# T U M

I N S T I T U T  F Ü R  I N F O R M A T I K

# The Collaborative Multi-User Editor Project IRIS

Michael Koch

TUM-I9524
August 1995

T E C H N I S C H E  U N I V E R S I T Ä T  M Ü N C H E N

TUM-INFO-08-1995-I9524-150/1.-FI


Typescript: ---

Druck: Mathematisches Institut und
Institut für Informatik der
Technischen Universität München

# The Collaborative Multi-User Editor Project IRIS

Michael Koch
Institut für Informatik
Technische Universität München
E-mail: kochm@informatik.tu-muenchen.de

August 1995

**Abstract**

The collaborative editing of documents by teams is a very common task nowadays. Writing groups are often distributed over many locations because of the globalization of organizations and the increasing interdisciplinarity of tasks. Since many writers already use computers for their jobs, providing computer support for the collaborative writing process has been identified as an important goal. Numerous tools for computer supported collaborative writing have already emerged but in most cases have not come into widespread usage. As the main reason for this lack, incorrect assumptions about the needs of the collaborating users have been identified.

The goal of our current research is to identify and to satisfy the real needs of writers collaborating in wide area network environments. In accordance with the results of several studies on collaborative writing our approach is to provide as much flexibility and integrability as possible for the user. Therefore we propose an integrated environment with a core data and information management service and several different tools and user interfaces around it. The supported classes for document content and structure can be easily extended by new content types and new document structure types because of the open architecture. For the same reason it is possible to integrate standard editor applications as user interfaces for editing document parts.

For evaluating our results, we are building a collaborative multi-user multimedia editing environment that fulfills all the requirements. In this report the architecture and concepts of that editing environment named IRIS are presented.

**Keywords:** Distributed Multi-User Editor, CSCW, Groupware, Distributed Systems, Concurrency Control, IRIS, Multimedia Documents

# Contents

# Chapter 1

# Introduction

## 1.1 CSCW and groupware

The history of computers is well summarized by the following citation of Tesler:

> "*Computers began as cumbersome machines served by a technical elite and evolved into desktop tools that obeyed the individual. The next generation will collaborate actively with the user*" [Tesle95]

Since the early 1980ies more and more so called *personal computers* are available at workplace. As a desktop tool for individuals, personal computer initially provided services for helping a single user with his work. The most important application classes were databases, word processors, graphic tools and spreadsheets. In the 1990ies these individual workstations got more and more wired together in local and wide area networks. The interconnection of computers was first used for distributed computation and data exchange. The next logical step is now not only to connect programs that are running on different computers but to connect the users themself [Wilso91]. The efforts are intensified by the emerging need for people to work in teams that are locally dispersed.

The research area that is concerned with computer support for collaborating teams is called *Computer-Supported Cooperative Work (CSCW)*[1]. CSCW is not a self-contained research area with its own technology but a very interdisciplinary area of which the main issue is to integrate different technologies in order to support collaborative work. Among others the following disciplines are part of CSCW: communication technology, distributed systems, user interfaces, man-machine-interaction, artificial intelligence and even sociology or organizational theory [Ellis91].

While CSCW is the name for the working area, the term *groupware* stands for the software solutions. This term usually refers to a huge class of computer systems which

[1] The term CSCW was firstly used with this meaning in the year 1984 by Irene Greif (Massachusetts Inst. of Technology) and Paul Cashman (Digital Equipment Corporation) to describe the scope of an interdisciplinary workshop [Baeck93, Greif88].

cannot be strictly distinguished from other classes of computer systems. Ellis et. al. define groupware in the following way:

**Definition 1.1 (Groupware)** *"[Groupware are] computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment" [Ellis91]*

Generally groupware are applications that (a) interact with different persons and (b) couple these persons [Dewan94]. For getting an overview of the variety of systems that emerged under the label groupware, the following taxonomy may be useful. It classifies groupware in several categories based on application-level functionality [Ellis91]:

**Message Systems**: These systems represent the largest class of cooperative systems. They all can be seen as descendants of the early electronic mail programs. Since the proliferation of such systems has led to the 'information overload' phenomenon [Hiltz85] recent systems often provide help for users to structure, filter and pre-process incoming messages (one example is INFORMATION LENS [Malon87]).

**Multiuser Editors**: Often members of a team have to work on data concurrently. Multiuser editors provide help for exchanging data and notifications and for avoiding or resolving conflicts emerging from concurrently accessing the same data. The editors provide the means of writing e.g. documents or source code without the necessity of manually dividing the manipulated object into separate parts, distributing it to the involved persons and reassembling it after completion of individual work.

**Group Decision Support Systems and Electronic Meeting Rooms**: The aim of these tools is to improve the productivity of decision-making meetings, either by speeding up the decision-making process or by improving the quality of the resulting decisions. Often GDSSs are implemented as electronic meeting rooms. One example is the CATEAM ROOM and the GROUPSYSTEMS software of the University of Hohenheim in Germany [Ferwa91].

**Computer Conferencing**: The computer serves as a communications medium in a variety of ways. One way is asynchronous or real-time (synchronous) conferencing. Systems supporting that task share many concepts with message systems they are, however, significantly different in that they are mostly centralized and structure is imposed in terms of how messages are grouped. A well known example for asynchronous systems is USENET NEWS.

**Coordination Systems**: Coordination systems address the problem of "integration and harmonious adjustment of individual work efforts toward the accomplishment of a larger goal"[Singh89]. Examples for coordination systems are electronic circulation folders [Karbe90] or workflow management tools.

As overlaps exist in these categories it is often not possible to say that one real system exclusively belongs to exactly one category. Hence one has to give a list of categories or just categorize by the primary emphasis and intent of an application.

## 1.2 Document processing

Today document processing is by far the primary area in which computers are used. On the other hand, most people who have to write larger texts use computers to support that task. In a survey Dorner discovered that in the year 1992 74% of a large number of professional writers already used computers for writing and another 11% were considering buying one [Dorne92].

Since tasks nowadays have become more and more complex, the size and amount of textual documents have drastically increased. The critical point at which some documents become too large for a single person to handle has long been reached. As a consequence, the creation of these documents is only manageable by teams. Because of the globalization of organizations the members of these teams are often distributed over many locations.

It is therefore a primary requirement to have tools supporting multiple distributed users working on the same document. Most of the time these groups will work together asynchronously, but synchronous interaction also has to be supported. This support has to be provided for different media types because most documents today consist of more than text [Olsen88]. The rapid development of communications technology makes the provision if that support easier and more economic even for non-academic teams. Specific editing applications supporting the task of collaboratively writing a document are needed [Beck93b, Dillo93, Neuwi94].

As an example for the use of a distributed collaborative editing environment consider the following scenario:

> *The authors A, B, C and D are working on the design documents of a larger project. The documents are consisting of text and graphic parts and are structured hierarchically. Most of the time the authors are working on different parts of the documents on their own. In this phase it is nevertheless crucial for them to know about the progress and changes on the other documents. Some documents (e.g. the monthly reports) are edited by more than one of the authors concurrently.*
>
> *Author A starts the editing environment. He gets an overview of all project documents and their structure. In the overview he navigates to the document part he is currently working on (automatic support by a profile that holds the last working position). By clicking the structure node representing the document or a subtree of a document structure an editor application is started that allows editing the document contents (and the structure of the subtree).*

*Author B is back from leave. She first looks into the structure overview to see which document parts have been changed in the last few weeks and where the main spots of activity are now (history and a display of areas with much activity). Next she visits the documents she is collaborating on and requests a detailed change history. While studying the history some questions that have to be discussed with the co-authors appear. B mentions in the session display that only A is currently working on the documents. That is why B sends the question to D per e-mail. The mailer can be called directly from the editing application. Then B starts a desktop video conference with A. While the video conference is running there is the possibility of exporting local windows to the conference for establishing WYSIWIS*[2].

*C is traveling to clients at the moment. The evening before he left, he connected his laptop to the enterprise network, loaded some documents and told the system before disconnecting that some of the documents should be cached locally. Hence C can work with the local copies while traveling or while staying at the client's. When he reconnects his laptop to the enterprise network all his changes are automatically distributed and the changes applied by other authors are loaded to his laptop. If conflicts are detected, two parallel versions of the affected document parts exist until the conflict is solved by one of the authors.*

## 1.3 Related Work

Many tools have already been proposed to support collaborative writing. We have to distinguish between tools mainly supporting synchronous work and those supporting asynchronous work.[3]

In the area of support for asynchronous collaborative editing we first have to mention tools providing a shared repository where document parts can be accessed and locked separately (e.g. distributed file systems like CODA [Satya90a] or code repositories like CVS [Berli90]). These tools are used to provide the document files for standard editors. Another approach is to build special editing applications tailored to the document production process upon the distributed file systems or repositories (e.g. CES [Greif86], QUILT [Fish88, Lelan88], PREP [Neuwi92, Neuwi90], GROUPWRITER [Malco91], SHARED BOOKS [Lewis88], MESSIE [Sasse93] or DUPLEX [Pacul94]).

Basic support for synchronous work is provided by application sharing systems like SHARED X [Rodde91] or XSHARE [Glick92]. These systems allow the sharing of

[2] WYSIWIS = What You See Is What I See

[3] The difference between synchronous and asynchronous work is not exactly that the one is done at the same time, the other at different times. It is more a question of when synchronization between different flows of work is done [Douri95b].

standard applications between workstations. The application window is identically displayed on multiple workstation screens. Input is accepted from one of the workstations. In most systems the workstation that can provide input is switched by floor passing. The main disadvantage of these systems is that everybody has the same display. Only one author can work, the others have to observe his actions.

The next step in support for editing are tools explicitly supporting synchronous editing of documents. Examples are group editors like GROVE [Ellis89, Ellis90], SASE and SASSE [Baeck93], CAVEDRAW [Lu91], GROUPDESIGN [Beaud92] and GROUPDRAW [Green92]. Most of these tools are limited to LAN environments. If wide area network support is provided this is done by pessimistic locking protocols and by some form of central control or central storage (e.g. MACE [Newma91]). Hence, it is not possible to access the document if the network is down.

According to Beck, who has studied co-authoring in academia, these tools (especially the synchronous ones) are not used by writing teams [Beck93b]. As the main reason for this many surveys (e.g. [Grudi90, Tatar91]) identify incorrect assumptions about human cooperation that have become enshrined in the design. This means that the problem is a missing or poor understanding of the way in which groups collaborate. Beaudouin states in [Beaud92] that in many applications technology has been used for the sake of technology and not to satisfy the demands of the users. Another huge problem is that the existing tools often attempted to change the way in which people interacted in their work environment [Grudi88].

## 1.4 Multi-user editor requirements

As a consequence of these experiences we must first investigate what people want to have before designing a (new) collaborative editor. This investigation should neither be restricted to academic writing groups nor to any other single group using computers to support their writing process. For my investigation I started with determining what cooperation/collaboration is. Then I used the result to specify what collaborative creation of documents means. Rather than starting a new case study I investigated already existing reports of numerous ethnological field studies, laboratory studies and theoretical abstracts on collaborative writing to get a comprehensive summary (e.g. [Beck93a, Beck93b, Dillo93, Murra94] and [Sharp93]). Some results of that investigation will be published in [Koch96], a more detailed description will appear in a later report.

If we summarize the results we end up finding one important point: There must not be any constraints to the work of an author. More precisely one can say that any collaborative writing software must support the writer's normal working practices [Sharp88]. A group writing tool should allow each individual author to plan and write in whatever style and format they feel most suited to [McAlp94]. This demand can be summarized by the following statement:

*Authors working with a group editor want to have at least the functionality and accessibility they have working with their single user editor.*

First a group editor has to be usable for the individual user. The authors need to:

- be able to read *and* update (write) any displayed text
  - no technical access restrictions for group members (use of social protocols instead; 'weak' locks and access modes to support social protocols)
  - the possibility to have private areas, adjustable granularity and chooseable time for making updates public
- get immediate answers to their actions (low response time)
- be able to use their favorite user interface or at least to be able to largely customize the standard interface to their needs

These are the most important requirements. Additionally there are less important requirements concerning support for 'cooperation'. First there should be the possibility for authors to communicate directly. Beyond direct communication, we need to consider the notion of awareness. Peripheral awareness of other people plays a vital and subtle role in cooperative work, enabling ad-hoc interaction, promoting knowledge of the state of activity, and supporting important behaviors such as monitoring and overseeing. The collaborative editor application should automatically and continuously provide awareness of the presence and actions of other users. Awareness of action involves showing what users are accessing and also what kind of access is being made (thus, reading might be distinguished from deletion).

The notion of awareness should also apply to past action. Thus, the user not only should be made aware of what changes have taken place in a database since last access, but one should also be told who was responsible for these changes.

The environment has to support this awareness by broadcasting notifications about the activities of the users. The following aspects have to be satisfied:

- history information about updates to the document (together with the differences between document versions),
- on-line information about co-authors and about their updates,
- communication among co-authors (synchronous, asynchronous, 1:1, 1:n),
- interfaces to allow integration of external tools for information.

Finally we still have some demands concerning the medium types, the structure of the editable documents and the integrability of standard applications. Firstly it should be possible to use standard software as user interfaces. The system should not be limited to one user interface. Secondly different medium types (text, graphic, video, audio, tabulars) and document structures (hierarchical, hypertext) should be supported or the system should at least be able to be easily extended to support new medium and structure types.

## 1.5 Project IRIS

It seems dear that there can be no solution that meets all these requirements with one single tool. Hence it is important to construct a tool for supporting collaborative document preparation as an integrated editing environment. To build such a collaborative multi-user multimedia editing environment is the primary goal of the IRIS[4] project.

IRIS should allow several authors to collaboratively work on documents asynchronously or synchronously. The following requests should be met:

- The collaboration can be asynchronous (at different times or without coupling) and synchronous (at the same time); asynchronous work or loosely coupled synchronous work is expected to be the most frequent type of work, nevertheless synchronous collaboration should be supported.
- The cooperating authors can be working at different locations that are connected in a wide area network or at locations that are not connected to the net for some time (autonomous workstations); if communication links fail, the authors must not be restrained.

To support the cooperation of several co-authors and to support especially the building of *group awareness*, communication through the common document should be enabled. Therefore changes of the document have to be displayed at all sites as fast as possible and information about current and former actions should be available. The following types of information should be available:

- history of changes (display with the possibility to filter and sort the information by different criteria)
- notifications about new propagated actions (filtering possible)
- list of authors working at the moment and of the regions they are working in
- information about all actions of another author (WYSIWIS)
- more information about other authors (name, picture, other activities, other projects, schedule)

Some of the information should only be provided on request (e.g. information for establishing WYSIWIS, additional information about co-authors). One reason for this is the protection of privacy.

In addition to the communication through the document direct communication should be possible. All possibilities should be supplied: synchronous 1:1 (talk, video conference), synchronous n:m (irc, video conference), asynchronous 1:1 (email), asynchronous n:m (news).

[4]In Greek mythology, Iris is a female messenger of the gods who reliably delivers messages from Zeus to human beings, and vice versa. This role of mediator in human-computer-human interaction together with high reliability is also demanded on our group editor.

## 1.6 History of the project

The efforts to build a multi-user editor have their roots in the year 1992. Then Dr. Uwe Borghoff (now Rank Xerox Research, France) was heading the team. The first goal was to examine whether or not pessimistic concurrency control and replica control algorithms that are in use in the area of distributed file systems may be adapted for a multi-user editor and CSCW applications in general. The result of the work was a prototypical editor application for editing linear text documents (user interface similar to VI). The text was stored fully replicated. Access to the document was synchronized at the granularity of paragraphs by a pessimistic locking- and voting-protocol. [Borgh93a, Borgh95, Koch92, Makio93]

On learning that it is important for concurrent access to partition the document we next examined the structuring of documents. Two aspects were of interest: structure to determine what parts of document should be locked on access and structure to support navigating in larger documents or collections of documents [Borgh93a, Borgh95, Borgh93b, Borgh93c]. In 1993 a graphical structure editor was implemented [Bauer93, Gesin93]. The main goal of the implementation was to find an accurate way of displaying and manipulating hierarchical document structures and to experiment with the combination of the structure editor with content editors.

After these first works our goal now is to build a complete collaborative editor environment for structured multimedia documents according to the requirements presented in Section 1.4. It should be possible to extend the environment by new media types, by new document structures and by different user interfaces. Therefore a new architecture was designed in the year 1994 and we began rebuilding the components of IRIS according to the new architecture.

## 1.7 Outline of this report

The goal of this report is to present the architecture, the current status and the goals of our multi-user editing environment IRIS. The report is organized as follows: In Chapter 2 we discuss the architecture of the system. There we identify a common storage and information layer as basic component of an editing environment. The basic concepts for building this basic storage component are discussed in Chapter 3. Here we also present the current status in implementing the layer. The following chapter presents the concepts and the current status of the user interface layer. Then some implementation aspects are highlighted (Chapter 5). Finally, in Chapter 6, some conclusions are drawn and our plans for the future are summarized.

# Chapter 2

# IRIS basic model

According to the requirements analysis mentioned in Section 1.4 the areas in which a tool supporting the collaborative writing of documents should provide support are:

- *data handling*:
  storing, retrieving, exchanging and versioning of the document data

- *information handling, indirect communication*:
  generating and distributing information needed to achieve group awareness; sources of information are actions carried out by the users and state changes detected by the distributed system

- *direct communication*:
  communication among the authors (synchronous, asynchronous, 1:1 and 1:n), WYSIWIS

- other means of working together not directly related to the document (e.g. time planers, workflow management, group decision support)

The editor tool and the user interface should be very flexible and adaptive and it should be possible to integrate external tools into the environment. A collaborative editing environment should be prepared for usage in heterogeneous environments that are distributed in wide area networks. Mobile workstations and disconnected work should be considered. The implications of this technical environment are discussed in more detail in Chapter 3.

In an article on the design of interactive systems, Dourish writes that the development of groupware and, more general the design of any interactive system "must not only be concerned with the traditional issues of system design, but also with the issues of providing a system that is amenable to evolution and adaption"[Douri95a]. Dourish focuses three particular aspects of this problem:

- open infrastructure

- dynamic and reactive systems

- adaptive and evolving systems

That is exactly what we identified in the requirements analysis as demands on a collaborative editing environment. In this chapter we present the document model and the basic architecture of our editing environment and finally focus on some integration aspects. It is especially discussed how the problems mentioned by Dourish, mainly the aspects of flexibility and integrability are tackled.

## 2.1 Documents

In a document editor the basic component are *documents*. Within IRIS a document is considered as a set of typed *objects* (text, graphic, video) with relations between them. The relationships can be seen as a *logical document structure*. In a document such as the present report we have text and postscript pictures that are organized in chapters, sections, subsections, figures, footnotes and bibliographic references.

The editing environment should be extensible for different and new medium types. Existing applications should not have to be changed if a new medium type is integrated. Additionally the document structure should be extensible or changeable. All this is achieved by strictly separating the document structure from the document content. A document consists of a structure that includes links to separate content objects (see Figure 2.1).

Examples of documents are:

- text documents consisting of several paragraphs (text objects) in a linear order
- text documents with paragraph objects ordered in a hierarchical document structure (e.g. chapter, section, subsection)
- text objects in any structure with references to bibliographic information or general annotation objects
- multimedia documents including different medium objects and a structure that includes support for 'displaying' objects parallel (in general: synchronization relations)
- hypertexts

The manner in which a document has to be partitioned in content objects is not regulated. The partition is dynamic and driven by the authors. The only restriction is for data of different media types to be stored in different objects. Hence, one might partition a document per paragraph, per chapter or partly per paragraph and partly in another granularity. This dynamic partition can be exploited for avoiding access conflicts. A user defined granularity of access can minimize the likelihood of conflicts.

In the first phase of the project we support only hierarchical document structures and annotation links. Figure 2.1 shows a part of the structure of the IRIS project

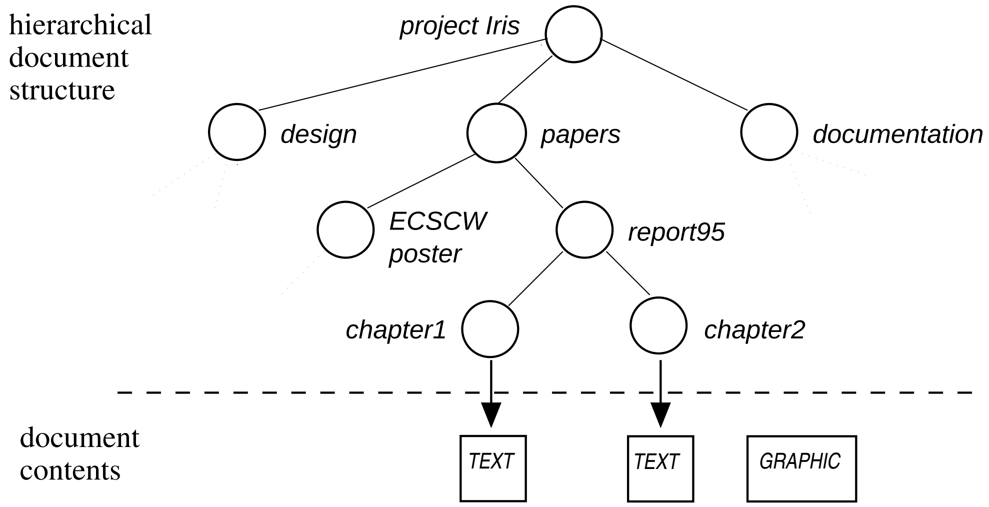


Figure 2.1: Example IRIS document

document. Here you can see that the hierarchical structure can be used as document directory containing several documents in subtrees. The leaves of the structure tree store references to the content objects. Possible content object medium types are text, graphic and binary (see Chapter 3). Nevertheless the architecture is already prepared for the addition of other structure or medium types.

The structure of a document is stored in an object in the same way as the content. There may be different applications in the IRIS environment for editing subtrees or subgraphs of the structures. Because of the separation of structure and content it is also possible to use different applications that structure the same contents in a hierarchical and in a hypermedia structure. Both applications can work collaboratively on the same shared content objects.

Our next step in extending the document structure will be to add synchronization information to the structure nodes in order to support real multimedia documents (see the Amsterdam Hypermedia Model [Hardm93] for more information on such structures for hypermedia documents).

## 2.2 Architecture of IRIS

Since we have to provide different user interfaces it is advisable to split the editor applications into two parts: the *access layer* and the *user interface layer*. A common access layer can provide reliable storage mechanisms and mechanisms for generating and distributing information for group awareness to different user interfaces.

- *access layer*: This layer is responsible for managing the data objects and the structure objects of the documents. It is the core service for storing and accessing documents with high availability in wide area network environments. The objects are stored in replicated form. The main task of a local instance of the access layer is to handle access to the local replica, communicate with remote instances of the access layer in order to synchronize access on the replicas and distribute information.

- *user interface layer*: On top of the access layer, user interfaces are built for user access to the documents. The user interface layer is a family of specialized editors rather than a single editor application. There may be different applications at the same time for editing the same object type.

The access and user interface layers are implemented as separate processes that communicate. An access layer process can serve several local user interface processes via an interface stub (C functions, C++ objects or a Tcl/Tk interface). The following groups of functions are available to the applications (see Chapter 3 for information on the implementation):

- *access*
  `get(objectID):access-object`
  `put(access-object):result`

- *management*
  `create():access-object`
  `delete(objectID):result`

- *synchronization*
  `set_intention(objectID,suppl):result`
  `unset_intention(objectID,suppl):result`

- *notification*
  `send_msg(notification):result`
  `receive_msg():notification`
  `mask_msg(notification-mask):result`

The main task of the first two functions is the transfer of data between the application and the access layer. This is done with the help of 'access objects'. An access object provides functions and data to access a portion of a specific media type. The implementation of an access object is media dependent (e.g. a text access object can hold a copy of the text and functions for reading and writing this text, a video access object can hold a reference to a video server and the protocols for reading and modifying the video sequence). Hence, depending on the media type, an invocation of `get` may either retrieve the actual data or simply provide a handle for data retrieval. For the implementation it is still important to realize that the functions `get` and `put` underly consistency constraints. It depends on the chosen concurrency handling protocol how this problem is solved.

For synchronizing concurrent operations on objects, the users need information about the other users' intentions of changes. The function `announce_intention` may be used for announcing an intention to write an object or a part of the object. If a pessimistic concurrency handling protocol is used `announce_intention` can be implemented by locking the object or parts of the object.

Notifications are usually generated and distributed by the access layer. A notification always consists of an object identifier, a type, such as 'locked for update', and additional data depending on the medium type of the object and the notification's type. An application may instruct the local access specialist to subscribe to specific notification types using the function `mask_asyncmsg`. The function `send_asyncmsg` can be used by the application to send its own notifications or to request information which will also be delivered in the form of notifications.

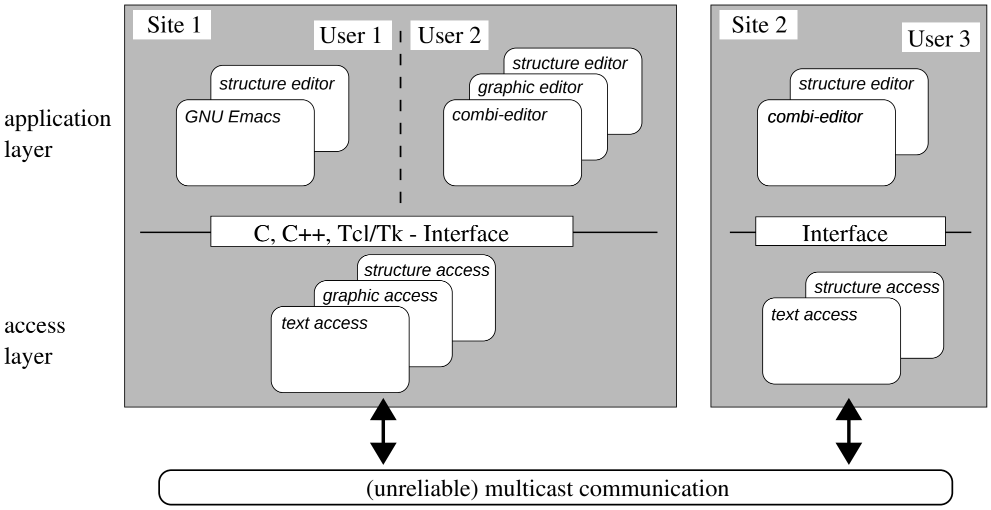


Figure 2.2: Structure of access and application layers

The access layer should be able to handle objects of different media types. To do that in an efficient and extensible way it is further separated into different parts (the so called medium-specific access specialists) [Teege94]. This makes it possible to handle the concurrency control or replication policy differently for different medium types. The resulting overall structure of the IRIS system is depicted in Figure 2.2.

Since we want to have a slight transition between asynchronous and synchronous work the distributed data storage service has to be integrated in the applications as far as possible. It will not suffice to load the document from the service at the beginning of an editing session and to store it back at the end of the session. The user interface layer not only makes use of the access layer for loading and saving the document but for all document access.

## 2.3 Integration aspects

The main idea with the architecture is not to build a tool that includes support for all areas mentioned at the beginning of this chapter. We propose an integrated environment with core applications for concurrently editing documents and with different external application for other collaboration and communication issues. The first step in constructing such an environment was the separation of user interface applications and access layer. With that separation it is easy to provide different user interface applications working together on one single document.

The second step is to separate functionality that is not concerned with the core editing service from the user interface applications. Support for tightly coupled co-operation (video conferences, window sharing, telepointers), direct communication (e-mail, news) or workflow management, brainstorming etc. is not integrated in the editor applications. There instead are interfaces for calling existing external applications.

In the context of integration the following functionality is needed:

- possibility to call other applications (especially applications to communicate with other co-authors) from the editor with transferring parameters to these applications,
- standard syntax for common identifiers (e.g. user names),
- possibility to display information provided by other applications,
- possibility to configure the way information is obtained: To show background information about another user one could contact the users editor (if running) and receive information from that process, to execute the 'finger' command or to contact a distributed calendar tool.

# Chapter 3

# IRIS access layer

The main component of the IRIS architecture is the *access layer*. It is responsible for managing the data objects of the document. If one wants to work efficiently in a wide area network environment the data must be replicated on the various sites. The replication, however, raises several questions: (a) How to efficiently replicate data? (b) How to maintain consistency? (c) What to do when several people modify the same portion of the document and the server later detects a conflict between the modifications? This chapter will first outline the basic principles of our access layer, where answers to these questions can be found. We will discuss the replication scheme and the concurrency control used in the approach. In the remaining parts of the chapter we present the components of the access layer that have up to now been implemented.

## 3.1 Data storage — basics

As already mentioned, the access layer has to handle objects of different media types. Every object can be identified by a unique identifier. We choose the following syntax for object identifiers[1]:

```
irisobj:mediatype:id
```

Identifiers begin with the fixed string '`irisobj`' followed by a type name and a string that is globally unique within the type class. For the type name we choose the primary type name of the object's MIME content-type (see RFC 1521). At present we use the following types: `text`, `x-graphic`, `x-struct`).

An object consists of the content data and additional attributes. The content data can be unstructured (e.g. text), structured (e.g. graphic, document structure) or a reference to external data (e.g. video?). The attributes are a list of '`aname=value`' pairs. They can be used for storing application specific information or collaboration specific

[1] The object identifier syntax was inspired by the URI (Universal Resource Identifier) syntax proposed in RFC 1630. It might be possible to use other URI's later as content object references in parallel with our objects.

information. We have, at this stage, defined the following attributes: user_visible_name, creator, creation_date, mimetype, history, application, references. Additionally, any other attributes can be added by the applications.

Technically the data access layer consists of several local specialist processes. The functionality of these specialists can be used by local applications via an RPC-like interface stub. The specialists are started on demand and stopped when they are no longer needed (that is when no local application has objects opened that they replicate). The specialists are persistent and survive machine breakdowns.

According to the function groups presented in Section 2.2 objects can be accessed by the applications through the following interface[2]:

| | |
|---|---|
| `Prefetch` | `GetSessionInfo` |
| `Open` | `GetAttr` |
| `Close` | `SetAttr` |
| `Create` | `ExistsMsg` |
| `Delete` | `GetMsg` |
| `Get` | `PutMsg` |
| `Put` | |
| `SetIntention` | |
| `UnsetIntention` | |

Since we aspire at availability and low response time a local access layer instance replicates all objects that are accessed (opened) by local applications until they are no longer needed by these applications. Not all possible objects have to be replicated on all sites. The functions `Open` and `Prefetch` are designated for expressing the interest of an application for a special object, the function `Close` is designated for telling the access layer that an application no longer needs a specified object.

The interest of an application in an object is announced by the functions `Open` or `Prefetch`. With the function `Close` an operation states that it is no longer interested in an object and that the replica can be deleted. `Get` and `Put` are the functions for reading the objects content data and for rewriting the content data. If an objects has internal structure (e.g. types struct, graphic) there are other functions for reading and updating the internal subobjects instead of these global functions.

Because of the replication and consistency constraints `Get` and `Put` cannot simply work on the local data. Their implementation has to follow a replication control protocol. The basic strategies of how this is handled internally are described in the next sections. More information can be found in the theses [Blume95, Fries95, Sewal95].

[2] The exact syntax of these functions depends on the implementation language: In the C interface `Open` returns an identifier that is used to access the object with the other functions, in the C++ interface `Open` returns an object or is implicitly done in the constructor of an object and the other functions are implemented as methods of this object.

## 3.2 Replication control

The basic component, the data access and information service, has to be implemented especially for IRIS. In this chapter we deal with providing full editing functionality (allowing every user to read and write each part of the shared document at any time). In this context it is important to consider the handling of concurrent access and the managing of how to join and leave sessions.

A number of criteria must be met when data is distributed across a network [Douri95b]:

- availability
- transparency
- consistency
- responsiveness

We have already mentioned some of these aspects when dealing with the requirements of a multi-user editor tool in Section 1.4.

The first decision to be made is whether the shared document should be stored on a central server or replicated on the workstations. Since access to document parts becomes impossible when the server is inaccessible the replication approach must be our choice. For the same reason partial replication cannot be used.

Hence, every co-author has to get his own copy (replica) of the objects he is working on to support full availability and responsiveness in the context of site and communication failures.

Co-authors working synchronously implies concurrent access. Concurrent reading is not a problem with a replicated document but concurrent writing is. If two users attempt to modify the same part of the document simultaneously, the outcome may be unpredictable. Hence, we have to come up with some method for handling concurrent access to the replicas.

Traditional methods for concurrency control are serialization or privileged access through locking. These pessimistic methods guarantee that no conflicts occur. Their main task is inconsistency avoidance. To establish this it is sometimes unavoidable to restrict access. Especially if the communication network is partitioned, access will only be possible in one of the partitions. Embedding these formal turn management policies and protocols in the interface is too restrictive. Hence, pessimistic concurrency control cannot be used in our model. What we need is an optimistic multiple-reader multiple-writer concurrency control that tolerates network partitions.

### 3.2.1 Optimistic replication control

Optimistic methods do not bother avoiding conflicting updates. All they guarantee is to detect conflicts after they have occurred. An update is applied to the local replica and then distributed to be applied to the other replicas. Hence low response time at the local application is possible. If a conflict between two updates occurs it has to be detected. Machinery is needed in the system for detecting conflicts, for automatic resolution when possible, and for confining damage and preserving evidence for manual repair. The group editor should mainly provide feedback to support the users in their own policies. It should notify about conflicts and support the resolution of conflicts by presenting the conflicting updates in a profitable way.

For implementing this optimistic concurrency control we have to assign a unique identifier (timestamp and site identifier for example) to every update. Then we have to maintain a history of updates together with the document replicas. These histories will be exchanged temporarily (or after an update occurred) in multicast or general epidemic propagation [Demer88]. By comparing the remote history with the local one, an instance of the editor can determine if there are outstanding updates and whether these updates conflict with other updates. This concept was first used in software repositories (e.g. RCS [Tichy82] and CVS [Berli90]).

If different parallel versions of an object exist in the local storage the editor has the problem of determining which version to display. We define the branch of the version tree that was first displayed as the current branch on the local machine (see Figures 3.1 and 3.2). This is important because the user should no be irritated by an automatic version switch. Nevertheless there has to be additional information about the existence of other parallel versions and the possibility to switch explicitly between versions must exist.

If conflicts occur they should be resolved in due time. This normally cannot be handled by the storage service. Here it will be the best to present the conflict to the co-authors. But there must not be the constraint of having to resolve conflicts immediately they are detected. It must be possible to continue work with conflicting versions.

As stated in the last section it should be possible to work on a part of the document privately. This means that updates to this part should not be presented to co-authors without the author's consent. This can be achieved by not distributing private updates. To avoid conflicting updates other authors should be informed that a newer version exists which is still private. In this way private versions can be used as a substitute for locks to show intentions on parts of the text. If another author decides to do an update even though a private version exists he/she should be allowed to do so.

In order to implement that information we have to at least distribute information that an edit is occurring and the location of the edit. Since the local access layer should be allowed to delete the local replica of a object when no application is working with it, we also must distribute the whole update at that time. That is why we decided to mark a private update with a special flag. If an update is marked as private it is distributed but not presented. Other authors see the last non-private version of the object.

Figures 3.1 and 3.2 show an example with three users during and after a network partition: The initial state of the document was 'x'. Then a network failure separated Site 3 from Sites 1 and 2. In every partition an update was carried out (leading to state 'y' in one partition and to state 'z' in the other partition). After that update User 2 changed the ‘y’ to ‘p’ as private update (Figure 3.1). Then the network failure that caused the partition disappeared and the sites exchanged their information. The conflict was recognized (branch in version trees) but the display did not change (Figure 3.2). The authors have to resolve the conflict by merging the different versions later.

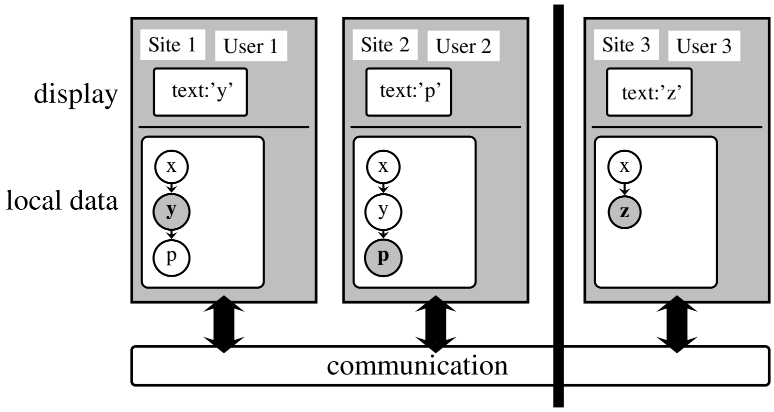


Figure 3.1: Version management in IRIS - during partition

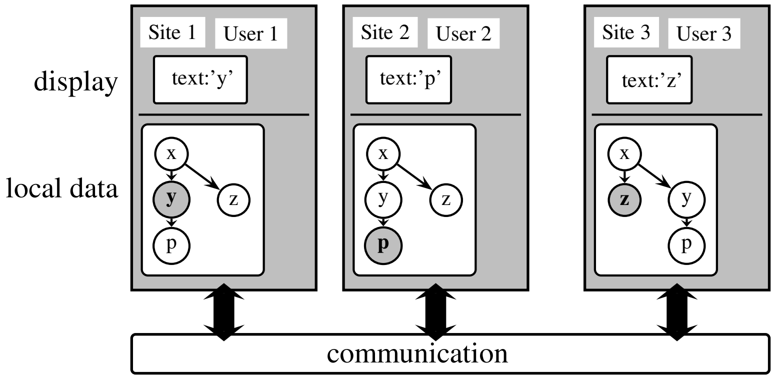


Figure 3.2: Version management in IRIS - after partition

Even though conflicts do not cause the loss of data it is desirable to minimize the likelihood of conflicting updates. As past experience has shown, it is profitable to divide the document into several parts (objects) and to update and read per object. We have to decide what we want to use as the granularity of updates. One possibility for dividing the document is to break it down based on some document model. This can once again be too restrictive for the co-authors. The best solution will be a dynamic granularity [Pacul94]. The granularity should be determined by the user. The user determined dynamic granularity can be extended by services of the user interface which automatically determine and set the granularity, based on the operations of the user. (See Section 4 for information on the solution in our prototype IRIS.)

With this framework we are able to read and write objects after having joined a session. The problem of how to present conflicts and who should be allowed to resolve conflicts will not be discussed in detail here.

### 3.2.2 Joining and leaving sessions

What must still be discussed is how a computer application that a co-author starts can enter a session and how it can leave a session[3]. When leaving a session it is important for nothing which the user has changed locally to be lost. If the local workstation is separated from the others by a network failure, the local data must not be deleted. But the user should also be allowed to leave the session whenever he wishes to do so.

The system therefore must hold the local replica even if it is no longer needed by an editor process until it becomes dear that all the information has been transferred to another editor or to a persistent repository. I call this concept 'passive replicas'. They are not passive in the normal meaning, but they are no longer changed by the users. The sole aim is to transfer their data to a reachable interested process.

In this context we also can very easily solve the problem of joining a session: The new editor just has to try to load the text from a passive copy or from a repository (which can be seen as special form of a passive copy). This can be done by sending a multicast to all applications which are interested in information about the document.

The requirement that a new user should be able to connect to the session even in the presence of network failures can be addressed by trying to guarantee that there is always a passive copy on a highly available host. This can be achieved by adding constraints to the transfer process of exiting editor processes (passive copies). For instance the editor process might only be allowed to transfer the data to another storage service in the local LAN-segment, rather than to any other active editor process. If the other hosts in the LAN have the same constraints and if we suppose that hosts in the same LAN are always available, the copy would be available when needed next time. Other applications of these constraints to the transfer process are currently being investigated (security, reliability issues etc.).

With persistent passive copies it is also possible to prepare a mobile workstation with local copies of document parts before the workstation is disconnected from the communication network. This technique is also mentioned as 'pre-fetching and hoarding' within the cache of the CODA distributed file system [Kistl92, Satya90b]. In CODA copies of critical files can be explicitly stored on the local workstation. When a disconnection occurs the local process logs all update activities and otherwise emulates server behavior. Upon reconnection the local updates are reintegrated by sending the log to the server for replay.

[3] In this context the term *session* describes the group of all co-authors currently working with an object. Joining a session means starting to work with the object, leaving a session means stopping to work on the object (for some time).

### 3.2.3 Information

The information that should be provided to enable group awareness has not yet been mentioned due to its irrelevancy to the access layer. The layer should just distribute all available information without selectivity. According to Section 1.4 this includes information about co-authors and updates. Since information about updates is already distributed automatically by propagating the updates we need only take care of the information about co-authors. This is implemented by distributing a list with the names of all local authors with every update and collecting the received lists. Detailed information about the authors is sent on demand. Another example for information that is distributed on demand is the information about keystrokes needed to establish WYSIWIS.

At present the display of group information is implemented in IRIS by doing just that. All available information is displayed. We plan, however, to enhance this further.

For the user interface it is important to know that not all pieces of information can be provided at any time with equal quality. The amount and quality of the information that can be provided depends on the status of the infrastructure. The application has to inform the user about the implications of the current state. For example in the case of network failures or mobile workstations one cannot determine exactly who is currently editing the document. So there should be a way to notify the user about the quality of displayed information. Possible sources of error are process failures, site failures (announced or unannounced) and network failures or the disconnection of mobile workstations. Some of these failures can be identified, others cannot. The state of the network can be viewed as an additional information to be displayed.

This possible change in quality of information can be expressed (and presented to the user) in different qualities of an information service. There may be automatic degradation and restoration of the service that can be notified to the user. With this abstraction there is the additional possibility for the user to choose a specific service quality so the system does not have to try to provide more.

Here are two examples of possible degradation of service:

- display of updates
  - 'immediate' display of all local and remote updates (considering the communication delays the local version is up-to-date)
  - some co-authors are reachable and their updates will be displayed but it is not dear whether all updates in the system will be displayed immediately
  - display of own updates (maybe of some remote updates but there can be no guarantee of its completeness)
  - display of local updates only
- display of the group composition
  - reliable display of the current composition

- hints about possible group members
- no display

Different substages may be inserted indicating how reliable the presented information is. All information about the infrastructure can be used here. The reachability of the co-authors is an especially important piece of information which should be determined. Possible states are: all co-authors reachable, a defined sub-class of co-authors can be reached, some co-authors that were announced as being unreachable cannot be reached, some co-authors are unreachable without announcement. Announced unreachability may happen with mobile workstations.

We are currently investigating these aspects in the context of the access layer that has to provide the additional information and in the context of the user interface that has to notify the user.

## 3.3 Import/export tools

In addition to the specialist processes the access layer consists of some tools for importing and exporting documents in standard document formats. The import tools use the structure node attributes for storing information that allows the export tools to reproduce the files in an identical form. Up to now we have built tools for LaTeX and SGML. Work on building tools for ODIF and formats of standard text processors (e.g. FRAMEMAKER) is under way.

The possibilities of the tools will now be explained at the example of the SGML-to-IRIS converter. We assume the following (simple) SGML file as input:

```
<!DOCTYPE demo [
<!ELEMENT demo - - (section+)>
<!ELEMENT section - - (title?, paragraph+)>
<!ELEMENT title - O (#PCDATA) >
<!ELEMENT paragraph - O (line+) >
<!ELEMENT line O O (#PCDATA) >
]>
<demo>
<section><title>Import/export tools</title>
<paragraph>
<line>In addition to the specialist process and the editor
<line>interfaces we provide some tools to import ...
<line>and one more line ...
<paragraph>
<line>And this is the last paragraph with a single line
</section>
</demo>
```

The import tool expects two or three arguments: the first argument is the name of the SGML file, the second argument is the identifier of a structure node that should be used as root for the new subtree. If an empty argument is given, a new structure object is created for the import. The last (optional) argument is a list of SGML elements that should not be further divided into different text objects. In our example using '`paragraph`' as third parameter will result in the IRIS tree in as shown Figure 3.3.

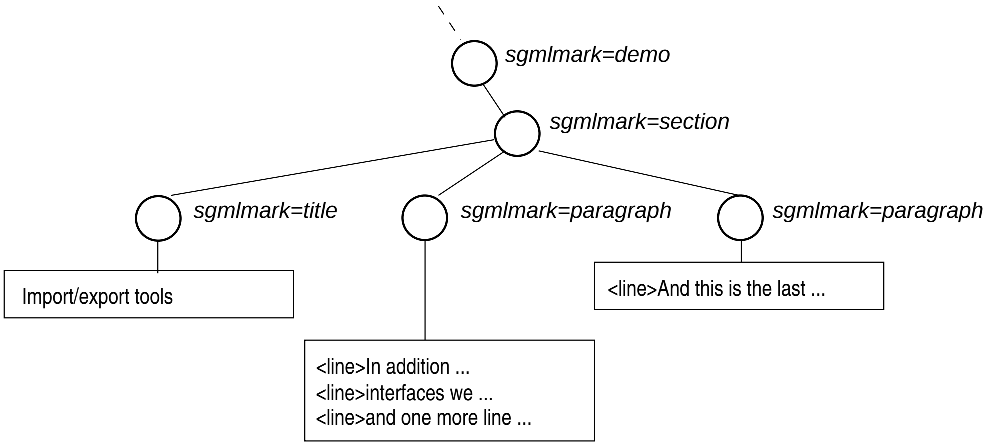


Figure 3.3: Imported SGML document

## 3.4 Annotations

For asynchronous collaboration on shared documents annotations are very important. At present annotations are not explicitly supported by IRIS but we are working on integrating them.

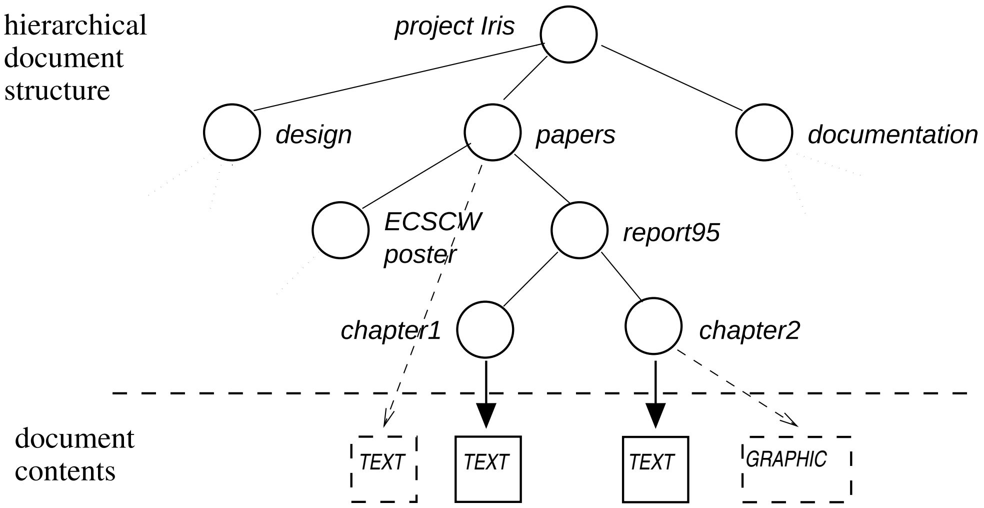


Figure 3.4: Document structure with annotations

The scheme that will be used is the following: annotations are stored as individual medium objects in the access layer. The document structure is extended to hold links from structure nodes to annotation objects. These links are stored as node attributes in the structure object.

Since the structure object will not be changed it will be possible for the current applications to work on the structure with annotations in the same way as on the old structure objects. We can still use the old applications and build new applications supporting annotations simultaneously.

One more problem with annotations is that it should be possible to link annotations not only to whole subtrees or whole objects but to parts of objects. This issue is not tackled now, but can easily be dealt with by concepts of the DEXTER hypertext reference model [Gronb94, Halas94]: in addition to the content every object stores a list of reference points or areas. Every reference has a unique identifier that can be used together with the object identifier for globally specifying the reference. In the example of Figure 3.4 the annotation link from leaf node 'chapter2' may have 'chapter2:12' as anchor. This means that the annotation link will start at reference point '12' in the text object of the leaf. This concept has already been implemented for structure objects: every node in the structure has a unique number that can be used to reference it.

## 3.5 Current status and ongoing work

As already mentioned in the project history (Section 1.6) we implemented access layer components with pessimistic concurrency control in earlier project phases. These components were first changed to provide the interface described in Section 3.1.

These specialists are working with pessimistic concurrency control (dynamic voting protocols and locking) and are using a central repository for loading and storing the document objects. That approach works well enough for testing the first user interfaces and for pre-evaluating the architecture. Synchronously we started implementing the optimistic specialists. That work is finished now for the text specialist and has begun for the structure specialist. See the next section for more details on the additional features of an optimistic access specialist.

At present we have specialists for the following object types:

- document structures (`x-struct`)
- text (`text`)
- graphic (`x-graphic`)

The specialists are implemented as separate processes that can be accessed by interprocess communication. At present client stubs are available as C libraries, C++ objects or Tcl/Tk-objects.

## Text specialist

First we built a text specialist [Borgh95, Kelln95, Koch92]. That is a process that can store text objects (MIME type '`text`'; binary strings without internal structure and a list of attributes). The application interface has exactly the functions listed in Section 3.1. As the specialist stores unstructured data it can also be used for storing binary objects (e.g. complete FRAMEMAKER documents). For distinguishing different formats stored by the text specialist we are using the MIME subtype or the object attributes.

The text specialist delivers notifications to its applications if an object that is opened by the application was (a) updated, (b) locked or unlocked or (c) if a user joined or left the session.

## Structure specialist

The structure specialist [Fries95] is responsible for structure objects (MIME type '`x-struct`'). These objects hold a tree structure consisting of nodes with several attributes. Some of the attributes are used for ordering the nodes in the tree (father, children) or for specifying the content objects at leaf nodes (contentid, contentmimetype). Additionally, there are the same attributes as described for the objects. The free definable attributes may be used for different purposes. One example is given in Section 3.3.

For the applications the structure specialist offers the standard object functions. Since structure objects have an internal structure, the specialist does not offer single `Get` and `Put` functions but a number of functions to access the structure. The following functions can be used to modify the structure and the node attributes:

```
StructCreateNode
StructDeleteNode
StructPutNode
StructGetNode
StructLockNode
StructUnlockNode
StructGetNodeAttr
StructSetNodeAttr
```

For identifying a node there are two sorts of identifiers. First we have temporary local identifiers that are returned by the `Open` function.[4] These identifiers are not globally unique. They may only be used within one session and only at one site. If one wants to have a reference to a node that lasts longer than a session and that can be used globally, one can get such a identifier by the `StructGetPartID` command. The `StructPartID2LocID` command can convert a global identifier to a local one.

[4] In the C++ interface we do not have these local identifiers because of the object oriented approach.

## Graphic specialist

The graphic specialist was designed to support the graphic editor described in Section 4.2. An object of the type '`x-graphic`' consists of graphic subobjects at distinct locations of a bitmap. Possible subobject types are: circle, ellipses, line, square and text. The subobjects have the following attributes: position, size (radius, length, height according to the subobject type), line style, line width, line color and fill color. Graphic subobjects may be grouped hierarchically. The access layer provides functions for creating, deleting, grouping, locking and unlocking subobjects and for updating attributes:

```
GraphicInsertNode
GraphicDeleteNode
GraphicTransformNode
GraphicReadNode
GraphicGetOrderList
GraphicSetLock
GraphicUnLock
GraphicMakeGroup
GraphicReleaseGroup
```

Additionally there are functions for setting and unsetting telepointers, creating and destroying private areas and for retrieving information on other users (see description of the user interface for more details on the application of these functions). Finally we have functions for grouping and ungrouping subobjects to hierarchical object groups. See [Giess94, Schra94] for more details.

## Ongoing work

We are currently working on finishing the optimistic text access specialist. The specialist provides a superset of the pessimistic specialist functionality to applications. Hence, our existing user interfaces are able to work with it.

The following functions were added or extended compared to the pessimistic text specialist:

- `Get`: It is now possible to get a history of updates and to retrieve other object versions other than the current one.

- `Put`: One can determine if the update is private or public. If it is private one might also choose to overwrite the last private update instead of creating a new version.

- `Merge`: This command allows merging two or more conflicting versions of an object (branches in the version tree). One of the versions is updated with the new content of the object and the other versions are marked as 'merged to x'.

- `Lock/Unlock`: These commands now have the names `SetIntention`, `UnsetIntention`. They just set or unset an attribute and distribute the attribute.

- Additional notifications are delivered when conflicts are detected.

In addition to the new functionality the passive copy scheme is implemented as described in Chapter 3. The specialist allows specifying sets of sites where copies have to resist for guaranteeing availability. These sets may be modified by functions of the access layer interface.

For the other specialists (graphic, structure) we have started investigating into how conflicts may be resolved automatically and when the user still has to participate in conflict resolution. We will finish implementing these specialists in 1996.

Finally, we are currently working on a video specialist. This work is carried out by my colleague Tristan Daude (e-mail: `daude@informatik.tu-muenchen.de`).

# Chapter 4

# IRIS user interface layer

On top of the access layer, user interfaces are built for user access to the documents. The user interface layer uses the access layer not only for loading and saving the document but for all document access to enable synchronous collaboration. According to the requirements the user interfaces should satisfy the following aspects:

- provide functionality and look-and-feel of single user applications
- establish group awareness by presenting information on current and past actions and stati
- allow direct communication among the co-authors
- allow the integration of different external applications

To satisfy these needs the user interface layer is a family of specialized editors rather than a single editor application. Each application supports one or more media types. In the simplest case there is a specific application for each medium type. There may be different applications for editing the same object type. (Figure 4.1 shows the architecture of access and user interface layer with the communication relationships among the components.)

Different editor applications exist within IRIS for editing parts of the document. These applications can be divided into the following three classes:

1. collaboration-aware applications that are written especially for the IRIS environment and that can use all the possibilities of the access layer
2. extended standard editor applications that have been updated to use the access layer for all document access; here we do not have full support of all features (e.g. display of group awareness information)
3. unchanged standard editor applications that are called through a shell application that loads the objects from the access layer, writes them to a file, calls the standard application with that file and finally retransmits the contents of the changed file to the access layer

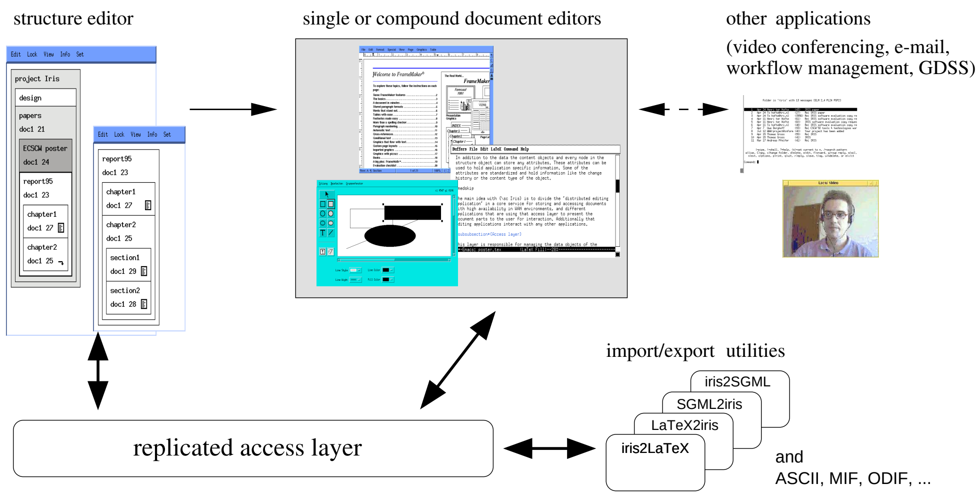


Figure 4.1: User view of the IRIS environment

All these applications are called with an object identifier as parameter. That identifier can specify one content object, a structure object or a subtree in a structure object (by specifying the root node of the subtree).

If an editor application can edit single objects only than it only accepts identifiers of those objects. More advanced interfaces try to linearize subtrees and allow the editing of all linked objects together.

These different applications are currently loosely integrated by a structure editor that allows navigating in the structure and changing the structure and calls the specific editor applications if content objects are to be edited.

The following possibilities are available when starting work with IRIS:

- `iristext irisobj:text:textobjid`
  for editing single text objects

- `iriscombi irisobj:x-struct:struktid:subid`
  for editing subtrees of a document structure

- `irisstruct irisobj:x-struct:struktid`
  for editing complete document structures in the structure editor

To summarize how IRIS may be used, we now present a generic session:

> The user first starts the structure editor with the identifier of a document structure as parameter:
>
> `irisstruct irisobj:x-struct:12988`
>
> After the initialization the structure editor contacts the structure access specialist and retrieves the structure tree of the whole document and the corresponding session information. This information is displayed in the

form of structure views. Within these views the user may obtain a general view of the structure and ongoing work, edit the structure or select a basic or composite logical object for editing.

If a part of the document is selected in the structure view (by selecting a basic or composite logical object), a medium specific editor is started. This editor uses the data from the structure specialist for linearizing the subtree and retrieving the identifiers of the objects referenced by the leaf nodes. It then instructs the corresponding media specialists to access the content objects.

The choice of what application to be started from the structure specialist when selecting a node for editing depends on

(a) the medium type of the node or if the node is the root of a subtree the medium type of the first leaf of the linearized subtree

(b) the application preset[1] for the medium type determined by (a)

(c) the value of the attribute `application` in the content object or in the structure leaf node

## 4.1 Structure editor

A central part of the IRIS environment is the structure editor that can change and display the document structures stored in structure objects.

A structure object consists of a tree of nodes. Every node has different attributes, the leaf nodes additionally have references to content objects. In the IRIS structure editor [Borgh93c, Gesin93, Nebel95] the structure is presented to the user in the form of one or more *structure views*. A structure view depicts one node from the structure in an iconic way. The node may be the logical root, or a basic or composite logical node. The actual appearance is a rectangular space on the screen. Additionally, the icon for an object may display one or more of the following items that are individually and interactively configurable by the user: the value of the user_visible_name attribute of the node, the global node identifier, or the sequence of iconic representations of all subordinate objects in their sequential order provided it is no leaf node. For leaf nodes it is also possible to insert an icon representing the type of the content object the node is refering to. See Figure 4.2 for an example of the editor's display.

The structure editor offers functions for

- *structure browsing*: Upon the start of the editor on a certain structure object, a single structure view for the root node is created automatically. Additionally,

[1] Such presets are stored in the a site or user specific `.irisrc` file.

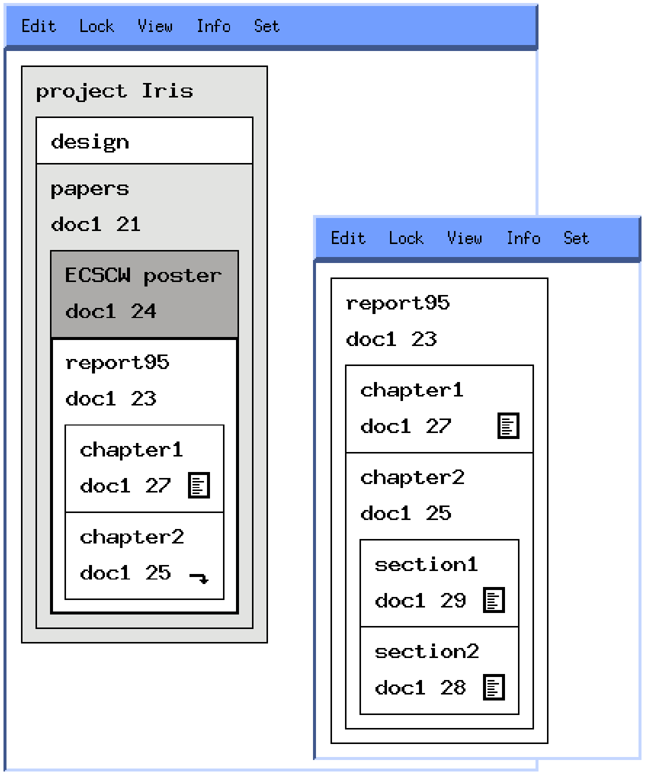


Figure 4.2: Structure editor user interface

another view for the last working position of the user is created. The views do not show any subnodes.

An existing structure view can be configured in the following general way. A depicted node may be selected and the display of its ‘user visible name’, the identificator or the subobjects may be enabled or disabled independently. Thus a structure view may be configured to show some parts in detail while hiding other parts.

Also, new views may be created by selecting a node depicted by an existing view. At any time a structure view may be closed. When the last view of a document is closed, the structure editor is closed.

- *structure editing*: There are five operations which change the structure. All of them may be initiated in the usual way using a structure view, i.e., by selecting nodes in the view. The operations are: *insert node*, *delete node*, *move subtree*, *copy subtree* and *update*. The last operation allows updating attributes, e.g. user visible name, content reference.

  All structure editing operations implicitly lock the involved structure nodes using the access layer. Additionally, it is possible to explicitly lock a node.

- *group awareness information*: The structure views display locked nodes in dark gray and nodes with intention locks (nodes that cannot be locked because child nodes are locked) in light gray. It is possible to configure the view so that the name of the lock holder is displayed in the node.

  The editor can display the object attributes, the node history, a list of the users currently working in the session and more information on the users in extra windows.

- *activating content editors*: The last kind of operation initiated with the structure editor is the starting of content editors. An editor is started by selecting a structure node. The structure editor then determines which editor application should be started and calls it with the identifier of the structure node as parameter. It depends on the type of the selected node, of the medium type of referenced content objects, of the node attribute `application` and of user presets which of these applications is started.

## 4.2 Graphic editor

The IRIS graphic editor [Giess94] is a multi-user object-oriented drawing program with features similar to those of most structured drawing packages. Users of the editor may create objects (such as lines and squares) that can then be manipulated. The purpose of building that tool was building a multi-user graphics editor within the IRIS architecture rather than implementing a full featured graphics editor.

The editor can be started from the structure editor by selecting a leaf node that contains a reference to a graphic object or by directly specifying the identifier of the graphic object in the command line.

The user interface is collaboration aware and presents information on the other collaborators and their actions[2]. Group features are

- *graphic echo*: When an operation (e.g. move object) has started, the object is locked. When the operation is completed by a remote user, not only the result of the operation is displayed on the local screen, but the whole operation is animated. If an object is moved from one position to another, it will not disappear and suddenly reappear at the new position but will move from the start position to the end position at constant speed.

- *localization*: In an additional window a small image of the whole graphic is displayed. There the working areas[3] of the other users can be displayed.

[2]The collaborative functionality is very similar to the functionality of the GROUPDESIGN tool of Karsenty, Tronche and Beaudouin-Lafon of the Univeristé de Paris-Sud [Karse93]

[3]The 'working area' is the area of the graphic that are displayed in the main window.

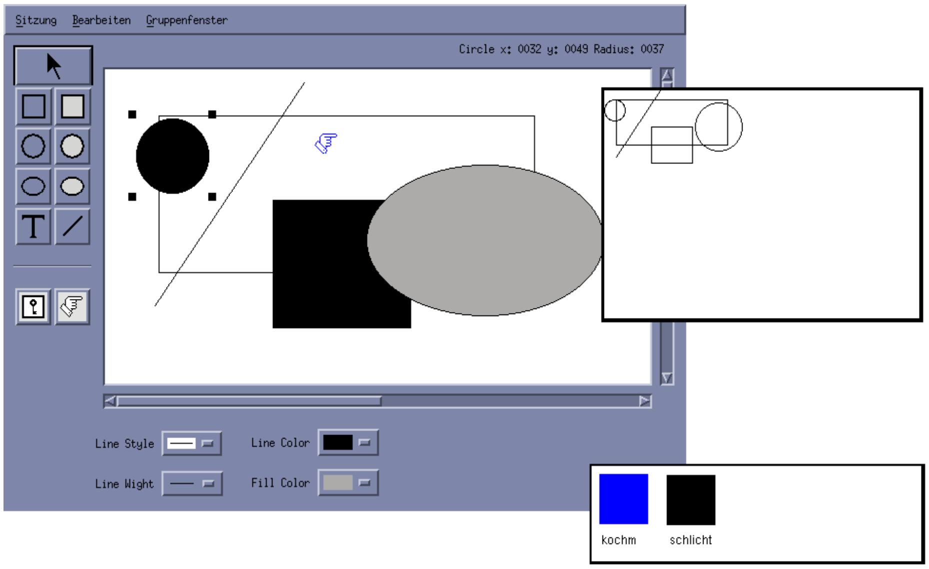


Figure 4.3: Graphic editor user interface

- *private editing*: A user may mark an area of the graphic for private editing. In that case the user's modifications do not appear on the other participant's windows (instead of the area at the other user's user interfaces a is 'curtain' displayed). When satisfied with the results the user can unlock the area and the 'curtain' disappears.

Most features use color to identify each user. The name of the users and their associated colors appear in a separate window[4].

## 4.3 Combi editor

In the structure editor it is possible to select a subtree of the structure for editing. An integrated editing application (a so called *combi editor*) is launched for the whole subtree. It linearizes the content objects and allows the user to edit them like one single object (see Figure 4.4 for an example). The combi editors primarily display the content objects, but they may also display some information about the structure to allow restricted structure editing functionality (insert, delete, merge and partitioning of objects). Hence, these applications combine limited structure editing functionality with the ability to edit and/or display contents of different medium types. A similar approach has already been used in some single user editors (QUILL [Chamb90] and EZ (ANDREW TOOLKIT) [Glied94]).

[4] The local user always has assigned the color 'black'.

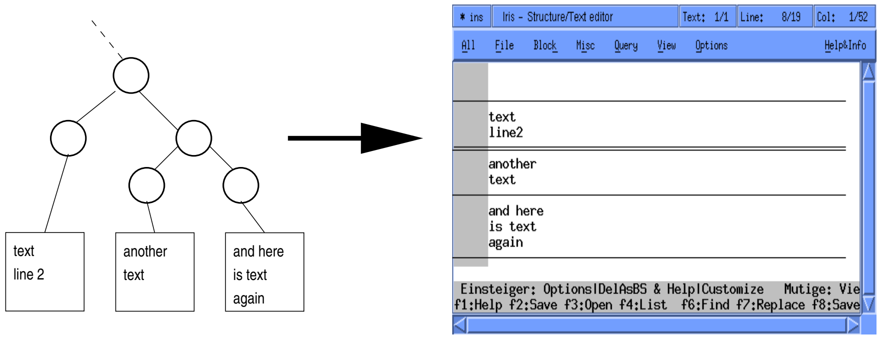


Figure 4.4: Example for linearization and display in combi editors

At present our combi editor only handles text objects. Other objects are displayed as sensible links that allow starting the corresponding object editor application in another window. One possible future extension would be to extend that application by editing functionality for other object types (see EZ for an example). We are further planning to apply the compound document editor concept presented in the OPENDOC and OLE standards [Adler95, Nelso95]. That means that we will have different part editors for all possible medium types and a combi editor shell that integrates these part editors in one display window.

## 4.4 Standard editor shells

One requirement for multi-user editors was that the users want to use their favorite interfaces in the environment.

The first possibility to provide standard interfaces within the IRIS environment is to change the editor source in a way that the editor uses the access layer for document access.

Since in many cases it is not possible to change the source code of an application the second best solution is to write an additional 'shell'-program that loads the objects from the access layer and writes them to a file to be accessed by the standard application. Then the standard application is started on the created file. The shell application monitors the file and writes updates back to the access layer immediately. When the standard application exits the shell writes the updated file back (perhaps after reconverting it) and exits. In addition to starting the standard application and monitoring the file the shell application may also open a window for displaying collaborative information (events, list of collaborating users).

The problem with that second approach is that it is very difficult to convert hierarchical IRIS documents to the file formats of the editors and to retransfer them back after editing. Hence, we also use, as third possibility, the storage of complete binary document files of external applications in text objects (subtype `x-blob` - binary large object). Here the shell application just stores the object content to a file and calls the standard application for editing that file.

At present we have changed the GNU EMACS editor in the first way. Shells of the second type were written for linearizing subtrees with text objects into ASCII files and into FRAMEMAKER MIF- and Book-files. The ASCII files can be edited with all available text editors. The binary file solution was tested with different Unix tools (e.g. XFIG, FRAMEMAKER)

## 4.5 Integration of other applications

Especially in the context of communication it is crucial to provide interfaces to other applications.

One project we are working on intends to combine the IRIS user interfaces with application sharing and video conferencing [Vojik94]. Our research is part of the DFN[5]-project "Reginales Testbed für Breitbandkommunikation" (Regional Testbed for Broadband Communication). The video conference system we are currently using is the BERKOM MMC system [Alten93]. That is a video collaboration system that is built on top of the BERKOM multimedia transport services (MMT). The setting provides support for tightly coupled cooperation (direct communication, window sharing, WYSIWIS).

At present the switch to a tightly coupled situation is achieved by explicitly joining a session within the MMC interface. It was not possible to integrate the conference initialization task in IRIS because MMC does not provide application interfaces. That is why we are currently trying to use other conferencing systems.

In addition to synchronous communication, tools for asynchronous direct communication (e-mail and news) have been integrated. The integration of a workflow management tool is currently in progress.

## 4.6 Current status and ongoing work

We have already mentioned in the different sections of this chapter what components of the user interface layer are already implemented. To summarize the state: we now have a structure editor, a graphic editor, shells for standard ASCII editors and for FRAMEMAKER and a first prototype of a combi editor for structure and text. All these applications have been implemented and tested with the pessimistic access specialists. Consequently they do not support object versions and conflict resolution.

Our ongoing work can be categorized as follows:

- *support of the optimistic access layer concepts at the user interface*: First we are extending the combi editor by support for different text versions and by support for helping the user in resolving conflicts.

[5] Deutsches Forschungsnetz Verein (German Researchnet Association)

- *compound document editors*: It is our goal to have a group aware compound document editor environment for editing a document (structure and content objects). We are currently exploring how the existing compound document models have to be changed to be applicable in synchronous multi-user environments. The result of that research will be a model for building IRIS component editors. That model then will be implemented for the different object types.

# Chapter 5

# Implementation

The components described in the previous chapters have been implemented on Hewlett Packard series 700 workstations using HPUX 9.0x, X11R5, Motif 1.2 (user interface builder UIM/X for the graphic editor). The implementation languages are C, C++ and Tcl/Tk. A version for Sun SOLARIS 2.4 has just been finished, a version for SGI Irix is planned. In this chapter we first describe some aspects of the environment (Section 5.1). Then the functionality of the libraries especially written for simplifying the programming of the specialist processes is presented (Section 5.2).

## 5.1 IRIS environment

The local IRIS environment consists of a number of binaries to be called by users (user interfaces, shell applications and converter applications), a number of binaries to be forked by the user applications (specialists) and some processes that should run all the time.

To install IRIS you have to:

1. copy the binaries (specialists, user interfaces, multicast daemon, object server) and some configuration files (e.g. a site specific `.irisrc` file including presets for editor configurations)

2. place the directory containing the binaries in the search path and set the environment variable `IRISROOTDIR`

3. edit the multicast initialization file (add all hosts that should be able to participate)

4. start the multicast daemon on all hosts mentioned in the multicast initialization file

5. start the object server on one of this hosts

6. now you can start working with IRIS by calling a user interface application

### 5.1.1 Multicast

For the communication among specialist processes a multicast service is used (see 5.2.1 for a description of its functionality). The central component of that multicast service are the multicast daemons. A daemon process accepts requests from the local clients and distributes multicast messages to all other daemons. These remote daemons then decide if the message is interesting for on of their clients and deliver it.

On all machines with applications participating in the group communication a daemon has to be running.

### 5.1.2 Object server

Another process that has to run all the time is the object server. That process is used by the pessimistic specialists to load objects on startup and to store them when closing a session.

The object server can be started on any machine within the group of multicasting hosts. The server subscribes for the multicast group '`iris:objserver`' and waits for request messages. Clients do not have to know where the server is running because they just have to send a request message to that group.

To access the object server a stub library with the following functionality is available:

```
objserv_insert(oid)
objserv_put(oid, data, len)
objserv_putmeta(oid, metadata, len)
objserv_get(oid, &data, &len)
objserv_getmeta(oid, &metadata, &len)
objserv_getlist(&listp, &len)
objserv_remove(oid)
```

With the optimistic specialists and their passive copy scheme the object server will no longer be needed.

### 5.1.3 Specialists

The specialist processes are dynamically started by the client stubs that are linked to the user interface applications. The binaries are searched in the standard search path and in the `$(IRISROOTDIR)/bin` directory.

## 5.2 Multilib

To implement components of distributed applications like IRIS (especially the access specialists) one needs communication mechanisms. Additionally, it is beneficial to have functions for dividing a process into different threads and for providing simple event registration and event handling. This functionality has been implemented for

IRIS in form of a multicast service and different libraries that are collected in the the so called `multilib` (multi-cast, multi-thread-library).

The multilib consists of four parts:

- functions for multi-threading and inter-thread communication
- functions and daemon process for multicasting
- functions for registrating events and for processing events
- functions for easily packing and unpacking variables in machine independent message buffers

### 5.2.1 Multicast

For communication between access specialists and for locating all members of an editing group we are using a multicast service, such that messages sent by one member are received by all participants in the same network partition. Hence, it is possible to communicate independently in different network partitions.

The service uses hierarchical group names (e.g. '`doc1.chapter2.section4`'). These group names can be any strings. For subscribing you may use wildcards with the group names. In IRIS we use the object identifiers as group names. All specialists subscribe for the object identifiers of their replicas.

#### Programming interface

The functions `mc_subscribe` and `mc_unsubscribe` are used to register for a multicast group. You can use wildcards in the group name and you may specify by a flag if messages are also delivered if their group name is only prefix of the given group name.

To send a message one has to specify the addressee (group name or hostname and process id), a message type (string) and the message buffer (pointer and length).

```
mc_bcastblock(mcastgroup, type, msgbuf, msglen, echo)
mc_sendblock(hostname-processid, type, msgbuf, msglen)
```

With the `type` field it is possible to shedule the message inside a process without unpacking the parameters (see Section 5.2.3). In addition there are more comfortable functions that accept lists of variables or buffers as parameters (see Section 5.2.4).

When a message for a process arrives it is queued until it is received by the command `mc_receive`. The functions returns the following data:

```
struct mc_message {
     char* key;
     char* type;
     mc_Address sender;
```

```
        MsgBuffer buffer;
        time_t sendtime;
} mc_Message;
```

Finally it is possible to request how many clients are locally or globally subscribed for a group name (`mc_query_local` and `mc_query_all`)[1].

**Architecture**

In the current version of the multicast service the functionality is implemented with primitive multicast daemons. Deamons have to run on all machines participating in the multicast service. A daemon has to have a list of all other machines that are participating. This list may be hierarchically divided into subnets. The daemon sends messages to any daemon in every subnet and the receiving daemon is responsible for distributing the message to the other daemons in the subnet.

We are currently working on better solutions for the message distribution. Therefore we are trying to use existing toolkits for achieving our functionality (e.g. HORUS [Renes93], GTS [Maffe95], IP multicast, MBONE).

Applications that want to use the multicast service are linked with the client library that consists of stub functions that communicate with the daemon process. When messages are received they are queued for the clients.

### 5.2.2 Multithread

Our thread library provides non-preemptive multi-threading. We implemented the functions from scratch for HPUX. For SOLARIS and further systems we implement the functions based on the standard pthreads library.

First there are functions for initializing the thread library, for forking a new thread and for waiting and explicit scheduling:

```
t_start()
t_fork(function, when, argcount, ...)
t_exit()

t_wait(threads, semaphores, messages, filein,
     fileout, fileerr, timeout, all)
t_select(nfds, readfds, writefds, exceptfds, timeout)
t_defer()
```

For inter-thread communication we have semaphors and messages:

```
sem_new()
sem_release(sem)
```

[1] The global value only counts the processes whose machines can be reached.

```
sem_release_now(sem)
sem_delete(sem)

t_send(thread, type, data, len)
t_send_now(thread, type, data, len)
t_receive(type)
t_msg_remove(msg)
```

### 5.2.3 Eventmanager

The main actions within a distributed application are waiting for events triggered by local users or remote instances and processing these events. To simplify the programming of such applications we built our `eventmanager`.

That library consists of different functions for registrating events and a loop function that checks for the events and forks the handling functions as threads.

Possible event types are:

- activity on input/output/alert file descriptors
- UNIX signals
- arriving multicast messages

See the following program for an example:

```
void main()
{
  mc_subscribe("demogroup", 0);
  t_addentry("msgtyp1", "demogroup", process_msg);
  t_addentry("msgtyp2", "demogroup", process_msg);
  t_addsignal(SIGINT, end_prg);
  t_addinput(0, process_stdin);
  t_mainloop();
}
void process_msg(char* type, mc_Message* msg)
{
  ...
}
void process_stdin(int type, int fd)
{
  read(fd, buffer, 100);
}
void end_prg(int sig)
{
  mc_unsubscribe_all();
  exit(0);
}
```

### 5.2.4 Message buffer

The last component of the multilib are functions for packing variables in machine independent buffers (C-type `MsgBuffer`). These buffers can be exchanged between heterogeneous machines.

At present the following variable types are supported for packing:

- int, array of int
- float, double, arrays of (float, double)
- char, array of char
- string (char* with '0' delimiter)
- mc_Address (address type used by the multicast service), array of mc_Address
- mc_Message (complete message structure as used by the multicast service)
- MsgBuffer

There are functions for initializing, freeing and printing a buffer. More important are the functions used for packing parameters in buffers. In these functions a format string and a list of parameters (for packing) or a list of pointers to parameters (for unpacking) has to be specified (similar to the standard C library functions `printf` or `scanf`):

```
msgbuf_pack(MsgBuffer *pbuf, char *formatstr, va_list argp)
/* msgbuf_pack(b, "%d %s %D", 12, s, iarr, iarrlen); */
msgbuf_unpack(MsgBuffer *pbuf, char *formatstr, va_list argp)
/* msgbuf_unpack(b, "%d%s%D", &i, &s, &iarr, &iarrlen); */
```

To simplify the sending of variables the packing is intergrated in additional versions of the send and bcast functions of the multicast service:

```
mc_mcast(groupname, type, formatstr, ...)
mc_send(address, type, formatstr, ...)
```

# Chapter 6

# Conclusion

In this report I have presented our group editor environment IRIS. The proposed environment has several advantages:

- It can be used in wide area networks.
- It supports disconnected workstations as well as mobile workstations with low bandwidth communication links. (The user or the user interface application can determine when to distribute updates and when to receive updates.)
- It models as a minimum the basic semantics of single user editors.
- It supports synchronous and asynchronous collaboration.
- Existing single user editors can easily be converted to user interfaces of the system.
- The system can easily be extended to support new medium types or structure types.

## Plans for the future

The editing environment now needs evaluation. In the last quarter of 1995 and in 1996 we intend to test IRIS by supporting the collaborative production of scientific reports among the research groups Forwiss München and Forwiss Erlangen. Additionally we are still extending the optimistic access specialists and user interfaces for different medium types.

The following aspects are considered interesting and will be addressed in the future:

- *Resolution of conflicts*: how to help the user in this task.
- *User interface*: how to present the information (differences between announced disconnection of mobile workstations and disconnections because of network failure).

- Integration of *annotations*.
- Support for *document layout information*.
- Support for *generic structures at the user interface*.
- *Compound document editor user interface*.

See http://www11.informatik.tu-muenchen.de/local/proj/iris/ for present information and for information on the availability of the software to public.

# Bibliography

[Adler95] R. M. Adler. Emerging Standards for Component Software. *IEEE Computer*, **28**(3):68–77, March 1995.

[Alten93] M. Altenhofen, J. Dittrich, R. Hammerschmidt, T. Käppner, C. Kruschel, A. Kückes, and T. Steinig. The BERKOM Multimedia Collaboration Service. *Proceedings of ACM International Conference on Multimedia* (Anaheim, CA), pages 457–464. ACM Press, New York, NY, August 1993.

[Baeck93] R. M. Baecker, D. Nastos, I. R. Posner, and K. L. Mawby. The User-Centred Iterative Design of Collaborative Writing Software. *Proceedings of ACM Conference on Human Factors in Computing Systems (INTERCHI'93)* (Amsterdam, The Netherlands), SIGCHI, pages 399–405, S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White, editors. ACM Press, New York, NY, April 1993.

[Bauer93] F. Bauernfeind. Erweiterung des verteilten Mehrbenutzereditors IRIS um ODA-Struktur. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, November 1993. Diplomarbeit.

[Beaud92] M. Beaudouin-Lafon and A. Karsenty. Transparency and Awareness in a Real-Time Groupware System. *Proceedings of 5th ACM Symposium on User Interface Software and Technology* (Monterey, CA), SIGGRAPH/SIGCHI, pages 171–180. ACM Press, New York, NY, November 1992.

[Beck93a] E. E. Beck. 6: A Survey of Experiences of Collaborative Writing. In M. Sharples, editor, *Computer Supported Collaborative Writing*, pages 87–112. Springer Verlag, Berlin, 1993.

[Beck93b] E. E. Beck and V. M. E. Bellotti. Informed Opportunism as Strategy: Supporting Coordination in Distributed Collaborative Writing. *Proceedings of 3rd European Conference on Computer Supported Cooperative Work* (Milan, Italy), pages 233–248, G. de Michelis, C. Simone, and K. Schmidt, editors. Kluwer Academic Publishers, Dordrecht, September 1993.

[Berli90] B. Berliner. CVS II: Parallelizing Software Development. *Proceedings of USENIX Conference Winter 1990* (Washington, DC), pages 341–352. USENIX Association, Berkeley, January 1990.

[Blume95] O. Blume. Dynamische, konfigurierbare Datenhaltung in Form von Replikaten. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, 1995. Diplomarbeit.

[Borgh93a] U. M. Borghoff. Möglicher Einsatz von Votierungsverfahren zur Nebenläufigkeitskontrolle in synchronen Groupware-Systemen. Technical report TUM–I 9326. Habilitationsschrift, Institut für Informatik, Technische Universität München, Munich, Germany, December 1993.

[Borgh93b] U. M. Borghoff and G. Teege. Application of Collaborative Editing to Software-Engineering. *SIGSOFT Software Engineering Notes*, **18**(3):A 56–64, July 1993.

[Borgh93c] U. M. Borghoff and G. Teege. Structure Management in the Collaborative Multimedia Editing System IRIS. *Proceedings of 1st International Conference on Multi-Media Modeling (MMM'93)* (Singapore), pages 159–173, T. S. Chua and T. L. Kunii, editors. World Scientific, Singapore, November 1993.

[Borgh95] U. M. Borghoff and J. H. Schlichter. *Rechnergestützte Gruppenarbeit — Eine Einführung in Verteilte Anwendungen*, Springer Lehrbuch. Springer Verlag, Berlin, 1995.

[Chamb90] D. D. Chamberlin. Managing Properties in a System of Cooperating Editors. *Proceedings of International Conference on Electronic Publishing, Document Manipulation & Typography* (Gaithersburg, MD), pages 31–46, R. Furuta, editor. Cambridge University Press, Cambridge, September 1990.

[Demer88] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. *Operating Systems Review*, **22**(1):8–32, January 1988.

[Dewan94] P. Dewan, R. Choudhary, and H. Shen. An Editing-Based Characterization of the Design Space of Collaborative Applications. *JOCOMP.*, **4**(3):219–239, 1994.

[Dillo93] A. Dillon. 5: How Collaborative is Collaborative Writing? An Analysis of the Production of Two Technical Reports. In M. Sharples, editor, *Computer Supported Collaborative Writing*, pages 69–86. Springer Verlag, Berlin, 1993.

[Dorne92] J. Dorner. Authors and Information Technology: New Challenges in Publishing. In M. Sharples, editor, *Computers and Writing: Issues and Implementations*. Kluwer Academic Publishers, Dordrecht, 1992.

[Douri95a] P. Dourish. Developing a Reflective Model of Collaborative Systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, **2**(1):40–63, March 1995.

[Douri95b] P. Dourish. The Parting of the Ways: Divergence, Data Management and Collaborative Work. *European Conference on Computer Supported Cooperative Work* (Stockholm, Sweden), September 1995.

[Ellis89] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. *Proceedings of ACM SIGMOD International Conference on Management of Data* (Portland, OR). Published as J. Clifford, B. Lindsay, and D. Maier, editors, *SIGMOD Record*, **18**(2):399–407. ACM Press, New York, NY, June 1989.

[Ellis90] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Design and Use of a Group Editor. *Engineering for Human Computer Interaction*, pages 13–25, G. Cockton, editor. North-Holland, Amsterdam, 1990.

[Ellis91] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware – Some Issues and Experiences. *Communications of the ACM*, **34**(1):38–58, January 1991.

[Ferwa91] T. Ferwagner. Experiences in designing the Hohenheim CATeam room. In J. M. Bowers and S. D. Benford, editors, *Studies in Computer Supported Cooperative Work*. North-Holland Publishers, Amsterdam, 1991.

[Fish88] R. S. Fish, R. E. Kraut, and M. D. P. Leland. Quilt: A Collaborative Tool for Cooperative Writing. *Proceedings of ACM SIGOIS/IEEE TC-OA Conference on Office Information Systems* (Palo Alto, CA). Published as R. B. Allen, editor, *SIGOIS Bulletin*, **9**(2&3):30–37, March 1988.

[Fries95] T. Frieß. Datenverwaltung im Strukturspezialisten des verteilten Mehrbenutzer-Editors Iris. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, 1995. Diplomarbeit.

[Gesin93] T. Gesing. Darstellung und Manipulation logischer Sichten auf die Dokumentstruktur des verteilten Mehrbenutzereditors IRIS. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, November 1993. Diplomarbeit.

[Giess94] T. Giessner. Eine Darstellungsschicht für Grafikdokumente im Mehrbenutzereditor Iris. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, 1994. Diplomarbeit.

[Glick92] J. Glicksman and V. Kumar. A SHAREd collaborative environment for mechanical engineers. *Groupware '93*, pages 335–347, D. D. Coleman, editor. Morgan Kaufmann Publ. Incorporated, Los Altos, CA, 1992.

[Glied94] T. Gliedt. EZ — As a Word Processor. *Linux Journal*, August 1994. available from the Andrew team (ftp.andrew.cmu.edu).

[Green92] S. Greenberg, M. Roseman, D. Webster, and R. Bohnet. Human and technical factors of distributed group drawing tools. *Interacting with Computers*, **4**(3):364–392, 1992.

[Greif86] I. Greif, R. Seliger, and W. Weihl. Atomic Data Abstractions in a Distributed Collaborative Editing System (Extended Abstract). *Proceedings of 13th ACM SIGACT/SIGPLAN Annual Symposium on Principles of Programming Languages* (St. Petersburg Beach, FL), pages 160–172. ACM Press, New York, NY, January 1986.

[Greif88] I. Greif, editor. *Computer-supported cooperative work: A book of readings*. Morgan Kaufmann Publ. Incorporated, Los Altos, CA, 1988.

[Gronb94] K. Grønbæk, J. A. Hem, O. L. Madsen, and L. Sloth. Cooperative Hypermedia Systems: A Dexter-Based Architecture. *Communications of the ACM*, **37**(2):65–74, February 1994.

[Grudi88] J. Grudin. Perils and pitfalls. *Byte*, **13**(13), December 1988.

[Grudi90] J. Grudin. Why Groupware Applications Fail. In B. Laurel, editor, *The Art of Human-Computer Interaction*. John Wiley, New York, 1990.

[Halas94] F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. K. Grønbæk and R. H. Trigg, editors, *Communications of the ACM*, **37**(2):30–39, February 1994.

[Hardm93] L. Hardman, D. C. A. Bulterman, and G. van Rossum. The Amsterdam Hypermedia Model: extending hypertext to support real multimedia. Technical report CS-R9306. Computer Science Department, Centrum voor Wiskunde, Amsterdam, The Netherlands, 1993.

[Hiltz85] S. R. Hiltz and M. Turoff. Structuring computer-mediated communication systems to avoid information overload. *Communications of the ACM*, **28**(7):680–689, July 1985.

[Karbe90] B. Karbe, N. Ramsperger, and P. Weiss. Support of Cooperative Work by Electronic Circulation Folders. *Proceedings of Conference of Office Information Systems* (Cambridge, MA), pages 109–117. ACM Press, New York, NY, April 1990.

[Karse93] A. Karsenty, C. Tronche, and M. Beaudouin-Lafon. GroupDesign: Shared Editing in a Heterogeneous Environment. *Computing Systems*, **6**(2):167–195. USENIX, Spring 1993.

[Kelln95] J. Kellner. Implementierung eines Text-Spezialisten für den verteilten Mehrbenutzereditor Iris. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, 1995. Fortgeschrittenenpraktikum.

[Kistl92] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, **10**(1):3–25, February 1992.

[Koch92] M. Koch. Konzeption und Implementierung einer Zugriffsschicht für einen verteilten Mehrbenutzereditor. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, November 1992. Diplomarbeit.

[Koch96] M. Koch. Design issues for a distributed multi-user editor. *Computer Supported Cooperative Work — An International Journal*, **5**(1), 1996. to be published.

[Lelan88] M. D. P. Leland, R. S. Fish, and R. E. Kraut. Collaborative Document Production Using Quilt. *Proceedings of 2nd International Conference on Computer Supported Cooperative Work* (Portland, OR), pages 206–215. ACM Press, New York, NY, September 1988.

[Lewis88] B. T. Lewis and J. D. Hodges. Shared Books: Collaborative Publication Management for an Office Information System. *Proceedings of ACM SIGOIS/IEEE TC-OA Conference on Office Information Systems* (Palo Alto, CA). Published as R. B. Allen, editor, *SIGOIS Bulletin*, **9**(2&3):197–204. ACM Press, New York, NY, March 1988.

[Lu91] I. M. Lu and M. M. Mantei. Idea Management in a Shared Drawing Tool. *Proceedings of 2nd European Conference on Computer Supported Cooperative Work* (Amsterdam, Netherlands), pages 97–112, L. Bannon, M. Robinson, and K. Schmidt, editors, September 1991.

[Maffe95] S. Maffeis, W. Bischofberger, and K.-U. Mätzel. A Generic Multicast Transport Service to Support Disconnected Operation. *Proceedings of 2nd USENIX Symposium on Mobile and Location-Independent Computing* (Ann Arbor, MI), April 1995.

[Makio93] A. Mäkiö. Konzeption und Implementierung einer Benutzerschicht für einen verteilten Mehrbenutzereditor. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, February 1993. Diplomarbeit.

[Malco91] N. Malcolm and B. R. Gaines. A Minimalist Approach to the Development of a Word Processor Supporting Group Writing Activities. *Proceedings of ACM SIGOIS Conference on Organizational Computing Systems* (Atlanta, GA), SIGOIS, pages 147–152. ACM Press, New York, NY, 1991.

[Malon87] T. W. Malone, K. R. Grant, F. A. Turbak, S. A. Brobst, and M. D. Cohen. Intelligent information-sharing systems. *Communications of the ACM*, **30**(5):390–402, 1987.

[McAlp94] K. McAlpine and P. Golder. A New Architecture for a Collaborative Authoring System: Collaborwriter. *Computer Supported Cooperative Work (CSCW)*, **2**(3):159–174. Kluwer Academic Publishers, Dordrecht, 1994.

[Murra94] D. Murray and B. Hewitt. 3: Capturing Interactions: Requirements for CSCW. In D. Rosenberg and C. Hutchison, editors, *Design Issues in CSCW*, pages 27–59. Springer Verlag, Berlin, 1994.

[Nebel95] C. Nebel. Benutzerunterstützung bei einem verteilten Mehrbenutzer-Struktureditor. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, 1995. Diplomarbeit.

[Nelso95] C. Nelson. OpenDoc and its Architecture. *The X Resource*, Issue 13:107–126. O'Reilly, Sebastopol, CA, January 1995.

[Neuwi90] C. M. Neuwirth, D. S. Kaufer, R. Chandhok, and J. H. Morris. Issues in the Design of Computer Support for Co-authoring and Commenting. *Proceedings of International Conference on Computer Supported Cooperative Work* (Los Angeles, CA), pages 183–195. ACM Press, New York, NY, October 1990.

[Neuwi92] C. M. Neuwirth, R. Chandhok, D. S. Kaufer, P. Erion, J. H. Morris, and D. Miller. Flexible Diff-ing in a Collaborative Writing System. *Proceedings of 4th International Conference on Computer Supported Cooperative Work* (Toronto, Canada), pages 147–154, J. Turner and R. E. Kraut, editors. ACM Press, New York, NY, October 1992.

[Neuwi94] C. M. Neuwirth, D. S. Kaufer, R. Chandhok, and J. H. Morris. Computer Support for Distributed Collaborative Writing: Defining Parameters of Interaction. *Proceedings of International Conference on Computer Supported Cooperative Work* (Chapel Hill, NC), pages 145–152, R. Furuta and C. M. Neuwirth, editors. ACM Press, New York, NY, October 1994.

[Newma91] R. E. Newman-Wolfe and H. K. Pelimuhandiram. MACE: A Fine Grained Concurrent Editor. *Proceedings of ACM SIGOIS Conference on Organizational Computing Systems* (Atlanta, GA), SIGOIS, pages 240–254. ACM Press, New York, NY, 1991.

[Olsen88] L. A. Olsen, R. R. Beattie, W. Brinkerhoff, D. Kmenta, and R. Santucci. Processing sponsored project proposals at twelve universities. UM EXPRES working paper. University of Michigan, 1988.

[Pacul94] Fran cois Pacull, A. Sandoz, and A. Schiper. Duplex: A Distributed Collaborative Editing Environment in Large Scale. *Proceedings of International Conference on Computer Supported Cooperative Work* (Chapel Hill, NC), pages 165–173, R. Furuta and C. M. Neuwirth, editors. ACM Press, New York, NY, October 1994.

[Renes93] R. van Renesse, K. P. Birman, R. Cooper, B. Glade, and P. Stephenson. The Horus System. Technical report CS TR. Department of Computer Science, Cornell University, Ithaca, NY, July 1993.

[Rodde91] T. Rodden. A Survey of CSCW Systems. *Interacting with Computers*, **3**(3):319–353, 1991.

[Sasse93] M. A. Sasse, M. J. Handley, and S. C. Chuang. Support for Collaborative Authoring via Email: The MESSIE Environment. *Proceedings of 3rd European Conference on Computer Supported Cooperative Work* (Milan, Italy), pages 249–264, G. de Michelis, C. Simone, and K. Schmidt, editors. Kluwer Academic Publishers, Dordrecht, September 1993.

[Satya90a] M. Satyanarayanan. Scalable, Secure, and Highly Available Distributed File Access. *IEEE Computer*, **23**(5):9–22, May 1990.

[Satya90b] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, **c–39**(4):447–459, April 1990.

[Schra94] K. Schrader. Zugriffsschicht für Grafikmanipulationen im Mehrbenutzereditor Iris. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, 1994. Diplomarbeit.

[Sewal95] B. Sewald. Optimistische Nebenläufigkeitskontrolle bei einem verteilten Mehrbenutzereditor. Technical report. Institut für Informatik, Technische Universität München, Munich, Germany, 1995. Diplomarbeit.

[Sharp88] M. Sharples and C. O'Malley. 17: A Framework for the Design of a Writer's Assistant. In J. Self, editor, *Artificial Intelligence and Human Learning*, pages 276–290. London: Chapman and Hall, 1988.

[Sharp93] M. Sharples, J. S. Goodlet, E. E. Beck, C. C. Wood, S. M. Easterbrook, and L. Plowman. 2: Research Issues in the Study of Computer Supported Collaborative Writing. In M. Sharples, editor, *Computer Supported Collaborative Writing*, pages 9–28. Springer Verlag, Berlin, 1993.

[Singh89] B. Singh. Invited talk on coordination systems. *Organizational Computing Conference* (Austin, TX), pages 13–14, November 1989.

[Tatar91] D. G. Tatar, G. Foster, and D. G. Bobrow. Design for Conversation: Lessons from Cognoter. *International Journal on Man-Machine Studies*, **34**(2):185–210, 1991. Republished in Greenberg, 1991.

[Teege94] G. Teege and M. Koch. Integrating Access and Collaboration for Multimedia Applications. *Proceedings of International Conference on Multimedia, Hypermedia and Virtual Reality* (Moscow, Russia), pages 170–176, P. Brusilowsky, editor, September 1994.

[Tesle95] L. G. Tesler. Networked Computing in the 1990s. *Scientific American: The Computer in the 21st Century*, **6**(Special Issue 1):10–21. Scientific American Inc, New York, NY, 1995.

[Tichy82] W. F. Tichy. Design, Implementation, and Evaluation of a Revision Control System. *Proceedings of 6th International Conference on Software Engineering* (Tokyo, Japan), pages 58–67, September 1982.

[Vojik94] F. Vojik. Video Support for the Distributed Multi-User Editor IRIS. *Proceedings of IASTED/ISMM Int. Conference on Distributed Multimedia Systems and Applications* (Honolulu, Hawaii), pages 277–280. IASTED/ISMM - ACTA Press, Anaheim, CA, August 1994.

[Wilso91] P. Wilson. *Computer Supported Cooperative Work: An Introduction*. Oxford, UK: Intellect Books, 1991.