *processes of work* [18]. Recent work such as the MILAN Conversation Model [46] and subsequent system development [5], Oval [107] and Freeflow [23] are all examples of this reconceptualisation .

Privacy and Control in Shared File Systems - it may be that designing such systems in terms of cues for privacy and mechanisms of accountability can remove the need for pervasive technical access controls. In the context of matrix work teams and the notion of virtual organisations where membership of teams may vary on a day-by-day basis, a flexible approach is clearly required [14] - and this dissertation has argued that technical controls cannot provide it. Instead a focus on cues and on effort may provide an alternative. As Bellotti notes in a recent paper, social control can provide this flexibility if systems are suitably engineered [19]. As yet few such systems exist and there appear to be no user studies of their effectiveness.

Flexible Systems and User Tailorability - it was argued that systems that do not encode social rules are substantially more flexible than those that do because they can allow for the inevitable changes in culturally accepted practices over time, a factor which most current CSCW system implementation ignores. Further, if users are able to add their own cues and actions to a system then the flexibility is further enhanced. The development of user-tailorable systems in terms of cues and actions, and studies of their utility may be extremely worthwhile as might exploring the affect that personalisation of an interface can have on the effectiveness of social controls.

The Trade-Off between Social and System Control - it was noted that there will always be certain system imposed constraints on users actions - if only because systems are designed to do particular things rather than everything. It would be interesting to examine the trade-off between system imposed constraints and flexibility through the framework of cues and actions in order to attain some idea of the appropriateness of each in different contexts. For example, should a safety-critical system always impose system control, and would such a strategy actually be successful given that there may be quite legitimate reasons for an operator to over-ride system constraints?

The Use of Field Studies - this dissertation has argued that refocusing design from processes onto cues and actions might produce better systems. It would be interesting to explore the utility of naturalistic fieldwork methods (such as those found in ethnography and ethnomethodology) in the elicitation of cues and actions as a resource for design.

Secondly, the issues raised by the use of the prototype, which might prove an extremely useful research vehicle for their exploration:

Privacy of Awareness Information - controlling what people can find out about each other in different contexts is the crux of designing for privacy. In the context of a global user location service, users may not wish certain information to made available to the general public but in the context of a local, focused group of colleagues, it may be useful to 'publish' more. There have been no user based studies of the affect that such awareness can have on people's perceptions and requirements for privacy and this would seem to merit considerable attention since without it, the multimedia telecommunications 'revolution' may never get far beyond the current telephone metaphor of binary access with no account taken of social factors [38].

Information Push vs Pull - the effectiveness of push vs pull for the distribution of information seems to have been little studied. An exploration of the viability of a global Internet based location service based on local 'push' GAP tools (such as TelePort), campus-based location demons and public 'pull' clients (such as World Wide Web browsers) would be of interest in this regard.

User Services and Invitation Types in MultiMedia Telecommunications - it was suggested that users will need to be presented with a set of given user services that have been designed for their context. It is unclear at present what these might be (although the Media Space work is providing some early indications for office workers) but an exploration of different communication and invitation types in different contexts of work may offer telecommunications companies a rich resource for the presentation of multimedia services and hence product differentiation.

Behaviour of the GAP protocol - whilst trial use and extensive simulation of the protocol have provided evidence of its utility and scalability, longer term and more wide-spread usage is required before its value can truly be assessed.

There is obviously much work to be done...!

Part V

Bibliography and Appendices

Bibliography

[1] K.R. Abbott and S.K. Sarin. Experiences with workflow management: Issues for the next generation. In cscw94 [44], pages 113–120.

[2] M.H. Agar. *The Professional Stranger: An Informal Introduction to Ethnology.* Academic Press, 1980.

[3] M.H. Agar. *Speaking of Ethnography: Qualitative Research Methods Vol. 2.* Sage, London, 1988.

[4] M.H. Agar and J.R. Hobbs. How to grow schemata out of interviews. In J.W.D. Dougherty, editor, *Directions In Cognitive Anthropology.* University of Illinois Press, 1985.

[5] A. Agostini, G. De Michelis, M.A. Grasso, W. Prinz, and A. Syri. Contexts, work processes, and workspaces. *Computer Supported Cooperative Work*, 5:223–250, 1996.

[6] P.E. Agre. Conceptions of the user in computer systems design. In P.J. Thomas, editor, *The Social and Interactional Dimensions of Human-Computer Interfaces*, pages 67–106. Cambridge University Press, 1995.

[7] P.E. Agre. From high tech to human tech: Empowerment, measurement, and social studies of computing. *Computer Supported Cooperative Work*, 3:167–195, 1995.

[8] B. Anderson. Teleport - a group awareness tool. Online Resource: http://pipkin.lut.ac.uk/~ben/PHD/teleport.html.

[9] B. Anderson. Gap: A group awareness protocol. Internet Draft: draft-ietf-mmusic-anderson-gap-1.0.ps, September 1996. Work in Progress.

[10] B Anderson and J.L. Alty. Everyday theories, cognitive anthropology and user-centred system design. In *People and Computers X, HCI '95*, pages 121–135. Cambridge University Press, Cambridge, August 1995.

[11] B Anderson, M Smyth, R Knott, M Bergan, J Bergan, and J.L. Alty. Minimising conceptual baggage: Making choices about metaphor. In G. Cockton, D. Draper, and G. Weir, editors, *People and Computers IX, HCI '94*. Cambridge University Press, Cambridge, August 1994.

[12] R.J. Anderson. Representations and requirements: The value of ethnography in system design. *Human Computer Interaction*, 9:151–182, 1994.

[13] R.J. Anderson, C.C. Heath, P. Luff, and T.P. Moran. The social and the cognitive in human computer interaction. *International Journal of Man-Machine Studies*, 38:999–1016, 1993.

[14] L. Bannon. From human factors to human actors: The role of psychology and human-computer interaction studies in system design. In J. Greenbaum and M. Kyng, editors, *Design at Work: Cooperative Design of Computer Systems*, pages 25–44. Lawrence Erlbaum Associates, 1991.

[15] L. Bannon and J. Hughes. The context of cscw. In K. Schmidt, editor, *Developing CSCW Systems: Design Concepts. Report of COST14 'CoTech' Working Group 4 (1991-92)*, pages 9–36. 1993.

[16] L. Bannon, M. Robinson, and K Schmidt, editors. *Proceedings of the 2nd European Conference on Computer-Supported Cooperative Work - ECSCW '91*. Kluwer Academic Publishers, London, September 1991.

[17] M. Beaudouin-Lafon and A. Karsenty. Transparency and awareness in a real-time groupware system. In *Proceedings of UIST '90*, pages 171–180. New York: ACM Press, November 1992.

[18] E Beck and V. Bellotti. Informed opportunism as strategy: Supporting coordination in distributed collaborative writing. In ecscw93 [58], pages 233–248.

[19] V. Bellotti. What you don't know can hurt you: Privacy in collaborative computing. In cscw96 [45]. To Appear.

[20] V. Bellotti and A. Sellen. Design for privacy in ubiquitous computing environments. In ecscw93 [58], pages 77–92.

[21] S.D. Benford. Requirements of activity management. In J.M. Bowers and S.D. Benford, editors, *Studies in Computer Supported Co-operative Work: Theory, Practice and Design*, pages 285–297. North-Holland, Amsterdam, 1991.

[22] D. Bentley, T. Rodden, P. Sawyer, and I. Somerville. Architectural support for cooperative multi-user interfaces. *IEEE Computer*, ?(?):???, 1994.

[23] R. Bentley and P. Dourish. Medium versus mechanism: Supporting collaboration through customisation. In ecscw95 [59], pages 133–148.

[24] M. Bittner. The concept of organisation. In R. Turner, editor, *Ethnomethodology*, pages 69–81. Penguin, 1974.

[25] J. Blomberg, J. Giacomi, A. Mosher, and P. Swenton-Wall. Ethnographic field methods and their relation to design. In Schuler and Namioka [130], pages 123–155.

[26] C Bormann, J. Ott, and C. Reichert. Simple conference control protocol. IETF Internet Draft: draft-bormann-mmusic-sccp-00-pre-0.txt, February 1996. Work in Progress.

[27] J. Bowers. The work to make a network work: Studying cscw in action. In cscw94 [44], pages 287–298.

[28] J. Bowers, G. Button, and W.W. Sharrock. Workflow from within and without. In ecscw95 [59], pages ?–?

[29] J. Bowers and J. Churcher. Local and global structuring of computer mediated comunication: Developing linguistic perspectives on cscw in cosmos. In L. Suchman, editor, *Proceedings of ACM SIGCHI CSCW 88*, pages 125–139, September 1988.

[30] J. Bowers and T. Todden. Exploding the interface: Experiences of a cscw network. In interchi93 [100], pages 255–262. Amsterdam, Nl.

[31] C.V. Bullen and J.L. Bennett. Groupware in practice: An interpretation of work experiences. In C. Dunlop and R. Kling, editors, *Computerization and Controversy*, pages 257–287. New York: Academic Press, 1991.

[32] G. Button, editor. *Technology in Working Order: Studies of work, interaction and technology.* Routledge, London, 1993.

[33] G. Button and P. Dourish. Technomethodology: Paradoxes and possibilities. In *Proceedings of ACM SIGCHI CHI'96*, pages 19–26, 1996.

[34] G. Button and R. Harper. Taking the organisation into accounts. In Button [32], pages 98–107.

[35] G. Button and W.W. Sharrock. Occasioned practices in the work of software engineers. In *Requirements Engineering: Social and Technical Issues* [103].

[36] *ACM Conference on Human Factors in Computing Systems, CHI'91.* ACM Press, New York, April 1991.

[37] *ACM Conference on Human Factors in Computing Systems, CHI'92.* ACM Press, New York, May 1992. Moneterey, Ca.

[38] A. Clement. Considering privacy in the development of multi-media communications. *Computer Supported Cooperative Work*, 2:67–88, 1994.

[39] P. Collett. *Social Rules and Social Behaviour.* Oxford: Basil Blackwell, 1975.

[40] C. Condon. The computer won't let me: Cooperation, conflict and the ownership of information. In S. Easterbrook, editor, *CSCW: Cooperation or Conflict*, pages 171–185. Springer Verlag, Berlin, 1993.

[41] C Cool, R.S. Fish, R.E. Kraut, and C.M. Lowery. Iterative design of video communication systems. In cscw92 [43], pages 25–32.

[42] S. K. Crad, T.P Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* London: Lawrence Erlbaum Associates, 1983.

[43] *ACM Conference on Computer Supported Co-operative Work, CSCW '92.* ACM Press, New York., November 1992.

[44] *ACM Conference on Computer Supported Co-operative Work, CSCW '94.* ACM Press, New York., November 1994.

[45] *ACM Conference on Computer Supported Co-operative Work, CSCW '96.* ACM Press, New York., November 1996.

[46] G. De Michelis and M.A. Grasso. Situating conversations within the language/action perspective: The milan conversation model. In cscw94 [44], pages 89–100.

[47] S Deering. Host extensions for ip multicasting. Master's thesis, Stanford University, 1987.

[48] J.D. Douglas, editor. *Understanding Everyday Life.* Routledge and Kegan Paul, London, 1970.

[49] P. Dourish. Godard: A flexible architecture for av services in a media space. Technical report, Rank Xerox EuroPARC, Cambridge U.K, 1991.

[50] P. Dourish. Culture and control in a media space. In ecscw93 [58], pages 125–137.

[51] P. Dourish. Accounting for system behaviour: Representation, reflection and resourceful action. In *Computers in Context, CIC '95*, pages 125–137, August 1995.

[52] P. Dourish. Developing a reflective model of collaborative systems. *ACM Transactions on Computer-Human Interaction*, ?(?):?, 1995.

[53] P. Dourish. *Open Implementations and Flexibility in CSCW Toolkits.* PhD thesis, Dept Computer Science, University College, London, 1996.

[54] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In cscw92 [43], pages 107–114.

[55] P. Dourish and S. Bly. Portholes: Supporting awareness in a distributed work group. In chi92 [37], pages 541–547. Moneterey, Ca.

[56] P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: Mediating between representation and action in workflow systems. In cscw96 [45]. To Appear.

[57] E.A. Dykstra and R.P. Carasik. Structure and support in cooperative environments: the amsterdam conversation environment. In S. Greenberg, editor, *Computer Supported Cooperative Work and Groupware*, pages 295–310. New York: Academic Press, 1991.

[58] *Proceedings of the third European Conference on Computer-Supported Co-operative Work - ECSCW '93.* Kluwer Academic Publishers, London, September 1993.

[59] *Proceedings of the European Conference on Computer-Supported Cooperative Work - ECSCW '95.* Kluwer Academic Publishers, London, 1995.

[60] W.K. Edwards. Session management for collaborative applications. In cscw94 [44], pages 323–330.

[61] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Design and use of a group editor. In G. Cockton, editor, *Engineering for Human Computer Interaction*, pages 13–25. Amsterdam: North Holland, 1990.

[62] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[63] R. Fish, R. Kraut, M. Leland, and M. Cohen. Quilt: A collaborative tool for co-operative writing. In *Proc. of the ACM Conference on Office Information Systems*, pages 30–37. ACM Press, New York, March 1988.

[64] R.S. Fish, R.E. Kraut, R.W. Root, and R.E. Rice. Evaluating video as a technology for informal communication. In chi92 [37]. Moneterey, Ca.

[65] F. Flores, M. Graves, B. Hartfield, and T. Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, 1988.

[66] S. Floyd, V. Jacobson, S. McCanne, Ching-Gung. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of ACM SIGCOMM '95.* ACM Press, 1995.

[67] C. O. Frake. Note on queries in ethnography. *American Anthropologist*, 66(3):132–145, 1964.

[68] D. Frohlich and P. Luff. Applying the technology of conversation to the technology for conversation. In *Computers and Conversation* [106].

[69] H. Garfinkel. *Studies in Ethnomethodology.* Prentice-Hall, London, 1967.

[70] H. Garfinkel. *Ethnomethodological Studies of Work.* Routledge and Kegan Paul, London, 1986.

[71] L. Gasser. The integration of computing and routine work. *ACM Transactions on Office Information Systems*, 4:205–225, 1986.

[72] W. Gaver, T. Moran, A. MacLean, L. Lovstrand, P. Dourish, K. Carter, and W. Buxton. Realising a video environment: Europarc's rave system. In chi92 [37], pages 27–35. Moneterey, Ca.

[73] W. Gaver, A. Sellen, C Heath, and P. Luff. One is not enough: Multiple views in a media space. In interchi93 [100], pages 335–341. Amsterdam, Nl.

[74] E Goffman. *Behaviour in Public Places: Notes on the Organisation of Gatherings.* Free Press, New York, 1963.

[75] W. Goodenough. *Cultural Anthropology and Linguistics*, volume 9 of *Georgetown University Monograph Series on Language and Linguistics.* Georgetown University Press, 1957.

[76] D Greatbatch, P. Luff, C Heath, and P. Campion. Interpersonal communication and human-computer interaction: An examination of the use of computers in medical consultations. *Interacting with Computers*, 5:193–216, 1993.

[77] S Greenberg. Personalizable groupware: Accomodating individual roles and group differences. In *Proc. ECSCW '91*, pages 17–32. Kluwer Press, Amsterdam, 1991.

[78] S Greenberg. Peepholes: Low cost awareness of one's community. In *ACM SIGCHI Conference Companion, CHI '96*, pages 206–207. ACM, New York, 1996.

[79] I Greif and S Sarin. Data sharing in group work. *ACM Transactions on Office Information Systems*, 7(2):187–211, 1987.

[80] J. Grudin. The computer reaches out: The historical continuity of interface design. In *Proc. CHI '90*, pages 261–268. ACM Press, New York, April 1990.

[81] C. Gutwin, S. Greenberg, and M. Roseman. Supporting awareness of others in groupware. In *ACM SIGCHI Conference Companion, CHI '96*, page 205. ACM, New York, 1996.

[82] M Handley. Sdr: Session description tool. ftp://cs.ucl.ac.uk/mice/sdr/. Public Software Release.

[83] M. Handley, J. Crowcroft, and C. Bormann. The internet multimedia conferencing architecture. IETF-MMUSIC-INTERNET-DRAFT, February 1996. Work in progress.

[84] M. Handley and V. Jacobson. Sdp: Session description protocol (draft 02.1). IETF-MMUSIC-INTERNET-DRAFT, November 1995. Work in progress.

[85] M. Handley and E. Schooler. Session invitation protocol. IETF-MMUSIC-INTERNET-DRAFT, February 1996. Work in progress.

[86] M. Handley, I. Wakeman, and J. Crowcroft. The conference control channel protocol: A scalable base for building conference control applications. In *SIGCOMM '95*, 1995.

[87] R.H.R. Harper and J.A. Hughes. 'what a f-ing system! send 'em all to the same place a expect us to stop 'em hitting': Making technology work in air traffic control. In Button [32], chapter 7, pages 127–144.

[88] J. Hartland. The use of 'intelligent' machines for electrocardiograph communication. In Button [32], pages 55–80.

[89] C Heath and P Luff. Disembodied conduct: Communication through video in a multimedia office environment. In chi91 [36], pages 99–103.

[90] C Heath and P Luff. Collaboration and control: Crisis management and multimedia technology in london underground control rooms. *Computer Supported Cooperative Work*, 1(1-2):69–95, 1992.

[91] C Heath and P Luff. Media space and communicative asymmetries: Preliminary observations of video-mediated interaction. *Human Computer Interaction*, 7(3):315–346, 1992.

[92] C. Heath and P. Luff. System use and social organisation: Observations on human-computer interaction in an architectural practice. In Button [32], pages 184–210.

[93] C. Heath, P. Luff, and A. Sellen. Rethinking media space: The need for flexible access in video-mediated communication. In *Proc. CSCW 93*, page ?? ACM Press, New York., 1993.

[94] J. Heritage. *Garfinkel and Ethnomethodology.* Cambridge: Polity Press, 1984.

[95] D.R. Hipp. Embedded tk. Online Tcl/Tk archive resource: ftp://src.ic.ac.uk/packages/.

[96] K. Holtzblatt and S. Jones. Contextual inquiry: A participatory technique for system design. In Schuler and Namioka [130], pages 177–219.

[97] J. Hughes, V. King, T. Rodden, and H. Andersen. Moving out of the control room: Ethnography in system design. In cscw94 [44], pages 429–439.

[98] J. Hughes, I. Somerville, R. Bentley, and D. Randall. Designing with ethnography: making work visible. *Interacting with Computers*, 5(2):239–253, 1993.

[99] J.A. Hughes, D. Randall, and D. Shapiro. Faltering from ethnography to design. In cscw92 [43], pages 115–122.

[100] *Proceedings of ACM INTERCHI '93.* ACM Press, April 1993. Amsterdam, Nl.

[101] H. Ishi, M. Kobayashi, and K. Arita. Iterative design of seamless collaboration media: From teamworkstation to clearboard. *Communications of the ACM*, 37(8):83–97, 1994.

[102] H. Ishi and M Ohkubo. Message driven groupware design based on an office procedure model. *Journal of Information Processing*, 14(2):184–191, 1990.

[103] M. Jirotka and J.A. Goguen. *Requirements Engineering: Social and Technical Issues.* Academic Press, 1994.

[104] S. M Kaplan, W.J Tolone, D.P Bogia, and C. Bignoli. Flexible, active support for collaborative work with conversationbuilder. In cscw92 [43], pages 378–385.

[105] L. Killey. Shredit 1.0: A shared editor for apple macintosh. user's guide and technical description. Technical report, Cognitve Science and Machine Intelligence Laboratory, University of Michigan, 1990. unpublished.

[106] P. Luff, N Gilbert, and D. Frohlich. *Computers and Conversation.* London: Academic Press, 1990.

[107] Y. Malone, K-Y. Lai, and C. Fry. Experiments with oval: A radically tailorable tool for cooperative work. In cscw92 [43], pages 289–297.

[108] S. Manandher. Activity server: You can run but you can't hide. In *Proceedings of Usenix Summer Conference '91*, pages 299–311, 1991.

[109] M.M. Mantei, R.M. Baecker, A.J. Sellen, W.A.S. Buxton, T. Milligan, and B. Wellman. Experiences in the use of a media space. In chi91 [36], pages 203–208.

[110] S. McCanne and V. Jacobson. vic: A flexible framework for packet video. In *Proceedings of ACM MultiMedia '95*, page ??, 1995.

[111] S. McCanne and V. Jacobson. *vat unix man pages*, 1996.

[112] S. McCanne and V. Jacobson. *vic unix man pages*, 1996.

[113] P.L. McLeod. An assessment of the experimental literature on electronic support of group work: Results of a meta-analysis. *Human Computer Interaction*, 7:257–280, 1992.

[114] G. Morgan. *Images of Organization.* Sage, London, 1986.

[115] D. A. Norman. *The Psychology of Everyday Things.* Basic Books, New York, 1988.

[116] D.A. Norman and S. Draper, editors. *User Centered System Design: New Perspectives on Human-Computer Interaction.* Lawrence Erlbaum Associates, 1986.

[117] G.M. Olson and J.S. Olson. Introduction to this special issue on computer-supported cooperative work. *Human Computer Interaction*, 7(4):251–256, 1994.

[118] J.K. Ousterhout. *Tcl and the TK toolkit.* Addison Wesley, New York, 1994.

[119] John K. Outserhout. *Tk 4.0: Unix Manual Pages.*

[120] M Robinson. Design for unanticipated use... In ecscw93 [58], pages 187–202.

[121] M. Robinson and L. Bannon. Questioning representations. In Bannon et al. [16], pages 219–233.

[122] R.W. Root. Design of a multimedia vehicle for social browsing. In *Proceedings of the Second Conference on Computer Supported Cooperative Work*, pages 25–38. New York: ACM, September 1988.

[123] M. Rouncefield, J.A. Hughes, T. Rodden, and S. Viller. Working with "constant interruption": Cscw and the small office. In cscw94 [44], pages ?–?

[124] H. Sacks, E.A. Schegloff, and G. Jefferson. A simplest systematics for the organisation of turn-taking for conversation. *Language*, 50:696–735, 1974.

[125] M.A. Sasse, M.J. Handley, and S.C. Chuang. Support for collaborative authoring via email: The messie environment. In ecscw93 [58], pages 249–?

[126] B. Schatz. Building an electronic community system. *Journal of Management Information Systems*, 8(3):87–107, 1991-92.

[127] K. Schmidt. Riding a tiger, or computer supported cooperative work. In Bannon et al. [16], pages 1–16.

[128] E Schooler. The connection control protocol: Specification. Technical report, USC/Information Sciences Institute, Marina del Rey, Ca., 1992.

[129] E. Schooler. A multicast based user directory service. IETF-MMUSIC Working Group Presentation., 1996. Slides available at ftp://ftp.isi.edu/confctrl/minutes/slides.3.96.tar.Z.

[130] D. Schuler and A. Namioka, editors. *Participatory Design: Principles and Practices.* Lawrence Erlbaum Associates, 1993.

[131] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. IETF-AVT-INTERNET-DRAFT, 1995. Work in progress.

[132] K. Scmidt and L. Bannon. Taking cscw seriously supporting articulation work. *Computer Supported Cooperative Work*, 1:7–40, 1992.

[133] K. Scmidt and C. Simone. Coordination mechanisms: Towards a conceptual foundation of cscw systems design. *Computer Supported Cooperative Work*, 5:155–200, 1996.

[134] D. Shapiro. The limits of ethnography: Combining social sciences for cscw. In cscw94 [44], pages 417–428.

[135] W.W. Sharrock and R.J. Anderson. *The Ethnomethodologists.* Ellis Horwood, 1992.

[136] H. Shen and P. Dewan. Access control for collaborative environments. In cscw92 [43], pages 51–58.

[137] Brian Smith and Lawrence Rowe. *Tcl-DP 3.3b1.* Available from ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/Tcl-DP.

[138] G. Smith and T. Rodden. An access model for shared interfaces. In cscw94 [44], page ??

[139] M. Sohlenkamp and G. Chwelos. Integrating communication, cooperation, and awareness: The diva virtual office environment. In cscw94 [44], pages 331–343.

[140] I. Somerville, R. Bentley, T. Rodden, and R. Sawyer. Cooperative systems design. *The Computer Journal*, 37(5):357–366, 1994.

[141] I. Somerville, T. Rodden, P. Sawyer, and R. Bentley. Sociologists can be suprisingly useful in interactive systems design. In *People and Computers VII, Proc. HCI '92*, page ?? Cambridge University Press, 1992.

[142] W.C. Sturtevant. Studies in ethnoscience. In S.P. Spradley, editor, *Culture and Cognition*, pages 129–167. Chandler, 1972.

[143] L. Suchman. Technologies of accountability: Of lizards and aeroplanes. In Button [32], pages 113–126.

[144] L. Suchman. Do categories have politics? the language/action perspective reconsidered. In ecscw93 [58], pages 1–14.

[145] L. Suchman and R. Trigg. Understanding practice: Video as a medium for reflection and design. In J. Greenbaum and M. Kyng, editors, *Design at Work: Cooperative Design of Computer Systems*, pages 75–89. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1991.

[146] L.A. Suchman. Office procedures as practical action. *ACM Transactions on Office Information Systems*, 1:320–328, 1983.

[147] L.A. Suchman. *Plans and Situated Actions. The problem of human-machine communication.* Cambridge University Press, Cambridge, 1987.

[148] L.A. Suchman and E. Wynn. Procedures and problems in the office. *Office: Technology and People*, 2:133–154, 1984.

[149] J.C. Tang and M. Rua. Montage: Providing teleproximity for distributed groups. In *Proc. ACM SIGCHI Conference on Computer Human Interaction (CHI) '94*, pages 37–43, April 1994.

[150] F.W. Taylor. *The Principles of Scientific Management.* Harper, New York, 1923.

[151] P. Thomas, editor. *The Social and Interactional Dimensions of Human-Computer Interfaces.* Cambridge University Press, 1995.

[152] J. Trevor, T. Rodden, and G. Blair. Cola: A lightweight platform for cscw. *Computer Supported Cooperative Work*, 3:197–224, 1995.

[153] UCL. Nte - a network text editor, 1996. Software available from: http://www-mice.cs.ucl.ac.uk/mice/nte/.

[154] UCL. Rat - a reliable audio tool, 1996. Software available from: http://www-mice.cs.ucl.ac.uk/mice/rat/.

[155] D.G. Wastell and P. White. Using process technology to support co-operative work: Prospects and design issues. In D Diaper and C. Sanger, editors, *CSCW in Practice: an Introduction and Case Studies*, pages 105–126. Springer-Verlag, London, 1993.

[156] D.L. Wieder. On meaning by rule. In Douglas [48], pages 107–135.

[157] R Williams. User location service. IETF-MMUSIC-INTERNET-DRAFT, February 1996. Work in progress.

[158] T. Winograd. A language/action perspective on the design of cooperative work. *Human Computer Interaction*, 3(1):3–30, 1987-88.

[159] T. Winograd. Categories, disciplines, and coordination. *Computer Supported Collaborative Work*, 2(3):191–197, 1994.

[160] D. H. Zimmerman. The practicalities of rule use. In Douglas [48], chapter 9, pages 221–238.

Appendix A

Frame Analysis Data

A.1 Introduction

This appendix provides the numerical data from which the figures in Chapter 4 were derived.

Action	**Open**	**Ajar**	**Closed**
Walk in	55	9	9
Knock and Enter	37	27	13
Knock and Wait	5	27	23
Glance	3	36	26
Leave a Message	0	0	9
Come back later	0	0	12
Talk to Secretary	0	0	0

Table A.1: Percentage Frequency of Responses For Each State

Action	**Friend**	**Boss**	**Boss's Boss**
Walk in	75	43	0
Knock and Enter	20	57	50
Knock and Wait	5	0	25
Glance	0	0	25
Leave a Message	0	0	0
Come back later	0	0	12
Talk to Secretary	0	0	0

Table A.2: Percentage Frequency of Responses For 'Open' State

Action	**Friend**	**Boss**	**Boss's Boss**
Walk in	20	0	0
Knock and Enter	40	20	0
Knock and Wait	0	40	100
Glance	40	40	0
Leave a Message	0	0	0
Come back later	0	0	0
Talk to Secretary	0	0	0

Table A.3: Percentage Frequency of Responses For 'Ajar' State

Action	**Friend**	**Boss**	**Boss's Boss**
Walk in	13	6	0
Knock and Enter	21	6	0
Knock and Wait	18	29	25
Glance	31	23	12
Leave a Message	5	10	25
Come back later	8	19	0
Talk to Secretary	5	6	38

Table A.4: Percentage Frequency of Responses For 'Closed' State

Appendix B

GAP: A Group Awareness Protocol

B.1 Introduction

This document is a very drafty draft of a group awareness protocol for the distribution of awareness information between user-awareness tools. An awareness tool provides critical support for effective group work by enabling users to assess the current activity of potential co-workers. The aim of this document is to prompt discussion and research within the MMUSIC WG that focuses on the development of awareness tools for distributed groups in order to provide further support for effective collaborative work via the Internet Multimedia Conferencing Architecture. This work can be considered complimentary to the User Location service and as an enabler for SIP.

This Appendix is also available as an internet draft [9]

B.2 Awareness and Group Work in The Internet

A recent Internet Draft [83] describes the evolving architecture to support multimedia conferencing, and hence synchronous, focused, group work in the Internet context. At present two models of user discovery and participation in such multimedia sessions exist: session announcement, and session invitation. In the first instance users see the 'public' announcement of a 'conference' of interest and choose to join. In the second, users are explicitly invited to join a 'conference' by another user who must have access to some sort of directory [129] or location service [157]. Whilst such an architecture provides the basic capability to support group work, a number of user-oriented requirements are, as yet, unsupported.

One such requirement is the need to support informal awareness:

> In everyday work, informal awareness involves knowing who's currently around, whether they're available or busy, and what sort of activity they're engaged in. [81, pp 205]

General informal awareness of what other members of a work group are doing; how busy they are, whether they can be interrupted, who they are talking to and whether they are available for collaboration has been identified as being a critical enabler of group work [20, 55, 78]. As these studies have shown, the ability to move smoothly between single-user and collaborative working very much depends upon users being able to assess the activity of others.

Therefore distributed work groups need to have a sense of who's around and what they are doing in order to successfully coordinate their work. At present there is little support for such 'informal awareness' in the evolving Internet Conferencing Architecture and as a result, the potential for effective, synchronous, group work over the Internet may not be fully realised. If the MMUSIC WG is to further the goal of supporting effective group work in the Internet context then we suggest that there is a need, and a place, for research on the development of such general informal awareness tools as part of the Internet Multimedia Conferencing Architecture (see Figure B.1).

This document attempts to provide a start point for such an effort and is a first attempt to draw together common strands from the CSCW, CHI and Communications research communities.

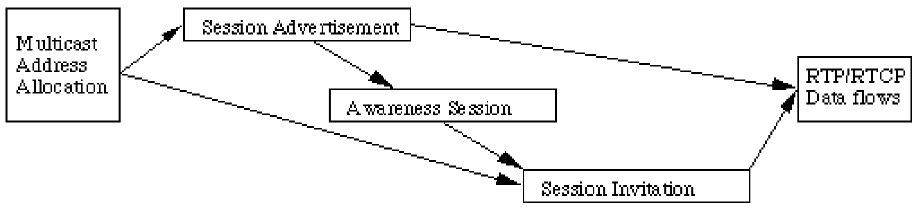

Figure B.1: GAP and the Internet Multimedia Conferencing Architecture

This document describes a proposed group awareness protocol (GAP) to support the distribution of "low-cost" awareness information to a user group which have some interest in that information. GAP draws its inspiration primarily from ongoing work in the IETF's MMUSIC and AVT working groups, most notably the CCP [128], CCCP [86], SDP [84] and RTP [131] protocols. It may be considered complimentary to the User Location service [157] and as an enabler for SIP [85] since users running a GAP tool may become aware of another user's activity and move smoothly, using SIP, into collaborative work.

B.3 GAP Usage

B.3.1 Use of IP Multicast

GAP uses IP multicast to distribute 'awareness information' between each user's GAP tool. Under normal conditions, a GAP tool will periodically multicast an information packet on a multicast address and port that are assigned at the tool's start-up time. GAP tools are not required to multicast to a well-known address/port combination. Rather, an awareness session is created using a multicast address allocation and session announcement tool (eg. sdr [82]). The awareness session must therefore be suitably scoped at this point in order to include all relevant participants. Users interested in a particular 'awareness session' can then launch their GAP tool from the SDP announcement.

The use of IP multicast provides a number of advantages:

Scalability/Efficiency - GAP tools send only one announcement to the group address rather than multiple announcements to each participant. Periodicity controls can regulate bandwidth requirements.

Scoping - Administrative scoping allows 'awareness sessions' to have a high ttl (in order to reach all potential participants) but are guaranteed not to be visible outside a scope zone.

Resistance to failure - Periodic multicast ensures that even if network outages occur, GAP tools do not completely fail due to loss of a centralised resource.

GAP therefore makes no assumptions about the consistency of global state, relying on the user interface to inform participants when a discrepancy is likely, and on the tendency for global state to re-converge following, for example, network outages.

B.3.2 Requirements and Extension

Since GAP was designed to support the sharing of 'awareness information' by groups of users, this draft outlines a core set of GAP information that may be considered necessary in any awareness sharing situation. An extension mechanism to enable particular GAP tools to add other information to this core set in order to enhance their 'awareness' support is defined below. Given the diverse information that different applications may require, such an extension mechanism appears vital.

Proposed requirements:

User's Name - User editable.

Local Time - Important for cross-time-zone groups.

URI of information relevant to session - eg. project group home page.

User's Email address

User's telephone number

User's current location - to be handed to a SIP tool to initiate a more focused session.

User's current activity status - eg. on phone or currently in a SIP-initiated session.

Other information - any application dependent information. This constitutes the extension mechanism.

In general, GAP should convey sufficient information about a particular user to enable a colleague to determine whether or not they are available for communication, and to enable that communication to be initiated.

B.3.3 Security

Security may be provided by encryption of GAP packets as has been suggested for SDP, RTP and SIP.

B.3.4 Multiple Parallel Sessions

By providing the ability to describe and announce GAP sessions using SDP, users may choose to join more than one GAP session at a time. This implies issues of conflict for resources that are not dealt with by the GAP protocol.

B.3.5 Usage Scenario

A distributed organisation with a broadband infrastructure may choose to provide multimedia conferencing applications at each employee's workstation. A particular project group may create a GAP announcement using an SDP aware tool which is suitably scoped to include the location of all members of that group. Each member of the group may then join that GAP session (or as many sessions as they are interested in) by launching a GAP-aware tool from their session directory. This GAP tool may then enable them to initiate more focused

Item	Example	Definition
0	v=GAP/1.0 INFO	Version Type
1	id=8975664	ID
2	ntp=457386765	NTP Time Stamp
3	cn=ben@128.125.110.123	Canonical name
4	seq=1	Sequence number
5	n=Ben Anderson (LUT,UK)	Name
6	t=14:22	Local Time
7	u=http://www-rs.cs.lut.ac.uk/ben	URI for more info
8	e=B.Anderson@lut.ac.uk	Email
9	p=+44 (0)1509 222689	Telephone
10	ca=158.125.5.11/5536747:Glancing at Jon	Current activity
11	ca=158.125.5.11/4345667:Whiteboard with Adam	Current activity
12	i=doorstate:open	State of user's 'door'
13	i=note:Gone to lunch	Short text message
14	i=havevideo	If user can send video
15	i=havespeaker	If user has speaker
16	i=havemic	If user has microphone

Table B.1: Payload of 'INFO' packet

interaction with other members of the project group by initiating conferencing calls. In this case, the GAP application will have informed the others of information relevant to its user's current activity (and, potentially, media transport preferences), and can then hand over the creation of a range of multimedia conferencing services to SIP.

B.4 GAP Specification

GAP awareness information is entirely text-based. Each GAP packet is made up of a number of lines of text of the form:

identifier=value

where *identifier* is a unique character or short string and where *value* is a text string whose structure depends on the preceding *identifier*. As with SDP [84] no white space is allowed each side of the '=' and the line is terminated by the newline character. Newline characters are, therefore, not allowed in the *value* text string.

Two packet types, 'INFO' and 'BYE', are currently defined each of which share a standard header.

B.4.1 GAP Standard Header

The first 5 elements of any GAP packet form a fixed header (see Table B.1 or B.2).

v= Protocol/Version Type 'Version' identifies the version of GAP (1.0 in the example above). This document describes version 1.0. Applications should ignore version numbers they do not recognise[1]. 'Type' denotes the

[1]Although this may be a useful way of informing users that a new version is available and being used...

kind of packet - currently defined to be one of INFO, BYE.

id= ID Unique identifier for source that sent the packet. This is generated at start-up after x seconds so that generation can take into account the ID of other sources. For this reason, no packets are sent by a source immediately following start-up (cf RTCP).

ntp= Time Stamp Time at which the packet was sent. NTP format.

cn= CName Canonical name (cf RTCP) of source. Unlike ID this does not change when a GAP tool is restarted. Provides location information for user and can be handed to a SIP tool.

seq= Sequence Sequence number for packet. Starts from 1 at start-up time (and hence may return to 1 for a particular source if the tool is restarted within a session).

B.4.2 INFO Packet

This packet contains awareness information from each source. A description, with examples, can be found in Table B.1.

The core awareness information suggested above is encoded as:

n= Name (and affiliation) of source for display in participant's list etc. Must be human readable and user editable.

t= Local time of source. Useful where participants are spread across time zones. Intended to be associated with source and displayed in user interface.

u= Universal Resource Identifier as used by World Wide Web clients. Can point to eg. a home page, a page giving information on the work group etc.

e= Email address of user. For format see [84].

p= Telephone number of user. For format see [84].

ca= Zero or more lines listing SIP (or otherwise) initiated sessions in which the user is participating[2].

i= Zero or more lines which provide additional information. This provides the main extension mechanism for GAP.

As an example of the extension mechanism, our experimental GAP tool - *TelePort* - [8] uses the following information fields (see Table B.1):

i=doorstate:*state* State of user's iconic door - defined to be one of "open", "ajar" or "closed".

i=note:*note* Text message displayed in user interface. User editable.

i=havevideo Source's video capability - presence indicates user can send video. Note assumption that reception requires no extra hardware.

[2]Integration or communication with a conference control module is needed here...

Item	Example	Definition
0	v=GAP/1.0 INFO	Version Type
1	id=8975664	ID
2	ntp=457386765	NTP Time Stamp
3	cn=ben@128.125.110.123	Canonical name
4	seq=132	Sequence number
5	t=(Text message)	t=Bye Bye

Table B.2: Payload of 'BYE' packet

i=havespeaker As above but for audio output.

i=havemic As above but for audio input.

Other examples might include the user's media preferences profile (eg H261 for video, maximum bandwidth of 128kb/s) or the url of documents currently being viewed or edited .

A GAP tool should gracefully ignore awareness information it does not know how to use.

B.4.3 BYE Packet

This packet is sent when a participant leaves a session and/or when a GAP tool intentionally quits. The packet is sent only once. A description, with example, is provided in Table B.2.

t=*message* Carries a text string providing a reason for sending the BYE packet. This is entirely optional and may be used in the user interface if present.

B.5 Scaling Issues

Since GAP relies on periodic multicast of state information, the greater the number of participants, the greater the number of packets being sent at any one time. In order to attempt to scale this bandwidth usage it is intended that GAP, as with RTCP, should keep track of the number of participants and the mean packet size and adjust the time interval between sending of INFO packets accordingly .

GAP uses the algorithms described in the RTP Draft Specification [131] and divides the bandwidth allocated to the session equally between the members. There is nothing in GAP analogous to the RTP data stream (RTCP is allocated 5% of the RTP data stream's bandwidth), and the required bandwidth is likely to be determined by the speed with which users want awareness information to be propagated. Since this will be dependent on the function of the tool using GAP, it is likely that the maximum bandwidth allocated to GAP will be defined on a per-application, or even per-session basis.

The goal of these methods is to attempt to ensure that the network activity of GAP sessions is relatively independent of the number of participants and generates more or less smooth data traffic.

B.6 Unresolved Issues

Is the arbitrary extension mechanism (ie. multiple information lines) a sensible solution?

Interworking with ITU T.120 suite?

Privacy of information.

B.7 Acknowledgments and Author

The research leading to this document was supported by a BT graduate scholarship.

`Ben Anderson`
`Department Computer Studies`
`Loughborough University`
`Loughborough`
`Leicestershire`
`UK.`
`Email:B.Anderson@lut.ac.uk`
`WWW:http://pipkin.lut.ac.uk/~ben`

Appendix C

TelePort: Code

C.1 Introduction

This appendix contains a listing of the C and Tcl-DP source code for the *TelePort* prototype. The listing has been split into two main sections - the C-like Embedded Tk source, and the Tcl-DP source with the latter further divided into logical units.

C.2 C source

The following listing contains the Embedded Tk code that is run through the ET macros in order to compile a binary of *TelePort*. The listing is in two parts - the main() function processes the Tcl-DP source code for compilation whilst the function defined as ET_PROC (getsecs) returns the current time of day in seconds. The latter is written using the Embedded Tk syntax for defining external C commands to be compiled into a tcl interpreter.

```
#-----------------------------------------------
** Copyright (c) 1995 Loughborough University of Technology.
** Copyright (c) 1995 British Telecommunications plc.
** All rights reserved.
**
** Author: B.Anderson@lut.ac.uk
**
** This file contains C code for a group awareness tool written using
** Embedded Tk.
**
** It adds one C function - 'getsecs' which returns the seconds since
** the epoch. Needed because Tcl 7.4 doesn't offer 'clock' functions.
** ------------------------------------------------------------------
*/

#include <stdio.h>
#include "tcl.h"
#include <time.h>

int main(int argc, char **argv){
  Et_Init(&argc,argv);
  if( Tdp_Init(Et_Interp)!=ET_OK ){
    fprintf(stderr,"Tdp_Init failed.\n");
    exit(1);
  }
  ET_INSTALL_COMMANDS;
  ET_INCLUDE( dp_init.tcl );
  ET_INCLUDE( teleport2.1-sunos.tcl );
  Et_MainLoop();
  return 0;
}

ET_PROC( getsecs )  {
time_t t; /* Number of seconds since epoch */
t = time(0);

sprintf(interp->result,"%d",t);
return ET_OK;
}
```

C.3 Tcl-DP source

The following listing contains the Tcl-DP source for the *TelePort* prototype divided into logical units.

C.3.1 Set Global Variables

```
#------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------

#create some useful global variables
```

```
# T - holds all important app info
global T
set T(play) audioplay
set T(Lname) "TelePort 2.1a3"
set T(Sname) "TP 2.1a3"
set T(DialogTimer) 10
set T(ef) {-*-times-medium-r-normal--*-*-*-*-*-*-*-*}
set T(bgc) {linen}
set T(inactive) gray50
set T(active) black
set T(preferences_file) {~/.GAPdefaults}
set T(logfile) {~/.GAP-TP2.1-log}
set T(icon_vol) 9
set T(DebugWin) .a
set T(waitDialogWin) .b
set T(CONNECT) "In a/v conference with "
set T(KNOCK) "Knocking on/at "
set T(GLANCE) "Glancing at "
set T(WHITEBOARD) "Working with "
set T(IDLE) {Not in conference}

# initialise winCount (used to set WinIDs)
global WinCount
set WinCount 1

# default list of media formats
global _FORMATS
set FORMATS(vic) [list H261 JPEG MPV CelB]
set FORMATS(vat) [list PCMU PCMA IDVI GSM]
set FORMATS(wb) [list wb]
set FORMATS(text) [list nt]

# default list of transports
global TRANSPORTS
set TRANSPORTS [list UDP RTP]

# list of  apps
global APPS
set APPS(video) [list vic]
set APPS(audio) [list vat]
set APPS(whiteboard) [list wb]
set APPS(text) [list nt]

# defaults for user's GAP values (overwritten by preferences file)
global GAP

# Initialise GAP header data
set GAP(Version) {GAP/1.0}

# Don't get ID yet, wait until we have recieved others
# so we can check for clashes
set GAP(BusyWith) 0
set GAP(Min_Time) 5000
set GAP(SendAfter) 5000
set GAP(TimeOut) 180
set GAP(TotalPackets) 0
set GAP(TotalBytes) 0
# GAP(MaxBandwidth) = total bw given over to GAP in bytes per sec
set GAP(MaxBandwidth) 0.5
# Mean packet size in bytes (ie. octets, characters)
# calculated from: 20 (IP) + 8 (UDP) + 4 (tcl-dp) + ??
set GAP(MeanPacketSize) 280
set GAP(Name) "Fred Bloggs, Useless Inc"
set GAP(Email) "fred@useless.com"
set GAP(Uri) {http://www.useless.com/~fred}
set GAP(Pots) {+01 (0)123 456 7890}
set GAP(Sequence) 1
set GAP(LastReceived) 0
set GAP(Sources) 1

# X-TP (Teleport) specific data
set GAP(Code) X-GAP1
set GAP(State) closed
set GAP(Note) "Being useless..."
set GAP(Have_Video) {1}
set GAP(Have_Mic) {1}
set GAP(Have_Speaker) {1}

#set some USCP (Conference control protocol) data
global USCP
set USCP(Version) USCP/1.0
set USCP(Logname) $env(USER)
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)

# set some SIP data
set SIP(Version) {SIP/1.0}
set SIP(Logname) $env(USER)
set SIP(Authority) {none}
set SIP(ReqCount) 0
set SIP(Sequence) 0
set SIP(ResendMax) 5
set SIP(ResendAfter) 1000
set SIP(Port) 9864

# set default  media as SDP types (see Internet Draft IETF-MMUSIC
# SDP v2 - Handley & Jacobson)
global MEDIA
```

```
# video
set MEDIA(videoA) "vic"
set MEDIA(videoP) "video port not set"
set MEDIA(videoT) "RTP"
set MEDIA(videoF) "H261"
set MEDIA(videoR) "128"

# audio
set MEDIA(audioA) "vat"
set MEDIA(audioP) "audio port not set"
set MEDIA(audioT) "RTP"
set MEDIA(audioF) "PCMU"

# whiteboard
set MEDIA(whiteboardA) "wb"
set MEDIA(whiteboardP) "whiteboard port not set"
set MEDIA(whiteboardT) "UDP"
set MEDIA(whiteboardF) "WB"

# text
set MEDIA(textA) "nt"
set MEDIA(textP) "text port not set"
set MEDIA(textT) "UDP"
set MEDIA(textF) "nt"

# define some urls
global URLS
set URLS(GAP) {http://pipkin.lut.ac.uk/~ben/PHD/public_docs/GAP.html}
set URLS(SIP) {http://pipkin.lut.ac.uk/~ben/PHD/public_docs/SIP.html}
set URLS(teleport) {http://pipkin.lut.ac.uk/~ben/PHD/teleport.html}
set URLS(mbone) {http://pipkin.lut.ac.uk/~ben/video/}
```

C.3.2 Main Section

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#-----------------------------------------

# main part of program

# make sure popups don't stay popped
bind Menubutton <ButtonRelease-1> {
tkMenuUnpost {}
}

proc start_Tools {} {
# Launch video tool but _not_ audio tool
  init_Video
}

proc get_Args {} {
   global argc argv T MEDIA
   # Get the command line arguments
  set argslist [split $argv "-"]
foreach e $argslist {
   set T([lindex $e 0]) [lrange $e 1 end]
}
set MEDIA(videoP) [expr $T(mport) + 2]
set MEDIA(audioP) [expr $T(mport) + 4]
set MEDIA(whiteboardP) [expr $T(mport) + 8]
check_mcaddress

set T(sessionName) $T(N)
set_cb_channel
}

proc init_GAP {} {
   # get our IP address and a unique identifier
   global GAP
set GAP(IP) [get_IP]
   set GAP(ID) [set_ID]
}

proc start_Up {} {
   global T
   set f $T(preferences_file)
destroy $T(initDialogWin)
if [file exists $f] {
source $f
Init
} else {
newuserDialog
}
}

proc Init {} {
global T GAP SIP
fb "Initialising network"
update idletasks
# Initalise multicast socket
set N(address) $T(maddr)
set N(port) $T(mport)
set N(ttl) $T(ttl)
new GAP N

set N(port) $SIP(Port)
new SIP N

#create main window
build_ui_mainWin
wm deiconify $T(mainWin)

# calculate first timer
set GAP(SendAfter) [new_Timer 1]

# Enter 'after' loop to
# multicast GAP packets
after $GAP(SendAfter) loop

# after Timeout seconds, decrease state of pts in case
# of network failure etc
fb ""
decrease_State
update idletasks
}

proc loop {} {
global T PTS GAP
set me $GAP(CName)
set PTS($me,netStatus) 4
send_GAP INFO

set GAP(SendAfter) [new_Timer 0]
after $GAP(SendAfter) loop
}

wm withdraw .

# initialise random number generator
randomInit [pid]

image create bitmap tp_grey -file "$env(TPHOME)/picsnds/tp_grey.xbm"
set img tp_grey
set msg "TelePort is starting up"
initDialog $img $msg

update idletasks

fb "Getting Arguments"
update idletasks
get_Args

fb "Initialising Protocol"
update idletasks

init_GAP

# initilise conference bus
fb "Initialising Conference Bus"
update idletasks
confbus_init

# initilise audio and video tools
fb "Initialising Video Tool"
update idletasks

start_Tools

# horrible, but this stops vic running out of colours
# might need to be longer timer if machine is slower
# Might go away as and when icons are internally defined (?)

after 5000
# initilise audio and video tools
fb "Configuring Video Tool"
update idletasks

# get_interp_name video
set GAP(CName) [get_CName]
set T(CName) $GAP(CName)
confbus video tp_no_quit

# do this last to try to avoid colour problems with vic
fb "Loading UI resources"
update idletasks
load_Images

start_Up
```

C.3.3 Main User Interface

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#-----------------------------------------


#-----------------------------------------
# ui-main.tcl
#
# Builds but does not show the main window.
#-----------------------------------------

proc change_State {door} {
global GAP T
set GAP(State) $door
$T(mydoorName) configure -image b$door
send_GAP INFO
}

proc Leave {} {
   global GAP_SENT GAP_GOT T GAP
send_GAP BYE
set msg [list tp_exit]
confbus video $msg
set f [open $T(logfile) a+]
puts $f "#next_record:[get_NTP]"
puts $f "#GAP_received"
   foreach name [lsort [array names GAP_GOT]] {
puts $f "GAP_GOT($name) = $GAP_GOT($name)"
   }
  puts $f "#GAP_sent"
  foreach name [lsort [array names GAP_SENT]] {
    puts $f "GAP_SENT($name) = $GAP_SENT($name)"
    }
close $f
exit
}

proc restart {} {
   init_Video
   init_Audio
 }

proc build_ui_mainWin {} {
global T GAP URLS USCP
set w .
  wm withdraw $w
  set T(mainWin) $w
#wm resizable $w 0 0
wm title $w "$T(Lname): $T(sessionName)"
wm iconname $w "$T(sessionName)"

frame .menubar -relief {raised} -bd 2

menubutton .menubar.file -menu {.menubar.file.m} \
-text {File}

menu .menubar.file.m
.menubar.file.m add command \
-command {Leave} \
-label {Quit}

menubutton .menubar.help \
-menu {.menubar.help.m} \
-text {Help}

menu .menubar.help.m

.menubar.help.m add command \
-command {help info} \
-label "On TelePort"

.menubar.help.m add command \
-command {help mbone}\
-label {On the MBONE}

menubutton .menubar.edit \
-menu {.menubar.edit.m} \
-text {Edit}

menu .menubar.edit.m
.menubar.edit.m add command -label {Preferences}\
-command {
if [winfo exists .p] {
   wm deiconify .p
   raise .p
} else {
   build_ui_preferencesWin
}
}

menubutton .menubar.show \
-menu {.menubar.show.m} \
-text {Show..}

menu .menubar.show.m
.menubar.show.m add command \
-label {Participants} \
-command {
if [winfo exists .plist] {
wm deiconify .plist
raise .plist
} else {
   create_List
   wm deiconify .plist
   raise .plist
}
}
.menubar.show.m add command \
-command {
if [winfo exists .ni] {
wm deiconify .ni
raise .ni
} else {build_ui_infoWin}
}\
-label {Network Info}

pack append .menubar \
.menubar.file {left frame center} \
.menubar.edit {left frame center} \
.menubar.help {right frame center} \
.menubar.show {left frame center}

frame .doors
frame .doors.current -relief {groove} -bd 2
label .doors.current.name -textvariable GAP(Name)
entry .doors.current.note \
-font $T(ef) \
-bg $T(bgc) \
-relief {sunken} \
-textvariable GAP(Note)
bind .doors.current.note <Return> {send_GAP INFO}
label .doors.current.act -textvariable USCP(CurrentActivity) -relief sunken
label .doors.current.icon -image b$GAP(State) -relief flat
set T(mydoorName) {.doors.current.icon}
frame .doors.current.options -relief {groove} -bd 2
label .doors.current.options.l -text {Options:}
button .doors.current.options.c -image sclosed\
-command {change_State closed} \
-relief raised -bd 2
button .doors.current.options.a -image sajar \
-command {change_State ajar}\
-relief raised -bd 2
button .doors.current.options.o -image sopen\
-command {change_State open}\
-relief raised -bd 2
pack .doors.current.options.l .doors.current.options.c \
.doors.current.options.a \
.doors.current.options.o -side left

pack .doors.current.name .doors.current.note \
.doors.current.act\
.doors.current.icon .doors.current.options\
 -side top -fill x -padx 2 -pady 2

frame .doors.media

frame .doors.media.v -relief groove -bd 2
frame .doors.media.a -relief groove -bd 2

pack .doors.media.v .doors.media.a -side top \
-expand 1\
-fill both

label .doors.media.v.l -text {Video Capability:}
radiobutton .doors.media.v.0 -image v0 -text {Can't Send Video}\
-value 0 -anchor w -variable GAP(Have_Video)
radiobutton .doors.media.v.1 -image v1 -text {Can Send Video}\
-value 1 -anchor w -variable GAP(Have_Video)

pack .doors.media.v.l .doors.media.v.0 .doors.media.v.1 -fill both -expand 1

label .doors.media.a.l -text {Audio Capability:}
radiobutton .doors.media.a.0 \
-text {No speaker}\
-image speaker0 -anchor w\
-value 0 -variable GAP(Have_Speaker)
radiobutton .doors.media.a.1 \
   -text {Have speaker}\
   -image speaker1 -anchor w\
-variable GAP(Have_Speaker) -value 1
radiobutton .doors.media.a.2 \
   -text {No microphone}\
-image mic0 -anchor w\
-value 0 -variable GAP(Have_Mic)
radiobutton .doors.media.a.3 \
-text {Have microphone}\
-image mic1 -anchor w\
-value 1 -variable GAP(Have_Mic)

pack .doors.media.a.l .doors.media.a.0\
.doors.media.a.1 .doors.media.a.2\
.doors.media.a.3 -fill x

pack .doors.current .doors.media -side left
```

```
frame .feedback
label .feedback.i -bitmap info -bd 2
label .feedback.t -textvariable T(fb) -anchor w -bd 2
pack .feedback.i .feedback.t -side left -padx 2 -pady 2

pack append  . \
.menubar {top frame center fillx} \
.doors {top frame center fillx} \
.feedback {top frame center fillx}

update idletasks

Set_Geom $w both
}


proc build_ui_infoWin {} {
global T GAP GAP_STATS
toplevel .ni
wm title .ni "$T(Sname): Debuging Information"
#wm resizable .ni 0 0

frame .ni.params -relief ridge -bd 2

frame .ni.params.address
label .ni.params.address.l -text {Address:}
label .ni.params.address.v -anchor {w} -textvariable T(maddr)
pack .ni.params.address.l .ni.params.address.v -side left

frame .ni.params.port
label .ni.params.port.l -text {Port:}
label .ni.params.port.v -anchor {w} -textvariable T(mport)
pack .ni.params.port.l .ni.params.port.v -side left

frame .ni.params.ttl
label .ni.params.ttl.l -text {TTL:}
label .ni.params.ttl.v -anchor {w} -textvariable T(ttl)
pack .ni.params.ttl.l .ni.params.ttl.v -side left

pack  .ni.params.address .ni.params.port .ni.params.ttl -side top -fill x

frame .ni.gap -relief ridge -bd 2
set w .ni.gap

frame $w.sources
label $w.sources.l -text {Number of Sources:}
label $w.sources.i -anchor w -textvariable GAP(Sources)
pack $w.sources.l $w.sources.i -side left
pack $w.sources -fill x

frame $w.timing
label $w.timing.l -text {Next INFO sent after:}
label $w.timing.i -textvariable GAP(SendAfter)
pack $w.timing.l $w.timing.i -side left
pack $w.timing -fill x

frame $w.psize
label $w.psize.l -text {Mean Packet Size (bytes):}
label $w.psize.i -anchor w -textvariable GAP(MeanPacketBytes)
pack $w.psize.l $w.psize.i -side left
pack $w.psize -fill x

frame $w.bw
label $w.bw.l -text {Bandwidth Devoted to GAP (bytes/s):}
label $w.bw.i -anchor w -textvariable GAP(MaxBandwidth)
pack $w.bw.l  $w.bw.i -side left
pack $w.bw -fill x

frame $w.gs
label $w.gs.l -text {Packets received per second:}
label $w.gs.p -anchor w -textvariable GAP_STATS(pps)
pack $w.gs.l $w.gs.p -side left
pack $w.gs -fill x

frame $w.m
label $w.m.l -text {Multiplier (should be 0.5 - 1.5):}
label $w.m.p -anchor w -textvariable GAP(Multiplier)
pack $w.m.l $w.m.p -side left
pack $w.m -fill x

frame $w.g -relief ridge -bd 2
canvas $w.g.c -height 200 -width 200
pack $w.g.c
pack $w.g -fill x -expand yes
set T(viewgraph) $w.g.c
set T(viewgraphyoffset) [$T(viewgraph) cget -height]

frame $w.fb
label $w.fb.l -text {Feedback:}
label $w.fb.p -anchor w -textvariable GAP(statsfb)
pack $w.fb.l $w.fb.p -side left
pack $w.fb -fill x

frame .ni.but
button .ni.but.d -text {Dismiss} -command {destroy .ni}
pack .ni.but.d
pack .ni.params .ni.gap .ni.but\
-fill x -side top -expand 1
pack .ni.params
```

```
}

proc add_point {x y tag} {
global T GAP
set y [expr $y/1000]
# the next line inverts the graph to standard  x y coordinates
set y [expr $T(viewgraphyoffset) - $y]

set GAP(statsfb) "Adding point at x=$x y=$y"
set new [$T(viewgraph) create oval [expr $x - 2] [expr $y - 2] \
[expr $x + 2] [expr $y + 2] -outline black -fill red -tags $tag]
}

##############################################################
# Procedures to manage the doors
# build_ui_userDoorWin {source} - creates door window for a source, called
# by clicking on participants list names
##############################################################

proc update_Door {w src} {
   global PTS

   if [winfo exists .$w] {
      wm title .$w $PTS($src,n)
     #.$w configure -bg $PTS($src,Nstate)
     # assumes this info exists - dangerous!
     # .$w.info.status.l configure -text $status
.$w.door.icon configure -image b$PTS($src,doorstate)
.$w.info.hw.sp configure -image speaker$PTS($src,havespeaker)
.$w.info.hw.mic configure -image mic$PTS($src,havemic)
.$w.info.hw.v configure -image v$PTS($src,havevideo)
}
# otherwise do nothing
}

proc build_ui_userDoorWin {cname} {
global PTS T WINID GAP
set s $WINID($cname)
set sme $T(Sname)
set w .$s

# if the source already has a window then just update
# images (textvariable look after the other updates),
# otherwise create the window.

if [winfo exists $w] {
   wm title $w "$PTS($cname,n)"
$w.door.icon configure -image b$PTS($cname,doorstate)
$w.info.hw.mic configure -image mic$PTS($cname,havemic)
$w.info.hw.sp configure -image speaker$PTS($cname,havespeaker)
$w.info.hw.v configure -image v$PTS($cname,havevideo)
} else {
toplevel $w
wm title $w "$PTS($cname,n)"
wm iconname $w "$sme: $PTS($cname,n)"
wm resizable $w 0 0
frame $w.info  -relief ridge -bd 3
pack $w.info -fill x

frame $w.info.lt -relief sunken -bd 2
label $w.info.lt.l -text "Local Time:" -anchor w
label $w.info.lt.time -textvariable PTS($cname,t)
pack $w.info.lt -side top -fill x
pack $w.info.lt.l $w.info.lt.time -side left -fill x -expand 1

frame $w.info.status -relief sunken -bd 2
pack $w.info.status -fill x -side top
label $w.info.status.l -textvariable PTS($cname,status) -anchor w
pack $w.info.status.l -fill x

frame $w.info.hw -relief sunken -bd 2
pack $w.info.hw -side top -ipadx 2 -ipady 2 -fill x -expand 1
label $w.info.hw.l -text "A/V Hardware:"
label $w.info.hw.mic -image mic$PTS($cname,havemic)
label $w.info.hw.sp -image speaker$PTS($cname,havespeaker)
label $w.info.hw.v -image v$PTS($cname,havevideo)
pack $w.info.hw.l $w.info.hw.mic $w.info.hw.sp $w.info.hw.v -side left -expand 1

frame $w.door -borderwidth 2 -relief ridge -bd 3
label $w.door.motd -textvariable PTS($cname,note)
label $w.door.icon -image b$PTS($cname,doorstate)
pack $w.door.motd
pack $w.door.icon
pack $w.door -expand 1 -fill x

frame $w.m -borderwidth 2
menubutton $w.m.b -relief raised \
-menu $w.m.b.options \
-text {Actions...}

menu $w.m.b.options
$w.m.b.options add command \
-command "glance $cname" \
-label {Glance}

$w.m.b.options add command \
-command "knock $cname" \
-label {Knock}
$w.m.b.options add command \
-command "connect $cname" \
-label {Knock and Enter}

$w.m.b.options add separator

$w.m.b.options add command \
-command "workspace $cname" \
-label {Open whiteboard}

$w.m.b.options add separator

$w.m.b.options add command \
-command "do_Email $cname $PTS($cname,e)" \
-label {Send Email}

$w.m.b.options add command \
-command "call_Browser $PTS($cname,u)" \
-label {View WWW homepage}

pack $w.m.b -fill x
pack $w.m -fill x
}
fb ""
}

proc glance {s} {
   global PTS T SIP GAP SIP_TAB env
   set result 1
   if ![string match $PTS($s,havevideo) 1] {
     set result [tk_dialog {.hum} {Are you sure?} {User cannot send video} {} {0} {Cancel} {Try anyway}]
   }
   if [string match $result 1] {
       set S(To) $s
set S(Type) REQ
set S(tpservice) GLANCE

# increment the request counter
incr SIP(ReqCount)

set ID "$SIP(Logname)@$GAP(IP)/[get_SIP_ID]"
set SIP($ID,Count) 0

set GAP(BusyWith) $PTS($s,cn)
set SIP_TAB($ID,ResendAfter) 1000
send_SIP S $ID
fb "Sent GLANCE, waiting for reply"
}
}

proc knock {s} {
   global PTS T SIP GAP SIP_TAB MEDIA env
set S(To) $s
set S(Type) REQ
set S(tpservice)  KNOCK
fb "Waiting for knock response"

incr SIP(ReqCount)
set ID "$SIP(Logname)@$GAP(IP)/[get_SIP_ID]"
set SIP_TAB($ID,ResendAfter) 1000
set SIP($ID,Count) 0
set GAP(BusyWith) $PTS($s,cn)
send_SIP S $ID
}

proc connect {s} {
   global MEDIA PTS T SIP GAP SIP_TAB env
   fb {Waiting to create connection}
incr SIP(ReqCount)
set ID "$SIP(Logname)@$GAP(IP)/[get_SIP_ID]"
set SIP($ID,Count) 0
set GAP(BusyWith) $s
   set SIP_TAB($ID,ResendAfter) 1000
set S(To) $s
set S(Type) REQ
set S(tpservice) CONNECT
send_SIP S $ID
}

proc workspace {s} {
   global T SIP GAP SIP_TAB env

set S(To) $s
set S(Type) REQ
set S(tpservice)  WHITEBOARD
# there's no point launching this until we get a reply
incr SIP(ReqCount)
set ID "$SIP(Logname)@$GAP(IP)/[get_SIP_ID]"
set SIP_TAB($ID,ResendAfter) 1000
set SIP($ID,Count) 0
send_SIP S $ID
fb "Sent WHITEBOARD, waiting for reply"
}
```

Various dialog procedures:

```
#-------------------
```

```
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------

#------------------------------------------
# ui-dialogs.tcl
#
# procs for drawing various dialog boxes - also procs
# called by them
#--------------------------------------------------------

proc newuserDialog {} {
global T

set w .intro
toplevel $w
wm title $w "$T(Lname): New User!"
frame $w.t
frame $w.b
pack $w.t
pack $w.b
label $w.t.label  -bitmap error -anchor nw
pack $w.t.label -side left -fill y
label $w.t.msg -width 40 -relief ridge -justify left \
-wraplength 250 \
-text "Aha, a NEW USER! \n\n There are a number of things \
that $T(Lname) needs to know before it can start using the \
GAP protocol. Since $T(Lname) can't find the file: $T(preferences_file)\
, you will need to edit the 'best guess' \
 entries that have just been created for you."
pack $w.t.msg -side left
button $w.b.ok -width 10 -text {OK} \
-command "destroy $w; newuserPrefs"
button $w.b.q -text "Quit now" -command exit
pack $w.b.ok -side right
pack $w.b.q -side left
}

proc newuserPrefs {} {
build_ui_preferencesWin
.p.b.cancel configure -state disabled
tkwait window .p
Init
}

proc initDialog {img msg} {
global T
toplevel .init
set w .init
set T(initDialogWin) $w
wm title $w "$T(Lname): Starting Up"
frame $w.f  -relief ridge -bd 2
set w $w.f
label $w.icon -image $img
label $w.fb -textvariable T(fb) -bg DarkSlateBlue -fg white -width 30
pack $w.icon $w.fb -side top -fill x -fill y -expand 1
pack $w -fill x -fill y -expand 1
}

proc waitDialog {img msg} {
global T
if [winfo exists .wait] {destroy .wait}
toplevel .wait
set w .wait
set T(waitDialogWin) $w
wm title $w "$T(Lname): Please Wait...."
frame $w.f
set w $w.f
label $w.icon -relief ridge -bitmap $img
label $w.msg -text $msg
pack $w.icon $w.msg -side top -fill x -fill y -expand 1
pack $w -fill x -fill y
}

proc connectDialog {s id} {
global T GAP
toplevel .cd
set w .cd
set T(connectDialogWin) $w
wm title $w "$T(Lname): Connection Control"
frame $w.text -relief raised -bd 2
frame $w.but

pack $w.text -fill both -expand 1
pack $w.but -fill both -expand 1

label $w.text.icon -bitmap info -anchor w
label  $w.text.t -wraplength 200
pack $w.text.icon -side left -fill y
pack $w.text.t -side left  -fill both -expand 1 -padx 2 -pady 2
set msg {Use the button below to close the connection}
button $w.but.c -text {Close Connection} -command "close_connection $s $id"
$w.text.t configure -text $msg
if $GAP(Have_Video) {
button $w.but.pv -text {Pause Video} -command "video_toggle_pause $w"
set msg {Use the buttons below to pause/un-pause the video, and to close the connection.}
pack $w.but.pv -side left
pack $w.but.c -side right
} else {
   pack $w.but.c
   }
}

proc video_toggle_pause {w} {
global videoIsPaused
if $videoIsPaused {
confbus video tp_start_sending
set videoIsPaused 0
$w.but.pv configure -text {Pause Video}
} {
confbus video tp_stop_sending
set videoIsPaused 1
$w.but.pv configure -text {Start Video}
}
}

proc close_connection {s id} {
global PTS T GAP USCP
if $GAP(Have_Video) {confbus video tp_stop_sending}
set msg [list tp_exit]
confbus audio $msg
unset T(vatPID)
unset T(vat)
if [winfo exists $T(connectDialogWin)] {destroy $T(connectDialogWin)}
set S(To) $s
set S(ID) $id
set S(Type) DISCONNECT
send_USCP S
fb ""
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)
}

proc todoDialog {url label msg} {
global T
toplevel .todo
set w .todo
wm title $w "$T(Lname): Unsupported Feature"
frame $w.top -bd 2
frame $w.but
set b $w.but
set w $w.top

pack $w -fill both -expand 1
pack $b

label $w.icon -bitmap warning -anchor w
frame $w.t -bd 2 -relief ridge
label $w.t.l -text $label
entry $w.t.e -justify center
$w.t.e insert end $url
label $w.t.t -text $msg -anchor w -wraplength 300
pack $w.icon -side left -fill y -padx 2 -pady 2
pack $w.t -side left  -fill both -expand 1
pack $w.t.l -fill x -ipadx 2 -ipady 2
pack $w.t.e -fill x -ipadx 2 -ipady 2
pack $w.t.t -fill x -ipadx 2 -ipady 2
button $b.d -text Dismiss -command "destroy .todo"
pack $b.d
}

proc errorDialog {t} {
# create a window to monitor $errorInfo variable
global T errorInfo
toplevel .ew
set w .ew
wm title $w "$T(Lname): Something bad happened"
frame $w.f
set w .ew.f
label $w.icon -bitmap warning
label $w.msg -relief ridge -textvariable $t -fg red
pack $w.icon $w.msg -side left
button .ew.d -text Dismiss -command "destroy .e"
pack .ew.f -fill x -fill y
pack .ew.d -side right
}

# dialog.tcl --
#
# This file defines the procedure tk_dialog, which creates a dialog
# box containing a bitmap, a message, and one or more buttons.
#
# @(#) dialog.tcl 1.19 95/09/27 09:51:36
#
# Copyright (c) 1992-1993 The Regents of the University of California.
# Copyright (c) 1994-1995 Sun Microsystems, Inc.
#
# See the file "license.terms" for information on usage and redistribution
# of this file, and for a DISCLAIMER OF ALL WARRANTIES.
#

#
```

```
# tk_dialog:
#
# This procedure displays a dialog box, waits for a button in the dialog
# to be invoked, then returns the index of the selected button.
#
# Arguments:
# w -Window to use for dialog top-level.
# title -Title to display in dialog's decorative frame.
# text -Message to display in dialog.
# bitmap -BitGAP to display in dialog (empty string means none).
# default -Index of button that is to display the default ring
# (-1 means none).
# args -One or more strings to display in buttons across the
# bottom of the dialog box.

# Edited a bit by Ben Anderson (LUT, 1995) to display a canvas to represent
# a timer. When the timer runs out, the dialog sends a mouse up event
# to the default button (be careful here...!)

proc timedDialog {w title text bitmap t default args} {
    global tkPriv T

    # 1. Create the top-level window and divide it into three
    # parts.

    catch {destroy $w}
    toplevel $w -class Dialog
    wm title $w $title
    wm iconname $w Dialog
    wm protocol $w WM_DELETE_WINDOW { }
    wm transient $w [winfo toplevel [winfo parent $w]]
    frame $w.top -relief raised -bd 1
    pack $w.top -side top -fill both
    frame $w.mid -relief flat
    pack $w.mid -fill both
    frame $w.bot -relief raised -bd 1
    pack $w.bot -side bottom -fill both

    # 2. Fill the top part with bitmap and message (use the option
    # database for -wraplength so that it can be overridden by
    # the caller).

    option add *Dialog.msg.wrapLength 3i widgetDefault
    label $w.msg -justify left -text $text \
    -font -Adobe-Times-Medium-R-Normal--*-180-*-*-*-*-*-*
    pack $w.msg -in $w.top -side right -expand 1 -fill both -padx 3m -pady 3m
    if {$bitmap != ""} {
label $w.bitmap -bitmap $bitmap
pack $w.bitmap -in $w.top -side left -padx 3m -pady 3m
    }

    # 2.1 Put the canvas in the middle and draw a small rectangle
    # get the canvas width from the time value

    canvas $w.canvas -relief sunken -width $t\c -height 0.5c -bd 2
  $w.canvas create rectangle 0 0 0 0 -tags rect
    pack $w.canvas -in $w.mid -fill x

    # 3. Create a row of buttons at the bottom of the dialog.

    set i 0
    foreach but $args {
button $w.button$i -text $but \
-command "set tkPriv(button) $i; after cancel do_Timer"
if {$i == $default} {
    frame $w.default -relief sunken -bd 1
    raise $w.button$i $w.default
    pack $w.default -in $w.bot -side left -expand 1 -padx 3m -pady 2m
    pack $w.button$i -in $w.default -padx 2m -pady 2m
    bind $w <Return> "$w.button$i flash; set tkPriv(button) $i; after cancel do_Timer"
} else {
    pack $w.button$i -in $w.bot -side left -expand 1 \
    -padx 3m -pady 2m
}
incr i
    }

    # 4. Withdraw the window, then update all the geometry information
    # so we know how big it wants to be, then center the window in the
    # display and de-iconify it.

    wm withdraw $w
    update idletasks
    set x [expr [winfo screenwidth $w]/2 - [winfo reqwidth $w]/2 \
    - [winfo vrootx [winfo parent $w]]]
    set y [expr [winfo screenheight $w]/2 - [winfo reqheight $w]/2 \
    - [winfo vrooty [winfo parent $w]]]
    wm geom $w +$x+$y
    wm deiconify $w

    # 5. Set a grab and claim the focus too.

    set oldFocus [focus]
    set oldGrab [grab current $w]
    if {$oldGrab != ""} {
set grabStatus [grab status $oldGrab]
    }
    grab $w
    if {$default >= 0} {
```

```
focus $w.button$default
    } else {
focus $w
    }

    # 6. Wait for the user to respond, then restore the focus and
    # return the index of the selected button.  Restore the focus
    # before deleting the window, since otherwise the window manager
    # may take the focus away so we can't redirect it.  Finally,
    # restore any grab that was in effect.

    set T(DialogTimer) $t
    after 500 do_Timer $w $t $default

    tkwait variable tkPriv(button)
    after cancel do_Timer
    catch {focus $oldFocus}
    destroy $w
    if {$oldGrab != ""} {
if {$grabStatus == "global"} {
    grab -global $oldGrab
} else {
    grab $oldGrab
}
    }
    return $tkPriv(button)
}

proc do_Timer {w t d} {
global tkPriv T
# puts stdout "Doing timer at time = $t"
set length [expr ($T(DialogTimer) - $t)]
if [winfo exists $w.canvas] {
     $w.canvas delete rect
     $w.canvas create rectangle 0.2c 0.2c $length\c 0.5c -fill red -tags rect
    }
set t [expr $t - 0.5]
   if {$t < 0} {set tkPriv(button) $d} {after 500 do_Timer $w $t $d}
}

#---------------------------------------------------------
# Creates the preferences window
#---------------------------------------------------------

proc Save_Prefs {} {
global GAP T MEDIA
fb "Saving preferences"
set fp [open $T(preferences_file) w]
puts $fp "# Teleport preferences file, do not edit or overwrite!"
puts $fp "global GAP"
puts $fp "set GAP(Note) {$GAP(Note)}"
puts $fp "set GAP(Name) {$GAP(Name)}"
puts $fp "set GAP(Uri) {$GAP(Uri)}"
puts $fp "set GAP(Email) {$GAP(Email)}"
puts $fp "set GAP(Pots) {$GAP(Pots)}"
puts $fp "set GAP(Have_Video) {$GAP(Have_Video)}"
puts $fp "set GAP(Have_Mic) {$GAP(Have_Mic)}"
puts $fp "set GAP(Have_Speaker) {$GAP(Have_Speaker)}"
foreach i [array names MEDIA] {
puts $fp "set MEDIA($i) {$MEDIA($i)}"
}
puts $fp "set T(icon_vol) {$T(icon_vol)}"
close $fp
fb "Save complete"
set A(To) all
}

# create the preferences window - this window has three main panels
# between which the user can toggle
proc build_ui_preferencesWin {} {
global T GAP
set T(dfgc) red
fb "Getting Preferences"
toplevel .p
wm withdraw .p
wm title .p "$T(Lname): Preferences"
wm iconname .p "$T(Sname): Prefs"
# create 2 frames, one for the info, one for buttons
frame .p.t
frame .p.b
button .p.t.pd -text {Personal} -relief raised\
 -disabledforeground $T(dfgc)\
 -state disabled\
 -command {
.p.b.help configure -command {help prefs}
#.p.t.ha configure -state active
.p.t.hc configure -state active
.p.t.int configure -state active
.p.t.pd configure -state disabled
build_ui_personalDetailsWin}

#  button .p.t.ha -text {Media Tools}\
#  -disabledforeground $T(dfgc)\
#  -command {
#  .p.b.help configure -command {help ha}
#  .p.t.ha configure -state disabled
#  .p.t.hc configure -state active
#  .p.t.int configure -state active
#  .p.t.pd configure -state active
#  build_ui_appsWin}

button .p.t.hc -text {Hardware}\
-disabledforeground $T(dfgc)\
-command {
.p.b.help configure -command {help hc}
#.p.t.ha configure -state active
.p.t.hc configure -state disabled
.p.t.int configure -state active
.p.t.pd configure -state active
build_ui_hardwareWin}

button .p.t.int -text {Interface}\
-disabledforeground $T(dfgc)\
-command {
.p.b.help configure -command {help interface}
#.p.t.ha configure -state active
.p.t.hc configure -state active
.p.t.pd configure -state active
.p.t.int configure -state disabled
build_ui_interfaceWin}

pack .p.t.pd .p.t.hc .p.t.int -side left

# three buttons to cancel, to save and to get help
button .p.b.help -width 10 \
-text {Help} \
-command {help prefs}
button .p.b.cancel -width 10\
-text {Cancel} \
-command {destroy .p}
button .p.b.save -width 10\
-text {Save} \
-command {
Save_Prefs
destroy .p}

pack .p.b.help -side left -padx 2 -pady 2
pack .p.b.cancel -side left -padx 2 -pady 2
pack .p.b.save -side right -padx 2 -pady 2

#pack the frames
pack .p.t -fill x
pack .p.b -fill x -side bottom

build_ui_personalDetailsWin
wm deiconify .p
}

proc build_ui_personalDetailsWin {} {
# create 5 frames, each with a label and an entry widget
# to hold user's preferences
global GAP T
if [winfo exists .p.int] {destroy .p.int}
if [winfo exists .p.m] {destroy .p.m}
if [winfo exists .p.h] {destroy .p.h}
if ![winfo exists .p.u] {
set width 20
frame .p.u -relief groove -bd 2
pack .p.u -after .p.t -fill x

# Name
frame .p.u.name
label .p.u.name.l -text {Name:} -width $width -anchor w
entry .p.u.name.e -relief sunken -font $T(ef) \
-width 40 -background $T(bgc) \
-textvariable GAP(Name)

bind .p.u.name.e  <Return> {
   set S(To) all
   send_GAP INFO
   }

pack .p.u.name -fill x
pack .p.u.name.l -side left
pack .p.u.name.e -side right -expand 1 -fill x

# Email
frame .p.u.email
label .p.u.email.l -text {Email:} -width $width -anchor w
entry .p.u.email.e -relief sunken \
-textvariable GAP(Email) \
-font $T(ef)  -background $T(bgc)
pack .p.u.email -fill x
pack .p.u.email.l -side left
pack .p.u.email.e -side right -expand 1 -fill x

# WWW URL
frame .p.u.url
label .p.u.url.l -text {WWW Home Page:} -width $width -anchor w
entry .p.u.url.e -relief sunken \
-textvariable GAP(Url)\
-font $T(ef) -background $T(bgc)
pack .p.u.url -fill x
pack .p.u.url.l -side left
pack .p.u.url.e -side right -fill x -expand 1

# POTS phone number
frame .p.u.pots
```

```
label .p.u.pots.l -text {Telephone:} -width $width -anchor w
entry .p.u.pots.e -relief sunken \
-textvariable GAP(Pots)\
-font $T(ef) -background $T(bgc)
pack .p.u.pots -fill x
pack .p.u.pots.l -side left
pack .p.u.pots.e -side right -fill x -expand 1

# MOTD (displayed over door)
frame .p.u.m
label .p.u.m.l -text {Message above Door:} -width $width -anchor w
entry .p.u.m.e -relief sunken \
-textvariable GAP(Note)\
-font $T(ef) -background $T(bgc)
pack .p.u.m -fill x
pack .p.u.m.l -side left
pack .p.u.m.e -side right -fill x -expand 1
}
fb ""
}

proc build_ui_appsWin {} {
global GAP FORMATS T MEDIA APPS
if [winfo exists .p.int] {destroy .p.int}
if [winfo exists .p.u] {destroy .p.u}
if [winfo exists .p.h] {destroy .p.h}
if ![winfo exists .p.m] {
frame .p.m
pack .p.m -after .p.t

# Video
frame .p.m.v -relief ridge -bd 2
label .p.m.v.t -text {Video:} -width 20
pack .p.m.v.t

set w .p.m.v.app
frame $w
pack $w -fill x -expand 1
label $w.l -text {Application:} -anchor w -width 15
pack $w.l -side left
foreach i $APPS(video) {
set f [string tolower $i]
radiobutton $w.$f -text $i \
-variable MEDIA(videoA) \
-command ""\
-relief flat -value $i -width 5 -anchor w
pack $w.$f -side left
}

# build_ui_Formats video

set w .p.m.v.p
frame $w
pack $w -fill x -expand 1
label $w.pt -text {Port:} -anchor w -width 15
label $w.po -textvariable MEDIA(videoP)
pack $w.pt -side left
pack $w.po -side left
set w .p.m.v.bw
frame $w
pack $w -fill x -expand 1
label $w.l -text "Max B/W (kb/s):" -anchor sw -width 15
scale $w.s -from 0 -to 500 -orient horizontal\
-length 200 -variable MEDIA(videoR)
pack $w.l  -side left
pack $w.s -fill x -side left -expand 1
pack .p.m.v  -fill x -padx 2 -pady 2

# Audio
set w .p.m.a
frame $w -relief ridge -bd 2
label $w.t -text {Audio:} -width 20
pack $w.t

set w $w.app
frame $w
pack $w -fill x -expand 1
label $w.l -text {Application:} -anchor w -width 15
pack $w.l -side left
foreach i $APPS(audio) {
set f [string tolower $i]
radiobutton $w.$f -text $i \
-variable MEDIA(audioA) \
-command ""\
-relief flat -value $i -width 5 -anchor w
pack $w.$f -side left
}

#build_ui_Formats audio

set w .p.m.a.p
frame $w
pack $w -fill x -expand 1
label $w.pt -text {Port:} -anchor w -width 15
label $w.po -textvariable MEDIA(audioP)
pack $w.pt -side left
pack $w.po -side left
pack .p.m.a  -fill x -padx 2 -pady 2
```

```
# Workspace tools
frame .p.m.ws -relief ridge -bd 2
set w .p.m.ws
label $w.l -text {Workspace Tools:} -width 20
pack $w.l
# this will have to cycle through a number of them
frame $w.wb
pack $w.wb -fill x -expand 1
checkbutton $w.wb.n -text {LBL whiteboard} \
-relief flat -width 15 -anchor w\
-onvalue WB -variable MEDIA(whiteboardF)
label $w.wb.l -text {Port:}
label $w.wb.p -textvariable MEDIA(whiteboardP)
pack $w.l
pack $w.wb.n $w.wb.l $w.wb.p -side left
pack $w -fill x -padx 2 -pady 2
}
fb ""
}

proc build_ui_hardwareWin {} {
# hardware configs
if [winfo exists .p.int] {destroy .p.int}
if [winfo exists .p.m] {destroy .p.m}
if [winfo exists .p.u] {destroy .p.u}
if ![winfo exists .p.h] {
global ef bgc GAP
frame .p.h -relief groove -bd 2
pack .p.h -after .p.t -fill both
frame .p.h.v -relief groove -bd 2
frame .p.h.a -relief groove -bd 2

pack .p.h.v .p.h.a -side left \
-padx 2 -pady 2 \
-expand 1\
-fill both

label .p.h.v.l -text {Video}
radiobutton .p.h.v.0 -image v0 -text {Can't Send}\
-value 0 -variable GAP(Have_Video)
radiobutton .p.h.v.1 -image v1 -text {Can  Send}\
-value 1 -variable GAP(Have_Video)

pack .p.h.v.l .p.h.v.0 .p.h.v.1 -fill x -expand 1

label .p.h.a.l -text {Audio}
radiobutton .p.h.a.0 -image speaker0 \
-text {No speaker}\
-value 0 -variable GAP(Have_Speaker)
radiobutton .p.h.a.1 -text {Have speaker}\
-image speaker1 -value 1\
 -variable GAP(Have_Speaker)
radiobutton .p.h.a.2 -image mic0\
-text {No microphone}\
-value 0 -variable GAP(Have_Mic)
radiobutton .p.h.a.3 -image mic1\
-text {Have microphone}\
-value 1 -variable GAP(Have_Mic)
pack .p.h.a.l .p.h.a.0 .p.h.a.1 .p.h.a.2\
 .p.h.a.3 -fill x -expand 1
}
fb ""
}

proc build_ui_interfaceWin {} {
# interface configs
if [winfo exists .p.m] {destroy .p.m}
if [winfo exists .p.u] {destroy .p.u}
if [winfo exists .p.h] {destroy .p.h}
if ![winfo exists .p.int] {
   frame .p.int
   pack .p.int -fill x -expand 1
scale .p.int.s -from 100 -to 0 -variable T(icon_vol) \
    -showvalue 1 -orient vertical
label .p.int.l -text {Audio Icon Volume}
pack .p.int.s -fill y -side left
pack .p.int.l -fil x -side left
}
wm deiconify .p
fb ""
}
proc build_ui_Formats {m} {
global FORMATS MEDIA
switch -exact -- $m {
audio {
if [winfo exists .p.m.a.format] {destroy .p.m.a.format}
set w .p.m.a.format
frame $w
pack $w -fill x -expand 1 -after .p.m.a.app
label $w.t -text {Format:} -anchor w -width 15
pack $w.t -side left
set app $MEDIA(audioA)
foreach i $FORMATS($app) {
set f [string tolower $i]
radiobutton $w.$f -text $i \
-variable MEDIA(audioF)\
-relief flat -value $i -width 5 -anchor w
pack $w.$f -side left
}
}
video {
if [winfo exists .p.m.v.format] {destroy .p.m.v.format}
set w .p.m.v.format
frame $w
pack $w -fill x -expand 1 -after .p.m.v.app
label $w.t -text {Format:} -anchor w -width 15
pack $w.t -side left
set app $MEDIA(videoA)
foreach i $FORMATS($app) {
set f [string tolower $i]
radiobutton $w.$f -text $i \
-variable MEDIA(videoF)\
-relief flat -value $i -width 5 -anchor w
pack $w.$f -side left
}
}
}
}
```

The help dialogs:

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------

#------------------------------------------
# ui-help.tcl
#------------------------------------------

proc help {subject} {
global URLS helptext T
set w .help
set p $w.f.pics
if ![winfo exists $w] {
toplevel $w
wm iconname $w "$T(Sname): Help"
wm withdraw $w
frame $w.f
pack $w.f
frame $p
pack  $p -side left -fill y
label $p.icon -image tp
pack $p.icon -padx 2 -pady 2 -fill x
label $w.f.l -width 40 -wraplength 3i -relief ridge -justify left -textvariable helptext
button $w.b -text {Dismiss} -command "destroy $w"
button $w.cb -text {See WWW pages}
pack $w.f.l -side right -fill x -fill y -padx 2 -pady 2
pack $w.cb -side left -expand 1
pack $w.b -side right -expand 1
wm deiconify $w
}
switch -exact -- $subject {
prefs {
wm title $w "Help on... Preferences"
set helptext "Please provide this information, it will identify you in any GAP-based application, \
as well as providing the means for others to contact you. If you have not used an GAP based application \
before, a 'best guess' has been entered which will probably need to be edited. These resources will be saved \
to the file ~/.GAPdefaults so you should not have to re-enter them. \n\n More information on GAP can be \
found at the url $URLS(GAP)"
$w.cb configure -state normal
$w.cb configure -command "
call_Browser $URLS(GAP)
destroy $w"
}
ha {
wm title $w "Help on... Media Tools"
set helptext "$T(Lname) uses these tools to enable you to communicate and interact with \
other participants. $T(Lname) expects to find the tools listed via your \$PATH, and will report an \
error if it cannot do so.\n One day $T(Lname) will allow you to define your own helper applications and/or \
add new ones.\n\nMore information on the audio and video tools can be found\
at the url $URLS(mbone)"
$w.cb configure -state normal
$w.cb configure -command "
call_Browser $URLS(mbone)
destroy $w"
}
hc {
wm title $w "Help on... Hardware Capabilities"
set helptext "This dialog allows you to be explicit about the audio/video hardware you have at your \
disposal - you may have a video card but no camera, or a speaker but no microphone for example. The icons\
 used here are replicated in the actual user-interface so you can see who has what capability before\
  trying to contact them..."
$w.cb configure -state disabled
}
info {
wm title $w "Help on... TelePort"
set helptext "TelePort is an experimental system built as a vehicle for a PhD research programme into\
aspects of user interfaces for broadband telecommunications services. More information on\
TelePort can be found at the url $URLS(teleport) \n\n Author = Ben Anderson,\
Dept Computer Studies, Loughborough University of Technology, Loughborough, Leics, UK.\
\n\nErrors and bug reports to: B.Anderson@lut.ac.uk\
\n\nThis research is supported by a Research Scholarship funded by British Telecommunications plc."
$w.cb configure -command "call_Browser $URLS(teleport); destroy $w"
}
email {
wm title $w "Help on... Email"
set helptext {TelePort has no built in email tool - you will\
need to use the X-selection to copy a participant's email\
address to your email tool.}
$w.cb configure -state disabled
}
mbone {
wm title $w "Help on...MBONE"
set helptext "The MBONE is a multicast backbone implemented within the current Internet\
using unicast tunnelling of multicast IP packets. This allows efficient, scalable distribution of data\
which is particularly useful for multiparty conferencing and broadcasting.\n\nMore information\
on a number of applications currently under development for the MBONE can be found at: $URLS(mbone)"
$w.cb configure -command "call_Browser $URLS(mbone); destroy $w"
}
default {
wm title $w {Default Help}
set helptext "There is no specific help for this context. You can get more detailed information about\
 this application from the online help pages at $URLS(teleport)"
$w.cb configure -command "call_Browser $URLS(teleport); destroy $w"
}
}
}
```

Image loading:

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#-----------------------------------------

#---------------------------------------------------------
# ui-images.tcl
#
# tcl script to load all gifs in picsnds directory
# Looks for ".gif" extension (exact match only) and ignores
# everything else.
#---------------------------------------------------------

proc load_Images {} {
# location of sounds and images
global picdir  env
set picdir "$env(TPHOME)/picsnds"

# automagically load all gifs
set popd [pwd]
cd $picdir
set imagelist [glob *.gif]
set loaded [image names]
foreach i $imagelist {
regsub ".gif" $i "" p
 if {[lsearch $loaded $p] == -1} {
image create photo "$p" -file $i
# puts stdout "Loading $i"
} else {
    #       puts stdout "$p already loaded, skipping"
}
 }

cd $popd
 }
```

C.3.4 Tcl-DP initialisation of network sockets

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#-----------------------------------------

#---------------------------------------------------------
# tcldp.tcl
# All the Tcl-DP network procedures are defined in this file
#---------------------------------------------------------

proc new {what N} {
global T
upvar $N d
switch -exact -- $what {
```

```
GAP {
set T(gap) [lindex [dp_connect -mudp $d(address) \
 $d(port) \
 $d(ttl)] 0]
set T(net_addr) [dp_address create $d(address) \
$d(port)]
# what to do if there's data on the net port
dp_filehandler $T(gap) r GAP_receive
}
confbus {
set T(confbus) [lindex [dp_connect -mudp $d(address) \
 $d(port) \
 $d(ttl)] 0]
set T(confbus_addr) [dp_address create $d(address) \
$d(port)]
# what to do if there's data on the confbus port
dp_filehandler $T(confbus) r cb_Receive
}
SIP {
   set T(sip) [lindex [dp_connect -udp $d(port)] 0]
   dp_filehandler $T(sip) r SIP_receive
   }
}
}

proc GAP_send {content} {
global T
dp_sendTo $T(gap) $content $T(net_addr)
#puts "Just multicast $content"
}

proc SIP_send {content id} {
global T SIP_srcs
dp_sendTo $T(sip) $content $SIP_srcs($id)
#puts "Just unicast sent $content"
}

proc USCP_send {content id} {
   global T SIP_srcs
   dp_sendTo $T(sip) $content $SIP_srcs($id)
}

proc SIP_receive {mode ufp} {
   global T SIP SIP_srcs USCP
   # puts stdout "Got $ufp"
set incoming [dp_receiveFrom $ufp]
set from [lindex $incoming 0]
set addr [dp_address info $from]
set temp [lrange $incoming 1 end]
set p [string trim $temp "/{ /}"]
set d [split $p \n]
set IP [lindex $addr 0]
set firstline [lindex $d 0]
set proto [lindex $firstline 0]
if [string match $proto $SIP(Version)] {
   # get the id
set id_line  [split [lindex $d 3] "="]
set id [lindex $id_line 1]

# set the tcl-dp address id for sending replies
set SIP_srcs($id) $from

set type [lindex  $firstline 1]
process_SIP_$type $id $d
} elseif [string match $proto $USCP(Version)] {
   set type [lindex $firstline 1]
   process_USCP_$type $d
   } else {warn "Received unknown protocol packet:$proto"}

}

proc GAP_receive {mode mfp} {
global T GAP
set incoming [dp_receiveFrom $mfp]
# if sent by tcl-dp, first element is a tcl-dp address identifier
# that should be ignored - might need to tidy this up to deal with
# GAP packets from a different tool.
set temp [lrange $incoming 1 end]
set p [string trim $temp "/{ /}"]
set d [split $p \n]
set v [split [lindex $d 0] =]
if [regexp $GAP(Version) $v] {
receive_GAP $d
} else {
warn "Received unknown protocol packet: [lindex [lindex $v 1] 0]"
}
}
```

C.3.5 Session Invitation Protocol

Parsers for SIP and also procedures to deal with responses.

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
```

```
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#-----------------------------------------

#-----------------------------------------------------------------
# Procs for processing Session Invitation Protocol Packets
#
# Based on SIP v 1.0 IETF-DRAFT-MMUSIC-SIP-00
#-----------------------------------------------------------------
proc get_SIP_ID {} {
   global GAP
   # returns a large random number
   return [expr int(100000 * [random])]
}

proc process_SIP_REQ {id d} {
global PTS GAP SIP_TAB USCP SIP T
set me $GAP(IP)
set busy $USCP(CurrentSession)

# split the SIP data packet into header, session desc, media desc
set h [lrange $d 1 5]
set sd [lrange $d 6 13]
set md [lrange $d 14 end]
set src [lindex [split $id "/"] 0]
set origin [lindex [split [lindex $d 1] = ] 1]
set version [lindex $origin 2]
set result [catch {lindex $SIP_TAB($id,v) 2} oldversion]
if [string match $result 0] {
# we've seen this id before
if [string match $version $oldversion] {
   # new version so need to parse it
   warn "Got new version of request $id"
} else {
   # ignore it
   say "Ignoring repeated request id = $id"
}
} else {
# we haven't seen this id before
unpack_SIP_Header $id $h
unpack_SIP_REQ $id $sd $md

if {[string match $busy 0] || [string match $busy $id]} {
   # we're not busy , or we're busy with sender, so proceed
   # update the service control table
   set USCP(CurrentSession) $id

if [info exists SIP_TAB($id,session_attr_tpservice)] {
   # then it is a teleport service
   received_$SIP_TAB($id,session_attr_tpservice) $id $src
} else {
   # it wasn't
   received_SIP_REQ $id
   }
   } else {
   # if none of them were true then
   # we're busy with someone else so send BUSY
   set S(Type) REP
   set S(Category) 2
   set S(ReplyTypeName) BUSY
send_SIP S $id
}
}
}

proc unpack_SIP_Header {id h} {
global SIP_TAB
set SIP_TAB($id,header) [lindex $h 0]
foreach line $h {
set l [split $line =]
set type [lindex $l 0]
set data [lindex $l 1]
set SIP_TAB($id,$type) $data
}
}

proc unpack_SIP_REQ {id sd md} {
   global SIP SIP_TAB
   #say "Processing SIP request: Desc=$sd Media=$md"
   # the rest of it is an SDP packet.
   # 1. Unpack the description
    foreach line $sd {
   set l [split $line "="]
   switch -exact -- [lindex $l 0] {
     a {
      set d [split [lindex $l 1] ":"]
       set result [llength $d]
       set flag [lindex $d 0]
       if [string match $result  1] {
          set value 1
       } else {
         set value [lrange $d 1 end]
        }
       set SIP_TAB($id,session_attr_$flag) $value
       }
   default {
     set SIP_TAB($id,[lindex $l 0]) [lindex $l 1]
     }
   }
   }

   # 2. Get the media info
   foreach line $md {
   set l [split $line "="]
   set type [lindex $l 0]
   set desc [lindex $l 1]
   switch -exact -- $type {
   m {
     # media field
   set media [lindex $desc 0]
   set SIP_TAB($id,$media\Port) [lindex $desc 1]
   set SIP_TAB($id,$media\Transport) [lindex $desc 2]
set SIP_TAB($id,$media\Formats) [lrange $desc 3 end]
}
b {
  # bandwidth field
   set SIP_TAB($id,$media\_b) [lindex [split $desc ":"] 1]
}
a {
   # attribute field
   set d [split $desc ":"]
       set result [llength $d]
       set flag [lindex $d 0]
       if [string match $result 1] {
        set value 1
       } else {
         set value [lrange $d 1 end]
          }
       set SIP_TAB($id,media_attr_$flag) $value
}
default {
   # any other field
   set SIP_TAB($id,$media\_$type) $desc
}
}
}
}

proc process_SIP_REP {id d} {
global PTS GAP SIP SIP_TAB T
set Category [lindex [lindex $d 0] 2]
set ReplyTypeName [lindex [lindex $d 0] 3]

foreach line $d {
set l [split $line "="]
set SIP_TAB($id,[lindex $l 0]) [lindex $l 1]
}
set src [lindex [split $SIP_TAB($id,To) "/"] 0]
set SIP_TAB($id,Got) $ReplyTypeName
fb "Response: $ReplyTypeName"
received_$ReplyTypeName $id $src
}

#-------------------------------------------------------------------
# Procedures to deal with replies
#-------------------------------------------------------------------
proc received_SUCCESS {id src} {
   global PTS T SIP_TAB SIP
     if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WHITEBOARD TEXT"] {
   if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin)}
   complete_$SIP_TAB($id,Service)\_sent $id $src
   fb "Got SUCCESS in response to $SIP_TAB($id,Sent)"
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got SUCCESS"
}
}

proc received_UNSUCCESSFUL {id src} {
global SIP_TAB PTS T
if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   set msg "Sorry, your $SIP_TAB($id,Sent) request was unsuccessful"
tk_dialog {.busy} {Request unsuccessful} $msg {info} {0} {OK}
fb ""
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got UNSUCCESSFUL"
}
}

proc received_BUSY {id src} {
   global SIP_TAB PTS T
     if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   set msg "Sorry, $PTS($src,n) did not respond - try again later"
tk_dialog {.busy} {Participant is busy} $msg {info} {0} {OK}
fb ""
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got BUSY"
}
}
proc received_DECLINE {id src} {
global SIP_TAB PTS T
if [ regexp $SIP_TAB($id,Sent) "KNOCK WORKSPACE"] {
   set msg "Sorry, your $SIP_TAB($id,Sent) request was declined by
   $PTS($s,n)"
tk_dialog {.busy} {Request declined} $msg {info} {0} {OK}
fb ""
} else {
```

```
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got DECLINE"
}
}

proc received_UNKNOWN {id src} {
global SIP_TAB PTS T
set s $SIP_TAB($id,To)
if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   set msg "Sorry, $PTS($src,n) was unknown."
tk_dialog {.busy} {User Unkown} $msg {info} {0} {OK}
fb ""
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got UNKOWN"
}
}

proc received_FAILED {id src} {
global SIP_TAB PTS
if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   set msg "Sorry, your $SIP_TAB($id,Sent) request failed because
   $SIP_TAB($id,NO)"
tk_dialog {.busy} {Request failed} $msg {info} {0} {OK}
fb ""
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got FAILED"
}
}

proc received_FORBIDDEN {id src} {
global SIP_TAB PTS
if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   msg "Sorry, your $SIP_TAB($id,Sent) request was forbidden."
tk_dialog {.busy} {Request forbidden} $msg {info} {0} {OK}
fb ""
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got FORBIDDEN"
}
}

proc received_RINGING {id src} {
global SIP_TAB PTS T
if [ regexp $SIP_TAB($id,Sent) "KNOCK WORKSPACE"] {
   play_sound knock
    if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin)}
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got RINGING"
}
}

proc received_TRYING {id src} {
global SIP_TAB PTS T
if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin)}
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got TRYING"
}
}

proc received_REDIRECT {id} {
global SIP_TAB PTS T
if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   set msg "Got a redirection for $PTS($src,n). Unimplemented"
if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin}
waitDialog info $msg
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got REDIRECT"
}
}

proc received_ALTERNATIVE {id src} {
global SIP_TAB PTS T
if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   set msg "Got an alternative for $PTS($src,n). Unimplemented"
if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin}
waitDialog info $msg
fb ""
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got ALTERNATIVE"
}
}

proc received_NEGOTIATE {id src} {
global SIP_TAB PTS T

if [ regexp $SIP_TAB($id,Sent) "GLANCE KNOCK CONNECT WORKSPACE"] {
   set msg "Got a negotiate from $PTS($src,n). Unimplemented"
if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin}
waitDialog info $msg
fb ""
} else {
return "Odd reply: sent $SIP_TAB($id,Sent) (id=$id), got NEGOTIATE"
}
}

#-------------------------------------------------------------------
# Procedures to deal with requests
#-------------------------------------------------------------------
proc received_SIP_REQ {id} {
   # this wasn't a teleport request
   say "received non-teleport request id = $id"
}

proc received_CONNECT {id src} {
   global PTS USCP T
   # Received a CONNECT
   set USCP(CurrentSession) $id
   set USCP(CurrentActivity) "In conference with $PTS($src,n)"
   set ip $PTS($src,IP)
set who $PTS($src,n)
fb "$PTS($src,n) is coming in!"
play_sound knock
complete_CONNECT_recvd $id $src
}

proc complete_CONNECT_recvd {id src} {
   global T GAP videoIsPaused MEDIA  SIP SIP_TAB PTS USCP
   # 1. Got a CONNECT
# 2. Got a KNOCK which we accepted
    play_sound door2

   fb {Launching/configuring media tools}

   #since we  received the request
   # we use the ip address given in the c field
set ip [lindex $SIP_TAB($id,c) 2]

# configure vic
set vport $SIP_TAB($id,videoPort)
# create new network video object using default bw if not set
if ![info exists SIP_TAB($id,videoBandwidth)] {
set SIP_TAB($id,videoBandwidth) 128
}
set msg [list tp_new_video $ip $vport $SIP_TAB($id,videoBandwidth)]
set result [confbus video $msg]

   # launch audio tool
   set aport $SIP_TAB($id,audioPort)
   init_Audio $ip $aport

   if [string match $result ""] {
# send the reply (might need some error handling here)
set S(Type) REP
set S(Category) "1"
set S(ReplyTypeName) "SUCCESS"
send_SIP S $id
# start sending video if we can
if $GAP(Have_Video) {
   confbus video tp_start_sending
   set videoIsPaused 0
   }
   # switch vic focus to src
set msg [list focus $src]
     confbus video $msg
      if [winfo exists .wait] {destroy .wait}
     # create connection dialog
     connectDialog $src $id
     set USCP(CurrentSession) $id
     set USCP(CurrentActivity) "In conference with $PTS($src,n)"
} else {
set S(Type) REP
set S(Category) "2"
set S(ReplyTypeName) "FAILED"
send_SIP S $id
set USCP(CurrentSession) $id
set USCP(CurrentActivity) $T(IDLE)
}
}

proc complete_CONNECT_sent {id src} {
   global T GAP videoIsPaused MEDIA  SIP SIP_TAB PTS USCP
   # 1. received SUCCESS after sending CONNECT
   # 2. received SUCCESS after sending KNOCK

    play_sound door2
     fb {Launching/configuring media tools}
     # we sent the request so we use IP address given by CH field
# of the reply
set ip [lindex $SIP_TAB($id,CH) 2]

# configure vic
set vport $SIP_TAB($id,videoPort)
# create new network video object using default bw if not set
if ![info exists SIP_TAB($id,videoBandwidth)] {
set SIP_TAB($id,videoBandwidth) 128
}
set msg [list tp_new_video $ip $vport $SIP_TAB($id,videoBandwidth)]
set result [confbus video $msg]

   # launch audio tool
   set aport $SIP_TAB($id,audioPort)
   init_Audio $ip $aport

   ##X What if a/v set-up here has failed?
   # start sending video if we can
if $GAP(Have_Video) {
   confbus video tp_start_sending
   set videoIsPaused 0
   }
   # switch vic focus to src
set msg [list focus $src]
     confbus video $msg
    if [winfo exists .wait] {destroy .wait}
     # create connection dialog
     connectDialog $src $id
     set USCP(CurrentSession) $id
     set USCP(CurrentActivity) "In conference with $PTS($src,n)"
}

proc received_KNOCK {id src} {
global PTS GAP T SIP SIP_TAB USCP

set who $PTS($src,n)
fb "$who is knocking"
set USCP(CurrentSession) $id
set USCP(CurrentActivity) "Knocked at by $PTS($src,n)"
set S(Type) REP
set S(Category) "3"
set S(ReplyTypeName) "RINGING"
send_SIP S $id

play_sound knock

set result [timedDialog .dialog {Knock Knock.. anyone there?} \
"$who is knocking.." {} 10 0 {Ignore} {Answer} {Refuse}]
switch -exact -- $result {
0 {
# send reply code BUSY
set S(Type) REP
set S(Category) "2"
set S(ReplyTypeName) "BUSY"
send_SIP S $id
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)
}
1 {
complete_CONNECT_recvd $id $src
}
2 {
  # send reply code DECLINE
set S(Type) REP
set S(Category) "2"
set S(ReplyTypeName) "DECLINE"
send_SIP S $id
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)
}
}
}

proc complete_KNOCK_sent {id src} {
global T SIP_TAB
# sent KNOCK, got SUCCESS
fb {Knock successful, launching audio tool}
    set USCP(CurrentSession) $id
complete_CONNECT_sent $id $src
}


proc received_GLANCE {id src} {
global PTS GAP MEDIA T SIP SIP_TAB USCP
fb "Glanced at"

set who $SIP_TAB($id,FR)
set USCP(CurrentSession) $id
set USCP(CurrentActivity) "Glanced at by $PTS($src,n)"
set IP [lindex $SIP_TAB($id,c) 2]

set S(Type) REP
set S(Category) "3"
set S(ReplyTypeName) "TRYING"
send_SIP S $id

if $GAP(Have_Video) {
play_sound creak
# create new network video object
set msg [list tp_new_video $IP $MEDIA(videoP) $MEDIA(videoR)]
set result [confbus video $msg]

# send the reply (might need some error handling here)

set S(Type) REP
set S(Category) "1"
set S(ReplyTypeName) "SUCCESS"
send_SIP S $id
# start sending video
set msg [list tp_start_sending]
confbus video $msg
after 10000
set msg [list tp_stop_sending]
confbus video $msg
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)
} else {
set S(Type) {REP}
set S(Category) "2"
set S(Reason) "User cannot send video"
set S(ReplyTypeName) "FAILED"
```

```
send_SIP  S $id
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)
}
}

proc complete_GLANCE_sent {id src} {
   global T SIP_TAB MEDIA PTS USCP
   # we sent a glance and we just got a 'success'
   # Glance happens on a hardwired port (vic tool already running) and low bw so:
set ip [lindex $SIP_TAB($id,CH) 2]
set msg [list tp_new_video $ip $MEDIA(videoP) $MEDIA(videoR)]
set result [confbus video $msg]

play_sound creak

# set focus of vic
# XX this might not work
set msg [list focus $src]
confbus video $msg
# use same wait time as 'glance'
after 10000
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)
fb ""
}

proc received_WHITEBOARD {id src} {
global T PTS GAP SIP_TAB USCP
fb "Workspace request"
set who $PTS($src,n)

set S(Type) REP
set S(Category) "3"
set S(ReplyTypeName) "RINGING"
send_SIP S $id

set msg "$who wants to start a workspace session..."
set result [timedDialog .wbr "WorkSpace Request" $msg \
{info} 10 1 {Ignore} {Refuse} {OK}]

switch -exact -- $result {
0 {
# send reply code 0 (ignored) with msg (this ought
# to be user-editable...
set S(Type) REP
set S(Category) "2"
set S(ReplyTypeName) "BUSY"
send_SIP S $id
set USCP(CurrentSession) 0
}
1 {
# send reply code 1 (refused) with msg (this ought
# to be user-editable...
set S(Type) REP
set S(Category) "2"
set S(Reason) "Refused by user"
set S(ReplyTypeName) "FAILED"
send_SIP S $id
set USCP(CurrentSession) 0
}
2 {
  complete_WHITEBOARD_recvd $id $src
}
}
}

proc complete_WHITEBOARD_recvd {id src} {
   global T SIP_TAB USCP
   # Received whiteboard request
   fb {launching whiteboard tool}
   #since we  received the request
   # we use the ip address given in the c field
set ip [lindex $SIP_TAB($id,c) 2]
set port $SIP_TAB($id,whiteboardPort)
set t $SIP_TAB($id,s)
set command "exec nice wb -C \"$t\" $ip/$port &"
# say $command
set result [catch  $command T(whiteboardPID)]
if [string match $result 0] {
# send the reply (might need some error handling here)
set S(Type) REP
set S(Category) "1"
set S(ReplyTypeName) "SUCCESS"
send_SIP S $id
set USCP(CurrentActivity) $t
} else {
set S(Type) REP
set S(Category) "2"
set S(ReplyTypeName) "FAILED"
send_SIP S $id
}
set USCP(CurrentSession) 0
fb ""
}

proc complete_WHITEBOARD_sent {id src} {
global T SIP_TAB USCP
if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin)}
```

```
# sent whiteboard request
   fb {launching whiteboard tool}
   # we sent the request so we use IP address given by CH field
# of the reply
set ip [lindex $SIP_TAB($id,CH) 2]
set port [lindex $SIP_TAB($id,m) 1]
set t $SIP_TAB($id,s)
set command "exec nice wb -C \" $t \" $ip/$port &"
# say $command
set result [catch $command T(whiteboardPID)]
set USCP(CurrentSession) 0
fb ""
}

proc received_TEXT {id src} {
global T PTS GAP SIP_TAB
fb "Workspace request"

set who $PTS($src,n)

set S(Type) REP
set S(Category) "3"
set S(ReplyTypeName) "RINGING"
send_SIP S $id

set msg "$who wants to start a text editing session..."
set result [timedDialog .wbr "Text Editing Request" $msg \
{info} 10 2 {Ignore} {Refuse} {OK}]

switch -exact -- $result {
0 {
# send reply code 0 (ignored) with msg (this ought
# to be user-editable...
set S(Type) REP
set S(Category) "2"
set S(ReplyTypeName) "BUSY"
send_SIP S $id
}
1 {
# send reply code 1 (refused) with msg (this ought
# to be user-editable...
set S(Type) REP
set S(Category) "2"
set S(ReplyTypeName) "UNSUCCESSFUL"
send_SIP S $id
}
2 {
  # send reply code 2 (OK) with msg
set S(Type) REP
set S(Category) "1"
set S(ReplyTypeName) " SUCCESS"
send_SIP S $id
start_Workspace text $s $id
}
}
}

proc complete_TEXT_sent {id src} {
global T
# we sent a TEXT and just got a SUCCESS
if [winfo exists $T(waitDialogWin)] {destroy $T(waitDialogWin)}
start_Workspace text $src $id
fb ""
}

proc send_SIP {S id} {
global SIP PTS SIP_TAB GAP T SIP_srcs
# Send an SIP packet
# S is the array holding details about the SIP to send
# id is the id of the request/reply to be sent, or to reply to
upvar $S s
set cname $GAP(CName)
# increment sequence number
incr SIP(Sequence)
switch -exact -- $s(Type) {
REQ {
   # send a request
   set to $s(To)
   set tpservice $s(tpservice)
   set SIP_TAB($id,Version) 0
   set SIP_TAB($id,Count) 1
   # create the standard header
   set SIPheader "$SIP(Version) REQ\n"
   append SIPheader "PA=$GAP(IP)\nAU=$SIP(Authority)\nID=$id\n"
   append SIPheader "FR=$GAP(Email)\nTO=$PTS($to,e)\n"
   # create standard session description
   set sd "v=0\n"
   append sd "o=$SIP(Logname) [get_NTP] $SIP_TAB($id,Version) IN IP4 $GAP(IP)\n"
   append sd "s=$T($tpservice) $PTS($to,n)\n"
   append sd "i=field left blank\ne=$GAP(Email)\n"
   append sd "c=IN IP4 $GAP(IP)\nt=0 0\n"
   # create the appropriate media description
   set media [build_req_$tpservice $id]
   # set the appropriate records
   set SIP_TAB($id,Service) $tpservice
   set SIP_TAB($id,To) $to
   set SIP_TAB($id,Got) 0
   set SIP_TAB($id,Sent) $s(tpservice)
   lappend SIP_TAB(Sent) $id
   set content "$SIPheader$sd$media"
   set SIP_TAB($id,REQ) $content

   foreach line [split $content "\n"] {
     set theline [split $line "="]
     set SIP_TAB($id,[lindex  $theline 0]) [lindex $theline 1]
   }
   # need to create a dp address record
   set SIP_srcs($id) [dp_address create $PTS($to,IP) $SIP(Port)]
   after $SIP_TAB($id,ResendAfter) "resend $id"
           }

REP {
# send a reply - of which there are 4 categories (at present)
set SIPhead "$SIP(Version) REP $s(Category) $s(ReplyTypeName)\n"
set content "$SIPhead"
append content [build_$s(ReplyTypeName) $id s $cname]
foreach line [split $content "\n"] {
     set theline [split $line "="]
     set SIP_TAB($id,[lindex  $theline 0]) [lindex $theline 1]
   }
   set SIP_TAB($id,REP) $content
}
}
# say "Sending SIP: $content"
   SIP_send $content $id
}

proc send_USCP {S} {
   global T USCP GAP
   upvar $S s
   # create standard header
   set h "$USCP(Version) $s(Type)\n"
   append h "s=$GAP(CName)\n"
     append h "t=[get_NTP]\n"
   #build_USCP_$s(Type) $s(To)
   USCP_send $h $s(ID)
}
#----------------------------------------------------------
#
# Procs to build SIP REQUESTS
#
#---------------------------------------------------------

proc build_req_GLANCE {id} {
global MEDIA
# create session attribute
   set msg "a=tpservice:GLANCE\n"
append msg "m=video $MEDIA(videoP) $MEDIA(videoT) $MEDIA(videoF)\n"
return $msg
 }

proc build_req_KNOCK {id} {
global MEDIA SIP_TAB
# create session attribute
   set msg "a=tpservice:KNOCK\n"
append msg "m=video $MEDIA(videoP) $MEDIA(videoT) $MEDIA(videoF)\n"
append msg "b=AS:$MEDIA(videoR)\n"
   append msg "m=audio $MEDIA(audioP) $MEDIA(audioT) $MEDIA(audioF)\n"
   # set appropriate records - may be altered later during negotiation
   set SIP_TAB($id,videoPort) $MEDIA(videoP)
   set SIP_TAB($id,videoTransport) $MEDIA(videoT)
   set SIP_TAB($id,videoFormat) $MEDIA(videoF)
   set SIP_TAB($id,videoBandwidth) $MEDIA(videoR)
   set SIP_TAB($id,audioPort) $MEDIA(audioP)
   set SIP_TAB($id,audioTransport) $MEDIA(audioT)
   set SIP_TAB($id,audioFormat) $MEDIA(audioF)
   return $msg
 }

proc build_req_CONNECT {id} {
global MEDIA SIP_TAB
# create session attribute
   set msg "a=tpservice:CONNECT\n"
   append msg "m=video $MEDIA(videoP) $MEDIA(videoT) $MEDIA(videoF)\n"
   append msg "b=AS:$MEDIA(videoR)\n"
   append msg "m=audio $MEDIA(audioP) $MEDIA(audioT) $MEDIA(audioF)\n"
   # set appropriate records - may be altered later during negotiation
   set SIP_TAB($id,videoPort) $MEDIA(videoP)
   set SIP_TAB($id,videoTransport) $MEDIA(videoT)
   set SIP_TAB($id,videoFormat) $MEDIA(videoF)
   set SIP_TAB($id,videoBandwidth) $MEDIA(videoR)
   set SIP_TAB($id,audioPort) $MEDIA(audioP)
   set SIP_TAB($id,audioTransport) $MEDIA(audioT)
   set SIP_TAB($id,audioFormat) $MEDIA(audioF)
return $msg
 }

proc build_req_WHITEBOARD {id} {
global MEDIA SIP_TAB
# create session attribute
   set msg "a=tpservice:WHITEBOARD\n"
   append msg "m=whiteboard $MEDIA(whiteboardP) $MEDIA(whiteboardT) $MEDIA(whiteboardF)\n"
   return $msg
 }

proc build_req_TEXT {id} {
global MEDIA SIP_TAB
# create session attribute
```

```
   set msg "a=tpservice:TEXT\n"
append msg "m=text $MEDIA(textP) $MEDIA(textT) $MEDIA(textF)\n"
return $msg
 }

#----------------------------------------------------------
# Procs to build SIP REPLIES
#---------------------------------------------------------

proc build_SUCCESS {id s cname} {
# a success
global GAP SIP SIP_TAB
upvar $s p
set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
append msg "CH=IN IP4 $GAP(IP)"
return $msg
}

proc build_UNSUCCESSFUL {id s cname} {
   # request not successful
   global GAP SIP SIP_TAB
   set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
return $msg
}

proc build_BUSY {id s cname} {
   # user is busy (a progress report)
   global  GAP SIP SIP_TAB
   set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
return $msg
}

proc build_FAILED {id s cname} {
   # request failed
   global  GAP SIP SIP_TAB
   upvar $s p
   set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
append msg "NO=$p(Reason)\n"
return $msg
}

proc build_RINGING {id s cname} {
   # found user, getting attention (a progress report)
   global  GAP SIP SIP_TAB
   set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
return $msg
}

proc build_TRYING {id s cname} {
   # trying to reach user (a progress report)
   global  GAP SIP SIP_TAB
   append msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
return $msg
}

proc build_DISCONNECT {id s cname} {
   # request cancelled/disconnected
   global  GAP SIP SIP_TAB
   # the connection could have been started by receiving a request or sending one
   if [info exists SIP_TAB($id,ID)] {
   set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
} elseif [info exists SIP_TAB($id,ID)] {
   set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"
} else {
   say shit
   }
return $msg
}

proc build_FURTHER {id s cname} {
# further action required by initiator
gglobal  GAP SIP SIP_TAB
   set msg "PA=$SIP_TAB($id,PA)\nAU=none\nID=$id\nCN=$cname\nFR=$SIP_TAB($id,FR)\n"
append msg "TO=$SIP_TAB($id,TO)\n"

upvar $s p
switch -exact -- $p(ReplyTypeName) {
   REDIRECT {
     append msg "CH=IN IP4 $GAP(IP)"
     append msg "$SIP(RE)"
     }
   ALTERNATIVE {
     # do some fancy stuff
     }
   NEGOTIATE {
     # do some fancy stuff
     }
   }
return $msg
}
```

```
#----------------------------------------------------------
# Proc to resend SIP packets by id
#---------------------------------------------------------
proc resend {id} {
# resends the SIP message with given id if no reply has been
# received
global SIP GAP T SIP_TAB
set SIP_TAB($id,ResendAfter) [expr $SIP_TAB($id,ResendAfter) * 2]
if {[string match $SIP_TAB($id,Got) "0"] && [expr $SIP_TAB($id,Count) < $SIP(ResendMax)]} {
# then we haven't received a reply and we haven't hit
# the max resend yet
incr SIP_TAB($id,Count)
# need to alter 'o' field
set msg "$SIP_TAB($id,REQ)"
say "Resending request id $id"
SIP_send $msg $id
fb "No response: Resent request after $SIP_TAB($id,ResendAfter) ms (Attempt $SIP_TAB($id,Count) of $SIP(ResendMax))"
after $SIP_TAB($id,ResendAfter) resend $id
} else {
   say "Cancelling resend id = $id (# times sent = $SIP_TAB($id,Count))"
   }
}
```

C.3.6 Group Awareness Protocol

Parsing and sending procedures for GAP.

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#-----------------------------------------

#-----------------------------------------------------------
# GAP.tcl
#
# All Multicast Awareness Protocol procs are defined here
#-----------------------------------------------------------
# First the procs for dealing with incoming data
#-----------------------------------------------------------

# GAP receiving proc
proc receive_GAP  {d} {
global GAP
# increment the packet count
incr GAP(TotalPackets)
# update total bytes received (includes those sent by default)
# NB: this is not entirely accurate as IP adds 20, UDP adds 8 and
# Tcl-DP adds some more (5?)
set GAP(TotalBytes) [expr $GAP(TotalBytes) + [string length $d]]
set GAP(MeanPacketBytes) [expr $GAP(TotalBytes)/$GAP(TotalPackets)]

set type [lindex [lindex $d 0] 1]
process_GAP_$type $d
}

proc process_GAP_INFO {d} {
global PTS GAP SOURCES WINID

# Add CName to list of sources
# there must be a more elegant way to do this
set src [lindex [split [lindex $d 3] =] 1]
set SOURCES($src) $src

# check if source is in array maping CName to window id.
# If not, give it an id
if ![info exists WINID($src)] {set_win_ID $src}

# record IP address (shouldn't change after first time...)
set PTS($src,IP) [lindex [split $src "@"] 1]

# increment netStatus
set PTS($src,netStatus) 2
set PTS($src,Colour) Black

# parse packet - nice and simple ;-)
foreach l $d {
set t [split $l =]
set def [lindex $t 0]
set data [lindex $t 1]
switch -exact -- $def {
i {
set i [split $data :]
set PTS($src,[lindex $i 0]) [lindex $i 1]
}
default {set PTS($src,$def) $data}
}
}

# if packet is not ours and the ID clashes with ours, reset
# our ID.
```

```
if ![isme $src] {
if [string match PTS($src,id) $GAP(ID)] {set_ID}
}

update_GAP_received $src $PTS($src,ntp)
update_Sources
}


proc process_GAP_BYE {d} {
global PTS SOURCES WINID
set src [lindex [split [lindex $d 3] =] 1]
unset SOURCES($src)
set win .$WINID($src)
if [winfo exists $win] {destroy $win}
set PTS($src,netStatus) -6
}

proc update_GAP_received {src time} {
   global GAP SOURCES T GAP_GOT
   set now [get_NTP]
   set n [array size SOURCES]
     set GAP_GOT($src/$time) $n
 }

proc update_GAP_sent {} {
   global GAP GAP_SENT SOURCES
   set n [array size SOURCES]
   set s $GAP(Sequence)
   set GAP_SENT($s) "$GAP(SendAfter):$n"
}

#-------------------------------------------------
# Second, the procs for dealing with sending data
#-------------------------------------------------

proc send_GAP {type} {
# send a Multicast Awareness Packet
global GAP T USCP SIP_TAB
incr GAP(Sequence)

set status $USCP(CurrentActivity)

# build the header
set H "v=$GAP(Version) $type\n"
append H "id=$GAP(ID)\n"
append H "ntp=[get_NTP]\n"
append H "cn=$GAP(CName)\n"
append H "seq=$GAP(Sequence)\n"

#build the INFO or BYE packet
switch -exact -- $type {
   INFO {
# build the standard content
append m "n=$GAP(Name)\n"
append m "t=[get_LocalTime]\n"
append m "u=$GAP(Uri)\n"
append m "e=$GAP(Email)\n"
append m "p=$GAP(Pots)\n"

# Create Teleport info lines
   set i "i=doorstate:$GAP(State)\n"
append i "i=status:$status\n"
append i "i=note:$GAP(Note)\n"
append i "i=havevideo:$GAP(Have_Video)\n"
append i "i=havespeaker:$GAP(Have_Speaker)\n"
append i "i=havemic:$GAP(Have_Mic)\n"
set Content "$H$m$i"
}
BYE {
append H "t=GAP Tool quiting\n"
set Content $H
}
}
set packetSizeChar [string length $Content]
set packetSizeBits [expr $packetSizeChar * 8]
#  set GAP(MeanPacketSize)
GAP_send $Content
# say "sent GAP: $Content"
update_GAP_sent
}

proc new_Timer {initial} {
global T GAP SOURCES
# Uses same algorithms as RTCP and divides bw equally
# For full details see: DRAFT-IETF-AVT-RTP-* Appendix 7
set GAP_min_time $GAP(Min_Time)
set n [array size SOURCES]
set GAP(Sources) $n

if $initial {
# first time we called this proc (ie at start-up)
set t [expr $GAP_min_time / 2]
return $t
} else {
set GAP_bw $GAP(MaxBandwidth)
set t [expr $n * $GAP(MeanPacketSize) / $GAP_bw]
#  we need a random number in range 0.5 - 1.5 to multiply t by
set m [randomRange 0.5]
set GAP(Multiplier) $m
set interval [expr $t * $m]
#  if the value is less than 5 seconds, set it to 5
if [expr $interval < $GAP_min_time] {set interval $GAP_min_time}
#  update the graph if it exists
if {[info exists T(viewgraph)] && [winfo exists $T(viewgraph)]} {add_point $n $interval gapsrc}
return [expr round($interval)]
}
}

proc set_ID {} {
global GAP PTS SOURCES
# Generate random number to use as Id, check none of the
# other sources are using it. If so, regenerate
set src [array names SOURCES]
set inuse 0
while {$inuse != "-1"} {
set ID [randomRange 1000]
set inuse [lsearch $src $ID]
}
set GAP(ID) $ID
}
```

C.3.7 User Service Control Protocol

Parsing and sending procedures for USCP.

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------


# USCP.tcl - User Service Control Protocol

proc process_USCP_DISCONNECT {d} {
global PTS GAP SIP SIP_TAB T USCP
set src [lindex [split [lindex $d 1] = ] 1]
set msg "$PTS($src,n) has closed the audio/video connection."
tk_dialog .disc {Disconnect Dialog:} $msg {info} 0 {OK, Close Media Tools}
if $GAP(Have_Video) {confbus video tp_stop_sending}
confbus audio tp_exit
if [winfo exists .cd] {destroy .cd}
after 1000
set USCP(CurrentSession) 0
set USCP(CurrentActivity) $T(IDLE)
}
```

C.3.8 Tcl/Tk Conference bus

Remote control of other tk applications such as vic and vat.

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------

#----------------------------------------------------------
# tk-confbus.tcl
#
# Conference bus using Tcl/Tk send command. To be superceded by
# LBL's confbus arch when there is time.
#
# This will work for vic and vat ONLY if they have been compiled
# with the versions of Tcl/Tk that are compatible with the versions
# TelePort is using. This is because the implementation of the
# 'send' command changed between Tk 3.X and 4.0
#----------------------------------------------------------

proc confbus_init {} {
say "Initialising Tcl/Tk conference bus"
}

# conference bus receiving proc if using Tcl/Tk bus
proc cb_Receive {msg} {
# can't happen!
}

# conference bus sending proc if using Tcl/Tk bus
proc cb_Send {msg int} {
# force the tool's interpreter to use it's own confbusHandler proc
# to minimise possibilities of errors. Note that confbusHandler
# in cf-confbus.tcl (for vic 2.7a / vat 4.0a) requires
# two parameters but doesn't seem to use the first (?)
set args [list $msg]
#  say "Sent this on Tcl/Tk conference bus: $int $args"
if [string match $msg  tp_exit] {
   return [send -async $int confbusHandler 1 $args]
   } else {
    return [send $int confbusHandler 1 $args]
  }
}

proc set_cb_channel {} {
}

proc get_interp_name {which} {
global T
# this is called in order to make sure we know
# which interp we are sending to
if [info exists T($which)] {
# we did this before and got the name of the tool
# so just return it
return $T($which)
} else {
foreach int [winfo interps] {
if [string match $which\* $int] {
   # say "sending to $int"
   catch {send $int pid} p
   if ![regexp -nocase {[a-z]} $p] {
   # then we got the pid (might find a pre tk4.0 tool)
if [string match $p $T($which\PID)] {
   set T($which) $int
   return $int
}
   }
   }
}
# if we got this far then we can't contact the tool (probably a pre tk4.0 app)
return 0
}
}


proc confbus {media msg} {
global MEDIA
set tool $MEDIA($media\A)
set int [get_interp_name $tool]
if [string match $int 0] {
   say "$tool tool unreachable - it's probably either a pre tk4.0 app or you are not using xhost..."
   } else {
set result [cb_Send $msg $int]
#  puts stdout "Sent $msg to $T($tool), got $result"
if [info exists result] {return $result}
}
}
```

C.3.9 Platform Specific Procedures

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------

proc get_CName {} {
global GAP env
# Generate canonical name using user@host
# XXX Fix this. For now...
return "$env(USER)@$GAP(IP)"
}
proc get_LocalTime {} {
# return local time in 24 hour clock format
# XXX Fix this. For now...
catch {exec date +%H:%M:%S} t
return $t
}
proc get_NTP {} {
# return NTP time stamp
set now [getsecs]
# shamelessly stolen from sdr (c) 1996, Mark Handley & University College London.
set off 2208988800
if {$now==0} {return 0}
return [format %u [expr $now + $off]]
}



proc get_IP {} {
# figure out our IP address
# XXX Fix this. For now...
global GAP
# puts "Fix get_IP, net.tcl line 82"
catch {exec hostname} hn
catch {exec grep $hn /etc/hosts} hl
set ip [lindex $hl 0]
set GAP(IP) $ip
return $ip
}

proc play_sound {s} {
global T env
   set dir "$env(TPHOME)/picsnds"
   set popd [pwd]
   cd $dir
   catch "exec $T(play) -i -v $T(icon_vol) $s.au &" result
   # check result is alphanumeric (ie a pid), if not
   # there was an error of some sort.
   # if [regexp [a-z] $result] {
#    warn "$result (so beeped instead)"
#    bell
#    }
   cd $popd
}
```

C.3.10 Launching Media Tools

Procedures to launch media tools with appropriate parameters.

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------

#--------------------------------------------------------------
# externals.tcl
#
# Procedures to launch media tools
# Also procedures to show email and www resources.
#---------------------------------------------------------------

# The following procedures initialise the audio and video tools
proc init_Video {} {
global MEDIA T GAP
set app $MEDIA(videoA)
fb "Launching $app tool"
switch -exact -- $app {
vic {
set r $MEDIA(videoR)
set f [string tolower $MEDIA(videoF)]
set p $MEDIA(videoP)
set a $GAP(IP)
#  dougal needs this line to switch off shared memory for vic
# catch {exec nice vic -s -C $T(sessionName) -B $r -f $f \
catch {exec nice vic -C $T(sessionName) -B $r -f $f \
-I $T(ch) \
$a/$p &} result
fb "Video tool launched"
set T(vicPID) $result
}
default {tperror "Can't launch video tool: $app, editing\
the code in externals.tcl might help"}
}
fb "Video Tool Launched"
}
proc init_Audio {ip port} {
global MEDIA  GAP T PTS
set app $MEDIA(audioA)
fb "Launching $app tool"
switch -exact -- $app {
vat {
set f [string tolower $MEDIA(audioF)]
catch {exec nice vat -f $f  \
-I $T(ch) \
 $ip/$port &} result
set T(vatPID) $result
}
default {warn "Can't launch audio tool: $app, editing the\
code in externals.tcl might help"}
}
after 5000
get_interp_name audio
confbus audio tp_no_quit
fb "Audio Tool launched"
}

proc start_Workspace {app source id} {
global MEDIA GAP PTS SOURCES SIP_TAB
# this proc fires up your workspace tool in unicast mode and points
# it at the source destination.
switch -exact -- $app {
wb {
```

```
fb "Launching $app tool"
set t "wb shared with $PTS($source,n)"
# if received a  request:
if [info exists SIP_TAB($id,ID)] {
   set port $SIP_TAB($id,whiteboardPort)
   set ip [lindex $SIP_TAB($id,c) 2]
   }
# or sent one: so should have got a SUCCESS reply
if [info exists SIP_TAB($id,ID)] {
   set port [lindex $SIP_TAB($id,m) 1]
   set ip [lindex $SIP_TAB($id,CH) 2]
   }
catch {exec nice wb -C $t $ip/$port &} T(whiteboardPID)
}
text {
fb "Launching $app tool"
set t "nt shared with $PTS($source,n)"
set myP $MEDIA(textP)
set myIP $GAP(IP)
set yourP $PTS($source,textP)
#XX elementary check - needs work
if ![string match $myP $yourP] {
set myP $yourP
}
set a $PTS($source,IP)
catch {exec nice nt $a/$myP &} T(textPID)
}
default {warn "Can't launch workspace tool: $app, editing the code\
at about line 2808 might help."}
}
fb "Workspace tool, $app, launched"
}

# script to show email resource

proc do_Email {s to} {
global T PTS
set label "The Email address of $PTS($s,n) is:"
set msg {Use the X-selection to copy this to your favourite Email tool.
  One day there may be an integrated email system here...but probably not!}
todoDialog $to $label $msg
}

# script to show a URL

proc call_Browser {url} {
global T
set label "The URI requested is:"
set msg {Use the X-selection to copy this to your favourite WWW browser.\
One day there may be an integrated browser here....but probably not!}
todoDialog $url $label $msg
}
```

C.3.11 Miscellaneous Utilities

```
#-------------------
#  TelePort 2.1a3
# Copyright (c) 1995 Loughborough University of Technology.
# Copyright (c) 1995 British Telecommunications plc.
# All rights reserved.
#
# Author: B.Anderson@lut.ac.uk
#
# For SunOS - tested under SunOS 4.1.4
#------------------------------------------

#---------------------------------------------------------
# Various useful procedures:
#---------------------------------------------------------

#
# Set_Geom: Sets a window's minimum size to the min width and
# height requested for it - stops user shrinking window horribly
#
proc Set_Geom {w which} {
set mwidth [winfo reqwidth $w]
set mheight [winfo reqheight $w]

wm minsize $w $mwidth $mheight
if {$which == "both"} {wm maxsize $w $mwidth $mheight}
}


# procs to generate random numbers for timer
# Borrowed from Welch's book and hacked about a bit

proc randomInit {seed} {
global RAND
set RAND(ia) 9301
set RAND(ic) 49297
set RAND(im) 233280
set RAND(seed) $seed
}

proc random {} {
global RAND
set RAND(seed) [expr ($RAND(seed) * $RAND(ia) + $RAND(ic)) % $RAND(im)]
return [expr $RAND(seed)/double($RAND(im))]
}

proc randomRange {lower} {
set r [expr [random] + $lower]
return $r
}

proc warn s {
global T
if [winfo exists $T(DebugWin)] {
    $T(DebugWin) insert end "$s\n"
   $T(DebugWin) yview -pickplace end
} {
   puts stdout "$T(Sname) warns: $s"
   }
}

proc say s {
global T
if  [winfo exists $T(DebugWin)] {
    $T(DebugWin) insert end "$s\n"
   $T(DebugWin) yview -pickplace end
} {
   puts stdout "$T(Sname) says: $s"
   }

}

proc check_mcaddress {} {
global T

# carry out same checks as Mark Handley's SDR

set MIP [split $T(maddr) .]
if {[llength $MIP] != 4} {
warn "$MIP is invalid: Wrong number of digits in multicast address, must be 4."
exit
}
set a [lindex $MIP 0]
if {($a < "224") | ($a > "239")} {
warn "$MIP is invalid: First digit not in range 224 to 239."
exit
}
for {set t 1} {$t <= 3} {incr t} {
set e [lindex $MIP $t]
if {($e < "0") | ($e > "255")} {
    warn "$MIP is invalid: digits 2 to 4 need to be in range 1 to 255."
}
    }

}

proc set_win_ID {s} {
global WINID WinCount
set WINID($s) $WinCount
incr WinCount
}

proc toggle_noisy {} {
global T
if $T(noisy) {set $T(noisy) 0} {set $T(noisy) 1}
}

proc fb {msg} {
   global T
   set T(fb) $msg
   update idletasks
}
```

Appendix D

TelePort: Published Papers

This chapter contains a reprint of a version of Chapter 7 which was published in the Proceedings of the BCS HCI '95 Conference - People and Computers X. The full reference is given as [10].