ESPRIT Project 6155 – EuroCODE

CSCW Open Development Environment

Deliverable D-5.6:
The Middle Road Demonstrator:

Concept, use, technical foundation and evaluation of a system for supporting communication within cooperative work settings

Doc-Id:	**CODE-NR-95-11**
Version:	**1.0**
Authors:	**Peter Holmes, Hilde Lovett, Frode Løbersli, Eigil Skorve**
Date:	**December 20, 1995**
Status:	**Final**
Availability:	**Confidential**
Distribution:	**Consortium, CEC, NFR**

Published by NR Norsk Regnesentral ANVENDT DATAFORSKNING

The Middle Road Demonstrator

Concept, use, technical foundation and evaluation of a system for supporting communication within cooperative work settings

Peter D. Holmes
Hilde Lovett
Frode Løbersli
Eigil Tapio Skorve

Norsk Regnesentral
December 20, 1995

Preface

The Middle Road Demonstrator (MRD) is a system for communication support within cooperative work settings. The MRD is one of three demonstrator systems developed within the framework of ESPRIT Project 6155 - EuroCODE: CSCW Open Development Environment. The system was designed and evaluated in cooperation with a pilot group from the Department of Radiology at Rikshospitalet.

The MRD supports cooperative work by allowing users to share applications independent of time and space. The MRD supports both synchronous and asynchronous communication modalities. In both of these modalities, the MRD provides communication facilities based upon a "show, point and talk" functional profile.

The MRD's synchronous communication modality is realized through it's desktop conferencing facility, while it's asynchronous modality is realized through a multimedia messaging facility called the Snapshot Composer.

This report documents the design, development and evaluation of the Middle Road Demonstrator. The chapters in the report are written such that they may be read, to a certain extent, as independent documents. The report is divided into several major parts:

- the Introduction
- the Outcome,
- the Process. and
- Technical Appendices.

The **Introduction** is comprised of two chapters. The first of these chapters provides an overview of the EuroCODE project, it's goals and objectives. The second chapter provides a description of the setting for and within which the MRD was developed and evaluated.

The **Outcome** focuses upon results generated throughout the MRD activity. This part of the report consists of six chapters. These chapters concern the MRD concept, use of the system, the MRD's technical foundation, evaluation results and a summary. A chapter is also added which briefly describes further work, including plans for commercializing the system. The material is described in as non-technical a manner as possible, presenting matters from a functional perspective. The Outcome is written such that it may be read, to a great extent, as an independent report.

The **Process** material is comprised of three chapters. The first contains information about the MRD's design and development process, while the second presents detailed information regarding the MRD's evaluation at Rikshospitalet. This includes figures depicting screen images from the first two versions of the system, along with some of the comments and feedback provided by the pilot user group. The third chapter in this part of the report briefly summarizes aspects of the MRD's development process, and closes with some concluding remarks.

The report's **Technical Appendices** turn once again to the MRD's technical foundation. In this part of the report, technical details are provided for each of the MRD's four primary software modules, called "toolkits". The chapters found in the technical appendices are the Conference Toolkit, the Global Window Toolkit, the Digitized Sound Toolkit and the Snapshot Composer Toolkit.

This report also contains a **Glossary** defining particular technical terms which appear. **References** are included for the documents explicitly named in the report, as well as other relevant literature.

Table of Contents

Part 2 Outcome

Chapter 3

Chapter 4

Chapter 5

Part 3 Process

Chapter 9

Chapter 10

Chapter 11

Part 4 Technical Appendices

Appendix A

Appendix B

Appendix C

Appendix D

Part 5 Glossary, Acknowledgements and References

Part I

Introduction

Chapter 1

The EuroCODE Project

Esprit Project 6155 - EuroCODE: European CSCW (Computer-Supported Cooperative Work) Open Development Environment was carried out as a three-and-one-half year project; extending from the beginning of the third quarter, 1992, to the end of the fourth quarter, 1995.

1.1 Goals and objectives

The purpose of the EuroCODE project has been to design products aimed to support collaborative work between and across geographically-dispersed companies.

The EuroCODE project has created an open development environment for CSCW applications, as well as a series of CSCW products based upon that environment. This has been achieved through three key elements, the goals of the project. These goals are described below, and the relationships amongst them are depicted in figure 1.

- *A Framework for Computer Support of Cooperative Working*; a framework allowing for the modelling of cooperative working, as well as seamless integration of computer support within the working environment.
- *An Open CSCW Shell based on the Framework*; an open CSCW Shell, based on the framework, which supports the generation and tailoring of specific CSCW applications. These applications interwork across networks and operating system platforms involving non-CSCW applications and systems.
- *Three demonstrator systems*; systems built using the Shell and deployed in real settings.

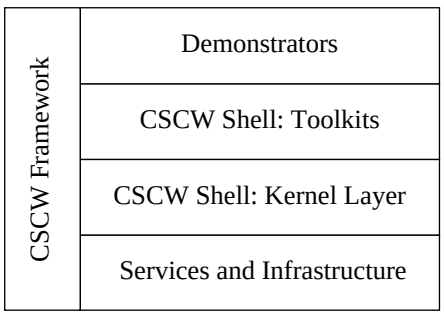

Figure 1 : *The EuroCODE Architecture.*

EuroCODE's products are designed to support three kinds of users:

- system integrators who build and modify CSCW applications and interfaces to other applications;

- third party developers of CSCW applications and systems; and,
- end users who can tailor and combine applications to suit needs which evolve with changing requirements.

1.1.1 The EuroCODE Framework

The EuroCODE Framework provides an abstract structuring of the field of cooperative working. Setting out from dimensions such as time, place, mobility, connectivity and communication bandwidth, the Framework has been developed and refined using investigations of trends in cooperative working in a range of user organizations. Substantial independence of specific technologies helps to ensure that EuroCODE systems can be properly integrated into user organizations and are robust in the face of continuing technological and infrastructural change.

1.1.2 The CSCW Shell

The CSCW Shell has been designed to provide an open, effective development environment for interworking CSCW applications. Employing an object-oriented approach, the CSCW Shell supports heterogeneous systems by defining how different technical attributes will be interpreted on different platforms under different constraints, including variations in data transmission bandwidth and images and sound resolution.

The design of the Shell included two layers, a Toolkit Layer and a Kernel Layer (see figure 1). Software elements within the Toolkit Layer were designed to be platform-independent. Underneath the Toolkit Layer, the Kernel Layer implemented application services and infrastructure. In order to support the Toolkits across different platforms, platform-dependent specializations of Kernel services were developed using object-oriented techniques.

In order to help ensure first-order interoperability of the Toolkits, the EuroCODE project adopted a reference platform. This platform was based upon the Solaris operating system and the X Window System.

1.1.3 The demonstrator systems

The three demonstrator systems have been chosen to span the field of cooperative working according to the dimensions of the EuroCODE Framework. The demonstrators highlight novel aspects of the CSCW Shell architecture and show how CSCW applications based on the Shell can work together across different work environments and with different underlying computers and network connections. At the same time these applications function as a reference for the development of EuroCODE products. These systems include:

- The Low-Road Demonstrator (LRD) which supports shared work management and mobile communication;
- The Middle-Road Demonstrator (MRD), supporting communication within cooperative work settings (e.g., distributed expertise); and,
- The High-Road Demonstrator (HRD) which supports the integration of electronic and paper documents.

1.1.4 Other project objectives

Other specific objectives of the project have included:

- exploring the benefits of
  - an objected-oriented approach to the CSCW Shell,
  - mobile computers designed to help mobile workers,
  - integrating digitized sound and video;
- improving the understanding of user-requirements in distributed work environments; and,
- using the CSCW Shell and Framework as a basis for developing and deploying powerful and effective systems supporting a broad spectrum of cooperative work.

EuroCODE development work has been based on a comprehensive requirements study of the area of cooperative working and existing support — both IT and non-IT — as well as detailed, in-depth studies within selected user organizations. Other project work concerns the identification of leading-edge applications of CSCW systems and the degree to which these kinds of systems can be realized using EuroCODE's CSCW Shell. Work has also been carried out concerning the identification and investigation of potential market segments for the CSCW Shell and the demonstrator systems.

In order to increase the possibility of producing useful results and particularly to further rapid technology take-up, user-centered techniques have played a prominent role throughout the project. In the early phases of the project user-centered requirements generation has been employed, and later evaluation of demonstrators by users has played a key role in the development of the Framework and the CSCW Shell, as well as in the design refinement of the demonstrators.

From a research perspective, the project makes a significant contribution to an improved theoretical understanding of human cooperation and to the design of open, object-oriented technologies supporting cooperative work in organizations. The findings and recommendations of EuroCODE constitute ongoing input for the work of European and world-wide standardization committees in different technological areas including distributed processing and communication.

1.2 The EuroCODE Consortium

The EuroCODE consortium has consisted of major IT companies, small-to-medium-sized enterprises (SMEs), research organizations and a representative user organization from three Member States and Norway. The partners lead the field of Computer Supported Cooperative Working (CSCW) in Europe, having rich expertise in object-oriented technologies, distributed processing and OSI communications as well as a substantial market potential.

EuroCODE has built upon the earlier Esprit II Project 5303: EuroCoOp — IT Support for Distributed Cooperative Work. This background has enabled the EuroCODE project to set off with an unrivalled understanding of the domain of CSCW as a basis for the develop-

ment of the Framework, as well as a number of immediately available components with which to develop the open CSCW Shell and demonstrator systems.

The Consortium finds collaboration technology to be of strategic importance and the exploitation potential very high. The large IT company partners currently work to introduce EuroCODE results into their product lines of business systems. The SMEs involved market the CSCW Shell, the demonstrator systems and other EuroCODE applications. Subsequently, they plan to provide consulting services within these important domains.

Chapter 2

The Development and Evaluation Setting

2.1 The Middle Road Demonstrator

2.1.1 Original plans

As mentioned above, the EuroCODE project was active for a period of 3.5 years, beginning in the third quarter of 1992. MRD-related activities were carried out together with the pilot group at Rikshospitalet (RH) throughout the entire project period.

Rikshospitalet was selected as a user organization for the EuroCODE project for several reasons. First, because of the expected need for a real-time conferencing system for professionals with access to networked workstations. Secondly, because Norsk Regnesentral already had a well-established and fruitful cooperation with the hospital, within the fields of image analysis and advanced medical computer applications. Thirdly, because the hospital has a special body, the Center for Medical Informatics, which works especially with advanced computer systems for hospitals.

Via contact with the Center for Medical Informatics, the Department of Radiology was selected as the group for which the Middle Road Demonstrator (MRD) would be introduced. The primary goal of the MRD was originally *to provide an improved means for establishing and engaging in synchronous communication.* At Rikshospitalet, this communication need exists amongst radiologists, clinicians, technicians, medical secretaries and other relevant hospital personnel; such is also the case within the Department of Radiology. In that department transpire both regularly-scheduled work routines as well as spontaneous, unscheduled and often acute medical situations. Many of these unscheduled medical situations require immediate, inter-departmental communication; to make the situation even more complex, one of the department´s four sections is not co-located with the rest.

With regard to introduction of the MRD at Rikshospitalet, the selection of the Department of Radiology was also firmly based upon the fact that that department produces and works with an extraordinary amount of medical imagery. These medical images must often be reviewed and discussed amongst different physicians within physically separated parts of the department. The MRD originally hoped to attain its primary goal by providing a facility for *real-time desktop conferencing*, including mechanisms providing “show, point and talk” functionality [30, 2, 42]. Within the context of use by the Department of Radiology, this would allow doctors to view and point within radiological images simultaneously, as well as carry out audio conversations via the data network.

2.1.2 Additional dimensions of communication support needed at Rikshospitalet

As work together with the pilot group progressed, it was necessary to re-evaluate just what purpose the MRD should serve within the Department of Radiology. The pilot group definitely wanted improved support for communication. However, the group insisted that the nature of their work — especially the degree to which their physical location within the department and hospital changes rapidly and relatively unplanned each day — markedly devaluated the potential benefit of a purely synchronous means of communication. Of course, the group wanted a means by which they could establish contact and immediately consult with others, should they be available. However, the pilot group also pointed out that should the desired party *not* be available immediately, the MRD should provide a communication facility by which they could leave a message, request or query for another — a message that could be answered when time permitted.

The material the pilot group handles involves a variety of media types. The messages they imagined sending would focus upon medical images. Along with the images in a message, the group wanted to send annotations upon the images (e.g., arrows which pointed at specific phenomena), as well as some text and/or audio material (e.g., in order to specify queries, replies, etc.).

Early on in the project, it became apparent that in order to satisfy the pilot group's communication needs, it would be necessary to extend the MRD's original, primary goal. The pilot group's day-to-day working-processes dictated that the MRD should include facilities for both synchronous and asynchronous communication support.

2.2 Rikshospitalet

Rikshospitalet is the main, state-owned hospital in Norway. Most Norwegian hospitals are operated by the counties. This includes local hospitals, county-hospitals and four of the five regional hospitals. Rikshospitalet is a regional hospital for one "health-region", and a central hospital for the whole country. It has only few functions as a local hospital. Thus, Rikshospitalet primarily receives patients with rare or severe illnesses. For example, Rikshospitalet is the only hospital in Norway performing complicated, pediatric heart surgery. It is often the case that patients must travel far in order to get to the hospital.

2.2.1 The Department of Radiology

Within Rikshospitalet, the main focus of the project was within the Department of Radiology. The Department of Radiology is a medical service department. The facilities within the department are used as a laboratory, to provide medical services for other departments within RH. Radiology concerns a lot more than conventional X-rays; other important techniques, or *modalities*, are angiograph, computer tomography (CT), magnetic resonance tomography (MR), gamma-camera and ultrasound.

The Department of Radiology is divided into four, more specialized sections:

- Pediatric radiology
- Thorax radiology
- Neuroradiology
- General radiology

The department is thus organized according to medical specialities, not according to radiological modalities. At the start of the project, the department had only one MR and two CTs, organized as specialized MR and CT Units. The MR and CT devices produced digital image data which was stored in the department's digital image archive. However, analog film copies of these images were also created; it was the analog film versions of the images which were used within the department's day-to-day work routines and processes.

Original images obtained from radiographic examinations are often physically transported between various sections and departments at RH. Such transportation is time-consuming, and it is not uncommon that images are temporarily misplaced or, at times, even disappear. To improve this situation, the Department of Radiology has been working on the introduction of a Picture Archiving and Communication System (PACS); the name of this system at Rikshospitalet is RiksPACS.

The aim of RiksPACS is to have a networked system which includes digitized pictures from all modalities. The introduction of RiksPACS was initiated within the Department of Radiology, specifically, within the sections for pediatric radiology and neuroradiology. At the start of the project, RiksPACS workstations were installed within each of those sections. These sections are located in different buildings at the hospital. The distance between these buildings is approximately 250 meters.

2.2.2 RiksPACS' influence upon user-group selection

With respect to the MRD effort, it was clear that any attempt to breach the myriad of possibilities for CSCW support within the *entire* Department of Radiology would be overwhelming. Meetings held with physicians from the Sections for Pediatric Radiology and Neuroradiology indicated a high degree of both formal and informal communication flow between these two sections. Furthermore, a great deal of that communication had its basis in radiographic imagery. Expectations were also such that the RiksPACS system would soon become used more and more regularly[1].

Given the physical locations of the RiksPACS workstations, along with the rich possibilities for CSCW support for the Sections for Pediatric Radiology and Neuroradiology, project studies first focused upon those two groups. It was soon determined that communication needs were especially great within the Section for Pediatric Radiology. The early pilot studies therefore focused much upon the work-processes and communication flow from that section's perspective [14, 59]. In the later development and evaluation phases

1) This expectation proved to be wrong. As a whole, Rikshospitalet has a centralized resource for matters concerning data system acquisition, installation, integration, maintenance, upgrade, etc. This resource is the hospital's Center for Medical Informatics. In regard to RiksPACS' thorough integration and refinement for the Department of Radiology, it is possible that Rikshospitalet was forced to focus the Center for Medical Informatics' work upon matters of greater urgency.

of the project, focus generalized once again back to communication needs existing at and between the Sections for Pediatric Radiology and Neuroradiology [2, 42, 40].

2.3 The Evaluation Setting

The MRD was tested and evaluated during a six week period at Rikshospitalet in Oslo. During the evaluation period, the MRD was integrated with the Department of Radiology's digital image database archive. There were two primary reasons for integrating the MRD directly with the archive, rather than being integrated with the department's RiksPACS system. On the one hand, the operating systems for the two systems were different. On the other hand, it was not fully known whether RiksPACS would be in clinical use when the time came for MRD evaluation.

Since the MRD was directly integrated with the department's digital image database archive, a simple application called MRD-PACS was written as a surrogate for RiksPACS. MRD-PACS allowed for listing the MR examinations for a given patient, as well as each examination's associated images. Using MRD-PACS, images could be selected for viewing on the MRD workstation screen.

The MRD evaluation was based upon real work scenarios (see chapter 4). These scenarios concerned communication between the neuroradiology section and the pediatric radiology section. Three machines were installed at Rikshospitalet during the evaluation. Two of the machines were placed in the conference room for the pediatric radiology section and the neuroradiology section, respectively. These rooms are located centrally within both sections. The radiologists do much of their preparation for morning conferences (i.e. meetings with clinicians) in these rooms. They also do some "finishing up" in these rooms, once the morning conferences are completed. The third machine was placed in a meeting room within the neuroradiology section. This machine was used as a server for the other two machines, and was used for development and testing without disturbing the actual work at the hospital.

As a whole, the evaluation effort concentrated upon obtaining feedback concerning functionality and user-interface design for the MRD's communication facilities. The evaluation effort also sought to obtain indications as to where and when MRD-like communication systems could be useful for the radiologists at Rikshospitalet, as well as for other working groups at Rikshospitalet.

In seeking to explore these issues, three questions were used to keep the system testing and evaluation sessions in focus. One question concerned the system's usability, while the other two concerned the system's usefulness:

Q1. Is the MRD easy to work with?

Q2. Can the MRD make work more efficient at the Department of Radiology?

Q3. Are there other application areas where MRD-like systems can be useful?

Further information about the evaluation process is presented in chapter 10. Details concerning the outcome of the evaluation are presented in chapter 6.

Part 2

Outcome

Chapter 3

The MRD Concept

In a single statement, one should understand the MRD to be:

...a system for communication support within cooperative work settings[1].

Underlying this conception of the MRD are certain views and ideas regarding the nature of cooperative work, communication and communication support. Section 3.1 primarily concerns itself with grounding definitions inherent in the formulation above, as part of the MRD's overall conceptual framework. Section 3.2 presents certain general requirements for systems designed to provide communication support. Section 3.3 describes the approach by which the MRD concept has been realized, while section 3.4 closes the chapter with a brief summary.

3.1 Conceptual Framework

3.1.1 Cooperative work

Many definitions exist for the term "cooperative work". Schmidt and Bannon [79] refer to and analyze different connotations of the term as it has been developed in the literature. Rather than reiterate Schmidt and Bannon's fine analyses, it is herein satisfactory to ground the definition of "cooperative work" using Webster [89]:

cooperate*: 1. to act or work together with another or others for a common purpose.*

Employing this definition as a basis necessitates that more than one person is involved in any cooperative effort. However, this definition makes no implications as to when and where the cooperation is taking place.

Of this definition, one could choose an interpretation of the phrase "...together...for a common purpose" to mean "physically co-located, but potentially unaware of the common purpose"; an example of this could be drawn from the workers' situation on the huge factory floors of the early 1900's. Alternatively, one could choose an interpretation of the same phrase to mean "not necessarily physically co-located, but definitely aware of the common purpose". This paper's presentation focuses upon the latter interpretation of this phrase. Furthermore, the interpretation chosen here goes on to include cooperation in which the common purpose and/or the awareness of that purpose may be developed within the context of the cooperative effort.

1) In this formulation, 'settings' are not to be conceived as static situations; instead 'settings' intends to refer to transient sets of conditions in effect from one moment to the next.

3.1.2 Communication

There are many aspects to communication, and different constellations of these aspects arise depending upon the context in which communication takes place. Several aspects of human communication are presented here, in order to create a meaningful basis for certain requirements discussed in section 3.2. To begin, Webster [89] will be used to define:

> ***communication**: 2.a). a giving or exchanging of information, signals, or messages in any way, as by talk, gestures, writing, etc.*

3.1.2.1 Forms of human communication

When considering human communication, Baird and Weinberg [8] classify certain aspects of communication as being verbal, paraverbal and non-verbal. In their work, the "verbal" aspects concern content (i.e., *what* is conveyed). The "paraverbal" aspects concern *how* the content is expressed; according to Baird and Weinberg, the paraverbal aspects involve expressive style, articulation, intensity, phraseology and interpretive expression. The "non-verbal" aspects involve what is done *during* the communicative interaction; during face-to-face communication, examples of this kind can include eye-contact, posture, gestures, etc. Within the limits set by the communication context, these forms of communication intentionally or unintentionally function so as to convey information.

3.1.2.2 Mutual understanding and grounding

Given a situation in which communicating parties are working for some common purpose, it is necessary that the interacting parties achieve and maintain some level of mutual understanding. Clark and Brennan [16] claim that during a communicative interaction, "common ground" — a basis for mutual understanding — cannot remain fully updated without a *grounding process*; the purpose of this process is to establish that the information which has been conveyed/exchanged has also been understood.

The grounding process can be observed in many conversations and discussions. At certain times, there is a need to focus upon objects and their respective identities. In such situations, it is often highly preferable that this identification process transpire quickly and effectively. For each object requiring identification, the main goal of the identification process is to create a *referential identity*; in other words, the goal is to create a common belief amongst the communicating parties that they have correctly identified the referent. Some techniques for grounding such references are alternative description, indicative gestures (e.g., pointing) and trial referencing [16].

3.1.2.3 Least collaborative effort

Creating and maintaining mutual understanding during a communicative interaction implies certain costs. These costs can be specific to the sender, specific to the receiver or incurred by both. In the context of cooperative efforts, Clark and Wilkes-Gibbs [17] suggest:

> ***The principle of least collaborative effort:*** *In conversation, the participants try to minimize their collaborative effort — the work that both do from the initiation of each contribution to its mutual acceptance.*

One can well consider how a generalization of this principle would apply to the kinds of information exchange available within systems designed to support cooperative work; such considerations are presented in section 3.2.3.

3.2 General requirements for communication support

Given the views above for understanding cooperative work and communication, a second tenet underlying the MRD concept is that:

...communication is a prerequisite for cooperative work.

This principle follows naturally from the definition that cooperative efforts involve more than one person and that the persons involved are or become aware of their common purpose within the effort. As an essential part of cooperative work, the MRD's purpose is to support such communication.

The aspects of communication presented in section 3.1.2 can be used to derive requirements for communication support. When functioning to support cooperative efforts, a communication system must include facilities which help enable the establishment and development of mutual understanding. This includes facilities:

R1. for conveying (the equivalent of) verbal and paraverbal forms of communication, as well as certain forms of non-verbal communication;

R2. which help enable grounding of referential identities; and,

R3. which require little effort to use.

3.2.1 Conveying the verbal, the paraverbal and the non-verbal

Clearly, a communication system must support exchange of the verbal (or, more generally, the content-related) aspects within a communicative interaction. It is argued here that exchange of the paraverbal aspects must be supported as well. This argument is supported by Reder and Schwab's work concerning multi-channel genres of communication [76]. Their cross-organizational studies of electronic mail use indicate that even computer-mediated, *written* conversations tend to include aspects of paraverbal communication common to oral discourse:

> *Some new genres of communication have arisen with the use of computer-mediated communication... several genres of electronic conversations have evolved, replete with characteristic linguistic forms and discursive conventions for signaling openings and closings, topic transitions, and techniques for initiating intra-conversational shifts to other channels.* ([76], p. 359)

Without the paraverbal aspects of communication, there is greater chance that communication content may be misconstrued and overall communication flow disrupted. In this

regard, Clark and Brennan point out that misunderstandings amongst communicating parties imply extra communication costs (i.e., *fault* and *repair costs*, see [16]).

Paraverbal information is a characteristic inherent within oral communication. Reder and Schwab's work indicates that paraverbal information is also used within electronic, written communication. Within a system for supporting communication, the requirement of facilities for conveying verbal and paraverbal information (R1) can usually be quite well satisfied by including mechanisms through which both textual- and audio-based content can be exchanged.

Here, it is not considered an absolute requirement that *all* non-verbal aspects of communication be conveyed. By this is meant that it is not considered an absolute requirement that facial expressions and body language be conveyed (through use of video technology, for example). Many investigations have been reported concerning the lack of this communication form. Two specific characteristics are common to these reports; that is:

- lack of non-verbal signals leads to fewer interruptions of the person currently speaking [78, 81]; and,
- communication becomes more task-oriented when the communication media being used is less rich [78].

These results strongly suggest that within many communicative interactions, the need to see the other parties is not of great priority. However, these results cannot be used to justify lack of such features, should requirements specific to a given cooperative situation call for them. A good example arises in business discussions between high-level managers and directors, wherein multi-party video teleconferences have been preferred to use of the telephone [82].

With regard to non-verbal communication forms, it is considered that pointing behavior *should* be supported by a communication system. The motivation underlying this standpoint is presented below.

3.2.2 Grounding referential identities

The need for facilities which enable the development of mutual understanding is an obvious one. As described in section 3.1.2.2, understanding one another often calls for the need to ground references to specific objects.

During many communicative interactions, people often point at an object instead of using a great number of words to specifically denote it. For this reason, it would be useful to support this kind of pointing behavior within machine-mediated communication contexts. For example, when communicating via machines, members of our pilot group demanded something to point *at* and something to point *with* [40].

With regard to conveying *visual content*, it is suggested here that the need to support pointing behavior within machine-mediated communication contexts is proportional to the level of detail within that content, as well as the degree to which such content is presented in parallel. The need to support pointing behavior may also be proportional to the volume of content to be conveyed, but not necessarily so. Therefore, systems designed to support communication about detail-rich and/or parallel visual content should provide facilities which allow the communicating parties to view the same material, and a mechanism by which they can point at specific details within that material[2].

3.2.3 Making it simple

Communication systems should be intuitive and easy-to-use. This point is suggested by Clark and Wilkes-Gibbs ideas about least collaborative effort (see section 3.1.2.3), and is certainly an experience shared by many. Perhaps the easiest systems to use are those which, when used, are taken for granted (or not noticed at all!). From a design perspective, this implies that communication systems should be *transparent* and *seamless*.

Here, a communication system's transparency is distinguished from it's seamlessness. The term 'transparency' is applied to the manner in which a specific communication medium affects communication content. A 'transparent communication medium' allows users to formulate, develop and exchange information content without being significantly distracted and/or limited by use of the medium.

In contrast, the term 'seamless' is applied to the manner in which a communication system affects overall task execution. Users of a 'seamless communication system' should not experience significant interruptions with regard to task execution; this notion can be defined along (at least) two dimensions:

- *seamless with regard to task-orientation:* use of the communication system should fit smoothly within the overall work process, leaving users free to keep their minds oriented upon their respective tasks[3]; and,
- *seamless with regard to shifts of modality:* should the communication system offer different modalities[4] for communication, shifts between system modalities should not significantly disrupt the state of the task.

3.3 The MRD Approach

To facilitate user-acceptance of the MRD, great effort has been made to satisfy the general communication support requirements (R1-R3) listed above. Addressing requirements R1 and R2 specifically, the MRD provides "show, point and talk" functionality [30, 2, 42] in both of it's communication modalities; this topic is further elaborated in section 3.3.2.

The MRD is also designed to support different users' favorite applications without affecting those applications' standard behavior. With regard to requirement R3, chapter 4 presents two thorough scenarios of MRD use; it is left to the reader to judge whether use of the MRD is simple and intuitive.

To facilitate technical acceptance, the MRD's architectural design is open and it's implementation follow standards. The system's design is object-oriented, in order to be flexible and extensible.

2) This addresses some of the issues concerning *permanent communication channels*, see Whittaker, et. al. [91].
3) The design of user-interfaces with respect to task-orientation is discussed in Gritzman, et. al. [31].
4) Consider, for example, real-time communication vs. message passing.

3.3.1 Two perspective view of cooperative work

The MRD has been designed to fit into the overall cooperative work process as unobtrusively as possible. To best understand this design fit, one can view the cooperative work process from two orthogonal, yet related perspectives. Here, these perspectives are discussed from an individual's point-of-view, rather than that of a group.

The first perspective concerns that of individual work upon tasks. One can easily draw examples in which cooperative efforts involve a sets of persons working on their own, communicating, exchanging and synchronizing their individual results with one another as need be. As each person works alone, they are focussed upon their own tasks.

The second perspective of the cooperative work process concerns exactly those settings and situations in which a person needs to, or is obliged or requested to, communicate with another person. It is exactly at these moments that the MRD stands ready to support the initiation and reception of communication contacts.

With regard to inter-personal communication contacts, at least two contextual dimensions can be distinguished:

- whether the communicative interaction transpires in real-time or not (i.e., *synchronous* vs. *asynchronous* contact); and,
- whether an individual is the one *initiating* a communication contact, or *receiving* such a contact.

The interaction of these two dimensions creates four communication contexts; these are illustrated in figure 2(a). Within a cooperative work setting, this figure depicts how these communication contexts are experienced from the individual's point-of-view, rather than that of the group as a whole.

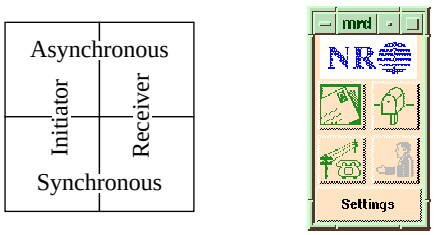

Figure 2 : *Communication contexts within cooperative work settings:*
(a) as experienced from the individual's point-of-view; and,
(b) as reflected within the MRD's top-level interface.

Figure 2(b) depicts how the design of the MRD's top-level interface directly reflects the four communication contexts illustrated in figure 2(a).

3.3.2 The MRD and communication support

The MRD is a communication system designed to support the initiation and reception of communication contacts, especially within the context of cooperative efforts. The MRD supports communication through it's capacity to transmit many different kinds of data, in each of it's two communication modalities. The MRD's transmission facilities can per-

form their work in a synchronous mode, enabling the possibility for desktop conferencing. In addition, the MRD's transmission mechanisms can perform their work in an asynchronous mode, providing a multimedia messaging facility for users.

Together, the MRD's desktop conferencing and multimedia messaging facilities lead to a new communication system concept. That is:

...the MRD allows for sharing applications independent of time and place.

The manner in which this concept is realized is the subject of following sections.

3.3.2.1 Synchronous contact: desktop conferencing

The MRD's desktop conferencing support is primarily intended for situations in which persons need to discuss task-related material(s) "right at that moment". The MRD's desktop conferencing functionality can be divided into two groups: (1) functionality directly related to communication content; and (2) administrative functionality.

The communication facilities provide show, point and talk functionality via:

- *show:* application sharing: mechanisms making it possible to simultaneously share task-related materials upon machines which may be geographically distributed;
- *point:* telepointers: electronic pointers making it possible to direct other participants' attention to specific areas-of-interest within shared documents, images, etc.; and,
- *talk:* conference audio: a mechanism for audio teleconferencing.

The administrative facilities include:

- a registrar service
  - which automatically keeps track of the machine upon which (potential) conference participants are logged in upon, and
  - provides the capacity to pre-define conference configurations; and,
- a conference manager which helps govern behavior within the conference (e.g., who is allowed to add/drop conference participants, who is allowed to add/drop applications, etc.).

The application sharing and telepointing mechanisms are implemented within the Global Window Toolkit [1], described below in section 5.4. The conference audio facility is implemented within the Digitized Sound Toolkit [38], described in section 5.5.

All of the MRD's desktop conferencing facilities are orchestrated by the EuroCODE conferencing architecture [85], implemented within the Conference Toolkit (discussed later in section 5.3). The conferencing architecture enables a uniform means for coordinating of a number of conferencing applications. It also offers a well-defined interface through which it is possible to integrate third-party applications.

3.3.2.2 Asynchronous contact: multimedia messaging

The MRD's multimedia messaging support is primarily intended for less time-critical situations in which persons need to exchange and/or pose questions about task-related ma-

terial(s). The multimedia messaging support can also be effectively used when someone is not available for a real-time conference.

The MRD's multimedia messaging facility is called the Snapshot Composer; this facility also provides show, point and talk functionality, though without the feedback characteristics inherent in real-time communication. When composing a multimedia message, these facilities allow one to:

- *show:* create a "snapshot" (i.e., a set of selected documents, or document references, along with certain application state information for each document) to be sent to and viewed by others;
- *point:* place simple annotation marks (e.g., arrows) atop[5] the documents within a snapshot; and,
- *talk:* include an audio and/or text message along with the snapshot and annotations.

When a user receives and opens such a multimedia message, the Snapshot Composer "reconstructs" the message in order that it's individual parts may be viewed. Here, 'reconstruction' means *re*-presentation of a multimedia message. During such re-presentation, an application is started for each document included in the message. Each document is loaded into it's respective application, and any state information associated with the document is used to help recreate the state the application was in when the message was created.

The simple text editor, annotation and snapshot mechanisms are implemented within the Snapshot Composer Toolkit, described below in section 5.6. The simple audio recorder/player is implemented within the Digitized Sound Toolkit [38], described in section 5.5.

3.4 Summing up

In section 3.2, general requirements for communication support were presented. These requirements derived from the consideration that communication systems designed to support cooperative efforts should enable their users to establish and develop mutual understanding. It has been maintained that such communication systems would require facilities:

- for conveying (the equivalent of) verbal and paraverbal forms of communication, as well as certain forms of non-verbal communication;
- which help enable grounding of referential identities; and,
- which require little effort to use.

The MRD directly addresses these general requirements by providing users with show, point and talk functionality. In both of it's communication modalities, the MRD allows:

5) Note: these annotations do not in any way become written into a document's contents. They appear atop a document, but a different logical layer is used to present them on the computer screen.

- work within applications to be shared;
- pointers to be used for directing attention; and,
- questions to be asked and answered orally (and/or textually[6])
- an intuitive, task-oriented user-interface which is easy to use (see chapter 4).

The desktop conferencing and multimedia messaging facilities ensure that the MRD allows for sharing applications independent of time and place. With it's mechanisms for real-time application sharing, telepointing and conference audio, the MRD's desktop conferencing facility helps eliminate barriers normally associated with geographically-distributed cooperative efforts. The MRD's multimedia messaging facility also helps eliminate barriers; in this case, barriers primarily associated with temporal distribution. The multimedia messaging facility provides the equivalent of a desktop conference, except for the real-time feedback.

It is in light of the views and perspectives presented herein that the MRD is defined to be a system for communication support within cooperative work settings.

6) Within a real-time conference, text-based communication can be achieved by sharing a text editor.

Chapter 4

Use of the MRD

4.1 Introduction

As mentioned in section 3.3.1, the MRD is designed to support communication contacts. Specifically, the MRD supports communication contacts along two different dimensions:

- synchronous, real-time contact vs. asynchronous, message-passing contact; and,
- user's role as initiator vs. receiver of the contact.

This "breakdown" of communication contacts into four communication contexts is immediately reflected in the MRD's top-level interface. As shown in figure 3, the MRD's top-level interface includes five buttons.

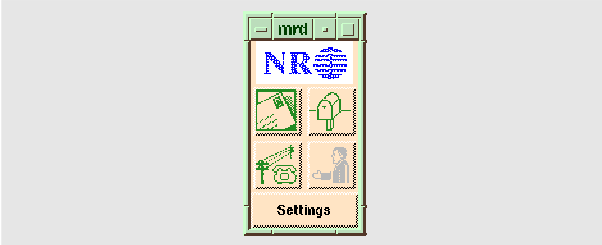

Figure 3 : *The MRD's top-level interface window*

The buttons in the upper row are used for asynchronous communication contexts, while those in the lower row are used for synchronous contexts. The buttons in the left column are used when initiating a communication contact, while those in the right column are used when on the receiving end of a contact. Employing scenarios, the use of these buttons is described below.

This chapter reintroduces the scenarios presented in the original MRD design document [42]. The scenarios illustrate how the MRD supports communication and cooperation needs between doctors at Rikshospitalet. Some of the key personnel roles found within the scenarios are:

- *Radiographer:* a medical technician responsible for carrying out examinations which rely upon the use of radiant energy (e.g., X-rays, CT, MR, etc.). These examinations yield radiographic images.
- *Secretary:* a secretary specialized in medical administrative work, responsible for typing dictated interpretations of radiographic images, for making appointments for various examinations and for administrative work in general.
- *Radiologist:* a physician responsible for the *interpretation* of radiographic images. Radiologists also perform (parts of) certain radiographic examinations themselves.
- *Clinician:* a physician responsible for the diagnosis and treatment of patients.

- *Pediatrician:* a clinician specializing in the diagnosis and treatment of children.
- *Pediatric radiologist:* a physician specializing in the interpretation of radiographic images obtained from children[1].
- *Neurologist:* a clinician specializing in diseases of the nervous system.
- *Neuroradiologist:* a physician specializing in the interpretation of radiographic images which concern the central nervous system.
- *Neurosurgeon:* a physician responsible for planning and carrying out surgical operations which involve the nervous system.

Further descriptions concerning the Department of Radiology's current work routines can be found in [14, 59]. Other details including illustrations of the department's physical layout can be found in [2].

The first scenario describes a situation in which a pediatrician suddenly needs to have a brief consultation with a pediatric radiologist; this scenario is just one instance of the kinds of situations involving unplanned, spontaneous consultations amongst doctors. Using the MRD's desktop conferencing facility, the doctors in the scenario are able to simultaneously view the patient's images and discuss the case in real-time.

The second scenario describes a situation in which a pediatric radiologist needs to receive some advice from a neuroradiologist. The communication requirements in the second scenario are not unlike the first. In the second scenario, however, the neuroradiologist is not immediately available. Using the MRD's multimedia messaging facility, the Snapshot Composer, the pediatric radiologist assembles together documents related to the nature of the desired consultation. These documents, along with annotation marks, text and voice messages, are then sent to the neuroradiologist.

4.2 Scenario 1: use of desktop conferencing

Gaute Eide a five-year old boy from Ørsta, is at RH for a scheduled control of the results of drainage from a ventricle in his brain. He was born with insufficient drainage, was operated upon early after his birth, and once later to install and modify the artificial drainage. He is living a normal life, but needs regular controls to ensure that the draining tubes are still in place while he continues to grow. It must also be judged whether the tubes can be expected to remain in place throughout the next year.

Gaute arrived at the Children's Clinic Tuesday evening together with his father. They have stayed overnight and the different examinations are to be carried out Wednesday morning. The examinations have all been scheduled in due time: the clinical examination to be performed by Dr. Peter, the pediatrician at the Children's Clinic, and the MR examination to be performed at 9:00 am at the Section for Pediatric Radiology. This section, though organizationally a part of the Department of Radiology, is physically located within the Children's Clinic, and is thus at a distance of approximately 250 meters from the rest of the Department of Radiology.[2]

1) When neuroradiological examinations are carried out on children, radiological diagnosis is the neuroradiologist's job, not the job of the pediatric radiologist.

At the Section for Pediatric Radiology, the requisition for Gaute's examination was received a couple of days earlier, and it was easy for the secretary to find a suitable slot for him in the examination program.

Gaute and his father had been informed about the examination schedule Tuesday night, and a bit before 9:00 am, a nurse reminds them of the appointment at the Section for Pediatric Radiology. Since they have been at the hospital many times before, Gaute and his father walk down the stairs to the Section for pediatric Radiology on their own. When down there, they wait for a while in the corridor, Gaute riding on a tricycle.

Ms. Lia, a radiographer, asks Gaute and his father to come in. They go into one of the labs for digital X-ray, where Gaute is gently and patiently instructed about how to position himself on the table. Gaute's genitals are covered by a lead apron. Since Gaute is a "big boy" by now, his father leaves the room while the images are taken instead of being there with Gaute. Since the staff is used to dealing with children, and since so many of the patients come back for regular controls, there is competence and motivation for being empathic with the children, making the examination go as smoothly as possible.

In all three images are made of Gaute, spanning from his brain to his stomach. When the images have been taken, they walk back to the ward where Gaute had been admitted. Ms. Lia enters the necessary data about Gaute into the imaging device (e.g. name and date of birth, id-number of the requisition, type of examination, etc.); the date and name of the hospital are automatically recorded on the images. Thereafter, she enters the necessary command to transfer the images to RiksPACS. By default RiksPACS caches in previous images of Gaute, such that they can be rapidly retrieved when the pediatric radiologist, Dr. Eigil, interprets the images. Ms. Lia checks off the requisition and puts it into a folder for examinations waiting to be interpreted. This folder is located next to the RiksPACS workstation.

At noon all other clinical tests and examinations indicate to Dr. Peter that Gaute appears to be well, and in no need of any treatment unless the X-rays should indicate otherwise. There is room on the plane to Ålesund (close to Ørsta) at 8:00 pm, and Gaute and his father would like to take that plane if possible. Actually, Dr. Peter would also like them to leave, since the department is full as usual. Dr. Peter therefore goes into his office to contact the pediatric radiologist he knows to be on duty, Dr. Eigil. On each of the doctors' workstations the MRD application is running[3]; it's window appears in the upper right corner of each of their screens.

Dr. Peter pushes the "telephone" button (lower-left) to start a conference. After pushing the telephone button a conference is initialized; the telepointer and conference audio applications are started automatically. Once these applications have started, Dr. Peter's screen appears as shown in figure 4.

2) Actually, this is a bit incorrect, since RH will move to a new site in 1997. Thereafter, the Section for Pediatric Radiology will be co-located with the rest of the department. The scenario has been left as it is, however, since situations of a similar nature will still arise.

3) Since the MRD is a fundamental communication tool for the doctors, it is automatically started when the user logs on.

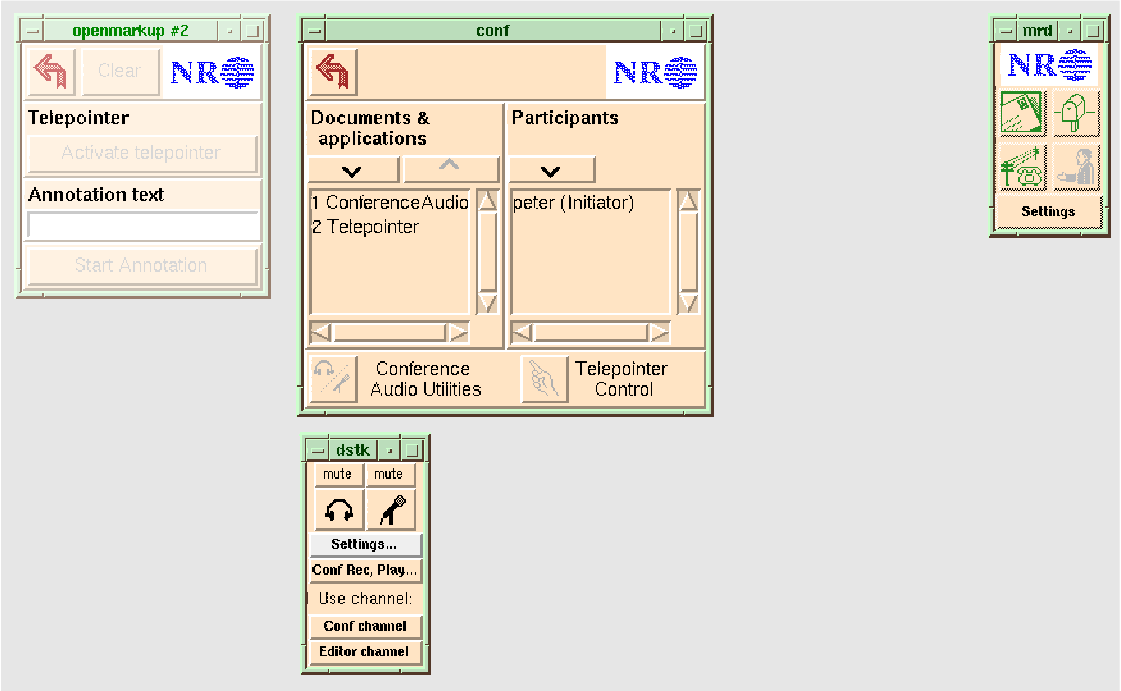

Figure 4 : *Initialization of the MRD's desktop conferencing facility. Clockwise from upper-left:*
a) the telepointer application (`openmarkup` window)
b) the Conference Manager (`conf` window), for adding/removing applications and participants
c) the top-level MRD window
d) the conference audio application (`dstk` window)

In the Conference Manager (`conf` window), the left panel shows applications which are already included in the conference; the right panel lists conference participants and their status with respect to the conference (e.g., "initiator", "asked to join", "joined", etc.). Including applications and participants within the conference is done in a consistent way: one clicks upon the '∨' button (above the left or right panel), then selects an item from the pull-down menu which appears. One such pull-down menu offers a list of applications which can be included in the conference (e.g., the MRD-PACS application[4]); the other offers a list of potential conference participants.

Removing applications from the conference is done by selecting the application to be removed, then clicking once upon the '∧' button (above the left panel). The MRD's currently implemented conference policy does not allow one participant to dismiss another from the conference; one can only dismiss oneself. To dismiss oneself from a conference, one pushes the "back-arrow" button in the Conference Manager's upper-left corner.

Normally, the telepointer application (`openmarkup` window) and the Conference Manager (`conf` window) do not appear on the screen until they need to be used. They can be made to appear and disappear from the screen using the "Conference Audio Utilities" and "Telepointer Control" buttons found in the lower row of the Conference Manager (`conf`).

4) In the two scenarios presented here, MRD-PACS functions as a surrogate for the RiksPACS system existing at Rikshospitalet (see section 2.3).

At this time, Dr. Peter asks Dr. Eigil to join the conference. Dr. Peter pushes the '∨' button above the right panel, then selects the name 'eigil' from the pull-down menu which appears. Having done so, Dr. Peter's Conference Manager will indicate that Dr. Eigil has been asked to join a conference.

On Dr. Eigil's screen, the "handshake" button (lower-right) turns yellow and several, audible beeps indicate that someone wants him to join a conference. When Dr. Eigil pushes the handshake button, a Conference Manager (`conf`) window will appear on his screen which includes an "accept" button and a "decline" button. The panels in Dr. Eigil's `conf` window list the applications currently included in the conference and the current set of participants (including their status). An illustration of Dr. Eigil's screen is given in figure 5.

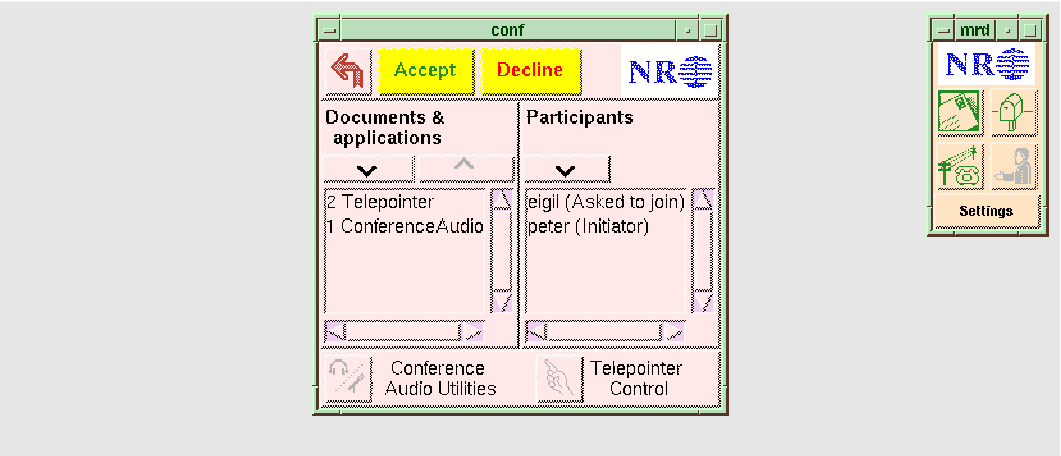

Figure 5 : *When requested to participate in a conference, the user has the possibility to accept or decline the request.*

Dr. Eigil pushes "accept", and the accept and decline buttons disappear. In doing so, Dr. Eigil will receive access to the applications already running within the conference.

Once Dr. Eigil has access to the applications in the conference, the doctors begin to speak with each other using the microphones and speakers connected to their respective workstations:

Dr. Peter: *Hi, Eigil... I need some advice about whether or not I can send home a patient tonight. Things are crowded down here as usual,... and it would be great if I could release this patient right away... there's still time for him to catch the evening plane to Ålesund...*

Dr. Eigil: ...yes,... *well,... I'm a bit busy now,... but I guess I have time for a quick look...*

Dr. Peter: ... *fine... Let me get these images up for you to see...*

Dr. Peter pushes the '∨' button above the left panel and selects "MRD-PACS" from the pull-down menu which appears. As soon as the application appears on Dr. Peter's screen, he can begin working with it. Dr. Peter is aware of the fact that almost immediately, the MRD-PACS will be replicated upon Dr. Eigil's display. Dr. Peter is also aware of the fact that as soon as the application appears on Dr. Eigil's display, Dr. Eigil can begin to provide

input as well. During the conference, the doctors use the audio channel to coordinate their joint work.

Within the MRD-PACS application, Dr. Peter types “Gaute” and pushes “search”; a scrolling list of matching items is presented. Within the list, he double-clicks on “Gaute Eide” then chooses the examination that took place on February 14, 1994. Finally, an image showing the condition of the draining tubes is selected.

Dr. Peter and Dr. Eigil can see the same windows on their workstations; they are free to organize the arrangement of these windows as they please. Together they discuss the images, using the telepointer to point out details of interest. A label is attached to each of their respective telepointers, in order to identify the owner of each. At this time, Dr. Peter’s screen appears as shown in figure 6. In comparing figure 6 to figure 4, note that Dr. Peter has used the Conference Manager’s “Conference Audio Utilities” and “Telepointer Control” buttons in order to temporarily remove the `dstk` and `openmarkup` windows from his screen; this allows him to focus even better upon the task-at-hand.

The drainage tubes are a bit hard to discern, but after applying a suitable image analysis function they appear more clearly. The tubes look OK, they agree, but their length and thus the probability that their size and positioning will be suitable throughout the next year is an issue. To discuss this they look at some of Gaute’s older images, also available within the digital archive[5]. They employ an image analysis tool to measure Gaute’s growth.

Still in question about Gaute’s case, they check whether Dr. Hegna, a neuroradiologist, is available to join their conference. Dr. Hegna accepts their request that he join, and all three go on discussing the case.

Together they conclude that it will be risky to wait for another year, but decide that half a year is safe. They say good-bye, all leave the conference[6]. Dr. Peter goes off to continue with other work. While at the workstation, Dr. Eigil dictates the interpretation of Gaute’s images using the dictaphone. He also tags the images (including some of the old ones) in order to prepare the demonstration of Gaute’s case. He takes the requisition out of the folder for examinations to be interpreted, and puts it in the folder for the demonstration to be held tomorrow. He takes the tape out of the dictaphone and puts it in the box where the secretary will pick it up for typing.

Dr. Peter informs the nurses about the doctors’ conclusion, then goes off to have a talk with Gaute and his father. He tells them that they can leave today, but that they cannot wait an entire year again before the next set of examinations. He assures the two that this does not mean that Gaute’s condition is any reason for concern.

5) Since Gaute’s case was to be reviewed that day, images from previous examinations of Gaute were cached into local memory in advance; this decreases retrieval response-time for old images.

6) The conference is ended when all the participants have left the conference. When the conference is terminated, all applications shared within the conference are also terminated.

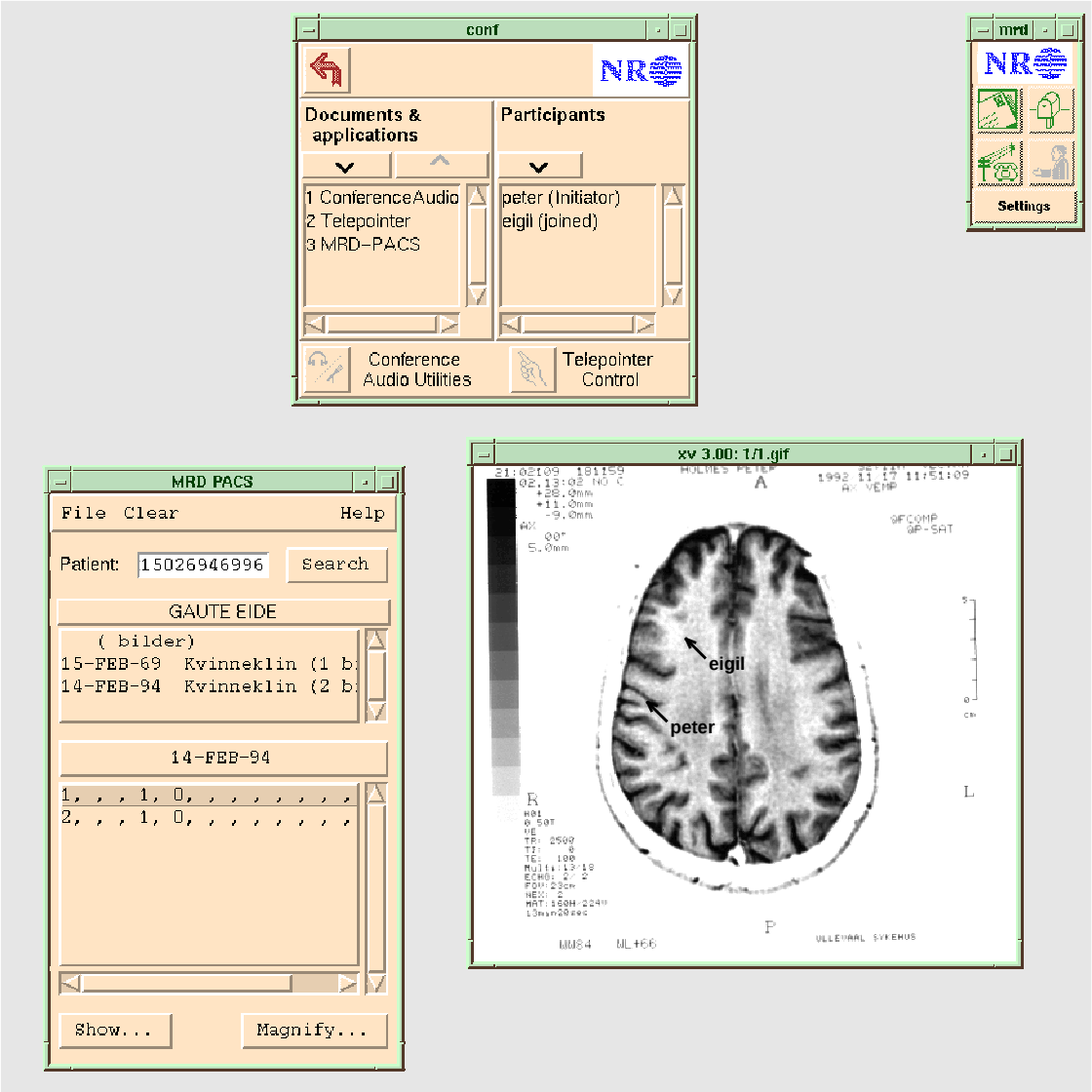

Figure 6 : *The MRD supports real-time communication while keeping users' favorite applications in focus.*

4.3 Scenario 2: use of multimedia messaging

The MRD seamlessly integrates synchronous desktop conferencing with asynchronous multimedia messaging. The MRD's multimedia messaging facility can be used when some person(s) are not available for a conference or, alternatively, when the feedback one needs is not of such a time-critical nature. Of course, it can also be used to send messages which do not require reply.

This scenario describes a situation in which a pediatric radiologist needs to receive some advice from a a neuroradiologist. Since the neuroradiologist is not available, the pediatric radiologist creates and sends a multimedia message. Two aspects of this communication form are described: the composition of the message and it's reception at the other end.

4.3.1 Composition of messages

Dr. Eigil, the pediatric radiologist, wants to discuss an examination with Dr. Larsen, a specialist in neuroradiology. He tries to reach her using the MRD's conferencing facility, but she does not immediately answer his request. Instead of waiting for Dr. Larsen, Dr. Eigil starts MRD-PACS within the conference[7]. He leaves active his request that Dr. Larsen join the conference, in the hope that she might soon be back and join him.

Dr. Eigil has the opinion that image 1 indicates a slight change in the state of the patient compared to an earlier image. Dr. Eigil wants to discuss this finding with Dr. Larsen, but she has still not joined the conference. Therefore Dr. Eigil decides to send a message to Dr. Larsen. He begins by arranging the windows on his screen as he prefers that Dr. Larsen shall see them when receiving the message. Already, he has planned to send her image 1 (since he believes it contains sufficient information for Dr. Larsen), some annotation marks and some short textual and audio remarks.

Dr. Eigil pushes the "letter" button (upper-left) to activate the MRD's Snapshot Composer. The Snapshot Composer lets the user select the documents he wishes to send, as well as place annotation marks atop those documents and include oral and written messages. After pushing the letter button, Dr. Eigil starts the small annotation application by clicking upon the "annotation" button within the Snapshot Composer's `send` window. At this time, Dr. Eigil's screen appears as shown in figure 7.

There are two points of interest within image 1. To mark these points, Dr. Eigil uses the annotation application, activating it by clicking on the "start annotation" button. (This button then turns into the "stop annotation" button.) The application enters a mode in which the mouse is grabbed. A click on the left mouse button leaves a labelled arrow where the mouse is pointing.

When Dr. Eigil is finished annotating he clicks on the "stop annotation" button. To briefly describe when he needs a reply from Dr. Larsen, he starts a simple text editor by clicking on the "text message" button. He then enters a text message asking Dr. Larsen to reply by 2:00 pm, if possible. Dr. Eigil is aware that he could have written the same text within the `send` window's "Subject" field; however, he believes that a large visible note to Dr. Larsen will have less chance of being overlooked.

7) Here, Dr. Eigil has started a conference in which he is currently the only member.

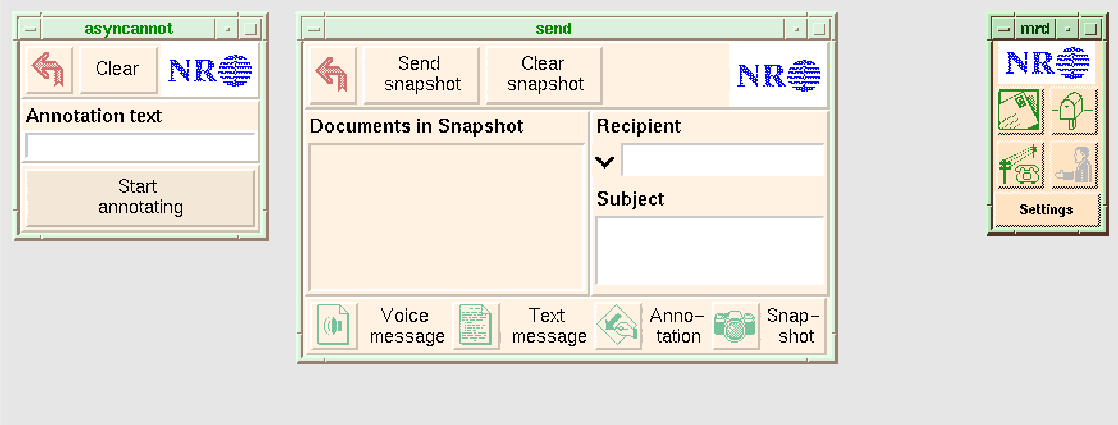

Figure 7 : *The Snapshot Composer's* `send` *window along with the "annotation-creator" application (*`asyncannot` *window).*

When the brief text message is written, Dr. Eigil wants to record a voice message concerning his finding to Dr. Larsen. By posing his queries orally, Dr. Eigil saves himself a great deal of time typing. In addition the annotation marks help him speed the formulation of his query: he can refer to 'Mark 1' and 'Mark 2' rather than having to provide precise anatomical descriptions. Dr. Eigil starts the simple audio recorder by clicking upon the "voice message" button.

Once the audio application is started, Dr. Eigil pushes the recording button (i.e., the button with the red circle) and starts talking. To stop recording, he pushes the stop button (i.e., the button with the black square). On those occasions in which he sends several images to someone, he uses natural terms to describe the relative locations of the images (e.g., "the image to the left"); he knows that when the message is opened and viewed by the receiver, the images will appear at the same positions they were in when the message was created.

Lastly, Dr. Eigil selects the documents to be sent to Dr. Larsen. To do so, Dr. Eigil clicks on the "snapshot" button. The cursor turns into a crosshair symbol ('+'). In this mode, the mouse position is used to determine within which window it is pointing. When Dr. Eigil clicks upon image 1, the reference to that image is recorded along with information about the state of the application presenting that image. Dr. Eigil knows that this "click" will include image 1 within the multimedia message.

Dr. Eigil then clicks upon the MRD-PACS application, which has the effect that information about MRD-PACS' application state will be included within the message. What Dr. Eigil understands is that by clicking upon the MRD-PACS application, it will be started automatically for Dr. Larsen when she opens the message; furthermore, he knows that the application will appear for Dr. Larsen just like it is now: Gaute Eide's examinations will be listed in the upper panel, and the listing of images associated with the February 14th examination will be listed in the lower panel.

When Dr. Eigil has selected these two elements, he quits the selection mode by simply pushing the snapshot button again. (The button has remained depressed while in the selection mode; pressing it again raises the button.)

The left panel within the Snapshot Composer's `send` window presents the user with a logical representation of the information elements to be included in the message. This

representation consists of an icon reflecting the kind of element included. Whenever text messages, voice messages and/or annotations are included in a multimedia message, icons are included for each of them. Icons representing selected documents are also included. In this way, the user has an easy way to see what's included within the multimedia message.

The `send` window offers a field in which it is possible to fill in a subject line; of course a field is available in which to specify a (set of) recipient(s). The Snapshot Composer's "clear snapshot" button makes it possible to clear the message and start again.

Dr. Eigil specifies Dr. Larsen as the receiver and types a short note in the subject field indicating the urgency of the message. Just prior to sending the message, Dr. Eigil's screen appears as shown in figure 8. Once the message is sent, the message-related application windows are closed (e.g., `asyncannot`, `sounded` and `texted`) and Dr. Eigil resumes with his work.

4.3.2 Reception of messages

In the early afternoon, Dr. Larsen enters her office and sits down in front of her workstation. She notices immediately that the MRD's "mailbox" button (upper-right) has turned yellow. This indicates that new multimedia messages have arrived. Dr. Larsen clicks on the mailbox button. In doing so, the Snapshot Composer's `receive` window opens, see figure 9.

In the `receive` window panel, the messages are listed. The listing specifies the date, sender and subject for each message; unread messages are marked as "NEW". By selecting (i.e., single-clicking upon) a message in the listing, a logical representation of that message's contents is presented just above the panel. As in the Snapshot Composer's `send` window, this representation is provided as a set of icons which reflect the kinds of information elements within the message.

Dr. Larsen sees the new message from Dr. Eigil which asks for a quick reply. Having already selected the message, she opens it by pushing the "open message" button. This operation starts a process which "reconstructs" the message composed by Dr. Eigil. For each information element within the message, an application is started in order to present that element; furthermore, these applications are placed on Dr. Larsen's screen using the positions in effect when Dr. Eigil composed the message.

Since Dr. Eigil's multimedia message includes a text element, the same simple text editor is started in order to present that element. For the audio element recorded and sent within the multimedia message, the same audio application used by Dr. Eigil is started on Dr. Larsen's workstation. In a similar fashion, the "annotation-viewer" application[8] (`make-annot` window) is started in order to present the annotation marks. The MRD-PACS application starts and sets it's state to be equivalent to the state it was in when Dr. Eigil, during composition of the message, clicked upon it.

8) The "annotation-viewer" application can only present annotations, it cannot create them. The annotation-viewer allows the user to hide (and then redisplay) annotations sent within a message.

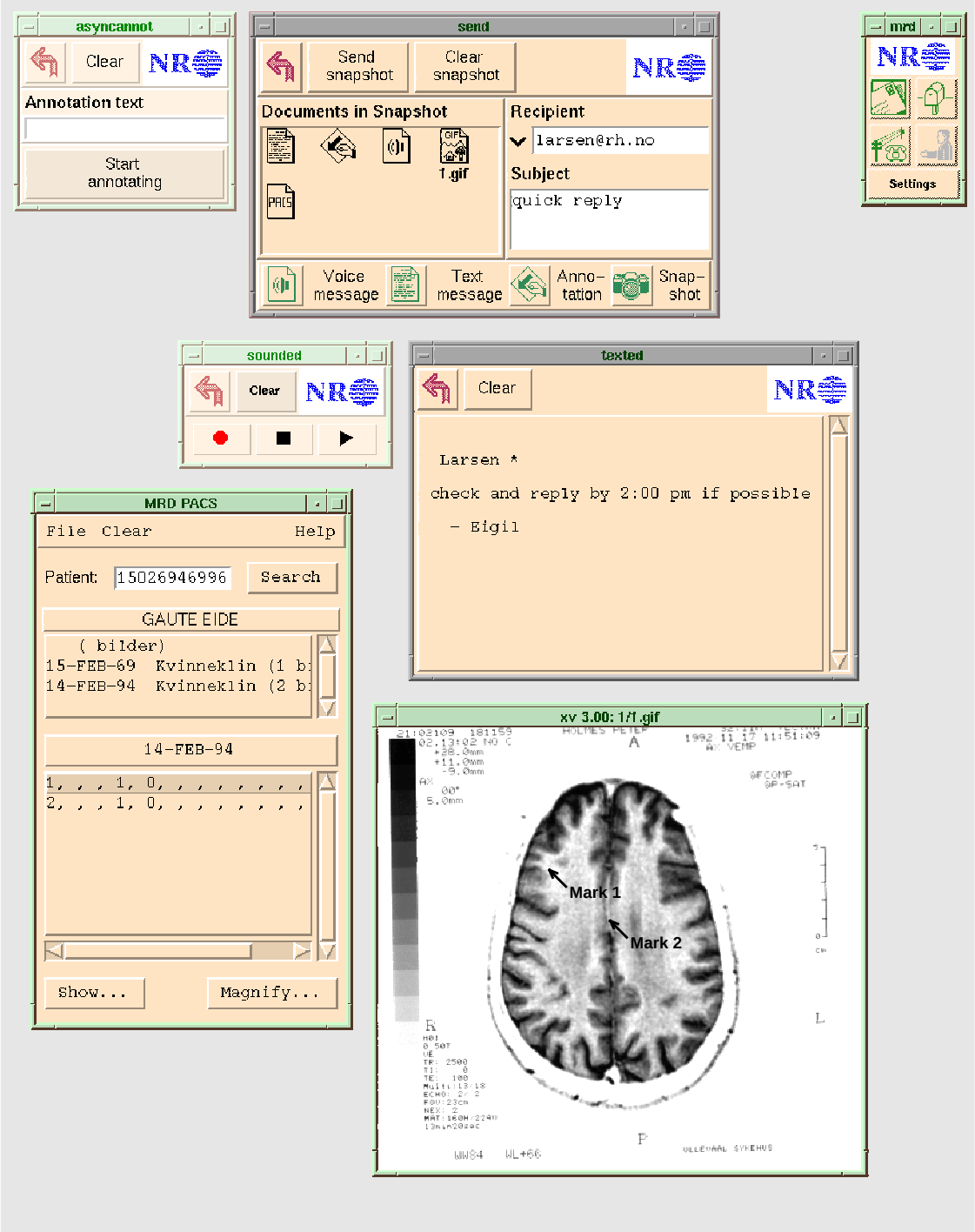

Figure 8 : *A multimedia message created using the MRD's Snapshot Composer.*

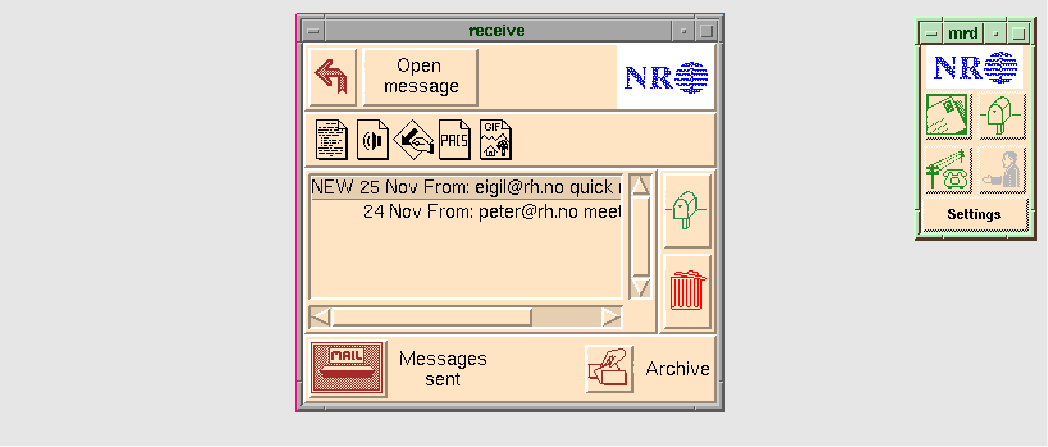

Figure 9 : *The Snapshot Composer's* `receive` *window.*

When the multimedia message is fully opened, Dr. Larsen screen appears as shown in figure 8, *except for two differences*:

- Dr. Larsen sees the Snapshot Composer's `receive` window instead of it's `send` window; and,
- she sees the "annotation-viewer" application (`makeannot`) running and visible at her workstation instead of the "annotation-creator" application (`asyncannot`).

Dr. Larsen sees that Dr. Eigil hopes for an answer before 2:00 pm. Having time to deliver a reply by then, she listens to his ideas about the findings in image 1. While listening, she studies the image, especially the areas annotated by Dr. Eigil.

Dr. Larsen essentially agrees with Dr. Eigil's findings, but wants further confirmation. Since the MRD-PACS has already been started, she simply selects and displays another image within the same examination. Upon seeing this second image, she's convinced about the matter. The time is almost 2:00 pm, so she deems it best to try and reach Dr. Eigil right away. In order to try and initiate a real-time conference with Dr. Eigil, Dr. Larsen pushes the "telephone" button within the MRD.

Chapter 5

Technical Foundation

5.1 Introduction

This section, along with section 5.2, are for general reading. The remainder of the chapter is slightly more technical. With respect to the report as a whole, this chapter may be omitted without loss of continuity.

As described in chapter 1, the EuroCODE architecture is comprised of four layers. The middle two layers comprise EuroCODE's CSCW Shell: the Kernel Layer and the Toolkit Layer. Within the Toolkit Layer [24] is a large assortment of toolkits; these toolkits include software libraries, application frameworks, basic applications, tools, etc. These toolkits provide application and system developers with a high-level programming environment in which advanced CSCW applications and systems can be developed. Implementation of EuroCODE's three demonstrator systems exemplifies this kind of shell-based system development.

The MRD itself has been implemented using UNIX and X Windows. The MRD also includes some public-domain software, as well as some software from the Mjølner BETA system. The MRD's architecture is open and it's software is based upon standard rather than proprietary solutions.

5.2 The MRD's Fundamental Toolkits

The MRD implementation is essentially comprised of four toolkits:

- the Conference Toolkit (CTK);
- the Global Window Toolkit (GWTK);
- the Digitized Sound Toolkit (DSTK); and,
- the Snapshot Composer Toolkit (SSCTK).

Briefly, the Conference Toolkit implements a real-time conferencing architecture. This architecture is used to administrate a "conference", i.e., a collection of applications together with a list of active conference participants. The architecture is open, including a well-specified interface by which it is possible to integrate new and/or third party applications.

The Global Window Toolkit implements a facility for window replication. Using this facility within a conference, it is possible to enable an ordinary, single-user X application to present it's output to a set of workstations. The facility also enables the possibility that such applications become capable of receiving input from a set of users, rather than a single one.[1] The GWTK also implements a telepointer mechanism, a facility with which

it is possible to electronically point within windows being replicated. These electronic pointers can be seen by all other conference participants.

The Digitized Sound Toolkit implements a facility for establishing real-time, audio teleconferences over the data network. It also includes applications for recording, editing and playback of audio, either locally at one's own workstation or within a conference.

The Snapshot Composer Toolkit contains functional elements designed to support the composition, reconstruction and presentation of multimedia messages. The SSCTK does not involve itself with the transmission of multimedia messages — that job is performed by the standard e-mail services to which it is integrated.

These individual toolkits' respective functionality is further presented in sections 5.3-5.6. **Note: for each of the toolkits presented, definitions for that toolkit's basic concepts are provided the Glossary (found at the end of the report).** It can be useful to read about each toolkit's basic concepts once having read the "Overview" for that toolkit.

More detailed technical information about the MRD's fundamental toolkits can be found in the MRD's Technical Appendices [41]. Information about other MRD-relevant toolkits can be found in section 5.7.

5.3 The Conference Toolkit

5.3.1 Overview

The Conference Toolkit (CTK) includes programs and source code for handling conference administration, as well as some conference applications to demonstrate use of the overall conferencing architecture. The administrative functionality allows one to predefine conferences (i.e., to pre-specify the membership and set of applications to be included in a particular conference).

The main program is the "conference". The conference is a sort of a switchboard, performing actions on applications and participants when instructed to do so by the "conference manager". A conference manager is a user interface for handling conferences.

Other important programs in CTK are the "registrar" and "confreg". The registrar keeps track of conferences, participants, and applications. It also provides a service for starting predefined conferences. "Confreg", a registrar client, is a user interface to the registrar for ordinary users.

5.3.2 Functionality

The Conference Toolkit is implemented in the BETA programing language [65]. It contains different application for administration of real-time conferences and source code for building such programs. Important programs and code are:

1) Note: in these case, the application is not "aware" that more than one user exists; it receives only a single input stream (which may be generated by multiple authors).

- conference (Starting the conference object)
- Conference managers
- Registrar
- confreg
- Superclass for building conference-aware applications
- A conference-aware wrapper for starting “external” conference-aware applications (i.e. conference-aware applications which are not specializations of the CTK’s superclass for building conference applications). Examples of such applications include MBONE applications like “vat”, “wb” and “nv”.
- A conference-aware application for starting conference-unaware application under GWTK.

The administration of a conference is done through a centralized conference object. This conference object keeps tracks of participants and applications in a conference. There is one conference object for each running conference. In principle it is possible that a person participates in different conferences at the same time. Each conference object will keep track of which participants and applications are included in the conference. The conference object receive instructions from the conference managers.

The conference managers are the users interface to conference objects. In a real-time conference, it is often desirable to implement different conference policies depending on the situation. This can be done in the CTK by implementing different managers. In the CTK, three different conference managers are implemented.

The registrar registers and stores information about participants, applications and predefined conferences. This information is used by conference objects and managers. The “confreg” program is a user interface to registrar.

In the CTK there is a superclass to be used for building conference applications. This superclass has all the necessary functionality for communication with the conference object — functionality such as: *quit*, *add participant* and *drop participant*.

A wrapper used to start conference-aware applications is also included in the toolkit. This wrapper starts the conference-aware application at each participant’s workstation. The participant’s workstation is registered with the registrar.

The last program of importance in the CTK is a conference-aware application for including stand-alone applications in a conference. The program, called “XyApplRepl”, uses the Xy-window replicator included in the GWTK, but the source code is written in an object-oriented manner which permits developers to change the replicator in an easy way. This is done through specialization of the superclass “applReplicator”.

5.4 The Global Window Toolkit

5.4.1 Overview

The Global Window Toolkit provides functionality for replicating windows, and communication facilities such as telepointing and annotations. Window replication is the means

for sharing application windows among users in real-time, and therefore serves as a basic component when designing conferences (see section 5.3.2 for more details). The MRD's window replication software be modularized, such that the overall system's architecture remain open; in this way substitution of improved window replication software at a later time can easily be achieved. At present, the GWTK uses the Xy package [13] for window replication.

Conference participants might want to show something to others by talking and pointing in shared windows. The audio part is covered by the DSTK (see DSTK descriptions for more details). The GWTK also covers pointing in shared windows. This is done by telepointing; that is, by making a participant's cursor visible on the other participants' screens.

Annotations are similar to telepointers since they are also visible for all participants in the conference. Annotations differ, though, by having static positions.

5.4.2 Functionality

Window replication is done through Xy [13]. Xy acts as an X server which communicates with Xy-agents. That is, an Xy-agent feeds the Xy server with input, and it expects output to transfer along to clients (applications) connected to it. Each user must have an Xy-agent running locally on his/her machine. In principle, every X application can be shared through the Xy window replication system. The reasons for choosing Xy instead of another window replicator were its ability (1) to handle different kinds of floor control[2] and (2) to share arbitrary X applications.

The functionality of telepointers is complicated. When sharing an application, the users involved may move the application's window(s) to different areas upon their respective screens. For each user, the telepointer must therefore detect where the shared window(s) are in order to point at the correct place on the user's screen. In the MRD, telepointers are only visible when pointing within shared windows.

Annotations may be placed while using the telepointer. Like the telepointer, annotations will be shown on each remote screen. In the MRD, each annotation has a label consisting of the name of the user who placed the annotation (optionally, a brief description) and a number. The annotations' positions on the screen may be saved, and can therefore be included in a multimedia message (see the SSCTK for more details).

5.5 The Digitized Sound Toolkit

5.5.1 Overview

The Digitized Sound Toolkit (DSTK) includes two primary applications. One application is a conference audio application; it is based upon the public-domain software called 'vat'[3], which exploits IP multicasting.

2) In certain conference policies, it is critical that there is no restriction upon providing input to (and receiving output from) the shared applications.

3) Copyright (c) 1993-1994 Regents of the University of California. All rights reserved.

The conference audio application provides real-time facilities for multi-party audio conversations over a data network. Within the application, facilities also exist for recording parts of such conversations, as well as editing and/or playing back such recorded information within the context of the same (or any other) audio conference.

The second primary application included in the DSTK is an audio editor; it is a copy of the 'audiotool' binary normally included in the standard Sun OS (and Solaris) software.

5.5.2 Functionality

The DSTK provides an IP[4] multicast-based application ('vat') for real-time conference audio. In addition, the audio conferencing application offers a basic facility for recording the audio contents of an ongoing conference. Once recorded, such information can be edited and/or played back within the same (or any other) conference.

The conference audio application included in this release is integrated with the EuroCODE Conference Toolkit (CTK). All applications included in this release can be run independently of the CTK. Audio files are easily integrated with the Distributed Hypermedia Toolkit (DHMTK) through use of the DHMTK's file component mechanism. Using this mechanism, however, it is only possible to create hypermedia links to entire audio files, not to audio segments within some audio file.

The DSTK also includes an audio editor called 'audiotool'. Audiotool allows for standalone recording, editing and playback of audio data. The conference audio application and audiotool have been integrated with one another. Audiotool is also integrated with the Snapshot Composer Toolkit (SSCTK).

Other functionality within the DSTK includes:

- an application for adjusting audio levels ('gaintool')
- basic recording and playback functions which can be invoked from the command-line ('codeRecord' and 'codePlay')
- a simple, audio recorder/player application ('sounded')
- an application for audio format conversions ('sox')

The implementation approach found in the DSTK is quite different than it's original design found in [38, 24]. However, much of the toolkit's original functional design has been satisfied.

5.6 The Snapshot Composer Toolkit

5.6.1 Overview

The Snapshot Composer Toolkit contains functional elements designed to support the composition, reconstruction and presentation of multimedia messages. The SSCTK does

4) The IP multicasting implementation is an instance of "true multicasting".

not involve itself with the transmission of multimedia messages — that job is performed by standard e-mail services.

Using the functional elements within the SSCTK, an application called the Snapshot Composer has been developed. This application offers access to all of the Toolkit's high-level functionality, as well as a tailorable user-interface.

The Snapshot Composer is fully integrated within the MRD; moreover, it is on equal footing with the MRD's Desktop Conferencing system. In other words, it is these two subsystems which comprise the MRD. The Snapshot Composer and the Desktop Conferencing system are also integrated with one another in such a way that the user can seamlessly shift from one communication context to another.

5.6.2 Functionality

The SSCTK consists of the public-domain packages called `MH` [75] and `metamail`, as well as new software written to exploit the features within those packages. One powerful feature within the SSCTK is that when creating multimedia messages, the SSCTK creates a logical representation of the message. Using that representation, it becomes possible to create alternative kinds of transmission-ready instances of the message (e.g., MIME, HTML, etc.).

Further details concerning the SSCTK's different kinds of multimedia messaging support are found in the MRD's Technical Appendices [41].

5.7 Other MRD-relevant toolkits

The MRD design and development process dovetailed with the design and development of all EuroCODE toolkits; the different processes provided both requirements and results for one another. At certain stages within the MRD prototyping effort, other EuroCODE toolkits were considered for use and integration along with the MRD's fundamental set of materials. Certain other toolkits were tested and evaluated [60]; some are used within the MRD, others are not. Despite the costs involved, the process of toolkit consideration and testing has been of great benefit. It has yielded an overall MRD design with which *future* integration of other EuroCODE toolkits should be a relatively straight-forward implementation task.

The other EuroCODE toolkits most directly relevant to the MRD are:

- the Distributed Application Toolkit (DATK);
- the Temporal Data Toolkit (TDTK);
- the Digitized Video Toolkit (DVTK);
- the User Interface Construction Toolkit (ICTK);
- the Distributed Hypermedia Toolkit (DHTK);
- the Multi-/Hypermedia Message Toolkit (MHMTK); and,
- the Enterprise Information Service Toolkit (EISTK).

5.7.1 The Distributed Application Toolkit

The Distributed Application Toolkit is a toolkit which enables the development of distributed applications and systems. The DATK offers a CORBA-compliant interface. With the DATK, it is possible to define client ↔ server protocols which allow process coordination and data exchange across heterogenous platform environments.

Within the EuroCODE project, the DATK was based upon a product-in-development. Part of that development included mechanisms for C++ ↔ BETA language interoperability. Another part concerned the creation of mechanisms for distributed exchange of temporal (high-volume) data streams[5]. Unfortunately, these two parts of the DATK were not available in time for full integration with the Conference Toolkit and the Temporal Data Toolkit, respectively.

For this reason, the DATK was not used within the current implementation of the MRD; instead, the MRD employs the distribution mechanisms found in the Mjølner BETA System. However, the original designs of the MRD's fundamental toolkits are based upon the DATK; this situation should make any later DATK-based implementation of the MRD a rather straight-forward matter.

5.7.2 The Temporal Data Toolkit

The Temporal Data Toolkit's design provides a uniform means by which to handle and exchange both real-time and stored temporal data in a distributed environment. The toolkit is written such that specializations for audio (i.e., DSTK) and video (i.e., DVTK) do not require inordinate amounts of work.

Originally, the Digitized Sound Toolkit's implementation was planned as an object-oriented specialization of the Temporal Data Toolkit. However, due to the late implementation of the TDTK, the TDTK was not used for the current implementation of the DSTK. Instead, several public-domain software packages were integrated in order to create a DSTK satisfying the required functional specification.

5.7.3 The Digitized Video Toolkit

The Digitized Video Toolkit is an object-oriented specialization of the TDTK. It provides a uniform means by which to handle and exchange both real-time and stored video data in a distributed environment. The DVTK is the toolkit with which video services could be added to the MRD.

5.7.4 The User Interface Construction Toolkit

The User Interface Construction Toolkit is designed to provide a platform-independent language for user-interface programming. In the current version of the MRD, it is used to implement the interface for the CTK's "registrar" class.

5) These streams sometimes require the maintenance of open data transfer connections.

5.7.5 The Distributed Hypermedia Toolkit

The Distributed Hypermedia Toolkit is a generic development framework suited for the development of both single-user and cooperative hypermedia systems. Hypermedia systems developed using the DHTK typically run in a distributed hardware configuration, where editors, hypermedia client processes and object-oriented database server processes communicate as distributed objects.

The first implementation of the MRD's multimedia messaging facility (the Snapshot Composer) employed the DHTK [36, 39, 43]. When evaluating this particular MRD element at Rikshospitalet, the pilot group was pleased with the communication support provided. They criticized the user-interface, however, commenting that too many windows appeared on the screen and that the functionality most central to their needs wasn't made available in a simpler fashion [40]. They also had certain reservations about the degree to which the DHTK-based implementation could support message privacy.

As explained in section 10.3.2.1, the interface-related problems were a result of the fact that Snapshot Composer was built as a *direct extension* of the Distributed Hypermedia Toolkit (DHTK) application framework. Alternatively, the DHTK could have been used as a hypermedia service provider. With such an approach, the DHTK's services are integrated directly into individual applications; in so doing, the DHTK application framework's user-interface is not visible to the user at all. Only the user's individual, hypermedia-enhanced applications appear on the screen. The reason this approach could not be used prior to the MRD's integration at Rikshospitalet was that the time limits and highly-parallel development plan prevented from doing so.

The current implementation of the MRD's multimedia messaging facility no longer employs the DHTK (see section 10.3.4, section 5.6 and chapter 4 for further details). Still, many situations can be identified for use of the DHTK within the overall MRD framework. The most immediate, MRD-related use for the DHTK is to provide tighter integration amongst applications by basing cross-application linking and annotation mechanisms upon the DHTK.

5.7.6 The Multi-/Hypermedia Message Toolkit

The Multi-/Hypermedia Message Toolkit allows applications to submit multimedia or hypermedia messages for transfer via X.400 using XT-PP[6]. Other message transfer services (e.g., SMTP) are also available through XT-PP. The MHMTK defines abstractions for the data types, operations and services needed for the composition, submission, reception and storage of messages.

When creating the final implementation of the MRD's Snapshot Composer, the MHMTK was considered. With regard to level of abstraction, it was judged that the MHMTK's facilities were slightly lower than desired. Furthermore, a good deal of the functionality offered within the MHMTK was already available within public domain software packages.

6) An X.400 product from NEXOR Ltd., England.

Implementation of the MRD's current Snapshot Composer led to the creation of a new EuroCODE Toolkit, the Snapshot Composer Toolkit. This toolkit employs public-domain software packages. Similarities and differences between the MHMTK and the SSCTK are discussed in the MRD's Technical Appendices [41] (see Appendix D, section D.2.3).

5.7.7 The Enterprise Information Service Toolkit

The Enterprise Information Service Toolkit (EISTK) provides access to a EuroCODE Kernel service called the Management Handler. Via this kernel service, the EISTK can provide user-specific information (e.g., organization, email address, telephone number, default machine name and machine characteristics, etc.) to applications. The EISTK is relevant to the MRD in that it could be used to provide information to and update information found in the CTK's Registrar object.

Chapter 6

Evaluation Results

The design, testing and evaluation periods of the MRD were carried out as iterative processes. This chapter primarily addresses the results obtained in the latter phase of the MRD evaluation[1]. In this phase, the evaluation concentrated upon obtaining feedback concerning functionality and user-interface design for the MRD's desktop conferencing and multimedia messaging facilities. This evaluation phase also sought to obtain indications as to where and when communication systems such as the MRD could be useful for the radiologists at Rikshospitalet — the pilot group — as well as for other working groups at Rikshospitalet.

In seeking to explore these issues, three questions were used to keep the MRD testing and evaluation sessions in focus. One question concerned the usability of the MRD, while the other two concerned the system's usefulness:

Q1. Is the MRD easy to work with?

Q2. Can the MRD system make work more efficient at the Department of Radiology?

Q3. Are there other application areas where MRD-like systems can be useful?

6.1 Application Areas for the MRD

This section combines responses received from the pilot group concerning questions Q2 and Q3. In regard to Q3, however, all of the application areas identified involve the Department of Radiology in one capacity or another. The pilot group did not speculate as to the potential usefulness of MRD-like systems for other working groups at Rikshospitalet.

The pilot group identified several different application areas for the MRD during the testing and evaluation periods. Some of the application areas were earlier identified and presented as scenarios during the functional design specification phase [42]. For the sake of completeness, all of the identified areas explicitly mentioned are presented below. The first five mentioned are *intra*-hospital application areas, while the sixth concerns all of the first five application areas in an *inter*-hospital communication setting:

- Ordinary consultations
- Consultation during surgery
- Morning demonstrations
- Remote supervision of radiographers
- Education
- Inter-hospital communication

1) More about the MRD design and evaluation processes is presented in chapters 9 and 10, respectively.

6.1.1 Ordinary consultations

Currently, clinicians at Rikshospitalet must either walk to the radiology department, or use the telephone, in order to consult a radiologist. This is not very efficient. First, it is very time-consuming to go to the radiology department from many of the clinical wards. Second, it is very difficult to discuss physical phenomena within radiological images via the telephone (often the radiologists *together* need to see and point within images, in order that explanations can be given). Third, the consultation must be performed in real-time. This latter point is due to the fact that there does not exist efficient and reliable support for asynchronous communication at Rikshospitalet today; this is especially true for the kinds of communication support required for radiological consultations.

Radiologists' work-days are constantly interrupted by the need to satisfy many sporadic, real-time consultations with clinicians, as well as other radiologists. Through use of the MRD, clinicians and radiologists could have the option to formulate their consultation requests as multimedia messages, including images, annotations, text and/or audio as necessary. If situations called for it, users would also have the possibility of reaching radiologists for real-time consultations[2].

Specific to the context of ordinary radiological consultations, a good organizational implementation of the MRD within the hospital would probably effect several benefits:

- the MRD could reduce the time used to initiate consultations;
- the MRD could reduce the number of interruptions during radiologists' work-days; and,
- the radiologists could plan their work-days much better.

6.1.2 Consultation during surgery

In some situations, surgeons need to consult with a radiologist during surgery. Such situations arise, for instance, when surgeons experience that the patient's true physiology significantly differs from the physiology depicted within the patient's radiological images.

These situations do not arise so very often, but they are very time-consuming for the surgeon, the surgical team and the radiologist (not to mention the patient!). The delays involve locating the radiologist, the radiologist's getting to the operating room and his preparation / sterilization prior to entering the surgical theater.

The pilot group identified this situation as a potential application area for the MRD's conferencing facility. In many circumstances, video images depicting the patient's physiology, as seen by the surgeon, would be a great advantage in such situations. The pilot group considered that the MRD would be very useful in these kinds of situations, especially due to the critical nature and time pressures which often exist at such times.

2) A good organizational implementation of the MRD includes routines for reading snapshot messages and for initiation of real-time consultations.

6.1.3 Morning demonstrations

The pediatric radiologists at Rikshospitalet demonstrate the examinations of children with heart diseases to the heart surgeons. These demonstrations are given at the heart surgeons' morning conferences. Their conference room is located in a building different than that in which the pediatric radiology department is located. Quite often, the pediatric radiologist only needs to demonstrate findings for one or two patients.

The pediatric radiologist in the pilot group meant that it was possible to use the MRD in order to carry out the morning demonstrations for the heart surgeons. This possibility would be especially important on days in which there is only one pediatric radiologist on duty.

The pilot group also stated that the MRD could be useful for recording morning demonstrations in general. Should a clinician miss a regular morning demonstration, that clinician could review and play back the part of the demonstration concerning their patient. In this case, the radiologists expected they would save a lot of time lost due to later interruptions[3].

6.1.4 Remote supervision and guidance of radiographers

Another application area identified by the pilot group involved the remote supervision and guidance of radiographers during the digital image acquisition phase of certain radiological examinations. This application area is especially important for the pediatric radiology department for two reasons. First, the MR and CT imaging devices are located within the main radiology building, approximately 250 meters from the pediatric radiology department. Second, the pediatric radiology department has the policy that images of children must be checked and approved by a radiologist while the patient is still at the radiological department[4].

The procedure for examinations of children is that a radiologist checks the images as soon as they are produced. If the images do not have the required quality, the pediatric radiologist provides new instructions to the radiographers as to which image planes to capture, specific device parameters to use, etc.

The production of an image series can take anywhere from 2-15 minutes or more. Once the series is acquired, the radiologist reviews the images. On occasion, the radiologist must ask for a new image series to be taken, in order that the patient's physical phenomena be depicted as clearly as possible by the device. In such cases, the pediatric radiologist requests the radiographers employ a slightly different image plane and/or device parameters when acquiring the next image series.

3) To completely fulfill the functional requirements involved with recording (the machine-based events within) a morning demonstration, the MRD would require a mechanism by which to record a log file consisting of all operations and cursor movements transpiring during some specified interval of time. Using such a log file, it would be theoretically possible to play back any part of that demonstration. At present, no such facility has been implemented within the MRD.

4) This policy is based on the desire to minimize the number of times the children must undergo these radiological examinations.

The examination process can be repeated several times and may last from 30-60 minutes. While waiting for the production of an image series, there is not sufficient time for them to return to their building and perform any other work.

Using the MRD, the radiologists in the pilot group meant that it is possible to remotely supervise and guide the radiographers during these kinds of examinations. While waiting for the production of an image series, the radiologists could spend their time performing other relevant tasks.

A prerequisite for satisfying the functional requirements inherent in this application area is that the imaging devices and PACS system make the images rapidly available.

6.1.5 Education

The last application area identified by the pilot group is within medical education. There exist many situations in this area where the MRD's functionality could be useful. In recent years, the teaching situation at Rikshospitalet has become more based upon problem solving, using case material from real patients. This pedagogical approach creates a need for supervision by the teacher, as well as encouraging cooperation between the students. The kinds of communication needs which arise within different teacher↔student↔ student groupings can be addressed using the MRD's synchronous and asynchronous communication facilities.

The pilot group meant that it is important to introduce MRD-like communication technology at this educational level. Allowing the students to become familiar with the kinds of communication support available *now* gives them the opportunity to create expectations and requirements about their future work environment later.

6.1.6 Inter-hospital communication

This final application area concerns hospital↔hospital communication. The pilot group stressed that should they have new means by which to communicate with partners outside the hospital — both synchronously and asynchronously — the benefits of the MRD would be much greater. All of the five application areas mentioned above could be application areas for the MRD, within an inter-hospital communication setting.

Members of the pilot group mentioned several hospitals and health organizations outside Rikshospitalet where the MRD could make the work more efficient. These ranged from local hospitals and health institutions in different parts of Norway, to medical experts located all around the world. The greatest advantages in this regard were that the patient, the specialist and/or the images might not need to travel.

6.2 Potential Consequences of MRD Use

The discussion above has focused upon application areas for the MRD; for some of those areas, the manner and/or degree to which the MRD would increase effectivity was also discussed. During the final evaluation phase, several other issues were also addressed.

These issues concerned expectations about the potential consequences of using the MRD within the Department of Radiology. Some of these issues addressed:

C1. quality

C2. security

C3. cost - benefit: organizational

C4. cost - benefit: economical

The radiologists in the pilot group meant that the MRD could help increase the quality of their services. In particular, the radiologists felt that the MRD would help enable them to carry out their daily plans, since there would likely be fewer interruptions during their days. At the same time, they would be available for real-time consultations in acute medical situations. In this regard, it is crucial that the MRD support both synchronous and asynchronous communication.

Security is an important issue in a hospital environment. Patient information is sensitive. The radiologists in the pilot group meant that the MRD system could help increase security.

Whenever images are moved from one place to another, there is always a risk that some images become lost. An electronic PACS system eliminates the need to physically move the original hardcopies of images throughout the daily, *routine* work within the department. The MRD supports communication needs which are often not routine — needs which may often arise "then and there"[5]. In this regard, the MRD enhances a PACS system with communication functionality such that messages about images can be sent directly to the receiver, again without involving the need to physically transport the hardcopies of those images.

With regard to point C3, Grudin [32, 35] stresses that an important reason why many CSCW application fail is because of the disparity between who does the work and who gets the benefit. In order to achieve a successful organizational implementation, many CSCW applications require that some people begin performing additional work. At the same time, it is not always the case that those people are the ones who perceive a direct benefit from the use of such systems.

One radiologist in the pilot group meant that the clinicians will achieve the greatest benefit from the MRD, while the radiologists' work-load will increase. The reason for this is that the radiologist will be more accessible for the clinicians. Still, there exist many factors of uncertainty with respect to this issue.

One of those factors concerns how the PACS system is used within the hospital. Since a PACS system can make it easier for clinicians to access radiographic images, it can also decrease the need for consultations with radiologist. Another factor concerns how PACS and MRD technology are organizational implemented.

Producing a cost - benefit analysis regarding the economic side of MRD use was considered an extremely difficult task. There exist innumerable factors that can be brought into any such analysis, and the question becomes thereafter one of scope. One perspective which developed during the evaluation period was this: should the MRD make work-

5) Although communication needs often arise "then and there", the *satisfaction* of that need, in many cases, can wait until a later time.

processes more time-efficient within the department, and should this savings increase patient throughput within the department, one could then argue that the MRD helps promote "more health per dollar" within the community.

Chapter 7

Summary and Conclusions

As described in section 1.1, the MRD is one of three demonstrator systems developed within the scope of the EuroCODE project [23]. Each of the systems aims to demonstrate some of the kinds of cooperative work support attainable when using differing degrees of bandwidth.

The Department of Radiology at Rikshospitalet, Oslo, was selected as the pilot group with which the Middle Road Demonstrator (MRD) would be designed, tested and evaluated. The selection of the pilot group was based upon the fact that that department produces and works with an extraordinary amount of medical imagery. Furthermore, communication and cooperation needs often arise amongst different physicians and clinicians: these medical images must often be reviewed and discussed by persons normally situated within physically separated parts of the hospital.

At the beginning of the EuroCODE project, the MRD was contractually predefined to include a desktop conferencing facility. The catch phrase for that facility at the time was *"show, point and talk..."* [30]. This catch phrase reflected the MRD's functional foci then; that is, via the data network, the MRD would enable a group of conference participants to simultaneously view, point within and discuss information presented by various applications.

At the outset, the functional characteristics defined within the MRD's desktop conferencing facility satisfied certain kinds of communication requirements (see sections 3.2.1 and 3.2.2). However, studies of the pilot group's concrete work-situation revealed other matters which needed to be taken into account: the radiologists moved about constantly between the examination rooms, the demonstration room, the laboratories, their offices, etc. A communication system supporting only real-time, synchronous conferencing was judged to be inadequate for their daily needs [42].

As mentioned in section 2.1.2, the requirements definition phase of the MRD led to the identification of a vital need for communication facilities which would support coordination and cooperation for this "highly mobile" user group. For this reason, the MRD design was extended to include facilities for multimedia messaging.

Despite this significant design extension, the MRD's original functional profile — the ability to show, point and talk — remained the same. The MRD's current, multimedia messaging facility allows one to create a message: (1) by selecting documents to send (via mouse-clicking upon them); (2) to place simple annotation marks (e.g., arrows) atop such documents; and/or, (3) to include an audio and/or text message along with the documents and annotations. Together, the MRD's desktop conferencing and multimedia messaging facilities led to a new communication system concept. That is:

...the MRD allows for sharing applications independently of time and place.

The MRD was developed using a shell-based approach. That is, EuroCODE's CSCW Shell was comprised of a set of inter-operable software elements called 'toolkits'. Various parts of the toolkits were assembled in order to create each of EuroCODE's three demonstrators. The MRD is primarily comprised of four different toolkits: the Global Window Toolkit, the Conference Toolkit, the Digitized Sound Toolkit and the Snapshot Composer Toolkit.

The MRD underwent a six-week evaluation at the pilot site. During that period, the system was integrated together with the Department of Radiology's digital image archive. The radiologists in the pilot group tested and experimented with preliminary versions of the system. They provided a great deal of feedback, especially in regard to the system's user-interface. Since the completion of the evaluation period at Rikshospitalet, a complete overhaul of the MRD's user-interface has been carried out. At the time of writing, however, this final version of the MRD has not been evaluated by the radiologists.

The evaluation at Rikshospitalet showed that the MRD concept has great potential for making the Department of Radiology more effective. Several application areas involving the Department of Radiology were identified, and the pilot group meant that the MRD could contribute to improve both security and quality within the department. Although it was only possible to test the MRD within simulated work situations, it is believed that the results from this evaluation are reliable and indicative.

With it's mechanisms for real-time application sharing, telepointing and conference audio, the MRD's desktop conferencing facility helps eliminate barriers normally associated with geographically-distributed cooperative efforts. The MRD's multimedia messaging facility also helps eliminate barriers; in this case, barriers primarily associated with temporal distribution. The multimedia messaging facility provides the equivalent of a desktop conference, except for the real-time feedback. In both of it's communication modalities, the MRD allows:

- work within applications to be shared;
- pointers to be used for directing attention; and,
- questions to be asked and answered orally.

The desktop conferencing and multimedia messaging facilities ensure that the MRD allows for sharing applications independently of time and place. Moreover, it is in light of these views and perspectives that the MRD is defined to be a system for communication support within cooperative work settings.

Chapter 8

Further Work

Numerous demonstrations for potential user organizations were performed during the project's final year. These demonstrations primarily addressed the health sector, though demonstrations were also performed for representatives within insurance, engineering, library and shipping businesses. All parties expressed positive response, and several expressed a desire to buy the MRD as a product.

In parallel, the MRD was presented for industrial parties, with focus upon actors within the health care market. One of the largest such enterprises in Norway is the Sysdeco Group. **The Sysdeco Group has decided to commercialize the MRD.** They see a market for communication systems, both intra- and inter-hospital. At the time of writing, the MRD is available only on the Unix platform. The Sysdeco Group plans to look into developing the MRD for the PC platform.

Another interested party in Norway is Uninett Ltd. They provide the academic community in Norway (e.g., universities, research institutions, etc.) with network infrastructure and services. Uninett Ltd. wants Norsk Regnesentral to provide the MRD via Uninett, as a collaboration tool for the academic community .

Part 3

Process

Chapter 9

MRD Design and Development

The goals of the EuroCODE project included the development of a CSCW Shell and the development of three demonstrator systems (see section 1.1). The CSCW Shell contains toolkits for the development of CSCW applications and systems, and the demonstrators were built using these toolkits.

EuroCODE's project plan for Shell and demonstrator creation was essentially phase-oriented, with supplementary activities aimed to provide feedback and revision. The primary, temporal sequence of phases within the EuroCODE project was (1) elicitation of requirements; (2) CSCW Shell design and development; (3) demonstrator implementation; and, (4) demonstrator evaluation. Within the EuroCODE project plan, the timing of these phases actually contained a substantial degree of parallelism. It was known from the beginning that there would be need for iteration as the project teams began working together with the different pilot groups.

The plan for creation of the MRD was situated within the framework of the EuroCODE project plan as a whole. The EuroCODE project plan was designed such that the MRD activity would contribute three[1] toolkits to the CSCW Shell. This situation necessitated that the MRD activity not only generate specific functional requirements for the MRD, but that it also generate specific functional requirements for the CSCW Shell. To the EuroCODE project as a whole, the MRD activity was also scheduled to deliver toolkit implementations satisfying the MRD-specific requirements.

This chapter briefly summarizes the requirements gathering, design and development phases. It describes these phases from an MRD perspective, rather than a toolkit perspective. In section 9.1, the pilot group is briefly introduced. Section 9.2 presents some of the techniques used when eliciting and analyzing user-requirements. Section 9.3 provides some Shell design and implementation information, while section 9.4 describes certain conditions which impacted system implementation and integration. A wealth of information concerning the topics in this chapter can be found in [2] and [42].

9.1 The pilot group

Throughout the different phases of the MRD activity, the MRD project group enjoyed fruitful cooperation with the pilot group at Rikshospitalet. The kernel of the pilot group consisted of four radiologists, all members of the Department of Radiology at Rikshospitalet. These radiologists were active in all phases of the project. They are quite technically-oriented, and especially interested in PACS[2] systems. It is their interest that Rikshospitalet

1) Ultimately, the MRD activity contributed four toolkits to the CSCW Shell; these are described in the MRD's Technical Appendices [41].

become a national PACS expertise center in Norway. They have therefore been interested in experimenting with PACS systems, as well as systems which can enhance the usefulness of such systems. The MRD, which can provide new communication possibilities with respect to images stored in a PACS system, creates value-added services for the PACS.

9.2 Requirements gathering

The functional requirements for all EuroCODE toolkits were acquired through the requirement acquisition and preliminary design phases for each of the demonstrators. As the different project teams worked together with their associated pilot groups, functional requirements were obtained and thereafter analyzed. Whenever a requirement's nature was general enough for one of the Shell's toolkits, it was passed along to the project team responsible for that toolkit; otherwise, that requirement was considered demonstrator-specific (and satisfied within that demonstrator's technical implementation).

The motivation underlying the selection of the radiological department as the pilot organization was presented in section 2.1.1, while the MRD's overall development and evaluation setting was generally presented in chapter 2. The gathering and analysis techniques used in the requirements acquisition phase are roughly summarized below. The order of the presentation basically reflects the order in which they were applied.

- Observing the work of the users by simply being there, walking along with them and asking question now and then.
- Interviewing the users and other relevant persons.
- Formally describing the work and workflow.
- Creating scenarios. These scenarios described both the way they worked in the present situation, as well as how they could work in the future, given the introduction and implementation of new technologies. These scenarios were developed in cooperation with the pilot user group at Rikshospitalet. In this way, the users could participate in the design phase, and explicitly or implicitly express their own design ideas.
- Using checklists in order to test the scenarios from different perspectives. The checklists used were both work-oriented and technical.
- Presenting different prototypes for the user group.
- Arranging meetings and workshops where design ideas were discussed.

Fundamental to our design approach was to root design in work practice. As part of the way of achieving this, the EuroCODE Conceptual Framework was used [15]. As a basis for *designing* systems, the Framework describes how to create and apply scenarios, in order to illustrate their use.

> *No common definition as to what constitutes a scenario is available. Still, the various conceptualizations of the term seem to share in common the idea that a scenario is hypothetical, it is selective, and it relates its elements temporally.*
>
> *(Bødker et al., [15], p. 77)*

2) Picture Archive and Communication Systems.

In the EuroCODE Conceptual Framework, all scenarios are not hypothetical, future work situations; some are descriptions of current work situations. To guide scenario making, the EuroCODE Conceptual Framework offers a *work-oriented checklist* and a *technical checklist.* The work-oriented checklist was useful when focusing on various aspects of the work context: the physical setting, actors involved, related activities, awareness issues, etc. The technical checklist was applied when characterizing the MRD itself. In short, these checklists guided the creation of MRD scenarios.

Three different scenarios describing current work situations within the Department of Radiology were created. Several scenarios describing future work situations involving use of the MRD were created. All of the "current-work-situation scenarios" and five of the future scenarios are documented in [42].

For the future scenarios, rather detailed sequences of display screen images were created. This was done in order to capture interface aspects as well as to clarify MRD functionality. This manner of extending the scenario part of the EuroCODE Conceptual Framework has been influenced by storyboard prototyping.

Similar to film production, the use of storyboards in system design is a way to "sketch out" the future system (see Madsen and Aiken [64] and Andriole [6]). The key to storyboard prototyping is to use sequences of display screen images, illustrating a task-driven view of future systems. Storyboards may actually be seen as one particular kind of scenario, a kind which focuses on the future computer system's user-interface.

The MRD scenarios were used in three ways: (1) to facilitate design discussions in within the system design group; (2) to facilitate discussions with the pilot group; and, (3) to communicate the design solution in different situations.

During this phase, other relevant system prototypes and CSCW applications were demonstrated for the pilot group. These demonstrations also provided important input to requirements for the MRD.

9.3 Shell design and implementation

The next phase of the project was Shell design and implementation. This phase was partly done in parallel with the requirements gathering phase presented above. Generic requirements for both the MRD and the Shell were captured, and general technical solutions were implemented within the Shell. Different modeling and specification techniques were used during this work. An example of a technique used during this phase is use of the specification language ECOOL [55].

In actuality, implementation of the toolkits began prior to their complete specification. Full-scale implementation began once the definitions of inter-relationships amongst the various toolkits became relatively stable. Within some toolkits, public-domain software and "shareware" products were used as basis for the implementation; other software within the toolkits was implemented from scratch.

9.4 Demonstrator implementation and integration: prevailing conditions

The original project plan specified that the MRD was first to be implemented, and thereafter integrated with the Rikshospitalet environment during the first quarter of 1995. For political reasons, the radiologists wanted to conclude and document an evaluation of the MRD — a version of the MRD integrated at Rikshospitalet — by late December, 1994[3].

In order to integrate the desired functionality with the Rikshospitalet environment within that time frame, a preliminary version of the MRD (the MiniMRD) was created. The first version consisted of a set of loosely integrated applications which, together, provided the basic functionality. These applications were not tightly integrated, nor did they offer a homogenous user-interface.

Evaluation and revision of the MiniMRD was carried out while the system was integrated with the Department of Radiology's digital image archive. Despite much hard work at that time, the different parts of the MiniMRD were never quite as tightly integrated as intended. On the other hand, the radiologists came with feedback which ultimately had an extremely significant impact on the system.

Following the evaluation of the MiniMRD at Rikshospitalet, the change requests from the radiologists were prioritized, and a final version of the MRD was created. The pilot group's comments led to a complete overhaul of the system's user-interface. In addition, the technical implementation of the MiniMRD's multimedia messaging facility (the Snapshot Composer) was also redone.This work also resulted in a new toolkit, the Snapshot Composer Toolkit (see section 5.6). We also arranged some workshops with a user-interface expert in this phase of the project.

The methodology used during the evaluation is presented in section 10.2, and details concerning the outcome of the evaluation are presented in chapter 6.

3) End-of-year submissions to national research organizations were due at that time. Since the radiologists were interested in Rikshospitalet's future with respect to PACS research, they wished to deliver a submission which included the MRD evaluation.

Chapter 10

The MRD Evaluation Process

The MRD's design, testing and evaluation efforts were carried out as iterative processes. In actuality, the MRD was evaluated throughout the entirety of the EuroCODE project. First as a concept and later, as demonstrations and workshops were held with the pilot group and their colleagues, more as a prototype and a "real" system. As the project effort proceeded, the radiologists became more and more eager for the system become finalized such that they could use it within their own working environment.

This chapter emphasizes primarily three things: (1) a preliminary assessment; (2) the evaluation methodology; and, (3) feedback obtained in the latter phase of system evaluation. Only short mention is made of the evaluation results here; more detailed information is presented in chapter 6.

The latter evaluation phase is best demarcated as the time during which the MiniMRD (see section 9.4) was integrated with the Department of Radiology's digital image database archive. During that time, two versions of the MiniMRD were tested and evaluated by the pilot group.

During the evaluation period at Rikshospitalet, the project group did not stress a differentiation between the MiniMRD and the final version of the MRD which was to be completed following the evaluation period. Still, the users were aware that they were testing and experimenting with an early version of the system. In the text which follows, the terms 'MRD' and 'MiniMRD' are occasionally used synonymously. The term 'MiniMRD' is used whenever a clear distinction is called for.

The latter evaluation phase concentrated upon obtaining feedback concerning functionality and user-interface design for the system's desktop conferencing and multimedia messaging facilities. This phase also sought to obtain indications as to where and when MRD-like communication systems could be useful for the radiologists at Rikshospitalet, as well as for other working groups at Rikshospitalet.

In seeking to explore these issues, three questions were used to keep the system testing and evaluation sessions in focus. One question concerned the system's usability, while the other two concerned the system's usefulness:

Q1. Is the (Mini)MRD easy to work with?

Q2. Can the (Mini)MRD make work more efficient at the Department of Radiology?

Q3. Are there other application areas where MRD-like systems can be useful?

Section 10.1 describes an advance assessment performed in order to help associate organizational benefits one might expect with specific, potential technological changes at the pilot site. Section 10.2 describes the evaluation methodology. Section 10.3 addresses primarily Q1; it includes many of the radiologists' comments about the MiniMRD's user-interface, and mentions what was done to improve it. Section 10.4 provides a very brief summary of the evaluation results, while section 10.5 closes with a description of some

of the problems which arose during the evaluation. It should be noted here that the presentation in this chapter assumes that the reader is relatively familiar with the material found in chapters 3 and 4, as well as the terminology developed in section 5.6.

10.1 Preliminary Assessment of Communication Support

In preparing to address questions regarding the MRD's usefulness, it was deemed appropriate to perform a preliminary assessment. The purpose of the assessment was to associate organizational benefits one might expect with specific, potential technological changes at the pilot site; in other words, the assessment would help when trying to determine to which technology a specific organizational benefit could be attributed.

The assessment concerned a comparison of the expected and/or potential functional differences in work routines within the Department of Radiology, given three different technological environments. The three environments were (1) the present-day situation; (2) a future situation in which the department has included a digital PACS[1] system implementation; and, (3) a future situation in which the department has included a digital PACS and MRD system implementation.

The prerequisites of the assessment included an understanding of the current, daily work routines at Rikshospitalet — work routines based upon hard-copies of radiological images. It was also necessary to have a basic understanding as to the way in which these routines could be improved given the use of a PACS system. Lastly, it was necessary to have a full understanding of the MRD. This latter prerequisite was the determining factor in deciding it best to perform the assessment without the involvement of the pilot group.

The assessment was structured using Johansen's time/place classification of collaboration [50]. Johansen's classification breaks collaborative work settings into two dimensions, time and space; within a specific setting, collaborating parties work at the same time or different times, and they work at the same place or different places. Johansen's classification is illustrated in figure 10. The different quadrants are numbered I-IV here, in order to ease discussion later.

I Same Time Same Place	II Different Times Same Place
IV Same Times Different Places	III Different Times Different Places

Figure 10 : *Johansen's time/place classification of collaboration [50].*

For each of the collaborative work settings (I-IV), a brief summary is given regarding functional differences within three different technological environments at Rikshospitalet.

1) Picture Archiving and Communication System.

10.1.1 Same time - same place (I)

Present situation

Within the radiological department at present, much of the cooperation happens at the same time and in the same place. This is due to the fact that the department's routines are based upon radiological images stored as hard-copies. This means that in a situation where someone wishes to discuss an image with a radiologist, the communication must transpire in a face-to-face context, and the image must be there then as well!

When interacting face-to-face, verbal, non-verbal and paraverbal information is communicated (see section 3.1.2).

PACS

The primary benefit achievable with the addition of this technology is derived from the use of digital images. With many PACS systems it is possible to invoke different kinds of image processing and enhancement routines, in order to better discern the nature of the phenomena in question. As a potential drawback with PACS systems when compared to light panels, the possibility to observe and browse through large number of images at the same time is significantly reduced.

PACS + MRD

The MRD does not add any new functionality within this collaborative setting.

10.1.2 Different times - same place (II)

Present situation

In the present situation, collaboration at the same place but different times is supported by today's work-routines (e.g., change of job shifts, etc.). It is also possible to leave notes and other (non-electronic) messages for each other.

PACS

In general, PACS systems provide no explicit support for collaboration within this particular setting. It is possible to mark and annotate interesting images for persons to be arriving later, but no explicit communication support is provided.

PACS + MRD

The MRD's multimedia messaging facility supports communication within this collaboration setting. For example, the radiologist who interprets a patient's pictures can send comments to the radiologist scheduled to present those images at the morning conference.

10.1.3 Same time - different places (III)

Present situation

At Rikshospitalet today, the telephone is the only alternative for synchronous communication within this collaboration setting. During unplanned radiographic consultations, the images are simply discussed via telephone. The telephone is an especially important communication media when clinicians wish to start treatment of the patient as soon as possible.

The disadvantage in this situation is that only one of the communicating parties has access to the hard-copies of the images.

PACS

With a PACS system, images are accessible in different places at the same time. This makes it easier for clinicians to study the images themselves. For those situations in which people wish to communicate synchronously, it is still possible to use telephones. In this way, the communicating parties can look at the same images on their screens.

The problem with this solution is that the communicating parties must perform "extra" work in order to synchronize the communication content (i.e., one person must tell the other which images they are looking at, and which PACS-related operations they are performing, in order the make sure that the other is seeing the same thing on their screen). The communicating parties must also use careful anatomical descriptions in order to guide one another's attention to the important phenomena within the images.

The radiologists in the pilot group say that this strategy is sufficient when talking about normal anatomy. However, it is often the case that patients with complex difficulties are referred to Rikshospitalet. In these cases, the standard conceptual framework normally used to describe human anatomy is not sufficient. In these cases and others, it is often necessary to point at phenomena within the images.

PACS + MRD

Technical integration of the MRD with a PACS system can serve to eliminate the two problems named above. When running in a "free-floor" mode, use of the MRD's desktop conferencing facility allows any of the conference participants to input data. This circumvents the need for users to tell one another about which images they are viewing, etc. The MRD's telepointer provides a mechanism through which each conference participant's pointer can be seen simultaneously by the other participants. The MRD's conferencing facility also provides for multi-party audio communication via the data network.

10.1.4 Different times - different places (IV)

Present situation

In this collaboration setting it is necessary to distinguish between routine work and unscheduled / unplanned cooperation. At present, coordination of routine work at different times and different places is primarily supported by requisition forms; these are filled out, and thereafter accompany the patients' cases as they are handled by different groups within the hospital [59]. These requisitions will also probably have an important role in coordinating routine work-processes in the future.

For non-routine work such as unscheduled and unplanned consultations, the only possible ways to communicate with others at Rikshospitalet is by writing letters, reports and notes.

PACS

In general, PACS systems do not give any explicit support within this collaboration setting (except as mentioned in setting II, see section 10.1.2).

PACS + MRD

As in setting II, the MRD's multimedia messaging facility supports communication by enabling the possibility to share applications asynchronously. For the pilot group at Rikshospitalet, a typical message could consist of images, annotations, text and/or audio messages.

The MRD's multimedia messaging would enable the staff at Rikshospitalet to communicate asynchronously in situations where immediate response was not necessary. With regard to messages which consist of images, annotations and text, the doctors at Rikshospitalet have communication means which are relatively reliable, but not so efficient.

10.2 Evaluation Methodology

The evaluation period was divided into three parts:

1. Introduction of the (Mini)MRD
2. Continuous use
   - Observation and support during use by the pilot group
   - Demonstration for clinicians, radiographers, and radiologists (other than the ones in the pilot group).
3. Final evaluation of the MiniMRD

Members of the pilot group were observed while using the system. In addition a semi-structured interview guide [40] was used as a basis for more directly addressing the general questions formulated as Q1-Q3 above. An abbreviated version of the questions appearing in the interview guide is given below:

G1. What do you think about the MRD-concept?

G2. What do you think about the user-interface?
(The MRD as a whole, and each of its components.)

G3. How could such a system influence the daily work at the hospital?

G4. In which situations do you expect that the hospital staff could benefit from MRD use, and in which situations do you expect that it would create extra work for the staff?

G5. Do you see other application areas for the MRD?

G1 and G2 were mainly used in the "introduction" and "continuous use" parts of the evaluation, while each of G1-G5 were used in the "final evaluation" part.

10.2.1 Software elements evaluated

During the evaluation period, both the synchronous and asynchronous communication modalities were tested. The following applications comprise the synchronous mode:

- Conference Manager - the interface through which a conference is controlled. Functionality concerns, for example, which persons and which applications are included in the conference, addition of participants and applications to the conference, etc.

- Conference Audio - controls the audio connection between the participants in the conference (e.g., mute, volume etc.).
- Telepointer - allows the display of participants' cursors for the other participants.
- Registrar - registers all ongoing conferences and gives the user the possibility to request joining an ongoing conference. The registrar can also be used to predefine conferences.

The following applications comprised the asynchronous mode:

- Snapshot Composer - makes a snapshot of the windows displayed on the screen. Messages can be sent to persons who could not attend a conference (based on the Hypermedia). The Snapshot Composer also controls a list of (incoming) snapshot messages.
- Audio Editor - gives the possibility to record a message to be included in the Snapshot

The final application was a simple application for which the MRD provided communication support:

- MRD-PACS - interacts with the Department of Radiology's digital image archive, allowing search for patient examinations and display of the associated MR images.

10.2.2 The evaluation setting

The MiniMRD was tested during a six week period at Rikshospitalet in Oslo. The work situation was communication between the neuroradiology section and the pediatric radiology section. These sections are located in different buildings at the hospital. The distance between these buildings is approximately 250 meters. This setting is not explicitly handled amongst the scenarios presented in the MRD design document [42]. In principle, however, it is equivalent to the "unscheduled consultation" scenario and the "consultation via messages" scenario. The difference rests in that here, the communication was between radiologists, rather than between a radiologist and a clinician, as described in the scenario.

Three machines were installed at Rikshospitalet during the evaluation. Two of the machines were placed in the conference room for the pediatric radiology section and the neuroradiology section, respectively. These rooms are located centrally within both sections. The radiologists do much of their preparation for morning conferences (i.e. meetings with clinicians) in these rooms. They also do some "finishing up" in these rooms, once the morning conferences are completed. The third machine was placed in a meeting room within the neuroradiology section. This machine was used as a server for the other two machines, and was used for development and testing without disturbing the actual work at the hospital.

10.2.3 Procedure

As mentioned above, the evaluation period was divided into three parts:

1. Introduction of the (Mini)MRD
2. Continuous use
3. Final evaluation of the MiniMRD

10.2.3.1 Introduction of the MRD

A first introduction to the MiniMRD took place at Rikshospitalet December 22, 1994. The concept of the MRD was demonstrated for the pilot group, and they used the system for the first time. All of the radiologists were located in the neuroradiology section, while one of the system developers was located in the pediatric radiology section. This was done such that it was possible to demonstrate the interactive parts of the MRD between these sections. All comments and suggestions from this introduction workshop were written down.

There were also individual introductions to the system with each member of the pilot group. The comments and suggestions from these individual introduction meetings were logged.

10.2.3.2 Continuous use

This part of the evaluation started in late January, 1995. The MiniMRD was tested during a six week period at Rikshospitalet. At least one system developer was present to provide continuous support, as well as observe the use of the system. Notes from these observations were written down in a log. When possible, informal semi-structured interviews based on the interview guide were carried out.

The MiniMRD was also demonstrated for other groups at Rikshospitalet. Comments and suggestions from the participants at the demonstration were logged. A final demonstration at Rikshospitalet was planned and carried out in cooperation with the pilot group; in fact, it was the pilot group who gave the demonstrations itself. Both the RiksPACS system and the MiniMRD were demonstrated. The participants at this demonstration were clinicians, secretaries, management staff, radiologists (others than in the test group) and the director of the hospital.

10.2.3.3 Final evaluation of the MiniMRD

At the end of the evaluation period, a workshop together with the pilot group was arranged at NR. Experiences with use of the (Mini)MRD were discussed. During this workshop, three other relevant systems were demonstrated. These systems were the MADE application, WWW and Osiris. The MADE application offers functionality for the aggregation, processing and presentation of multimedia information (including video); it is tailored for creating report templates — and instantiating them — for ultrasound laboratories. Developed at the University of Geneva's Digital Imaging Unit[2], Osiris is an advanced application for viewing, manipulating and annotating radiological images.

10.2.4 Instruments

Only paper and pencil were used at Rikshospitalet during the evaluations. This due to the fact that patient information is sensitive. The radiologists strongly advised against recording tapes, neither video nor sound.

2) See http://expasy.hcuge.ch/www/UIN/UIN.html

10.3 Prototype Development and Pilot Group Feedback

10.3.1 Background

As mentioned earlier, the pilot group evaluated the MiniMRD (see section 9.4), rather than final version of the MRD presented in chapter 4. The applications within the MiniMRD were not tightly integrated, nor did they offer a homogenous user-interface.

The fact that the radiologists had a very coarse version of the system to evaluate had the effect that many of the comments were concentrated around details in the user-interface, rather than the functionality the system offered[3]. At the same time, however, the pilot group came up with some new basic ideas about the user-interface which had not captured in earlier design phases. These new ideas ultimately had a very strong influence on the design of the MRD's mechanism for multimedia messaging.

This section covers the evaluation of the MiniMRD's first two versions; these were tested and evaluated at Rikshospitalet during January and February 1995. The user-interface and functionality found within the final version of the MRD (see chapter 4) is based upon the comments and requirements derived during this evaluation phase. At the time of writing, the final version of the MRD has not yet been evaluated by the radiologists.

10.3.2 The first version of the MiniMRD at Rikshospitalet

The first version of the MiniMRD was started by selecting "MRD" within a popup-menu activated from the background window. Figure 11 shows the appearance of the screen just after having started the first version of the MiniMRD. The figure illustrates that the components lack a consistent user-interface.

To achieve seamless transition between asynchronous and synchronous communication modalities (see section 3.2.3), all applications required to support both those modalities were started when the MiniMRD was started.

In the upper-left corner of the figure is the MRD-PACS application. Just under that application is the top-level window of the first, DHTK-based version of the Snapshot Composer; below, one of the Snapshot Composer's sub-windows lists the multimedia messages found in one user's mailbox ("frode: InBox").

In the lower-right corner of the figure are found the Conference Manager, the conference audio application ("dstk") and the telepointer application ("xytel"). In the middle of the figure is an audio editor ("AudioTool v3").

3) For this reason, most of the material presented in this section addresses question Q1 regarding the usability of the MRD (i.e., "Is the MRD easy to work with?").

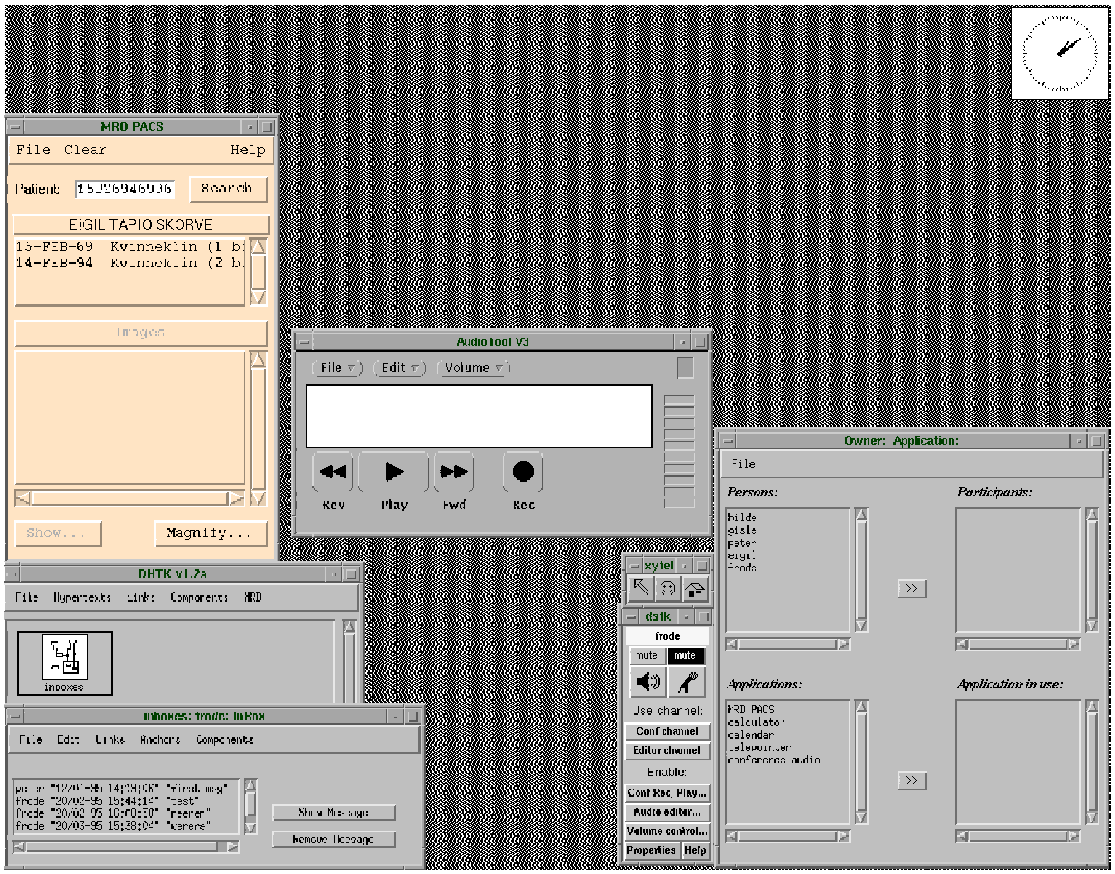

Figure 11 : *The user-interface for the first version of the MiniMRD.*

The main problem with this version of the MiniMRD was that it's appearance was very complex. It had too many windows, and the windows had different layouts. A related problem is that the operations by which to interact with the different applications were not consistent. One example is that it was necessary to click on the right mouse button to save a file in the audio editor yet, at the same time, the right mouse button had no functionality in other applications. The pilot group complained that this was too complex. In addition, they complained that the MRD occupied too much space on the screen. Efficient use of screen "real-estate" is critical when working with radiographic images, since they require so much screen space.

With regard to overall system complexity, one of the radiologists had the following comments:

> *The system must be so easy to use that an assistant physician can use it on her first day at the hospital!*
>
> *I have seen a system in America which was operated by using 4 colors. The MRD should be as easy to use.*

The pilot group also commented on each of the components. The discussion below groups the components with respect to the communication modality in which they are used[4].

4) The MRD-PACS application is not a true part of the MRD; it is an application for which the MRD provides communication support. For this reason, no comments regarding the MRD-PACS application are presented below.

10.3.2.1 Asynchronous mode

The components specific to the MRD's asynchronous communication modality are the Snapshot Composer and the audio editor. The pilot group's main comments regarding these components were:

C1. Too many levels and windows when using the Snapshot Composer.

C2. Non-logical manner for deciding what should be included in a snapshot and what should not.

C3. A user's multimedia mailbox (inbox) did not automatically appear when the user started the Snapshot Composer.

C4. The audio editor's user-interface was quite different than that of the other applications; it was also clumsy to use for certain operations (e.g., saving data).

C5. Lack of a facility for creating static annotations.

Snapshot Composer

The pilot group had very positive feedback concerning this concept, but there were also many problems in using it. The main complaints about the Snapshot Composer are summarized in comments C1-C3.

Most of these problems were a result of the fact that Snapshot Composer was built as a *direct extension* of the Distributed Hypermedia Toolkit (DHTK) application framework. The DHTK application framework[5] is a large system with many possibilities; it's user-interface, in turn, is rather complex.

To elaborate upon points C1 and C2, the problem was that the user had to go through several levels/steps in order to compose and send a message. First, the user had to decide which images which should be part of the snapshot, and display them (should they not already appear on the screen). As another step, the user needed to create audio and/or text messages, if such were desired. At this point, the number of windows is already quite large.

The user then had to locate the Snapshot Composer's top-level window; it was there the "MRD" menu was located. Once the menu was found, the user would then select the "snapshot" item from the menu. This selection activated a new window in which the user could specify the recipient of the message, as well as (optionally) include information regarding the "Subject:" of the message. Once these were specified, the user pressed an "OK" button within the same window.

At that point, the system would automatically create a snapshot, which appeared for the user as yet another window; technically, a snapshot was a hypermedia *table-top composite* [36]. This table-top composite's window contained an icon representing each snapshot-

5) As an object-oriented application framework, the DHTK allows one to immediately create hypermedia-based applications by simply adding and specializing object classes to the DHTK's set of existing object classes; this is exactly how the DHTK-based version of the Snapshot Composer was created. The decision for using this approach was based upon the abbreviated schedule within which the MRD was to be integrated with Rikshospitalet's environment.
Alternatively, the DHTK could have been used as a hypermedia service provider. With such an approach, the DHTK's services are integrated directly into individual applications; in so doing, the DHTK application framework's user-interface is not visible to the user at all. Only the user's individual, hypermedia-enhanced applications appear on the screen. The reason this approach could not be used was that the time limits and highly-parallel development plan prevented doing so.

aware application (see section 5.6) on the screen. In the majority of cases, this implied that the snapshot contained more information elements than those the user really intended to include in the message. In order to avoid sending these extra elements in the message, it was necessary to remove them from the table-top composite. To do so, the user was forced to select the icon he wanted to omit from the message and press a "Delete" button; moreover, the user was forced to do this on an element-by-element basis.

The radiologists found all these steps to be rather exasperating, and it made them feel that the system was very complex. The pilot group meant that the solution with deleting elements from the snapshot was not logical. Instead they suggested that it was much more logical to select what to send by clicking on the windows. This desired functionality has been implemented within the latest version of the MRD. As yet, however, it has not been tested by the pilot group at Rikshospitalet.

The users' complaint C3 was that instead of having their own "private" multimedia mailbox open up when the Snapshot Composer started, they were presented with a list of mailboxes (much like a set of postal shelves at an ordinary office). From this list, they were forced to select and open their own mailbox. The reason it was not possible to have each user's mailbox open up when the Snapshot Composer started is that the technical implementation of the DHTK did not allow such a possibility.

The pilot group mentioned different problems associated with this situation. First, it created an unnecessary step that the users had to perform before they could read their messages. A second problem was that the users had access to other persons' mailboxes. This could well be considered a serious security problem at a hospital, since patient information is very sensitive. In the latest version of the MRD, each user's mailbox is completely private.

Audio editor

The other component used within the MRD's asynchronous communication modality was the audio editor. This application was an application that allowed recording, playback and simple editing of audio data. The pilot group had some different opinions about this application, but they agreed that the user-interface should have been consistent across all MRD applications.

They also agreed that it should have been much easier to save audio messages. A few operations were necessary in order to save the audio data; these included entering the name of the file in which to save it. In the latest version of the MRD, saving audio messages requires no operations whatsoever.

There were different opinions about the usefulness of the audio editing facilities. The question was ultimately whether they needed to edit audio messages at all. The conclusion was that only recording, playing and rewinding are necessary[6]. The latest version allows only recording and playback; no time was available in which to implement the rewind mechanism.

Missing Functionality

Functionality the users found lacking within the MRD's asynchronous communication modality was the possibility to create annotations. They wished to point out or mark different structures within images. The reason for this was that it would have made it easier

6) These are exactly the same operations available on the dictaphones they use daily.

to reference different phenomena within the images and, consequently, use fewer words within the message[7].

One of the radiologists said that, in many circumstances, simply an arrow pointing at some phenomenon could constitute a sufficient message. The radiologist knew that when some receivers of such a message see the arrow, their knowledge and experience tell them what the problem is. Furthermore, the sender's personal knowledge of the recipient would help the sender know just how much information he needs to include in order to specify the query[8].

For the radiologists' situation when communicating asynchronously, annotation functionality is a absolute requirement. The functionality they wanted did not necessarily need to be so very sophisticated. In the latest version of the MRD, a simple, application-independent annotation facility is included.

10.3.2.2 Synchronous mode

The components specific to the MRD's synchronous communication modality are the Conference Manager, the telepointer application and the conference audio application. The pilot group did not comment heavily upon these applications. It is believed that the reason for this is that they had become familiar with these components, having seen them earlier on several occasions. The main comments about these components were:

C6. Lack of a shared conference manager.

C7. Confusing offset with the telepointer.

C8. The conference audio application provided sufficiently good sound quality.

C9. Lack of a facility for creating static annotations.

Conference Manager

In the first version of the MiniMRD, only the initiator of a conference had access to the Conference Manager. This form for conference policy can be called "fixed-floor" mode (or "dictatorship", for fun).

The pilot group meant that all conference participants should have equal access to the Conference Manager. This form of conference policy can be called "free-floor" (or "anarchy"). When every participant has equal access to the Conference Manager, it is important that the users coordinate their actions using the audio connection.

The pilot group meant that the user-interface for the Conference Manager was intuitive.

Telepointer

Regarding the telepointing facility, there was one main comment. It concerned an offset between the telepointer and the normal cursor. This offset was purposefully created in order to distinguish the telepointer and the cursor. The pilot group found this offset quite confusing, but it has not been corrected yet due to other software coding priorities.

7) This requirement concerns precisely that of grounding referential identities, see section 3.2.2.

8) This touches upon Clark and Wilkes-Gibbs principle of least collaborative effort, see section 3.1.2.3 and [17].

Conference Audio

Concerning the conference audio application, the pilot group meant that the audio quality was sufficient for their purposes. They also meant that they could use the audio channel to coordinate floor-control during the conference.

Missing Functionality

A comment indirectly related to the telepointer was that the MRD didn't support the creation of static pointers ("light-weight" annotations). The users wanted to be able to "drop" static pointers onto images — pointers which could be viewed by all conference participants. This kind of facility would allow the radiologists to point at phenomena within different images at the same time, for example. The pilot group suggested that a more sophisticated version of this facility could allow for other kinds of annotations (e.g., circles and boxes).

10.3.3 The second version of the MiniMRD

The second version of the MiniMRD aimed to alleviate some of the problems the users experienced with regard to the system having too many levels and windows. Figure 12 illustrates how the two communication modalities of the MRD could be started.

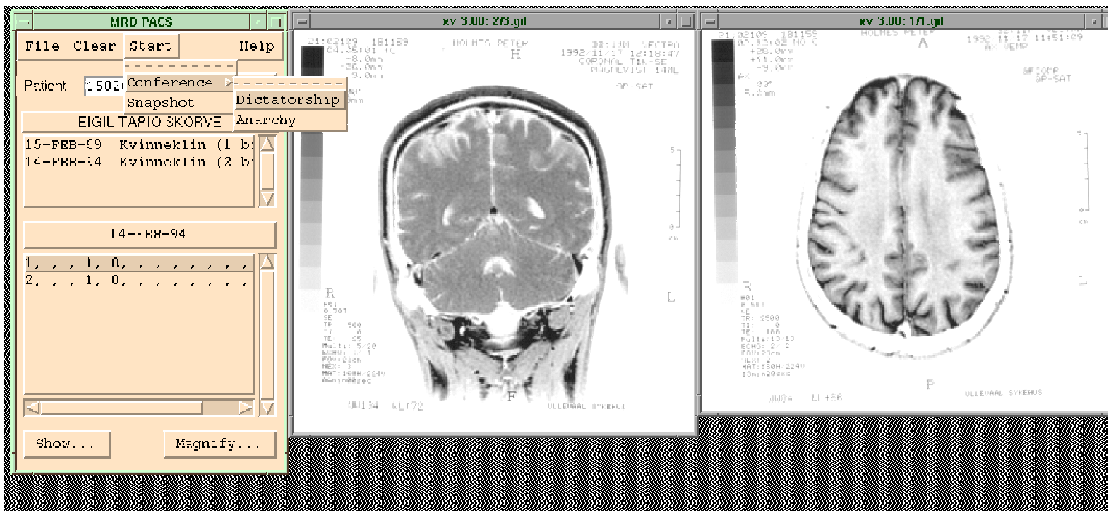

Figure 12 : *The user-interface when starting the second version of the MiniMRD.*

In this version, the MiniMRD had been integrated within the pilot group's "favorite" application[9]. The scenario was that the user (a radiologist) normally investigates and interprets patients' images throughout the day. Occasionally, he has a need to consult with other radiologists. At that time, the user could initiate either a synchronous or asynchronous communication contact; this initiation operation would be accessible directly from within his favorite application. If initiating a synchronous contact, he would have that choice of establishing a fixed- or free-floor conference policy.

With regard to these changes, the response from the pilot group was very positive. They said that this scenario was exactly as it would be in real use.

9) During the evaluation, this application was MRD-PACS.

These comments implied that it was important to establish a close, *logical* connection between the users favorite applications and the communication support system. The problem with integrating the communication support facilities within each user group's favorite applications is that each such application must be slightly modified to include the new menu(s), etc. This kind of approach is expensive and, for some applications, may be impossible.

The final version of the MRD executes as an application-independent process, rather than being integrated within the users' favorite applications.

10.3.4 The final version of the MRD

The comments from the radiologists led us to using "task orientation" as the driving force for the final user-interface design. In a task-oriented setting, Norman [71] suggests that the user and the task-at-hand guide the design of the user-interface. For specific and well-defined tasks, Gritzman et al., [31] offer ideas for design of user-interfaces for computer systems and consumer electronics

The final version of the MRD is presented in chapter 4. This version offers a homogenous and thoroughly integrated user-interface. It also includes the following improvements:

- Task-oriented design and user-interface.
- Intuitive methods to select applications for inclusion within snapshots.
- Annotation possibilities in both synchronous and asynchronous communication modalities.
- A reduced number of levels and steps.
- Seamless transitions across communication modalities.

10.4 Evaluation Results

10.4.1 Application areas for the MRD

This section combines responses received from the pilot group concerning questions Q2 and Q3. In regard to Q3, however, all of the application areas identified involve the Department of Radiology in one capacity or another. The pilot group did not speculate as to the potential usefulness of MRD-like systems for other working groups at Rikshospitalet.

The pilot group identified several different application areas for the MRD during the testing and evaluation periods. Some of the application areas were earlier identified and presented as scenarios during the functional design specification phase [42]. For the sake of completeness, all of the identified areas explicitly mentioned are presented below. The first five mentioned are *intra*-hospital application areas, while the sixth is concerns *inter*-hospital communication:

- Ordinary consultations
- Consultation during surgery

- Morning demonstrations
- Remote supervision of radiographers
- Education
- Inter-hospital communication

Further details about these application areas, as well details concerning the manner in which the MRD contributes to greater effectivity can be found in section 6.1.

10.4.2 Potential Consequences of MRD Use

The discussion above has focused upon application areas for the MRD; for some of those areas, the manner and/or degree to which the MRD would increase effectivity was also discussed. During the final evaluation phase, several other issues were also addressed. These issues concerned expectations about the potential consequences of using the MRD within the Department of Radiology. Some of these issues addressed:

C1. quality

C2. security

C3. cost - benefit: organizational

C4. cost - benefit: economical

Further details about these various issues appear in section 6.2.

10.5 Problems and Issues Arising during the Evaluation

In testing and evaluating the MiniMRD, one great difficulty concerned the fact that it was necessary to reschedule the system's integration at Rikshospitalet. The new schedule demanded integration several months ahead of the original schedule. This situation led to the fact that certain software bugs couldn't be ironed out in time. It also contributed to the fact that the applications and toolkits comprising the first version of the MiniMRD were not as well integrated with one another as had been intended.

Aside from these two contributing factors, other highly significant problems were encountered:

- The Department of Radiology's RiksPACS system was not in regular use.
- The members of the pilot group were not free from their regular, daily duties.
- The pilot group was small.
- Since MRD installation was geographically distributed, it was impossible to observe the communicating parties at the same time.
- New radiographic images were not available in real-time.
- Network limitations at Rikshospital made it impossible to involve more than two machines at a time when using the MRD's desktop conferencing facility.

The department's RiksPACS system is not in regular use at Rikshospitalet. This made it difficult to address which problems at Rikshospitalet could be solved by a PACS system alone, and which communication problems required use of the MRD. As a consequence, a good deal of feedback was received concerning the design of PACS systems.

Another problem encountered was that fact that the radiologists are busy people. The pilot group was highly motivated, but could only test the MRD amidst all of their other daily tasks; whenever their "beepers" sounded, they were forced to leave. This made it especially difficult to evaluate the real-time features of the MRD.

Though the radiologists were very busy, they used a good deal of their time evaluating the MiniMRD. The evaluation of the MRD was limited to two communication scenarios (see chapter 4).

Since the MRD installation was geographically-distributed, it was impossible for a single person to observe both users at the same time. Technically, it was feasible to employ synchronized video films, but video recording at that level of detail was not permitted.

A fifth problem encountered was the lack of direct access to the digital image archive. One reason for this was that writing to the jukebox was prioritized higher than reading from it; in other words, delaying any examination in order to read images for the MiniMRD evaluation was not allowed. A second reason had purely to do with data security. Lack of direct access to the archive implied that, on occasion, the radiologists were required to wait between three to fifteen minutes to view brand new images.

A problem involving the network infrastructure at Rikshospitalet made it impossible to have real-time desktop conferences with more than two participants. This limited evaluation of the MRD's desktop conferencing facility to conferences involving only two persons.

In addition, other organizational, juridical and political issues were encountered during the creation of the MRD. Some of these are mentioned in Løbersli [59].

Chapter 11

Summary and Conclusions

The design, testing and evaluation periods of the MRD were carried out as iterative processes. Throughout the entire course of the project, the pilot group responded enthusiastically to interviews, workshops and prototype demonstrations. They critiqued written reports concerning the project team's understanding of their work-processes and routines, and they participated in the development of scenarios concerning potential application areas for the MRD. These radiologists remained highly motivated and involved, despite the fact that they had solely their own free time to contribute.

The EuroCODE Conceptual Framework [15] describes principles, techniques and methodological tools for designing CSCW applications systems. Within this framework, emphasis is placed upon the fact that the framework itself should:

> *- be under **constant negotiation**, helping designers from different communities of practice in their joint effort to reach a shared understanding of how to use the EuroCODE Shell and for what purposes,*
>
> *- be **interactive**, as it is necessary to support design processes which are by nature creative, communicative and non-linear, and*
>
> *- help designers to **handle change** as a social as well as a technical phenomenon.*
>
> *(Bødker, et. al., [15], p. 28)*

While using the Framework's techniques and tools together with such an actively involved yet busy pilot group, it was necessary at times to modify these techniques, etc., as well as their application. Thus, the Framework's built-in flexibility came to be of use during the development of the MRD.

The Framework's methodological descriptions and examples concerning scenario-based system design were of extremely great value. The scenarios presented in chapter 4 regarding use of the MRD were developed, in an iterative fashion, together with the pilot group. These scenarios remained a focal element throughout the rest of the project.

The scenarios were based upon the Framework's work-oriented checklist, to ensure that the MRD's functional requirements were as complete as possible. The Framework's technical checklist was also used when developing the scenarios, to help ensure that the technical design in fact satisfied the functional requirements known at that time. During technical design, use of detailed object-diagrams had the effect that inter-object protocols were defined and in place early.

As a result, the scenarios underwent little change, *from a functional perspective*, since their original creation (first reported in the MRD design document [42]). In fact, a great deal of the original text is intact; it was primarily the figures and images which changed. From the user's perspective, the greatest part of that which changed concerned the manner in which the user would interact with the system. The kinds of functionality available were essentially the same; it was the *access* to this functionality which was different.

During the evaluation, important feedback was gathered for the MRD's functionality and user-interface design; some of this information is also applicable to other kinds of communication systems. Given the pilot group's tight schedules, it was not possible to test the MRD within real work situations (as intended). Still it is thought that the results from this evaluation are reliable and indicative.

The evaluation confirmed that both the MRD's synchronous and asynchronous communication support facilities were useful. The Department of Radiology's foremost communication needs were well addressed by the MRD's multimedia messaging facility, and the pilot group identified important application areas for the MRD's desktop conferencing system.

The evaluation has also shown that the MRD concept has great potential for making the Department of Radiology more effective; it is not so unlikely that this is true for Rikshospitalet as a whole. The pilot group meant that the MRD could contribute to improve both security and quality within the department. They expressed strong interest in acquiring the system, as soon as it was stable enough to be used within a clinical setting.

Before reaping any potential benefits, however, it is necessary to thoroughly plan and carefully introduce new technology within the department and/or hospital's technical, social and work-process environments. An electronic infrastructure supporting communication involving radiological images must be defined and implemented. Such an infrastructure would likely include a PACS system. Should the MRD be selected as a communication system, the planning and implementation of these two technologies would best be done such that each technology takes into account the functional and technical capacities of the other.

In addition, a new culture regarding new types of communication and some of the possible ways of using it must be established within the department / hospital. Effort must also be made to inform new users that certain ways of using the new communication mechanisms can in fact be detrimental to the organization as a whole (e.g., gossiping via the conferencing system can potentially get in the way of other, more patient-critical discussions).

The MRD's communication support facilities are general in nature; they are in no way bound to use solely within medical work-contexts. It is strongly believed that other user-groups can benefit from the general set of facilities implemented within the MRD. In Braa, et. al. [14], for example, scenarios are described which involve direct use of the MRD at the Great Belt. Other exploitation plans and marketing contexts are described in [25] and [26], respectively.

In regard to the MRD's concrete future, the Sysdeco Group, Norway, has decided to commercialize the system. They see a market for communication systems, both intra- and inter-hospital. More details about future plans for the MRD are found in chapter 8.

Part 4

Technical Appendices

Appendix A

The Conference Toolkit (CTK)

A.1 Abstract

The Conference Toolkit (CTK) includes programs and source code for handling conference administration, as well as some conference applications to demonstrate use of the overall conferencing architecture.

The main program is the “conference”. The conference is a sort of a switchboard, performing actions on applications and participants when instructed to do so by the “conference manager”. A conference manager is a user interface for handling conferences.

Other important programs in CTK are the “registrar” and “confreg”. The registrar keeps track of conferences, participants, and applications. It also provides a service for starting predefined conferences. “Confreg”, a registrar client, is a user interface to the registrar for ordinary users.

The CTK also includes:

- a superclass for building conference applications,
- a conference application that integrates the GWTK with the CTK, and
- some other applications built for demonstrating the conferencing architecture.

A.2 Toolkit Functionality

A.2.1 Basic Concepts

- *Conference object*; is a server process that keeps track of participants and applications in a real-time conference. It includes and removes participants and applications from a real-time conference.
- *Conference manager*; is the user interface to a conference. It passes instructions from the participants to the conference object. A conference with one or more participants will have at least one conference manager.
- *Registrar*; is a server process used by conference objects and managers. It keeps track of information about participants (e.g., user name, host, display etc.) and applications.
- *Conference-aware application*; is an application that knows it is running under a conference. A conference application can be centralized, distributed or pseudo-distributed. A centralized conference application has one process running on one

machine, but a distributed conference application has a process for every participants. A pseudo-distributed conference is neither fully centralized nor distributed.

A.2.2 Functionality implemented in the Toolkit

The Conference Toolkit is implemented in the Beta programing language. It contains different application for administration of real-time conferences and source code for building such programs. Important programs and code are:

- conference (Starting the conference object)
- Conference managers
- Registrar
- confreg
- Superclass for building conference-aware applications
- A conference-aware wrapper for starting external conference-aware applications (i.e. conference-aware application that is not a specialization of CTK superclass for building conference applications). Examples of such application is MBONE application like “vat”, “wb” and “nv”.
- A conference-aware application for starting conference-unaware application under GWTK.

The administration of a conference is done through a centralized conference object. This conference object keeps tracks of participants and applications in a conference. There is one conference object for each running conference. In principle it is possible that a person participates in different conferences at the same time. Each conference object will keep in track which participants and applications included in the conference. The conference object will get instructions from the conference managers.

The conference managers are the users interface to conference objects. In a real-time conference, it is often desired to implement different conference policies depending on the present situation. This can be done in the conference toolkit by implementing different managers. In CTK there is implemented three different conference managers. These are called confman, confmanICTK and shadowConfman. Confman is a simple motif based conference manager. This manager has not been used for a while, but can bee seen as an example of how to implement a Motif based manager. It is not integrated with the registrar. The second conference manager is based on the EuroCODEs User Interface Construction Toolkit. This manager support dictatorship as a conference policy (i.e. only one participant have the power to do operation on the conference object), but an anarchy conference policy can in principle be obtained by running this conference manager under the window replicator in GWTK. The last conference manager, the “shadowConfman” is a conference without a “real” user interface, instead other external application communicate with it using unix ports. This has made it possible to make a Tcl/Tk user interface for the conference.

Registrar registers and stores information about participants, applications and predefined conferences. This information is used by conference objects and managers. Confreg, is also based on UICTK, is a user interface to registrar.

In CTK there is included a superclass to be used for building conference application. This superclass have all the necessary functionality for the communication with the conference object like *quit*, *add participant* and *drop participant*.

A wrapper used to start conference-aware application is also included in the toolkit. This wrapper starts the conference-aware application at each participants workstation. The participants workstation is registered in the registrar.

The last program of importance in CTK, is a conference-aware application for including stand alone applications in a conference. The program, called XyApplRepl, uses the Xy-window replicator included in GWTK, but the source code is written in a object oriented manner that permits developers to change replicator in a easy way. This is done through specialization of the superclass “applReplicator”.

A.3 Technical Description

This section describes only the main programs (or interfaces), such that the overall design of the Toolkit can be understood. Supporting classes and low-level functionality is not described.

The CTK has been tested for the UNIX platform on Sun 4 and Sun 10 workstations under SunOS operating system, release 4.1.3, 5.3 and 5.4. The source code compiles with the Mjolner Beta Compiler release 3.0. (Beta System version 5.0).

All programs in the CTK requires that the ensemble demon from the Beta distribution library is running on every participant’s workstation before they are included in a conference. This can be done running the command:

```
unix> startensemble
```

This command is found in the Mjolner Beta Release 3.x.

A.3.1 Environments variables

In order to use CTK following environments variables have to be declared:

- BETALIB <location on the disc were the Beta compiler is installed>
- BETA_NAMESERVER <A machine>.
  This environment variable is used in connection to the Beta distribution library.
- REGISTRAR_STORE <A directory on the disc>
  Tell the registrar were it shall store its data. This directory must be writable for all persons allowed to run confreg.
- XYHOST <a machine>
  Used to specify which machine the Xy window replicator (See GWTK) shall be started when invoked from the conference architecture.

A.3.2 Conference and the conference manager

The conference object requires that the registrar is running before it can be started. The conference object is started from the command line with the command:

```
conference [-m manager][-a appl][-u user][-s subject][-ns]
```

<manager> is the manager that should control this conference

<appl> is various applications that should be running initially

<user> is a participant that should be included initially

<subject> is a more friendly name on the conference

<-ns name> the conference should register with the beta name server with the specified name.

The conference manager can either be started automatically from the conference object or directly from the command line. Starting it from the conference object is done with specifying the “-m” option. It can be done as following from the command line:

```
confmanICTK -c <Name>
```

Where <name> is the name the conference is registered by in the name server.

When the conference object and manager is started, the users can start including participants and application.

A.3.3 Registrar and confreg

The registrar is started from the command line:

```
UNIX> registrar &
```

If there already is stored information about the participants and applications then can the conference object and manger start asking the registrar about information. Otherwise the user have to register at least persons and applications to be included in the conference. This is done with invoking the following command:

```
Unix> confreg
```

And the choose “New participant” and “New application” from the file menu. New participant should be intuitive to fill in. The rest of this section contains a example of how to include the conference-aware demo application “xConfappCent”, the standard X11 calculator program “xalc” under the window replicator (See GWTK) and the conference audio program “vat” described in the DSTK documentation. The “New application” dialog box has three entries: name, command and parameters. The command must be specified relative to $CTK_HOME.

“xConfappCent”:

Name: xConfappCent (Or an another user friendly name)

Command: demos/xConfappCent

Parameters:

“xcalc”:

Name: Calculator

Command: demos/XyApplRepl

Parameters: xcalc

To register the telepointer application write “xytelepoint” instead of “xcalc”.

“Conference audio”:

Name: ConferenceAudio

Command: demos/confappWrap

Parameters: vat -C %U -t 8 -u <Full path to DSTK-home>/src/tcl/confAudio.tcl 224.2.0.1/4532

A.4 The Application Developer’s Interface

For a detailed description of the use of the toolkit look at the README file included with the source code. The current external interface to the conference object, registrar object, and conference applications are presented below.

A.4.1 Registrar

The registrar object has the following external interface:

‘StartConference’ starts the conference described in the conference descriptor.

```
StartConference: entry
    (# cd: ^ConferenceDescriptor;
    display: ^text;
    ok: @boolean;
    enter (display[], cd[])
    exit ok
    #);
```

‘Hello’ is invoked by a newly created conference object.

```
Hello: entry
    (# regName: ^text;
    manager: ^conferenceManagerIf;
    conf: ^conferenceIf
    enter (regName[],manager[],conf[])
    #);
```

‘ConferenceGoodBye’ is invoked by a conference object when it terminates.

```
ConferenceGoodBye: entry
    (# confName: ^text;
    enter confName[]
    #);
```

‘savePerson’ and ‘deletePerson’ add and remove persons to/from the registrar.

```
savePerson: entry
    (# p: ^participant;
    enter p[]
    #);
deletePerson: entry
    (# uid: ^text;
    enter uid[]
    #);
```

‘GetAllPersons’ returns the set of all persons that the registrar knows.

```
 GetAllPersons: entry
    (# pers: ^PartList;
    exit pers[]
    #);
```

‘saveApplication’ and ‘deleteApplication’ add and remove applications to/from the registrar.

```
 saveApplication: entry
    (# appl: ^ApplicationDescriptor;
    enter appl[]
    #);
deleteApplication: entry
    (#               id: @integer;
    enter id
    #);
```

‘GetAllApplications’ returns the set of all applications that the registrar knows.

```
GetAllApplications: entry
    (#               appl: ^applList;
    exit appl[]
    #);
```

‘GetConferenceList’ returns a list of all predefined conferences that the registrar knows.

```
GetConferenceList: entry
    (# cl: ^ConfList;
    exit cl[]
    #);
```

‘GetPreDefPart’ and ‘GetPreDefAppl’ gets the set of predefined participants/applications from the conference with name ‘confname’.

```
GetPreDefPart: entry
    (# confName: ^text;
    plist: ^PartList;
    enter confName[]
    exit plist[]
    #);
GetPreDefAppl: entry
    (# confName: ^text;
     alist: ^ApplList;
    enter confName[]
    exit alist[]
    #);
```

PredefineConf’ saves the information about a conference so it can be used later.

```
PredefineConf: entry
    (#                cd: ^conferenceDescriptor;
    enter cd[]
    #);
```

‘DeleteConference’ removes the information about a predefined conference from the registrar.

```
deleteConference: entry
    (# confName: ^text;
    enter confName[]
    #);
```

A.4.2 Conference

The conference has the following external interface.

´AddParticipant’ includes a participant to a conference.

```
AddParticipant: entry
    (# part: ^Participant;
    status: @integer;
    enter part[] exit status
    #);
```

‘DropParticipant’ removes a participant from a conference.

```
DropParticipant: entry
    (# partUID: ^text;
     status: @integer;
    enter partUID[] exit status
    #);
```

‘GetParticipantList’ returns the participants in the conference.

```
GetParticipantList: entry
    (# list: ^PartList;
    exit list[]
    #);
```

‘StartApplication’ is invoked by the Conference Manager when one wants to start a new application.

```
StartApplication: entry
    (# ad : ^applicationDescriptor; id: @integer;
    enter ad[]
    exit id
    #);
```

‘DropApplication’ is invoked by a conference manager when one wants to quit an application.

```
DropApplication: entry
    (# applID: @integer; ok: @boolean;
    enter applID exit ok
    #);
```

‘GetApplicationList’ returns the list of applications running in the conference.

```
GetApplicationList: entry
    (# list: ^ApplList;
    exit list[]
    #);
```

‘setConferenceManager’ tells the conference about it’s manager.

```
SetConferenceManager: entry
    (# manager: ^ConferenceManagerIf; ok: @boolean;
    enter manager[]
    exit ok
    #);
```

‘GetConferenceManger’ returns a reference to the current conference manager.

```
GetConferenceManager: entry
    (# manager: ^ConferenceManagerIf;
    exit manager[]
    #);
```

‘MayIJoin’ handles the conference policy.

```
MayIJoin: entry
    (# p : ^participant;
    value: @boolean;
    enter p[]
    exit value
    #);
```

‘Hello’ is invoked by a newly started Conference Application to tell the conference about the conference application.

```
Hello: entry
    (# no: @integer;
    appl : ^ConferenceApplicationIf;
    name : ^text;
    enter (no, appl[], name[])
    #);
```

‘Die’ terminates the conference.

```
 Die: entry (# #);
```

‘GetConferenceName’ returns the name of the conference.

```
GetConferenceName: entry
    (#                name : ^text;
    exit name[]
    #);
```

‘GetMulticastParameters’ returns a multicast address.

```
GetMulticastParameters: entry
    (# IPAddress : ^text;
    tl : @integer;
    exit (IPAddress[], ttl)
    #);
```

A.4.3 Conference application

‘AddParticipant’ adds a new participant to the conference application. The conference application will then set up the necessary user interface etc. for the new participant.

```
AddParticipant: entry
    (# part: ^Participant;
    status: @integer;
    enter part[] exit status
    #);
```

‘DropParticipant´ tells the conference application to remove a participant.

```
DropParticipant: entry
    (# partUID: ^text;
    status: @integer;
    enter partUID[] exit status
    #);
```

‘GetConference’ returns a reference to the conference the application is running under.

```
GetConference: entry
    (# conf : ^ConferenceIf;
    exit conf[]
    #);
```

‘Die’ terminates the conference application.

```
Die: entry (# #);
```

A.5 Rationale for Design Changes

The CTK is implemented after the concepts presented in D-2.5, but the names of some routines have been changed. We have not made the conference managers conference-aware using the conference application superclass yet, but the shadowConfman can be used to make the conference manager that from the users perspective are conference-aware.

A.6 Possible Further Improvements to the Toolkit

We have not been abel to specialize our superclass for building distributed conference-aware applications yet. When this problem is solved, it will be a improvement for the toolkit.

An another improvement would have been to integrate the registrar with the Enterprise Information Toolkit (EISTK). This will make it easier to maintain information about participants.

Appendix B

The Global Window Toolkit (GWTK)

B.1 Abstract

The Global Window Toolkit provides functionality for replicating windows, and communication facilities such as telepointing and annotations. Window replication is the means for sharing application windows among users in real-time, and therefore serves as a basic component when designing conferences (see the CTK description for more details).

Conference participants might want to show something to others by talking and pointing in shared windows. The audio part is covered by the DSTK (see DSTK descriptions for more details). The GWTK also covers pointing in shared windows. This is done by telepointing; that is, by making a participant's cursor visible on the other participants' screens.

Annotations are similar to telepointers since they are also visible for all participants in the conference. Annotations differ, though, by having static positions.

B.2 Toolkit Functionality

B.2.1 Basic Concepts

- *Window replication:* is defined in a *technical perspective;* it refers to the ability for the same window to be presented on different computer screens at the same time. Moreover, replicated windows shall make output visible, and allow input to be given for all users according to defined rules, often termed as floor control.
- *Application sharing*: is defined in a *user perspective*; it is used to denote the situation during which an application's output is replicated and thus shared among users synchronously.
- *Telepointing*: is the ability to show other users where you point with the cursor in a shared application. A telepointer should be labeled by a user name (or some other unique characteristic) in order that it's owner be identifiable.
- *Annotation:* is a labeled and numbered static mark which, in a conference, refers to a single point in a shared window. In an asynchronous communication mode, annotations can be used to mark any part of the screen; in most cases, however, such annotations are used to point within some application's window.

B.2.2 Functionality implemented in the Toolkit

Window replication is done through Xy [13]. Xy acts as an X server which communicates with Xy-agents. That is, an Xy-agent feeds the Xy server with input, and it expects output to transfer along to clients (applications) connected to it. Each user must have an Xy-agent running locally on his/her machine. In principle, every X application can be shared through the Xy window replication system. The reasons for choosing Xy instead of another window replicator were its ability (1) to handle different kinds of floor control[1] and (2) to share arbitrary X applications.

The functionality of telepointers is complicated. When sharing an application, the users involved may move the application's window(s) to different areas upon their respective screens. For each user, the telepointer must therefore detect where the shared window(s) are in order to point at the correct place on the user's screen. In the MRD, telepointers are only visible when pointing within shared windows.

Annotations may be placed while using the telepointer. Like the telepointer, annotations will be shown on each remote screen. In the MRD, each annotation has a label consisting of the name of the user who placed the annotation (optionally, a brief description) and a number. The annotations' positions on the screen may be saved, and can therefore be included in a multimedia message (see the SSCTK for more details).

B.3 Technical Description

Xy[2] provides telecooperation and mobility for the X Window System. Every existing X application can be displayed on, as well as receive input from, multiple displays at the same time, requiring a modification of neither the Xlib nor the X server nor the X application itself. The term "multiple" includes also the possibility of having zero displays connected to the X application, which leads to the mobility aspect. Mobility means that you can move your X environment (i.e. all X applications) from one X server to another without the need of terminating all running X applications only for restarting them again on a different display.Using Xy you can log out, log in again and reconnect to your X environment that is still running.

In order to understand Xy it is helpful to consider its design. Xy consists of an XyServer (see also Xyserver(1)) and zero or more XyAgents (see also xya(1)). An XyServer is a permanent X server running in memory without access to peripheral devices. For X applications there is no difference between XyServers and "real" X servers. XyAgents are responsible for delivering input to the XyServer. With no XyAgent connected, an XyServer behaves like an X server without user input, except that it does not display its output on a screen. XyAgents provide the interface between XyServers and local X servers. Every XyAgent is running on a single local X server which is called the Target X server. All relevant events happening on the Target X server are reported to the XyServer which handles these events like the X server does with peripheral device events.

1) In certain conference policies, it is critical that there is no restriction upon providing input to (and receiving output from) the shared applications.
2) The technical description of Xy is to a great extent taken from [13].

The telepointer is an independent application specialised for use with Xy server. The telepointer application needs to be connected both to your local display and to the Xy server. The display that the Xy server accepts connection needs to be specified. The telepointer application can make use of Xy specific features without using the xyc program, if the agent for the local display is specified (-agentname option). Usually one would like to specify a name for the telepointer (normally this is done by setting the -pointername option and passing it to the telepointer application).

Annotations are equivalent to a label and a number. By default the label will be “Mark” but a parameter can be set to an arbitrary name for the label. A tag indicates whether the annotation is locally or globally visible.

Annotations can be used in both synchronous and asynchronous mode. When the telepointer is active in a conference, it is possible to switch to annotation mode. Then a click on the left mouse button will create an X-event which will be sent to the annotation application. The original X-event will still be sent to the application pointed to by the cursor. In this way, the focus point is not changed and the mouse will not be grabbed when annotating in conference mode.

Annotations put in a conference will be seen by all the participants in the same way as they see ordinary telepointers. Annotations are slightly different in asynchronous than in synchronous mode. Applications built for asynchronous annotations will have to grab the mouse. Thus it is not possible to work with applications as long as annotation mode is on. For each annotation x and y positions are stored in a file so that they can be included in a snapshot.

It is possible to make application which gives the user the opportunity to switch between asynchronous and synchronous mode. If this is done while having a conference only the synchronous annotation will be visible to the other users. In such cases an extra ‘:’ character is put on the label for the asynchronous annotation to distinguish it from the synchronous annotations.

B.4 The Application Developer’s Interface

B.4.1 The window replicator - *xy*

This API consists of three command call to the operating system. (their intended use is describes below).

- xy [display] (start an Xy server)
- xyc [-register display user] [-add user] (register a participant and host)
- xya [-port portnumber] (start an agent and connect it to an Xy server)

The Xy server is the basic component, as it does the windows replication. Hence it must be started first. The Xy server acts as another X server to our programs but it uses another display number than usual. This can be selected at will, e.g. as display number :3. X display numbers actually denote socket numbers that clients can connect to in order to display on

that display. Adding the value 6000 to the display number yields the real socket number used for communication between the server and the client.

You might also want to configure Xy, by setting the screen size, screen type, floor control etc. This is done by using a configuration file and by pointing to it with the environment variable XY_CONFIG_FILE.

(If the default configuration file resides in ~code/lib/xy as xy.config, this could be specified as:

```
vega:~> setenv XY_CONFIG_FILE ~code/lib/xy/xy.config)
```

To start the Xy server with display number of 3, the following should be issued

```
vega:~> xy :3 &
```

Starting agents

The agents are required to get output from the Xy-server. They connect to the local display on the participants machine, and expect the Xy server to connect them before they do anything at all. There are several options that can be specified when starting a new agent, which is because different machines might not have the exact same hardware configuration. This is especially a problem with colours, but can also be a problem, even if all participants are using the same colour screens. Local (nonshared) programs could occupy part of, or the entire default colourmap, forcing the agent to allocate a private colourmap. this is usually leads to colour flashing, when the input focus changes from shared applications to locally running applications.

You might choose to leave these details to the agent, or you can choose to specify that a private colourmap should not be allocated. This could be done by specifying the option -donotSwitchColormap or -staticCells when running the agent.

Another thing that has to be decided on when running the agent, is what port number it should expect the connection from the Xy server. This is done by specifying the raw port number 6006 which really means the display number 6 as explained above.

To run the agent on the local machine, with a display number of 6 and to show the output on the default local display, (the one specified in the DISPLAY environment variable), you specify:

```
wangelsol:~> xya -port 6006 &
```

Adding participants

When the agent is started it is time to connect it to the Xy-server. This is done in two steps with the xyc program. This program discovers where Xy is running, and on which display number it accepts clients to connect by reading the XYSERVER environment variable. This variable could be set as shown below:

```
vega:~> setenv XYSERVER :3
```

Just the same way as normal X servers are specified.

The next step tells the Xy server who you are and where your agent is running.

```
vega:~> xyc -register wangelsol:6 eigil
```

This statement says that the agent runs on the host wangelsol with a display number 6, that is, port number 6 + 6000, and the participants name is eigil.

To be sure that there always will be a connection between the agent and the Xy server a program may be started. The simple xlogo program will be used in this example.

```
vega:~> xlogo -display :3 &
```

Now, only the connection from the Xy server to the agent is remaining. This is done by issuing

```
vega:~> xyc -add eigil
```

This is enough since the name eigil already is registered with the agentname wangelsol:6.

To share an application within Xy, just run the application on the display of the xy server.

B.4.2 Telepointing and annotations in conferences

The API is given for Tcl/Tk applications, but functions marked with an asterisk is defined in C/X. They are given below with their C-interface.

B.4.2.1 Telepointer functions

- `(*)setup(char* local_server, char* remote_server, char* agent_name, char* pointer_name, char* pointer_font, int update_rate);`
- `(*)update() TimerUpdate(ClientData c);`
- `TStartUpdate(ClientData c, Tcl_Interp* i, int argc, char** arg);`
- `TStopUpdate()`

B.4.2.2 Telepointer and conference annotation function

(The parameter w indicates which pointer to be operated on. By convention 0 is the telepointer, other values refer to annotations).

- `(*)map(int w)`
- `(*)unmap(int w)`
- `(*)share(int w)`
- `(*)unshare(int w)`
- `(*)clear(int w)`

B.4.2.3 Conference annotation functions

- `(*)setup2(char* local_server, char* remote_server, char* agent_name, char* pointer_name, char* pointer_font, int update_rate, int xpos, int ypos);`
- `TStartAnnotation(Clientdata c, Tcl_Interp* i, int argc, char** arg);`

- `TStopAnnotation(Clientdata c, Tcl_Interp* i, int argc, char** arg);`
- `TSaveAnnotation(Clientdata c, Tcl_Interp* i, int argc, char** arg);`
- `MakeAnnotation(ClientData c, XEvent* e)` (convenience function)
- `(*)Do_InitTk(Clientdata c, Tcl_Interp* i, int argc, char** arg);`
- `Nearest(int xpos, int ypos);` (not used)

B.4.3 Annotation in asynchronous mode

(The parameter w indicates which pointer to be operated on. By convention 0 is the telepointer, other values refer to annotations).

- `(*)setup3(char* local_server, char* remote_server, char* agent_name, char* pointer_name, char* pointer_font, int update_rate, int xpos, int ypos);`
- `(*)map3(int w)`
- `(*)share3(int w)`
- `(*)clear3*(int w)`

B.5 Rationale for Design Changes

In response to user feedback an application developer's interface for supporting synchronous and asynchronous annotations was made. In the first place, synchronous annotations which are globally visible were provided so that discussion about images was made easier during conferences. Second, with the development of SSCTK, asynchronous annotations were added to the API. It is very simple in the sense that it does not take care of scrolling, resizing of windows etc. Some applications provide their own ways of (fancy) annotations.The annotation facilities offered by this toolkit is not meant to replace these, but rather to ensure that annotations always will be present either in asynchronous or synchronous mode, since it is strongly believed that this is of great importance in the context of mrd.

The library for supporting synchronous and asynchronous is extensions of the telepointer library.

B.6 Possible further improvements to the toolkit

Telepointing and annotations slow down the systems considerably. This is due to that each update of the telepointer requires loading and masking of bitmaps and quite a lot computation as well. In practice, conference participants are encouraged to minimise the use of

telepointing. A further elaboration of the telepointer could improve the efficiency of telepointing and annotation and thereby the usability of the toolkit.

The functionality of the telepointing and annotation facility is limited. The primary goal is to enable a simple telepointing and annotation facility for all X applications. An extension of functionality could be to use a foil metaphor for more complex annotation. That is, on every shared application a “foil-window” is put, which give the users the opportunity to draw, annotate. use a marker pen etc. Moreover, a future improvement of the toolkit would be to use the DHTK to make annotations “inside” applications.

As replication functionality is built more and more into the operating systems, this toolkit should provide a front-end directly to the operating systems. Even though a faster and more reliable window replicator would make the toolkit better, it should be kept open to take continuously advantage of the best technology for window replication.

Appendix C

The Digitized Sound Toolkit (DSTK)

C.1 Abstract

The Digitized Sound Toolkit (DSTK) includes two primary applications. One application is a conference audio application; it is based upon the public-domain software called 'vat'[1], which exploits IP multicasting.

The conference audio application provides real-time facilities for multi-party audio conversations over a data network. Within the application, facilities also exist for recording parts of such conversations, as well as editing and/or playing back such recorded information within the context of the same (or any other) audio conference.

The second primary application included in the DSTK is an audio editor; it is, in fact, a copy of the 'audiotool' binary normally included in the standard Sun OS (and Solaris) software.

Audiotool allows for stand-alone recording, editing and playback of audio data. The conference audio application and audiotool have been integrated with one another, as implied above. Audiotool is also integrated with the Snapshot Composer Toolkit (SSCTK).

The conference audio application included in this release is integrated with the EuroCODE Conference Toolkit (CTK). All applications included in this release can be run independently of the CTK. Audio files are easily integrated with the Distributed Hypermedia Toolkit (DHMTK) through use of the DHMTK's file component mechanism. Using this mechanism, however, it is only possible to create hypermedia links to entire audio files, not to audio segments within some audio file.

The implementation approach found in the DSTK is quite different than the design found in EuroCODE Deliverable D-2.5. However, much of the toolkit's original functional design has been satisfied.

C.2 Toolkit Functionality

C.2.1 Basic Concepts

This section introduces and briefly explains special concepts and terminology relevant to the Toolkit.

- *multi-party conference audio*: Multi-party audio conversations over a data network.

1)

- *multicasting*: This term concerns transmission and reception of information. From a logical perspective, it concerns communication of information from a single sender to some multitude of receivers (and vice-versa). With regard to low-level data messages, there exist two fundamentally different technical implementations:
  - a round-robin technique which sends the message from the sender to the first receiver, then from the sender to the second receiver, and so on.
  - so-called "true multicasting", wherein all parties both send to and read from a single (multicast) address; in this way, a sender only needs to send out a single instance of the message in order for it to reach all receivers.
- *audio file*: A collection of audio samples, sometimes prefixed by a header which provides information about file's audio format.
- *audio format*: A collection of parameters such as sample rate, number of channels, precision, encoding (e.g., A-law, linear), etc., which describe how some given audio data can be interpreted. For more information, see .../dstk/doc/formatInfo/fileFormats*.

C.2.2 Functionality implemented in the Toolkit

The DSTK provides an IP[2] multicast-based application ('vat') for real-time conference audio. In addition, the audio conferencing application offers a basic facility for recording the audio contents of an ongoing conference. Once recorded, such information can be edited and/or played back within the same (or any other) conference.

This version of the DSTK also includes an audio editor called 'audiotool'. Due to it's intuitive user interface, use of the audio editor is essentially self-explanatory; more details can be found in the documentation located in ...dstk/man/man1/audiotool.1 or in the corresponding (compressed) postscript file, see ...dstk/doc/postScript/audiotool.ps.Z.

Other functionality within the DSTK includes:

- an application for adjusting audio levels ('gaintool')
- basic recording and playback functions which can be invoked from the command-line ('codeRecord' and 'codePlay')
- a simple, audio recorder/player application ('sounded')
- an application for audio format conversions ('sox')

C.3 Technical Description

Much of the software in the DSTK has been obtained from the public domain. Effort has been made to integrate those different pieces of software where applicable. In some cases, it was necessary to make less-desirable compromises due to limitations (or "features") within imported binaries. These compromises will be mentioned in the text below.

Furthermore, the DSTK release includes "man" pages for all of the applications which have been obtained via public domain, see ...dstk/man/man*/*.

2) The IP multicasting implementation is an instance of "true multicasting".

C.3.1 Environment variables

The environment variables used within the DSTK are: DSTK_HOME, AUDIODIR, AUDIOSCRIBE, VAT_TTL, MXRECTIME (optional), DSTK_GEOM, ATOOL_GEOM, GTOOL_GEOM and ATOOLPATH. When the MRD is installed, it is necessary for the system installer to ensure that these variables are set[3] appropriately.

1. DSTK_HOME:
   The value for this variable is dependent upon $CODE_HOME, the value of the top-level installation point for the EuroCODE system. Typically, the value for DSTK_HOME is:
   ~ecode/shell/toolkits/dstk.
2. AUDIODIR:
   The variable specifies a directory to be used for locating files for both recording and playback operations. Typically, the value is:
   /local/home/$USER/audio/audioFiles.
3. AUDIOSCRIBE: (...here is one of the compromises...)
   The vat binary is compiled in such a way that the (public-domain) software for recording from (and playing back within) audio conferences must be run on a machine other than those upon which the audio conferencing itself is taking place. For example, in a case where two persons are talking to each other via vat, a third machine is required in order to record (or playback) the audio content of the conference[4]. If vat were compiled such that a third machine was not necessary, the process load (on the recording machine) would be more than doubled. In any case the source code for vat is not available, which certainly limits the possible alternatives here.
   Given this clarification, the value of AUDIOSCRIBE must be the name of a machine upon which recording and playback of audio conferences will be performed.
4. VAT_TTL:
   As a multicasting application, audio data created by vat will travel along the transmission medium and, once reaching a router/bridge, spread itself onto all lines leading out from the router/bridge (assuming that the router/bridge is programmed to correctly handle such multicast traffic). When started from the command-line, vat allows the possibility to control how may routers/bridges it crosses by setting a value for the “-t” flag.
   When starting conferences within the MRD, the conference audio application is started automatically. At that moment, the value of VAT_TTL is read in order to delimit the spreading of multicast data from vat. Spreading along a particular line stops when the data has crossed VAT_TTL number of routers/bridges; the value for VAT_TTL must be an integer.
5. MXRECTIME:
   This variable specifies, in minutes, the maximum length of time a user is allowed to record conference audio, for each execution of that operation. There is a default value of 12 minutes per recording. The default value can be overridden by (re-)setting the environment variable MXRECTIME. Setting the value for MXRECTIME is optional, though when set, it must be an integer.

3) The environment variables for the MRD are found in ~ecode/demonstrators/mrd/config/codeenv.
4) Note that the all machines must have multicast kernels and be equipped with /dev/audio resources.

6. DSTK_GEOM:
   This value specifies the (x, y) coordinates at which vat will be located when started via the MRD.
7. ATOOL_GEOM:
   This value specifies the (x, y) coordinates at which audiotool will be located when started via the MRD.
8. GTOOL_GEOM:
   This value specifies the (x, y) coordinates at which gaintool will be located when started via the MRD.
9. ATOOLPATH:
   This value specifies where the Snapshot Composer[5] application will look for audio files, whenever such files are included in multimedia messages via that application. The value for this variable is a string which specifies one or more directory paths, where the paths are separated by one or more white spaces. Normally, one of the paths in this string is $AUDIODIR/au (see following section).
   It is possible for individual users to append new paths to ATOOLPATH using the MRD's 'concatStrs' function; in the ~/.cshrc file, a user might have for example:
   source ~ecode/demonstrators/mrd/config/codeenv
   setenv ATOOLPATH `concatStrs $ATOOLPATH "/local/home/$USER/*"`
   This expression will effectively add all directories *immediately under* /local/home/$USER/ to the set of paths in which the Snapshot Composer searches for audio files.

Note that although the DSTK's environment variables are set for all users when sourcing the MRD's set of environment variables, they can be modified locally by the users following that source statement (as shown above). Obvious candidates for modification/extension are AUDIODIR, MXRECTIME and ATOOLPATH.

C.3.2 Directories and formats for recorded audio data

The DSTK offers three applications for recording data: `audiotool`, `sounded` and `recplay`. Recplay is the application for recording and playback of audio within conferences. Recplay (and vat) use an audio format based upon multicast data packets; herein, this format will be called "vat format". The other two applications, audiotool and sounded, use a simple format in which the audio samples are single channel, 8000 bits/sec. (sampling rate) and U-law (micro-law) encoded; herein, this format will be called "au format". Due to its simplicity, au format can be handled by most UNIX workstations.

Vat format and au format differ dramatically. Through use of the DSTK, the vat-formatted files are automatically kept in a different directory than the au-formatted files. The vat-formatted files are stored under $AUDIODIR/vat/, while the au-formatted files are stored under $AUDIODIR/au/.

It is worth remarking here that when creating audio files "manually" (e.g., via audiotool or sounded), it can be very useful to have these files located under $AUDIODIR/au/.

5) See documentation for the Snapshot Composer Toolkit (SSCTK).

C.3.3 Functions and applications

This section provides a short description of the higher-level functions and applications included in the DSTK.

C.3.3.1 vat

Vat[6] is an IP multicast-based application for audio conferencing over data networks. The DSTK's conference audio application is based upon vat. When initiating or joining desktop conferences via the MRD, the DSTK's conference audio application is started automatically. Therewith, the placement of vat on the screen is based upon the value of $DSTK_GEOM.

To start vat manually, one can simply type "vat", or use a command such as:

```
vat -C <conf_name> -t $VAT_TTL -u $DSTK_HOME/src/tcl \
confAudio.tcl 224.2.0.1/<port_number>
```

The confAudio.tcl file named above contains the Tcl/Tk code used to create the user-interface for the DSTK's conference audio application. Vat's original Tcl/Tk interface code is found in ...dstk/src/tcl/vat_v3.3.tcl; this is the interface displayed when executing simply "vat".

The confAudio.tcl file has been modified and extended from the original vat interface. It presently includes the interface code for "recplay" (described below), as well as hooks for integrating the Distributed Hypermedia Toolkit. Furthermore, the confAudio.tcl interface includes two buttons: "Conf channel" and "Editor channel". These buttons serve to direct the audio I/O to either all participants within the audio conference, or solely to the local audio devices, respectively.

For more technical details concerning vat, see ...dstk/man/man1/vat.1 or the corresponding (compressed) postscript file, ...dstk/doc/postScript/vat.ps.Z.

C.3.3.2 recplay

Recplay is the application for recording and playback of audio within conferences. It can only be started by the "Conf Rec, Play..." button within the DSTK's conference audio application; therefore, recplay is not an application in it's own right. Files recorded by this application are stored under $AUDIODIR/vat/.

For more technical details concerning the functions[7] underlying this application, see ...dstk/man/man1/vat_play.1 and ...dstk/man/man1/vat_record.1.

C.3.3.3 audiotool

Audiotool has been discussed quite a bit already. It is copy of the program normally provided with the standard SUN software. It is for this reason that the DSTK, when considered as a totality, can only be run upon the SUN architecture.

---

6) Copyright (c) 1993-1994 Regents of the University of California. All rights reserved.
7) Copyright (c) 1992, Anders Klemets (klemets@sics.se).

To start audiotool, type:

`audiotool` , or
`audiotool <file_name>`

Audiotool will accept arguments for the display and geometry flags, if provided. For more information, see ...dstk/man/man1/audiotool.1 or the corresponding (compressed) postscript file, ...dstk/doc/postScript/audiotool.ps.Z.

Keep in mind that it can be very useful to save files created by audiotool under $AUDIODIR/au/.

C.3.3.4 mrdAudiotool

MrdAudiotool is a perl script which ensures that audiotool is started upon the same machine indicated by the $DISPLAY variable. It also ensures that audiotool is started under $AUDIODIR/au/. MrdAudiotool can be started using:

`mrdAudiotool`
`mrdAudiotool <audioFile> -geometry <geometry>`
`mrdAudiotool -geometry <geometry> <audioFile>` , or
`mrdAudiotool -geometry <geometry>`

If <audioFile> is not provided, audiotool opens with a uniquely defined filename having no contents. If <geometry> is not provided, audiotool is placed on the screen using the value of $ATOOL_GEOM.

C.3.3.5 gaintool

Gaintool is a small application which can be used to control the audio input and output levels, as well as the device to which output is directed (i.e., speaker or headphones).

The gaintool binary provided with the DSTK is a copy of the program normally provided with the standard SUN software. Like audiotool, this application can only be run upon the SUN architecture.

To start gaintool, type:

`gaintool`

Gaintool will accept arguments for the display and geometry flags, if provided. For more information, see ...dstk/man/man6/gaintool.6.

C.3.3.6 startGaintool

StartGaintool is a perl script. When invoked upon some machine (*m*), startGaintool ensures that at most one instance of gaintool is started upon machine *m*. In other words, if gaintool is already running on *m*, startGaintool exits; otherwise, gaintool is started on *m*.

StartGaintool can be started using:

`startGaintool`

If $GTOOL_GEOM is set, startGaintool uses that value for gaintool's screen placement.

C.3.3.7 codeRecord

CodeRecord is a command-line function by which to start audio recording into a file. The format of the recorded files is “raw”, a format which differs from au format solely in that there is no header information found in raw audio files.

A typical invocation of codeRecord looks like:

```
codeRecord -t <record_time> > <file_name>
```

Here, <record_time> is specified in seconds.

CodeRecord can be run upon both Sun4 and Solaris operating systems.

Keep in mind that it can be very useful to save files created by codeRecord under $AUDIODIR/au/.

C.3.3.8 codePlay

CodePlay is a command-line function by which to start playing an audio file. CodePlay will play both raw- and au-formatted audio files.

A typical invocation of codePlay looks like:

```
codePlay <file_name>
```

CodePlay can be run upon both Sun4 and Solaris operating systems.

C.3.3.9 sounded

Sounded is a tiny Tcl/Tk application for recording and playback of audio files. It is based upon the codeRecord and codePlay functions described above. Sounded can be invoked using:

```
sounded <file_name>
sounded <file_name> _NEW
```

When starting up, sounded will try to play <file_name> by default. This behavior can be prevented by adding the “_NEW” flag after <file_name>. When <file_name> does not exist, an error message will be generated when trying to play that file. When this happens, he sounded application does not crash; furthermore, any recording made will be saved in <file_name>.

Sounded can be run upon both Sun4 and Solaris operating systems.

Keep in mind that it can be very useful to save files created by sounded under $AUDIODIR/au/.

C.3.3.10 startSounded

StartSounded is a perl script which ensures that sounded is started upon the same machine indicated by the $DISPLAY variable. StartSounded can be started using:

```
startSounded <file_name>
startSounded <file_name> _NEW
```

For further details, see sounded above.

C.3.3.11 sox

Sox[8] is an application for performing conversions between different audio file formats. The binary found in this release can be executed both Sun4 and Solaris operating systems. In the DSTK release, makefiles are included by which to compile sox onto a number of other platform architectures.

For further technical information about sox, see .../dstk/src/sox10/README , ...dstk/man/man1/sox.1 and .../dstk/doc/txt/sox.txt. For further information about audio formats, see .../dstk/doc/formatInfo/fileFormats*.

C.3.3.12 vat2raw

Vat2raw is a program for transforming a vat-formatted file into a raw audio file. Thereafter, sox can be used to transform the raw file into an au-formatted file. This allows for conference audio data recorded by recplay to be lifted into audiotool for editing.

For further technical information, see .../dstk/src/vat2raw/vat2raw.README.

C.3.3.13 raw2vat

Sox can be used to transform an au-formatted file into a raw file. Raw2vat is a program for transforming a a raw audio file into a vat-formatted file. This pair of sequenced operations allows for audio data created by audiotool or sounded to be played within an audio conference.

For further technical information, see .../dstk/src/vat2raw/raw2vat.README.

C.4 The Application Developer's Interface

The DSTK consists primarily of public-domain software. Therefore, the application developer's interface (API) available from the DSTK is inherited, for the most part, from that software. When constructing the DSTK therefore, the preparation of an application developer's interface was not heavily prioritized.

Despite the lack of a formal API, the user-interfaces for the DSTK's most relevant applications (except for audiotool) are written using the Tcl/Tk programming language. These applications include `vat`, `recplay` and `sounded`. With some experience, Tcl/Tk applications are easy to modify in order to change their look and/or functionality.

The interfaces to the DSTK's other primary functions and applications have been described and/or referenced further in the section above.

8) Creator & Maintainer: Lance Norskog (thinman@netcom.com).

C.5 Rationale for Design Changes

According to the design work laid out in EuroCODE deliverable D-2.5, the DSTK was to have been based directly upon the stream-based architecture of the Temporal Data Toolkit (TDTK). For this implementation of the DSTK, this is not the case. This section attempts to provide the essential rationale for why the DSTK's original technical design was so radically modified, as well as a brief summary of the implications of the technical design chosen.

C.5.1 Problem areas

C.5.1.1 DAIS and X

Conference Audio, the fundamental application within the DSTK, was originally designed to be built as a specialization of the TDTK's TDStreamAppl class. Here, the TDTK planned use of DAIS-based stream communication in order that chunks of temporal information could be transmitted in a distributed environment. As a specialization from the TDTK, Conference Audio would inherit the use of DAIS.

As an application, Conference Audio was designed to function as a client-server. It would offer an X-based interface with which the user could control sound levels, muting/unmuting of selected audio sources, start/stop playing of a source, etc. However, late into the schedule for implementation of the DSTK, the integration of X and DAIS-based server applications was still not adequately solved. This situation was one of several factors which had an influence upon the decision to change the technical design of the DSTK.

C.5.1.2 DAIS streams and multicasting

In the EuroCODE Technical Annex, it is specified that the DSTK would implement "multicast sound communication facilities." For its pilot group at Rikshospitalet, the MRD also considered exploiting multicasting for the transmission of radiological images.

Early, yet successful experiments in transmitting temporal chunks used DAIS-based RPC communication. Again, late into the schedule for implementation of the DSTK, DAIS-based stream communication was still under implementation. Without a firm basis upon which to build and test multicast, stream-based communication, DSTK development was hindered.

C.5.1.3 DAIS and BETA

From the start, Conference Audio was designed to be tightly integrated with the Conference object class from the Conference Toolkit (CTK). The CTK is written in the BETA language.

Given the TDTK's use of DAIS, implementation of Conference Audio using the TDTK as its basis would have required the creation of an application which successfully integrated BETA and DAIS. Yet again, late into the schedule for implementation of the DSTK, there did not exist a stable integration of BETA and DAIS.

Given this collection of difficult technical problems which remained unsatisfactorily solved prior to the delivery of the DSTK, it was tenable to think that the DSTK could not be implemented according to schedule. Therefore, the current implementation exploiting public-domain software was selected.

C.5.2 Implications of the implementation

The DSTK, in its current implementation, fulfills much of the original functional design. What is lost in this implementation, however, is a more integrated EuroCODE Shell approach. From both a functional and technical standpoint, this implementation of the DSTK cannot support the synchronization of audio and video; this condition is a direct result of the fact that this implementation of the DSTK is not based upon the TDTK.

C.5.3 Status of the implementation

This implementation of the DSTK fulfills the Technical Annex's requirement that the sound communication facilities implement multicast. Each of DSTK/CTK and DSTK/SSCTK integration are completed; furthermore, hooks for DSTK/DHMTK integration are in place (see below).

C.6 Possible Further Improvements to the Toolkit

This section describes functionality which could be included in the Toolkit, in order to improve it.

- *Direct integration of the DSTK with the DHMTK.*
  Hooks currently exist for integrating the DSTK with the DHMTK. These hooks are established both at the command-line level and within the user-interface (confAudio.tcl). Some minor modifications are required in the confAudio.tcl file, and the following scripts need to be filled in: hmAddEndp hmCreaComp, hmFollowL and hmNewLink.

Appendix D

The Snapshot Composer Toolkit (SSCTK)

D.1 Abstract

The Snapshot Composer Toolkit contains functional elements designed to support the composition, reconstruction and presentation of multimedia messages. The SSCTK does not involve itself with the transmission of multimedia messages — that job is performed by standard e-mail services.

The SSCTK consists of the public-domain packages called `MH` [75] and `metamail` (see ...ssctk/man/man1/metamail.1), as well as new software written to exploit the features within those packages. One powerful feature within the SSCTK is that when creating multimedia messages, the SSCTK creates a logical representation of the message. Using that representation, it becomes possible to create alternative kinds of transmission-ready instances of the message (e.g., MIME, HTML, etc.).

Using the functional elements within the SSCTK, an application called the Snapshot Composer has been developed. This application offers access to all of the Toolkit's high-level functionality, as well as a tailorable user-interface.

The Snapshot Composer is fully integrated within the MRD; moreover, it is on equal footing with the MRD's Desktop Conferencing system. In other words, it is these two subsystems which comprise the MRD. The Snapshot Composer and the Desktop Conferencing system are also integrated with one another in such a way that the user can seamlessly shift from one communication context to another.

D.2 Toolkit Functionality

D.2.1 Basic Concepts

This section introduces and briefly explains special concepts and terminology relevant to the Snapshot Composer Toolkit.

- *Snapshot Composer (SSC)*: The SSC is the primary application within the SSCTK; it comprises essentially all of the Toolkit's fundamental functionality. This functionality is offered through the SSC's top-level windows, "send" and "receive".
- *Multimedia messaging:* Within the context of the SSCTK, this is the ability to compose, send, receive and reconstruct messages containing more than one media type.
- *Message composition:* From the user's point-of-view, this is the creation of a multimedia message. From the SSCTK's perspective, this is (1) the creation of a logical

representation of a multimedia message; followed by (2) the construction of an electronic file having a format suitable for transmission by standard e-mail services.

- *Snapshot*: From the user's point-of-view, this is a set of user-selected documents (e.g. images, spreadsheets, text files, etc.) to be included as part of a multimedia message. From the SSCTK's perspective, this is a logical representation of a collection of document references, along with corresponding application state information for each.
- *Document*: Herein, this term intends to denote (e.g., textual, audio, video) material. Usually, this material can be presented/played by some application and is associated with some file. 'Document' can also refer to a collection of such material, where the collection can be addressed via a set of references.
- *Application state information:* A set of values describing the state of a running application. With respect to the SSCTK's snapshot mechanism (implemented by the `snapZoo` program), the type and specificity of state information which can be captured often varies from one X application to another. Still, it is usually always possible to obtain the name of the document being presented and the application's display geometry.
  Whenever complete application state information is sent along with documents (or document references) within a multimedia message, it is possible to present the documents and reconstruct the state the applications were in when the message was created. Less-than-complete state information simply implies less-than-complete reconstruction of application state when a document is presented.
- *Message reconstruction and presentation:* Herein, 'reconstruction' concerns the *re*-presentation of a multimedia message. During such re-presentation, applications are started and specific documents are loaded into them. Applications are displayed at specific locations upon the user's screen and, in some cases, the associated documents may be turned to specific pages and scrolled to specific positions. Other state-relevant information may be used during message reconstruction; this depends upon the manner and degree to which an application is snapshot-integrated.
  Within the SSCTK, the instructions for message reconstruction are found in each user's .mailcap file, a file describing the relationship between content-types and action to be taken for each type.
- *Content-type:* Within MIME messages, this is a header field which can be used to specify the type and subtype of data in the body of a message, as well as fully specify the native representation (encoding) of such data. Further technical details can be found in RFC 1521.
- *Annotations:* Within the context of the SSCTK, these are markers (e.g. arrows) associated with numbered labels for which x- and y-positions are known. A basic annotation application is integrated the SSC such that annotations can be used to help distinguish areas-of-interest within multimedia messages.
- *Snapshot-aware application:* The SSCTK's `snapZoo` program preferentially looks in an X application's "WM_COMMAND" window property when trying to capture state information from a running application. Snapshot-aware applications are those which purposefully update that window property when relevant changes in application state take place.
- *Snapshot-integrated application:* For an X application which doesn't update it's "WM_COMMAND" window property, the SSCTK includes exception-handling

software in order to integrate that X application with the SSCTK. As the name implies, snapshot-integrated applications are those which are integrated with the SSCTK. The set of snapshot-integrated applications is a superset to that of snapshot-aware applications. For all snapshot-integrated applications it is clearly defined what state information can be captured and how an application's state can be restored using that information.

D.2.2 Functionality implemented in the Toolkit

This section describes the functionality implemented in SSCTK. The functional elements within the SSCTK can be assembled as desired by an application developer. **The SSC is a specific assembly of the SSCTK's functional elements.** As such, there exist operations, interface elements, etc., which are available exclusively through the SSC.

In writing this document, care has been taken to try and distinguish exactly which operations, functions, etc. are available exclusively through the SSC; in such cases, these are specifically noted as being so. If not noted, the functional elements described herein are a general part of the SSCTK.

As mentioned earlier, the SSCTK provides specific support with respect to the composition, reconstruction and presentation of multimedia messages. The SSCTK does not involve itself with the transmission of multimedia messages — that job is performed by standard e-mail services. The SSCTK's different kinds of multimedia messaging support will be elaborated in the sections below.

D.2.2.1 Composing multimedia messages

Users may often find it useful to include snapshots within multimedia messages. The SSCTK therefore includes a snapshot mechanism implemented via the Toolkit's `snapZoo` program; this program is a central part of the SSC.

Composition of a multimedia message, via the SSCTK's snapshot mechanism, transpires in two phases. In the first phase, the snapshot mechanism allows the user to select the documents to be included in the message. When selecting, the user simply points within the application window in which the document is being presented and clicks once. For each document selected, the snapshot mechanism captures certain information about the presentation state of the application. A "snapshot" is a *logical* representation of the collection of document references along with the corresponding application state information for each.

For each document selected, the snapshot mechanism attempts to capture as much information as possible about the presentation state of the application; the type and specificity of state information captured depends upon the degree to which the application is snapshot-integrated.

In addition to including a snapshot within a multimedia message, the user may also include an audio recording, text message and/or simple annotation marks. Generation of these message parts is accomplished via the `sounded`, `texted` and `openmarkup` applications, respectively. Texted is found in the SSCTK, sounded in the DSTK and openmarkup, in the GWTK. Each of these applications is fully integrated with the SSC.

The SSC's user-interface allows the user to enter a name (or group) to which the message is to be sent, as well as fill in a brief "Subject:" line, if desired.

The second phase of multimedia message creation involves the construction of a multimedia message having a format suitable for transmission by standard e-mail services. Messages suitable for such transmission will also be called "physical messages" herein. In this SSCTK implementation, functions exist for the creation of MIME-formatted messages; the format of these messages is based upon an extended version of that required by `mailto` (for further information, see ...ssctk/man/man1/mailto.1). The BNF specification for the SSCTK's logical message syntax is described later.

The SSCTK's logical message format is easy to interpret, in order to suit different MIME message composers. As an alternative, it would be easy to generate HTML-formatted messages from the logical message. At present, this is not implemented within the SSCTK.

D.2.2.2 Transmission and reception of multimedia messages

Once the MIME messages are constructed, they can be sent with ordinary e-mail to the named recipients. In order to be able to receive and open SSCTK-based messages, the recipients require a mail reader which is MIME compliant.

The MRD is designed to filter SSCTK-based messages and direct them to special mail folders. This is accomplished using `MH` routines (available from the public domain). From a general perspective however, the management and handling of incoming e-mail for a user-site is not part of the SSCTK. To correctly install the SSCTK at a new site, system managers should consider the e-mail integration alternatives offered within the SSCTK's design and select the technique most suitable for their site. More technical details regarding this topic are found below.

For interpreting SSCTK-based multimedia messages, the SSCTK includes a function called `ssctkmetamail`. This function reads a file containing a MIME message, and uses the .mailcap file (described below) to reconstruct and present each part according to its content-type.

D.2.2.3 Opening and viewing multimedia messages

When reviewing multimedia messages, the SSCTK provides facilities for: obtaining a logical overview of each message, based upon that message's content-types; and, opening (i.e., reconstructing and presenting) each message in it's entirety.

The logical overview (or, "content-type overview") for a message is displayed as a set of icons representing the message's content-types; these icons normally look like the icons associated with various applications (e.g., netscape, xv, etc.). The association as to which icon appears for each content-type is specified in the codeenv file (described below).

The purpose of the content-type overview is such that a user can obtain a rough idea as to the contents and potential size of received messages (before opening them). The logical overview also can serve to provide a reminder as to the contents of a message once it has been read.

The SSCTK naturally provides an operation by which to open multimedia messages, in order to review their respective contents in full. This operation is performed by calling the

`ssctkmetamail` program. This function reads a file containing a MIME message, and uses the .mailcap file (described below) to open each part according to its content-type. Each part of the message is reconstructed and presented in parallel.

When reviewing multimedia messages, it is of course possible to retrieve and/or delete old messages.

D.2.3 Distinctions between the SSCTK and MHMTK

It is important to understand the distinctions between the SSCTK and the Multi/Hypermedia Message Toolkit (MHMTK).

The MHMTK allows an application developer to write applications which can offer facilities for creating, sending and retrieving messages; these messages can include multi- and hypermedia attachments. The MHMTK is built on top of the X.400 Communication Handler (which employs DAIS). The MHMTK offers extensive facilities for Folder navigation, access and manipulation, as well as operations for intra-message attribute access and status control.

The SSCTK consists of public-domain software packages, as well as new software written to exploit the features within those packages. The SSCTK allows an application developer to write applications offering facilities for creating and reviewing multimedia messages.

Functionality shared by the SSCTK and the MHMTK is that both offer mechanisms by which to create transmission-ready instances of multimedia messages. However, one striking contrast between the two Toolkits is that when a creating multimedia message using the SSCTK, the underlying mechanisms can be used to create a logical representation of the message.

Once the logical representation of the message is available, the SSCTK can be used to create a MIME-formatted, transmission-ready instance of the message from that representation. Using the MIME format as a basis for multimedia messaging, arbitrary mail applications which understand that format can be used (in principle) for sending, transporting, and receiving such messages (including the MHMTK).

Another powerful feature within the SSCTK again concerns the creation of multimedia messages. In particular, the SSCTK's snapshot mechanism captures application state information when selecting documents for inclusion within a multimedia message. Together, document references and associated application state information can be used to create multimedia messages which can be reconstructed when opened by the receiver.

Lastly, the SSCTK includes a ready-to-use application called the Snapshot Composer (SSC). The SSC offers access to all of the SSCTK's high-level functionality, as well as a tailorable user-interface.

D.3 Technical Description

This section describes only the top-level routines within the SSCTK, such that the overall design of the Toolkit can be understood. Supporting routines and low-level functionality are discussed in a later section.

D.3.1 Environment variables

The environment variables used within the SSCTK are: SSCTK_HOME, SUN4_SNAPAPPLS, SUN4_SNAPSERVER, FORKEDorSUFX, LOCALDISKAPPS, <$APPL_i$>SUFX, <$APPL_i$>PATH, TMP_SNAP_AREA, SNAPSCR_NEW and SNAPTERMCMD. When the MRD is installed, it is necessary for the system installer to ensure that these variables are set[1] appropriately.

1. SSCTK_HOME:
   The value for this variable is dependent upon $CODE_HOME, the value of the top-level installation point for the EuroCODE system. Typically, the value for SSCTK_HOME is:
   ~ecode/shell/toolkits/ssctk.
2. SUN4_SNAPAPPLS:
   This variable need only be specified when there exist snapshot-integrated applications which must be run on a SUN4 platform (rather than Solaris). This may occur when a user site owns a SUN4 license for an application, but not a Solaris license.
   When specified, the value for this variable must be a string which specifies one or more application names, where the names are separated by one or more white spaces. An example might be: “exclaim xdvi”.
3. SUN4_SNAPSERVER:
   This variable need only be specified when the value of SUN4_SNAPAPPLS is specified. The value for this variable must be the hostname of a network-accessible SUN4 platform.
4. FORKEDorSUFX:
   This variable need only be specified when there exist snapshot-integrated applications which either (1) implicitly or “automatically” fork when the binary (or start-up script) is executed; and/or (2) by default, seek document names having a specific suffix.
   Case (1) can be tested by simply starting the application and checking whether the system returns to the command-line. Case (2) is more subtle, and can usually be found in an application’s documentation. If one still can’t determine whether case (2) applies, it will soon be discovered when trying to open a document which has been sent using the SSCTK’s mechanisms.
   When specified, the value for this variable must be a string which specifies one or more application names, where the names are separated by one or more white spaces. An example might be: “exclaim imaker”; in this example, exclaim is an application which looks for documents ending with the suffix “wkz”, while imaker (i.e., FrameMaker) is an application which implicitly forks when started.
5. LOCALDISKAPPS:
   This variable need only be specified when there exist snapshot-integrated applications which require that documents to be opened must be accessible on the user’s local disk. (This is a bug, for example, in the SUN4 release of exclaim v1.0.)
   When specified, the value for this variable must be a string which specifies one or more application names, where the names are separated by one or more white spaces. An example might be: “exclaim”.

1) The environment variables for the MRD are found in ~ecode/demonstrators/mrd/config/codeenv.

6. <*APPL*$_i$>SUFX:
   An instance of this variable must be specified for each application name appearing in FORKEDorSUFX. For instance, when the value of FORKEDorSUFX is "exclaim imaker", then two environment variables must be declared and values specified for them. The names of the variables must be *upper-case* versions of the application names found in FORKEDorSUFX, where each such variable has the characters "SUFX" appended to it. Continuing with the example, we must declare EXCLAIMSUFX and IMAKERSUFX, where:
   EXCLAIMSUFX *must* be "wkz" (since exclaim looks for documents ending with the suffix "wkz"; and,
   IMAKERSUFX must have some value (e.g., "doc"). Note that since imaker forks, a suffix *must* be provided; however, since imaker doesn't require a *specific* suffix, the value provided for IMAKERSUFX is arbitrary.
7. <*APPL*$_i$>PATH:
   Instances of this variable must be specified for certain snapshot-integrated applications. Briefly, the applications for which this is necessary are those which do not provide full path- and file-name information for the SSCTK when snapshots are created.
   An instance of this variable should have a name which reflects the snapshot-integrated application with which it is associated (e.g., XVPATH for `xv`, ATOOLPATH for `audiotool`, XDVIPATH for `xdvi`, etc.).
   For each instance of an <*APPL*$_i$>PATH environment variable, the value for that variable is a string which specifies one or more directory paths, where the paths are separated by one or more white spaces. For example, when XDVIPATH is specified, it's value might be "/local/home/$USER /local/home/$USER/* /tmp". This value specifies where the SSCTK's perl routines will look for dvi-formatted files, whenever such documents are selected for inclusion within multimedia messages.
   In this example, the SSCTK's routines will first look for the sought file under /local/home/$USER. If not found there, the search will continue within all directories *immediately under* /local/home/$USER/, and thereafter under /tmp until the sought file is found.
   It is possible for individual users to append new paths to <*APPL*$_i$>PATH variables using the MRD's 'concatStrs' function; in the ~/.cshrc file, a user might have for example:
   source ~ecode/demonstrators/mrd/config/codeenv
   setenv XDVIPATH `concatStrs $XDVIPATH "/local/home/$USER/appl/xdvi"`
   This expression will add /local/home/$USER/appl/xdvi to the set of paths in which the Snapshot Composer searches for dvi-formatted files.
   More technical information regarding this topic is presented later.
8. TMP_SNAP_AREA:
   The value of this variable is used when interpreting a MIME message and packing up it's contents. The value of TMP_SNAP_AREA specifies the default directory to which any files included in the MIME message are written. Normally, the value of TMP_SNAP_AREA is set to "/local/tmp". Using "/tmp" as a value can lead to problems, depending upon the way in which a site's file systems are mounted.
9. SNAPSCR_NEW:
   SNAPSCR_NEW is a reference to the script $SSCTK_HOME/bin/snapZoo. SnapZoo is the script which allows the user to select a set of documents to include in a

multimedia message's snapshot.
The output of snapZoo is a logical representation of a collection of document references, along with corresponding application state information for each. The output of snapZoo is written to /local/home/$USER/tmp/mrdSnapshot.

10.:SNAPTERMCMD
The value of SNAPTERMCMD is used to terminate execution of the snapZoo program. As mentioned above, the user simply clicks within application windows in order to specify documents for inclusion within multimedia messages. Technically, snapZoo is executing a while loop at this time and, upon each loop, reading the name of the window selected.
When a window is selected which has a title bar equal to SNAPTERMCMD, snapZoo terminates. snapZoo does not return document and status information for that window.

Note that although the SSCTK's environment variables are set for all users when sourcing the MRD's set of environment variables, they can be modified locally by the users following that source statement (as shown above). Obvious candidates for modification/extension are SUN4_SNAPSERVER and instances of <$APPL_i$>PATH variables. For the application developer, SNAPTERMCMD may likely be modified/reset.

D.3.2 Special files

D.3.2.1 The $MRD_HOME/config/snapConfig file

The snapConfig file is composed of two parts. The first part of the snapConfig file specifies the set of applications which are snapshot-integrated. For each such application, a bitmap is also specified. These bitmaps are used (1) when creating snapshots and (2) when reviewing the logical contents of messages which have been received.

The second part of the file contains MRD-internal data and should not be edited nor modified.

More technical information regarding this file is presented later.

D.3.2.2 The $MRD_HOME/lib/perl/collect_*.pl files

To support the snapZoo script (described below), there exist a number of collect_*.pl files. Examples are

- `collect_audiotool.pl`,
- `collect_dockit.pl`,
- `collect_xdvi.pl`,
- `collect_netscape.pl`, etc.

Each file is comprised of exactly one perl subroutine, where the subroutine has the same name as the file (minus the ".pl" suffix). Each subroutine is used in order to integrate an existing X application with the SSCTK.

Each of the subroutines is application-specific; when examining the subroutines, however, one will notice that their structure and content is highly similar. From a functional per-

spective, the subroutines are used to capture application state information when creating snapshots.

More technical information regarding these files is presented later.

D.3.2.3 The ~/.mailcap file

Normally, each user has a .mailcap file. It is also possible to configure things such that there exists a default .mailcap file to which various users have (symbolic) links.

In general, a .mailcap file contains a set of specifications. Each specification associates a content-type with a command for treating/interpreting that type. For each snapshot-integrated application, the SSCTK requires such a specification. Some examples are:

```
application/x-xv;       xv %s -geometry %{geometry}
application/x-audiotool;mrdAudiotool %s -geometry %{geometry}
application/x-netscape; netscape -geometry %{geometry} %{o1} &
```

With respect to the SSCTK, these specifications are used when invoking the command for reconstructing and presenting the full contents of a multimedia message. At that time, the various parts of the selected MIME message are unpacked and, depending on the content-type of each part, a specific command is executed in order to open the corresponding part of the MIME message.

For more technical information regarding this file, see ...ssctk/man/man1/metamail.1.

D.3.3 Composing and sending multimedia messages

D.3.3.1 Creating the snapshot (snapZoo)

As mentioned above, snapZoo is the script which allows the user to select a set of documents to include in a multimedia message's snapshot. In the SSC's user-interface, it is invoked by clicking once upon the camera icon found in the "send" window. Once started, the cursor changes to the cross hairs symbol; this indicates that the user may begin selecting documents to include in a snapshot. To select a document, the user simply points within the application window in which the document is being presented and clicks once.

The value of SNAPTERMCMD is used to terminate execution of the snapZoo program. As mentioned above, the user simply clicks within application windows in order to specify documents for inclusion within multimedia messages. Technically, snapZoo is executing a while loop at this time and, upon each loop, reading the name of the window selected. When a window is selected which has a title bar equal to SNAPTERMCMD, snapZoo terminates. snapZoo does not return document and status information for that window.

Once the desired documents have been selected, the script is terminated by clicking once again upon the camera icon found in the SSC's "send" window.

The output of snapZoo is a logical representation of a collection of document references, along with corresponding application state information for each. By default, the output of snapZoo is written to /local/home/$USER/tmp/mrdSnapshot. It is possible to direct snapZoo's output to stdout using:

```
snapZoo -stdout
```

More technical information regarding the snapZoo script is presented later.

D.3.3.2 Creating the physical, multimedia message (ssctkmailto)

The ssctkmailto program handles creation of physical multimedia messages; it can also handle sending such messages. It is an extension of the `mailto` program found in the `metamail` package (for further information, see ...ssctk/man/man1/mailto.1 and metamail.1, respectively). Even though ssctkmailto offers the same functionality as `mailto`, only the extensions and features developed in order to create and handle snapshots will be described here.

ssctkmailto serves as a low-level mail program which accepts:

1. a subject line preceeded by the -s option,
2. a filename on stdin where applications and state information are given (compatible with the BNF syntax presented below), and
3. a list of recipients.

It then generates a MIME-message which will be written to a file or sent to the persons specified in the recipient list.

Normally, the ssctkmailto program filters mail through `splitmail` (for further information, see ...ssctk/man/man1/splitmail.1). This may lead to a situation in which large messages are spilt into several smaller messages. On the receiver's side, a message may therefore arrive in several pieces. (This will always be indicated in the header of the file.)

On the other hand, if the -f option is turned on, ssctkmailto will write the physical instance of the message to a file instead of passing it to splitmail. If the users of the system are sharing file systems, it is actually not necessary to use mail to pass snapshots. With the -f option, the message can be piped to individual recipient's snapshot folders. (If e.g. `MH` is used, the message can be piped to `rcvstore` which in turn, directs the message to the actual mail folder; for further information, see ...ssctk/man/man1/rcvstore.1).

ssctkmailto also gives the possibility to write the MIME message to stdout with the -S option. This overlaps with the functionality of the -f option, since stdout can be redirected to a file. The use of the -S option will generate, in contrast to the -f option, some header lines which are usually added by mail programs to each MIME message. First, it generates a *date* field compatible with that of `sendmail`, second, it puts a *from* field which is always set to *Demonstrator@nr.no*; for further information, see ...ssctk/man/man8/sendmail.8.

Note that ssctkmailto is a somewhat limited implementation of the MIME standard. For future versions, ssctkmailto should be extended to handle references to documents instead of sending the whole content of the document.

The BNF-syntax for the logical form of the snapshot file is given below. The symbol "nl" indicates that newline is required, and the atoms filename, data and geodata, <NODOC> and <NOOPTIONS> specify valid values which will not be further described in this document).

```
START -> ~* nl ENCODING | ~.
ENCODING -> 1 nl FILE
```

```
FILE -> FILENAME nl TYPE
FILENAME -> <NODOC> | filename
TYPE -> application/TYPES nl OPTIONS
OPTIONS -> <NOOPTIONS> | OPTIONLIST
OPTIONLIST -> ; OPTIONITEM OPTIONLIST | nl START
OPTIONITEM -> geometry={geodata} | o1={data} | o2={data} |
.... | on={data}
...
TYPES -> x-xv
TYPES -> x-audiotool
TYPES -> x-conf-pacs-db
TYPES -> x-osiris
TYPES -> x-netscape
TYPES -> x-madeup
TYPES -> x-exclaim
TYPES -> x-dvi
TYPES -> x-dockit
TYPES -> x-annot
```

D.3.4 Receiving and opening multimedia messages

D.3.4.1 Spooling messages (.mailfilter / MH)

Upon arrival, SSCTK-based multimedia messages are distinguished from other e-mail and directed to the respective user's snapshot folder. NR's e-mail system employs PP mail. Therefore, the message filtering job is taken care of by users' ~/.mailfilter files. For extracting SSCTK-based multimedia message using .mailfilter files at NR, the .mailfilter file looks like:

```
PATH="/local/mh/lib/mh:/local/pp/bin";
folder = FALSE;
if ($(subject) == /mrd::/) {
pipe "rcvstore +snaps"; folder = "snaps";
} else {
unixfile "/usr/spool/mail/$(userid)";
}
```

Note: If `sendmail` is used as MTA (mail transport agent), filtering has to be done by piping all incoming messages through the `slocal` program (which should be called from the .forward file in the user's home directory). `slocal` consults the users' ~/.maildelivery files (which have contents similar to that of .mailfilter files). For further information, see ...ssctk/man/man8/sendmail.8 and ...ssctk/man/man1/slocal.1, respectively.

SSCTK-based multimedia messages are stored in `MH` folders. `MH` has functionality for searching in headers, keeping track of unseen messages etc. The `rcvstore` program places messages in a specified folder. Each SSCTK-based message is given a unique file name (an integer), thus retrieval of messages is easy. For further information, see ...ssctk/man/man1/mh.1 and rcvstore.1, respectively.

D.3.4.2 Getting messages (scan / lsc)

`lsc` is a small program which returns a list of files for a given directory (latest files first). `scan` is a program which takes a filename as a parameter and returns the filename (including some status information), the data, the sender and the subject.

By using these to functions it is easy to construct an interface which keeps track of SSCTK-based messages. For further information, see ...ssctk/man/man1/scan.1.

D.3.4.3 Reviewing messages' logical contents

The facility for reviewing the logical contents of SSCTK-based messages is implemented by "grep'ing" on the MIME file for occurrences of 'Content-type'. Thereafter, each line returned from grep is checked to see whether it denotes a document. This is easily done by using regular expressions. Note that in the current implementation of the SSCTK, it is usually file contents (not references) which are passed within messages. For each document included in a message, the content-type overview therefore displays an icon representing the application normally used to view that kind of document.

D.3.4.4 Opening messages (ssctkmetamail)

ssctkmetamail interprets a MIME message file and starts applications according to specifications found in a user's .mailcap file. ssctkmetamail is an extension of `metamail` (see ...ssctk/man/man1/metamail.1); therefore, only the functionality of metamail as related to the Snapshot Composer will be described here.

Within the SSCTK, `metamail` is activated by writing a command line with the following options.

```
ssctkmetamail -d < <filename>
```

The -d option tells ssctkmetamail to run the applications without interacting with the user, that is, in a silent mode. The SSCTK opens message-parts in a parallel mode, since this may be crucial in certain situations. The content of `<filename>` is passed through stdin.

If an SSCTK-based message is split up into several pieces, ssctkmetamail must be started for each of the pieces. ssctkmetamail will not open any part of the message before this is done.

In some cases, the use of ssctkmetamail requires that message-parts be decoded and written as temporary files. The default directory for temporary files is /tmp, but this should (in many cases) be overridden by setting the global variable TMP_SNAP_AREA to "/local/tmp" (or some other directory to which all users have access).

D.3.4.5 Handling non-standard conditions when opening messages (fixStartup)

The fixStartup program is employed within the .mailcap file. It is used to handle non-standard conditions when preparing to open some part of an SSCTK-based messages. The conditions it "fixes" are:

- starting applications which must can only be run upon a SUN4 platform (see SUN4_SNAPAPPLS above);
- starting applications which either:
  - implicitly or "automatically" fork when the binary (or start-up script) is executed; and/or
  - by default, seek document names having a specific suffix; (see FORKEDorSUFX and <$APPL_i$>SUFX above); and/or
- starting applications which require that documents to be opened must be accessible on the user's local disk (see LOCALDISKAPPS above).

Use of fixStartup is simple. For example, the SSCTK-related specification for starting `xv` in the .mailcap file is normally:

```
application/x-xv;xv %s -geometry %{geometry}
```

Had xv been an application which always sought files ending in "gif", the specification would instead be:

```
application/x-xv;fixStartup xv %s -geometry %{geometry}
```

In such a case of course, it would be necessary to (1) add xv to the set of applications found in FORKEDorSUFX, and (2) declare XVSUFX and set it to "gif".

D.4 The Application Developer's Interface

The SSCTK consists of the public-domain packages called `MH` [75] and `metamail` (see ...ssctk/man/man1/metamail.1), as well as new software written to exploit the features within those packages. Therefore, some parts of the SSCTK's application developer's interface (API) are inherited from that software. For the most relevant functions from those packages, "man pages" have been included under ...ssctk/man/man*/. Interfaces to the SSCTK's other primary functions have been described and/or referenced further in the section above.

The Snapshot Composer's two top-level windows ("send" and "receive", found under ...ssctk/src/send and ...ssctk/src/rec, respectively) are written using the Tcl/Tk programming language. With some experience, Tcl/Tk applications are easy to modify in order to change their look and/or functionality.

As mentioned earlier, the `sounded`, `texted` and `openmarkup` applications are fully integrated with the SSC. Texted is found in the SSCTK, and is realised through the text widget in Tcl/Tk (see ...ssctk/man/mann/text.n). Sounded and openmarkup are found in the DSTK and the GWTK, respectively; see that documentation for further information about these applications.

D.5 Rationale for Design Changes

The SSCTK is a new Toolkit, designed and developed following the MRD evaluation at Rikshospitalet, Oslo. No previous design documents exist for this Toolkit, thus it is not relevant to speak of design changes here.

D.6 Possible Further Improvements to the Toolkit

This section briefly describes functionality which could be included in the Toolkit, in order to improve it.

- *Integration with the MHMTK:* In principle, it is possible to integrate the SSCTK with the MHMTK. For the user, this kind of integration would provide new support for work with Mail Folders, message archiving and access, etc.
- *Integration with the CTK's Registrar*: If the SSCTK were integrated with the CTK's Registrar, it would be possible to provide a much richer context within which user's could search for others' addresses, affiliations, etc. Again, this kind of functionality is not yet implemented.

D.7 An Example of Toolkit Use

This section presents a much more detailed technical clarification as to how the SSCTK's snapshot facility works. This clarification points out how values within special files are used in order to create snapshot-integrated applications.

D.7.1 The $MRD_HOME/config/snapConfig file

The snapConfig file is composed of two parts. The first part of the snapConfig file specifies the set of applications which are snapshot-integrated. For each such application, a bitmap is also specified; the files corresponding to these bitmaps are found in $MRD_HOME/bitmaps/. These bitmaps are used (1) when creating snapshots and (2) when reviewing the logical contents of messages which have been received. The second part of the snapConfig file contains MRD-internal data and should not be edited nor modified.

The snapConfig file is read each time snapZoo is executed. Lines beginning with the symbol '#' are comments and not treated; blank lines are not treated either.

An example of the snapConfig file could be:

```
## exclaim
exclaim
bitmap:  exclaim.xbm

## FrameMaker
dockit
bitmap:  imaker.xbm
```

```
## Mrd Pacs
conf-pacs-db          confPacsDb
bitmap:  pacs.xbm

## xv
xv
bitmap: xv.xbm

## audiotool
audiotool
bitmap:  Dev_mag_tape.xbm

### Netscape
netscape
bitmap:  netscape.xbm

####  ----- MRD internal applications follow below.
####  ----- Specifications which follow are
####  ----- only for definition of bitmaps, not for
####  ----- definition of applications the user will
####  ----- be selecting/picking

####  -----  DO NOT EDIT BELOW THIS LINE ! --------------

## texted
mrdInternal: texted
bitmap:               text2.xbm

## sounded
mrdInternal: sounded
bitmap:               sound2.xbm

## annot
mrdInternal: annot
bitmap:               App_install.xbm
```

Specification lines in the snapConfig file — lines which relate output from `xprop` to snapshot-integrated applications — contain one or two parameters. `xprop` is a function used within snapZoo; it is discussed immediately below.

D.7.1.1 xprop

For a given window, `xprop` returns information about the properties for that window. For the snapshot mechanism, the important properties are usually WM_CLASS and WM_NAME. As an example, consider a window which displays a *FrameMaker* document. When invoking `xprop` for that window, we receive (among other things):

WM_CLASS(STRING) = “dockit”, “Maker”
WM_NAME(STRING) = “/nr/holt/u1/peter/frame/own_templates/fm4/NR_fax.doc”

Note that it may be such that some other script (e.g., a script called `imaker`) was used to start the *FrameMaker* application. Even in such cases, `xprop` will return “dockit” as the first element in the WM_CLASS property for windows which display *FrameMaker* documents.

D.7.1.2 Specification of snapshot-integrated applications

Most specification lines within a snapConfig file consist of one parameter; some have two parameters. Examples from the file listed above are:

```
dockit
conf-pacs-db          confPacsDb
xv
```

The first parameter in a specification line does **not** necessarily specify the name of the application, nor the script used to start the application. Instead, it specifies the value[2] of the first element in the WM_CLASS property for a window presented by some application (e.g., windows in which documents are displayed).

The second parameter, when supplied, functions as a sort of alias; this value must be supplied whenever the value returned from xprop contains characters other than a-z, A-Z, 0-9 or underscore (‘_’). When no alias is explicitly supplied in a specification line, an implicit alias is created which has the same value as the first (only!) parameter in that line.

D.7.2 The $MRD_HOME/lib/perl/collect_*.pl files

To support the snapZoo script, there exist a number of collect_*.pl files. Examples are

- `collect_exclaim.pl`,
- `collect_dockit.pl`,
- `collect_confPacsDb.pl`,
- `collect_xv.pl`,
- `collect_audiotool.pl`,
- `collect_netscape.pl`, etc.

As mentioned earlier, each file is comprised of exactly one perl subroutine, where the subroutine has the same name as the file (minus the “.pl” suffix). Each subroutine is used in order to integrate an existing X application with the SSCTK.

Each of the subroutines is application-specific; when examining the subroutines, however, one will notice that their structure and content is highly similar. From a functional perspective, the subroutines are used to capture application state information when creating snapshots.

2) If you are not sure of this value, execute `xprop` and click the mouse upon the desired window.

D.7.3 snapZoo

D.7.3.1 snapZoo, the collect_*.pl files and the snapConfig file

SnapZoo is a perl script. It contains general-purpose code for creating a logical representation of a snapshot. In order to capture application-specific state information, however, snapZoo relies upon the perl subroutines found in the collect_*.pl files. This arrangement of general-purpose and application-specific software makes it quite easy to integrate new X applications with the SSCTK.

When snapZoo is executed, the snapConfig file is read. For each specification line, the (implicit or explicit) alias therein is used in order to locate a corresponding collect_*.pl file. When located, the file's perl subroutine is included in snapZoo. For example the specification line:

`dockit`
causes snapZoo to locate the collect_dockit.pl file and thereby include the collect_dockit subroutine. Alternatively, the specification line:

`conf-pacs-db confPacsDb`

causes snapZoo to locate the collect_confPacsDb.pl file and thereby include the collect_confPacsDb subroutine.

D.7.3.2 User input

Once the collect_* subroutines have been included, snapZoo enters a state in which it accepts mouse clicks from the user. The user moves the cursor into an application window, clicks once, moves to another application window, clicks once,... and so on. When the user is finished clicking upon application windows, he returns to the camera icon (found within the SSC's "send" window) and clicks upon it. snapZoo recognizes this final click as a signal to terminate the user's input and to enter a state wherein that input is processed.

D.7.3.3 Input processing by snapZoo

snapZoo employs the `xwininfo` command in order to facilitate user input. While the user is entering each click, snapZoo reads the output from `xwininfo` and records a correspondence between the (unique) window id and the geometry of that window.

Once the input termination signal is captured, snapZoo begins invoking the `xprop` function for each window id recorded. The output from `xprop` is scanned in order to find out, for a given window id, which type of application window corresponds to that id; it is here that the first element of the WM_CLASS property is read.

Given the value of that element, the correspondences derived earlier (while reading the snapConfig file) determine which perl subroutine shall be used in order to extract application state information for that application instance.

As display state information, most '`collect_`' subroutines return the name of the document being presented within the selected window. Exceptions to this rule will be discussed later.

D.7.3.4 Output from snapZoo

The output from snapZoo is written into /local/home/$USER/tmp/mrdSnapshot. The output is read and interpreted in order to generate a (physical) mail message in MIME format.

An example of output from snapZoo could be:

```
~*
1
/local/home/peter/images/gif/CT_1_s01.gif
application/xv
; geometry="510x417+498+31"
~*
1
/local/home/peter/audio/audioFiles/au/angioMsg.au
application/audiotool
; geometry="428x189+520+486"
~*
1
<NODOC>
application/conf-pacs-db
; geometry="278x441+38+68"; o1="1"; o2="2"
~.
```

Note: to be entirely precise, the EOF characters ('~.') are not part of the output from snapZoo. Upon the return from snapZoo, they are appended by the SSCTK subroutine which invokes snapZoo.

Part 5

Glossary, Acknowledgements and References

Glossary

This glossary is organized into four major sections. The sections are divided according to toolkit. In this way, the sections function so as to collect related terms together, and present them in as logical a fashion as possible. The list of terms below is alphabetized, and the number following each term indicates where it can be found within the glossary as a whole.

- annotation (8, 21)
- application sharing (6)
- application state information (18)
- audio file (11)
- audio format (12)
- conference-aware application (4)
- conference manager (2)
- conference object (1)
- content-type (20)
- document (17)
- message composition (15)
- message reconstruction and presentation (19)
- multi-party conference audio (9)
- multicasting (10)
- multimedia messaging (14)
- registrar (3)
- snapshot (16)
- Snapshot Composer (13)
- snapshot-aware application (22)
- snapshot-integrated application (23)
- telepointing (7)
- window replication (5)

Basic CTK concepts

1. *Conference object*; is a server process that keeps track of participants and applications in a real-time conference. It includes and removes participants and applications from a real-time conference.
2. *Conference manager*; is the user interface to a conference. It passes instructions from the participants to the conference object. A conference with one or more participants will have at least one conference manager.

3. *Registrar*; is a server process used by conference objects and managers. It keeps track of information about participants (e.g., user name, host, display etc.) and applications.
4. *Conference-aware application*; is an application that knows it is running under a conference. A conference application can be centralized, distributed or pseudo-distributed. A centralized conference application has one process running on one machine, but a distributed conference application has a process for every participants. A pseudo-distributed conference is neither fully centralized nor distributed.

Basic GWTK concepts

5. *Window replication:* is defined in a *technical perspective;* it refers to the ability for the same window to be presented on different computer screens at the same time. Moreover, replicated windows shall make output visible, and allow input to be given for all users according to defined rules, often termed as floor control.
6. *Application sharing*: is defined in a *user perspective*; it is used to denote the situation during which an application's output is replicated and thus shared among users synchronously.
7. *Telepointing*: is the ability to show other users where you point with the cursor in a shared application. A telepointer should be labeled by a user name (or some other unique characteristic) in order that it's owner be identifiable.
8. Annotation: is a labeled and numbered static mark which, in a conference, refers to a single point in a shared window. In an asynchronous communication mode, annotations can be used to mark any part of the screen; in most cases, however, such annotations are used to point within some application's window.

Basic DSTK concepts

9. *multi-party conference audio*: Multi-party audio conversations over a data network.
10. *multicasting*: This term concerns transmission and reception of information. From a logical perspective, it concerns communication of information from a single sender to some multitude of receivers (and vice-versa). With regard to low-level data messages, there exist two fundamentally different technical implementations:
    a) a round-robin technique which sends the message from the sender to the first receiver, then from the sender to the second receiver, and so on.
    b) so-called "true multicasting", wherein all parties both send to and read from a single (multicast) address; in this way, a sender only needs to send out a single instance of the message in order for it to reach all receivers.
11. *audio file*: A collection of audio samples, sometimes prefixed by a header which provides information about file's audio format.
12. *audio format*: A collection of parameters such as sample rate, number of channels,

precision, encoding (e.g., A-law, linear), etc., which describe how some given audio data can be interpreted.

Basic SSCTK concepts

13. *Snapshot Composer (SSC)*: The SSC is the primary application within the SSCTK; it comprises essentially all of the Toolkit's fundamental functionality. This functionality is offered through the SSC's top-level windows, "send" and "receive".
14. *Multimedia messaging:* Within the context of the SSCTK, this is the ability to compose, send, receive and reconstruct messages containing more than one media type.
15. *Message composition:* From the user's point-of-view, this is the creation of a multimedia message. From the SSCTK's perspective, this is (1) the creation of a logical representation of a multimedia message; followed by (2) the construction of an electronic file having a format suitable for transmission by standard e-mail services.
16. *Snapshot*: From the user's point-of-view, this is a set of user-selected documents (e.g. images, spreadsheets, text files, etc.) to be included as part of a multimedia message. From the SSCTK's perspective, this is a logical representation of a collection of document references, along with corresponding application state information for each.
17. *Document*: Herein, this term intends to denote (e.g., textual, audio, video) material. Usually, this material can be presented/played by some application and is associated with some file. 'Document' can also refer to a collection of such material, where the collection can be addressed via a set of references.
18. *Application state information:* A set of values describing the state of a running application. With respect to the SSCTK's snapshot mechanism (implemented by the `snapZoo` program), the type and specificity of state information which can be captured often varies from one X application to another. Still, it is usually always possible to obtain the name of the document being presented and the application's display geometry.
    Whenever complete application state information is sent along with documents (or document references) within a multimedia message, it is possible to present the documents and reconstruct the state the applications were in when the message was created. Less-than-complete state information simply implies less-than-complete reconstruction of application state when a document is presented.
19. *Message reconstruction and presentation:* Herein, 'reconstruction' concerns the *re*-presentation of a multimedia message. During such re-presentation, applications are started and specific documents are loaded into them. Applications are displayed at specific locations upon the user's screen and, in some cases, the associated documents may be turned to specific pages and scrolled to specific positions. Other state-relevant information may be used during message reconstruction; this depends upon the manner and degree to which an application is snapshot-integrated.
    Within the SSCTK, the instructions for message reconstruction are found in each

user's .mailcap file, a file describing the relationship between content-types and action to be taken for each type.

20. *Content-type*: Within MIME messages, this is a header field which can be used to specify the type and subtype of data in the body of a message, as well as fully specify the native representation (encoding) of such data. Further technical details can be found in RFC 1521.
21. *Annotations:* Within the context of the SSCTK, these are markers (e.g. arrows) associated with numbered labels for which x- and y-positions are known. A basic annotation application is integrated the SSC such that annotations can be used to help distinguish areas-of-interest within multimedia messages.
22. *Snapshot-aware application:* The SSCTK's `snapZoo` program preferentially looks in an X application's "WM_COMMAND" window property when trying to capture state information from a running application. Snapshot-aware applications are those which purposefully update that window property when relevant changes in application state take place.
23. *Snapshot-integrated application:* For an X application which doesn't update it's "WM_COMMAND" window property, the SSCTK includes exception-handling software in order to integrate that X application with the SSCTK. As the name implies, snapshot-integrated applications are those which are integrated with the SSCTK. The set of snapshot-integrated applications is a superset to that of snapshot-aware applications. For all snapshot-integrated applications it is clearly defined what state information can be captured and how an application's state can be restored using that information.

Acknowledgments

The creation and completion of the Middle Road Demonstrator would not have been possible without the individual efforts of a number of others. We would like to recognize and thank those persons here.

Norsk Regnesentral:

Pål Sørgaard - for his leadership, creative ideas, thoroughness and dedication throughout the first two years as project leader of our EuroCODE team.

Gisle Aas (known locally as “null-comma-null Gisle”) - for his deep technical insight and visions, as well as his capacity to write programs in zero flat.

Birger Møller-Pedersen - for his razor-sharp distinctions when discussing, developing, critiquing and explaining object-oriented designs and implementations.

Jørn Braa - for his hard work in eliciting and documenting functional requirements together with the pilot group.

Steinar Kristoffersen - for his thorough efforts, especially in regard to hunting down, analyzing and testing different window replication software.

Torgeir Veimo - for his creative design solution for an application-independent telepointer.

Michael Gritzman - for his clear mind and unquestionable expertise in designing the final version of the MRD’s user-interface.

Stein Myrseth and Håvard Hegna - for helping to check out, explore and test cross-platform possibilities.

Morten Salomensen and *Staale Heier* - for their dedication and service-mindedness in helping to establish and maintain a stable program development environment.

Department of Radiology, Rikshospitalet, Oslo:

Søren Bakke, Bjarne Smevik, Hans Jørgen Smith and *Andreas Abildgaard* - for their enthusiasm, creativity, attention to detail and lastly, but more than anything else, their incredible patience in testing and evaluating the early and somewhat unstable versions of the system.

Aarhus University:

Morten Kyng - for his capacity to create order in the midst of chaos, anytime, anyplace.

Ole Lehrmann Madsen and *Søren Brandt* - for their knowledge of and willingness to navigate within the deep, mystical world of BETA (sometimes even with only minimal clues as to where they might need to go).

Kaj Grønbæk and *Lennart Sloth* - for their hard work, dedication and unbelievable turn-around time when helping to develop the first, hypermedia-based version of the Snapshot Composer.

Elmer Sandvad and *Kim Halskov Madsen* - for their various efforts throughout the project, in particular their creativity and work during the development and refinement of the MRD use scenarios.

Jydsk Telefon:

Leif Elstrøm - for his coordinating skills and eternal optimism.

John B. Hansen, Palle B. Nielsen and *Erik Nielsen* - for their efforts throughout the project, especially in regard to managing all of the technical complications which arose in connection with the EuroCODE roadshow.

Palle Bach Nielsen - for his early work in the project, especially in regard to the Temporal Data Toolkit.

Rank Xerox, Cambridge:

Wendy Mackay - for her great fantasy and strong determinism in working to realize the many possibilities inherent in the High Road Demonstrator design.

Petri Launiainen and *Lone Faber* - for their imaginative work in exploring and defining possible constellations of HRD and MRD functionality, especially in regard to Distributed Hypermedia and Temporal Data Toolkit functionalities.

References

[1] Aas, G., “Design of the Global Window Toolkit”, CODE-NR-93-17, Norsk Regnesentral, Oslo, December, 1993.

[2] Aas, G., Holmes, P., Lovett, H., Møller-Pedersen, B., Sørgaard, P., EuroCODE deliverable: “D-5.1: CSCW Shell Requiremnents for the Middle Road Demonstrator”, CODE-NR- 92-8, Norsk Regnesentral, Oslo, December 30, 1992.

[3] Aas, G., Johansen, G., “Real time multimedia conference with voice and global window: User’s Guide -- Implementation Notes”, appendix to Report no. 852, Norwegian Computing Center, Oslo, March, 1992.

[4] Ahuja, S.R., Ensor, J.R., Lucco, S.E., “A Comparison of Application Sharing Mechanisms in Real-Time Desktop Conferencing Systems”, *SIGOIS Bulletin 11* (2/3), 1990, pp. 238-248.

[5] Altenhofen, M., Dittrich, J., Hammerschmidt, R., Käppner, T., Kruschel, C., Kückes, A., Steinig, T., “The BERKOM Multimedia Collaboration Service”, Proc. of the First ACM Intern. Conf. on Multimedia, Aug. 1-6 1993, Anaheim, CA, pp. 457-463.

[6] Andriole, S., *Rapid Application Prototyping*, Boston, MA, QED Technical Publications (1992).

[7] Anupam, V., Bajaj, C.L., “Collaborative Multimedia Scientific Design in SHASTRA”, Proc. of the First ACM Intern. Conf. on Multimedia, Aug. 1-6 1993, Anaheim, CA, pp. 447-456, 479-480.

[8] Baird, J.E., Weinberg, S.B., “Elements of Group Communication”, in *Small Group Communication (Fifth Edition)*, Cathcart, R.S., Samovar, L.A. (editors), Wm. C. Brown Publishers, 1988, pp. 260-274.

[9] Baldeschwieler, J.E., Gutekunst, T., Plattner, B., “A Survey of X Protocol Multiplexors”, ACM SIGCOMM, Vol. 23, No. 2, April 1993, pp. 16-24.

[10] Benson, I., Ciborra, C., Proffitt, S., “Some Social and Economic Consequences of Groupware for Flight Crew”, Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 119-129.

[11] Bly, S.A., “A Use of Drawing Surfaces in Different Collaborative Settings”, Proc. of the Conf. on Computer-Supported Cooperative Work, Sept. 26-28, 1988, Portland, OR., pp. 250-256.

[12] Bly, S.A., Minneman, S.L., “Commune: A Shared Drawing Surface”, Proc. of the Conf. on Office Information Systems, April 1990, Boston, MA., pp. 184-192.

[13] Bormann, C., Hoffmann, G., “Xmc and Xy: Scalable Window Sharing and Mobility — or — From X Protocol Multiplexing to X Protocol Multicasting”, Proc. of the 8th Annual X Technical Conference, Boston, January, 1994.

[14] Braa, J,. Sørgaard, P., Holmes, P., Mogensen, P., Kyng, M., Thüring, M., Robinson, S., Kreifelts, T., Mackay, W., EuroCODE deliverable: “D-1.2: Requirements for EuroCODE Systems”, March, 1, 1993.

[15] Bødker, S., Christiansen, E., Grønbæk, K., Madsen, K.H., Mogensen, P., Robinson, M., Kühn, H., Robinson, S., Thüring, M., Hinrichs, E., Sørgaard, P., Hennessy, P., EuroCODE deliverable: “D-1.1: The EuroCODE Conceptual Framework: Preliminary”, June, 17, 1993.

[16] Clark, H.H., Brennan, S.E., “Grounding in Communication”, in *Perspectives on Socially Shared Cognition*, Resnick, L.B., Levine, J.M., Teasley, S.D. (editors), American Psychological Association, 1991, pp. 127-149.

[17] Clark, H.H., Wilkes-Gibbs, D., “Referring as a collaborative process”, Cognition 22 (1986), pp. 1-39.

[18] Craighill, E., Lang, R., Fong, M., Skinner, K., “CECED: A System for Informal Multimedia Collaboration”, Proc. of the First ACM Intern. Conf. on Multimedia, Aug. 1-6 1993, Anaheim, CA, pp. 437-445.

[19] Crowley, T., Milazzo, P., Baker, E., Forsdick, H., Tomlinson, R., “MMConf: An Infrastructure for Building Multimedia Applications”, Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 329-342.

[20] Degen, L., Mander, R., Salomon, G., “Working with Audio: Integrating Personal Tape Recorders and Desktop Computers”, Proc. of CHI ‘92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 413-418.

[21] Dourish, P., Bly, S., “Portholes: Supporting Awareness in a Distributed Work Group”, Proc. of CHI ‘92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 541-547.

[22] Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E., Pier, K., Tang, J., Welch, B., “Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration”, Proc. of CHI ‘92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 599-607.

[23] EuroCODE: CSCW Open Development Environment, ESPRIT Project 6155, Technical Annex.

[24] EuroCODE Consortium (various authors from AU, GMD, NR, RXE, NX, ICL), EuroCODE deliverable D-2.5: “Design of the Toolkit Layer of the CSCW Shell”, CODE-AU-93-23, Aarhus University, Aarhus, Danmark, February 14, 1994.

[25] EuroCODE Consortium, EuroCODE deliverable D-8.1.6: “EuroCODE Exploitation Plan”, December 31, 1995.

[26] EuroCODE Consortium, EuroCODE deliverable D-8.2.5: “Markets and Application Contexts for EuroCODE Systems”, December 31, 1995.

[27] Galegher, J., Kraut, R.E., “Computer-Mediated Communication for Intellectual Teamwork: A Field Experiment in Group Writing”, Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 65-78.

[28] Gaver, W., Moran, T., MacLean, A., Loevstrand, L., Dourish, P., Carter, K., Buxton, W., “Realizing a Video Environment: EuroPARC’s RAVE System”, Proc. of CHI ‘92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 27-35.

[29] Gieskens, D.F., Foley, J.D., “Controlling User Interface Objects Through Pre- and Post-Conditions”, Proc. of CHI ‘92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 189-194. Gray, W.D., John, B.E., Atwood, M.E., “The Precis of Project Ernestine or An Overview of a Validation of GOMS”, Proc. of CHI ‘92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 307-312.

[30] Grimstad, T., Maartmann-Moe, E., Aas, G., “Real time multimedia conference with voice and global window”, Report no. 852, Norwegian Computing Center, Oslo, Dec. 1991.

[31] Gritzman, M., Kluge, A., Lovett, H., “Task Orientation in User Interface Design”, in *Human Computer Interaction - Interact ‘95*, K. Nordby, P. Helmersen, D.J. Gilmore, S.A. Arnesen (eds.), Chapman and Hall, London, 1995, pp. 97-102.

[32] Grudin, J., “Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces”, Proc. of the Conf. on Computer-Supported Cooperative Work, Sept. 26-28, 1988, Portland, OR., pp. 85-93.

[33] Grudin, J., “The Case Against User Interface Consistency”, *Communications of the ACM*, 1989, 2(10), pp. 1164–1173.

[34] Grudin, J., “CSCW: The Convergence of Two Development Contexts”, Proc. of CHI ‘91 (Human Factors in Computing Systems) April 27-May 2, 1991, New Orleans, Louisiana, pp. 91-97.

[35] Grudin, J., “Groupware and Social Dynamics: Eight Challenges for Developers”, *Communications of the ACM*, January 1994, 37(1), pp. 92-105.

[36] Grønbæk, K., “Composites in a Dexter-Based Hypermedia Framework”, Proceedings of the ACM European Conference on Hypermedia Technology (ECHT‘94), Edinburg, UK, September 18-23, 1994, pp. 59-69.

[37] Hannemyr, G., *Åpne systemer: teknologi, strategi og praksis* (Open systems: technology, strategy and practice), Universitetsforlaget AS, Oslo, 1992.

[38] Holmes, P., “Design of the Digitized Sound Toolkit”, CODE-NR-93-18, Norsk Regnesentral, Oslo, December, 1993.

[39] Holmes, P., “Hypermedia, Desktop Conferencing and PACS”, presented at EuroPACS ‘94: The 12th International EuroPACS Meeting, Geneva, Switzerland, September 22-24, 1994.

[40] Holmes, P., Lovett, H., Løbersli, F., “Evaluation of the Middle Road Demonstrator at Rikshospitalet”, CODE-NR-95-5, Norsk Regnesentral, Oslo, August 31, 1995. Also found in EuroCODE deliverable: “D-1.4.2: EuroCODE Demonstrator Evaluation Report”, EuroCODE Consortium, August 31, 1995.

[41] Holmes, P., Lovett, H., Løbersli, F., Skorve, E.T., EuroCODE deliverable: “D-5.6: The Middle Road Demonstrator: Concept, use, technical foundation and evaluation

of a system for supporting communication within cooperative work settings", CODE-NR-95-11, Norsk Regnesentral, Oslo, December 20, 1995.

[42] Holmes, P., Lovett, H., Møller-Pedersen, B., Sørgaard, P., Aas, G., Madsen, K.H., and Sandvad, E., "D-5.2: Design of the Middle Road Demonstrator", CODE-NR-93-12, Norsk Regnesentral, Oslo, August 31, 1993.

[43] Holmes, P., Sørgaard P., "Hypermedia, Desktop Conferencing and PACS", EuroPACS Newsletter, Vol. 8, Nr. 1, 1995, (http://www.nr.no/eurocode/papers/EUroPACS/).

[44] Ishii, H., "TeamWorkStation: Towards a Seamless Shared Workspace", Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 13-26.

[45] Ishii, H., Arita, K., "ClearFace: Translucent Multiuser Interface for TeamWorkStation", Proc. of the Second European Conf. on Computer-Supported Cooperative Work, Bannon, L., Robinson, M. & Schmidt, K. (eds.), Sept. 25-27, 1991, Amsterdam, pp. 163-174.

[46] Ishii, H., Arita, A., Yagi, T., "Beyond Videophones: TeamWorkStation-2 for Narrowband ISDN", *Proceedings of the Third European Conference on Computer-Supported Cooperative Work*, G. De Michelis, C. Simone, K. Schmidt (eds.), Sept. 13-17, 1993, Milan, Italy, pp. 325-340.

[47] Ishii, H., Kobayashi, M., "ClearBoard: A Seamless Medium for Shared Drawing and Conversation with Eye Contact", Proc. of CHI '92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 525-532.

[48] Ishii, H., Miyake, N., "Toward an Open Shared Workspace: Computer and Video Fusion Approach of TeamWorkStation", Comm. of the ACM, December 1991, pp. 37-50.

[49] Jeffay, K., Lin, J.K., Menges, J., Smith, F.D., Smith, J.B., "Architecture of the Artifact-Based Collaboration System", Proc. of the ACM 1992 Conf. on Computer-Supported Cooperative Work (CSCW '92), Oct. 31 - Nov. 4, 1992, Toronto, Canada, pp. 195-202.

[50] Johansen, R., *Groupware: Computer Support for Business Team*s, The Free Press, New York, 1988.

[51] John, B.E., Vera, A.H., "A GOMS Analysis of a Graphic, Machine-Paced, Highly Interactive Task", Proc. of CHI '92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 251-258.

[52] Johnson, B., "TreeViz: Treemap Visualisation of Hierarchically Structured Information", Proc. of CHI '92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 369-370.

[53] Knister, M.J., Prakash, A., "DistEdit: A Distributed Tookit for Supporting Multiple Group Editors", Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 343-355.

[54] Knudsen, J.L., Löfgren, M., Madsen, O.L., Magnusson, B., *Object-Oriented Environments: The Mjølner Approach*, Prentice Hall, New York, 1994.

[55] Knudsen, J.L., Madsen, O.L., Sandvad, E., Møller, K.J., Grønbæk, K., Hansen, N.D., Kyng, M., "ECOOL: The EuroCODE Common Object-Oriented Language", CODE-AU- 93-08, Aarhus University, Aarhus, July 1, 1993.

[56] Kristoffersen, S., Holmes, P., Sørgaard, P., "Computer Supported Cooperative Work: Basic Concepts, Some Examples and Theories", NR Note: ITIP/01/94, Norwegian Computing Center, Oslo, January, 1994.

[57] Lauwers, J.C., Joseph, T.A., Lantz, K.A., Romanow, A.L., "Replicated Architectures for Shared Window Systems: A Critique", *SIGOIS Bulletin 11* (2/3), 1990, pp. 249-259.

[58] Lu, I.M., Mantei, M.M., "Idea Management in a Shared Drawing Tool", *Proc. of the Second European Conf. on Computer-Supported Cooperative Work*, Bannon, L., Robinson, M. & Schmidt, K. (eds.), Sept. 25-27, 1991, Amsterdam, pp. 97-112.

[59] Løbersli, F., "Kommunikasjon og samarbeid omkring radiologiske bilder — Konsekvenser for distribuerte sanntids-konferensesystemer" (Communication and cooperation about radiological images — Implications for distributed real-time conferencing systems), Hovedfagsoppgave, Universitet i Oslo, Institutt for informatikk, 13 May 1994.

[60] Løbersli, F., Aas, G., Holmes, P., "Toolkit Evaluation Report for DATK, ICTK, DHMTK, TDTK", CODE-NR-94-12, Norsk Regnesentral, Oslo, September 30, 1994.

[61] Mackay, W., "EVA: An Experimental Video Annotator for Symbolic Analysis of Video Data", *SIGCHI Bulletin*, Vol. 21, No.2 (October 1989), pp. 68-71.

[62] Mackay, W., Davenport, G., "Virtual Video Editing in Interactive Multimedia Applications", *Comm. of the ACM*, Vol. 32, No. 7 (July 1989), pp. 802-810.

[63] Mackay, W., Velay, G., Carter, K., Ma, C., Pagani D., "Augmenting Reality: Adding Computational Dimensions to Paper", *Comm. of the ACM*, Vol. 36, No. 7 (July 1993), pp. 95-96.

[64] Madsen, K.H., Aiken, P., "Some Experiences with Cooperative Interactive Storyboard Prototyping", *Communications of the ACM*, Vol. 36, No. 4 (1993). Also in Muller, J., Kuhn, S., Meskill, J. (eds.): Proceedings of the Participatory Design Conference, Cambridge, MA, MIT Press.

[65] Madsen, O.L., Møller-Pedersen, B., Nygaard, K., *Object-Oriented Programming in the BETA Programming Language*, Addison-Wesley Publ. Co., Wokingham, England, 1993.

[66] Mander, M., Salomon, G., Wong, Y.Y., "A 'Pile' Metaphor for Supporting Casual Organization of Information", Proc. of CHI '92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 627-634.

[67] Mantei, M., "Capturing the Capture Lab Concepts: A Case Study in the Design of Computer Supported Meeting Environments", Proc. of the Conf. on Computer-Supported Cooperative Work, Sept. 26-28, 1988, Portland, OR., pp. 257-270.

[68] Markus, M.L., Connolly, T., "Why CSCW Applications Fail: Problems in the Adoption of Interdependent Work Tools", Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 371-380.

[69] Minneman, S.L., Bly, S.A., “Managing a` trois: a study of a multi-user drawing tool in distributed design work”, Proc. of CHI ‘91 (Human Factors in Computing Systems) April 27-May 2, 1991, New Orleans, Louisiana, pp. 217-224.

[70] Moran, T.P., Anderson, R.J., “The Workaday World as a Paradigm for CSCW Design”, Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 381-393.

[71] Norman, D.A., “Why Interfaces Don’t Work”, in *The Art of Human-Computer Interface Design*, B. Laurel (ed.), Addison-Wesley Publishing Co., 1990, pp. 209-219.

[72] Nygren, E., Lind, M., Johnson, M., Sandblad, B., “The Art of the Obvious”, Proc. of CHI ‘92 (Human Factors in Computing Systems) May 3-7, 1992, Monterey, CA, pp. 235-239.

[73] Pagani, D.S., Mackay, W.E., “Bringing Media Spaces into the Real World”, Proceedings of the Third European Conference on Computer-Supported Cooperative Work, G. De Michelis, C. Simone, K. Schmidt (eds.), Sept. 13-17, 1993, Milan, Italy, pp. 341-356.

[74] Patterson, J.F., Hill, R.D., Rohall, S.L., Meeks, W.S., “Rendezvous: An Architecture for Synchronous Multi-User Applications”, Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 317-328.

[75] Peek, J.D., *MH & xmh — Email for Users and Programmers* (Third Edition), O’Reilly and Associates, Inc., Sebastopol, CA, 1992.

[76] Reder, S., Schwab, R.G., “The Communicative Economy of the Workgroup: Multi-Channel Genres of Communication”, Proc. of the Conf. on Computer-Supported Cooperative Work, Sept. 26-28, 1988, Portland, OR., pp. 354-368.

[77] Robinson, M., “Design for unanticipated use...”, Proc. of the Third European Conference on Computer-Supported Cooperative Work”, September 13-17, 1993, Milan, Italy, pp. 187-202.

[78] Rutter, D.R., *Communicating by Telephone*, Permagon Press, 1987.

[79] Schmidt, K., Bannon, L., “Taking CSCW Seriously - Supporting Articulation Work”, *Computer Supported Cooperative Work (CSCW)* Vol. 1, No. 1-2 (1992) pp. 7-40.

[80] Shepherd, A., Mayer, N., Kuchinsky, A., “Strudel - An Extensible Electronic Conversation Toolkit”, Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 93-104.

[81] Short, J., Williams, E., Christie, B., *The Social Psychology of Telecommunications*, John Wiley and Sons, 1976.

[82] Solheim, I., “Bildekommunikasjon som virkemiddel for informasjonsutveksling og samarbeid — Erfaringer fra Telenor” (Video communication as a tool for information exchange and cooperative work — Experiences from Telenor), NR Report 902, ISBN 82-539-0400-2, Norsk Regnesentral, Oslo, December 1995.

[83] Stefik, M., Foster, G., Bowbrow, D.G., Kahn, K., Lanning, S., Suchman, L., “Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings”, *Communications of the ACM*, Vol. 30, No. 1, January 1987, pp. 32-47.

[84] Sørgaard, P., “Object-Oriented Programming and Computerised Shared Material”, Proc. of the European Conference on Object-Oriented Programming, August 15-17, 1988, Oslo, Norway, pp. 319-334.

[85] Sørgaard, P., “Design of the Conference Toolkit”, CODE-NR-93-16, Norsk Regnesentral, Oslo, December 9, 1993.

[86] Tang, J.C., Leifer, L.J., “A Framework for Understanding the Workspace Activity of Design Teams”, Proc. of the Conf. on Computer-Supported Cooperative Work, Sept. 26-28, 1988, Portland, OR., pp. 244-249.

[87] Tang, J.C., Minneman, S.L., “VideoWhiteboard: Video Shadows to Support Remote Collaboration”, Proc. of CHI ‘91 (Human Factors in Computing Systems) April 27-May 2, 1991, New Orleans, Louisiana, pp. 315-322.

[88] Watabe, K., Sakata, S., Kazutoshi, M., Fukuoka, H., Ohmori, T., “Distributed Multiparty Desktop Conferencing System: MERMAID”, Proc. of the Conf. on Computer-Supported Cooperative Work, Oct. 7-10, 1990, Los Angeles, CA., pp. 27-38.

[89] *Webster’s New World Dictionary of the American Language, Second College Edition*, D.B. Guralnik (ed.), The World Publishing Co., New York, 1970.

[90] Wellner, P., “Digital Desk”, *Comm. of the ACM*, Vol. 36, No. 7 (July 1993), pp. 86-95.

[91] Whittaker, S., Brennan, S.E., Clark, H.H., “Co-ordinating Activity: An Analysis of Interaction in Computer-Supported Co-operative Work”, Proc. of CHI ‘91 (Human Factors in Computing Systems) April 27-May 2, 1991, New Orleans, Louisiana, pp. 361-367.