COMIC

ESPRIT BASIC RESEARCH PROJECT 6225

Computable Models and Prototypes of Interaction

Abstract

This document forms Deliverable 4.2. of the COMIC project. The input for this deliverable comes from two tasks within workpackage 4, Task 4.2 "*Computer modelling and prototyping of interaction within virtual computer spaces*" and Task 4.3 "*Modelling, prototyping and assessment of multi-user interaction with and through shared artifacts*". Each of these tasks is addressed by a separate part of the deliverable, although this is not to say that they present unrelated work. To the contrary there has been significant cooperation between the partners involved in both of these tasks.

The first portion of the deliverable focuses on the development and prototyping of a computational model of interaction within virtual computer spaces.

The second portion of the deliverable presents the work undertaken within task 4.3 of the COMIC project. The work described in this section of the deliverable focuses on the sharing of objects across a community of users.

Document ID	COMIC-D4.2
Status	Final
Type	Deliverable
Version	3.0
Date	October, 1994
Editors	S. Benford, A. Bullock, L. Fuchs, J. Mariani
Task	4.2, 4.3

Project coordinator:

Tom Rodden
Computing Department
University of Lancaster
Lancaster LA1 4YR
United Kingdom
Phone: (+44) 1524 593 823
Fax: (+44) 1524 593 608
Email: tom@comp.lancs.ac.uk

The COMIC project comprises the following institutions:

Gesellschaft für Mathematik und Datenverarbeitung (GMD), Bonn, Germany
Risø National Laboratory, Roskilde, Denmark
Royal Institute of Technology (KTH), Stockholm, Sweden
Swedish Institute for Computer Science (SICS), Stockholm, Sweden
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
University of Amsterdam, Amsterdam, The Netherlands
University of Lancaster, Lancaster, United Kingdom (Coordinating Partner)
University of Manchester, Manchester, United Kingdom
University of Milano, Milano, Italy
University of Nottingham, Nottingham, United Kingdom
University of Oulu, Oulu, Finland

Editors of this report:

Steve Benford
Department of Computer Science
University of Nottingham
Nottingham NG7 2RD
United Kingdom
Phone: (+44) 115 951 4203
Fax: (+44) 115 951 4254
Email: sdb@cs.nott.ac.uk

Adrian Bullock
Department of Computer Science
University of Nottingham
Nottingham NG7 2RD
United Kingdom
Phone: (+44) 115 951 4225
Fax: (+44) 115 951 4254
Email: anb@cs.nott.ac.uk

Ludwin Fuchs
GMD FIT
Schloss Birlinghoven
53754 Sankt Augustin
Germany
Phone: +49 224114 2699
Fax: +49 224114 2084
Email: fuchs@gmd.de

John Mariani
Computing Department
University of Lancaster
Lancaster LA1 4YR
United Kingdom
Phone: (+44) 1524 593797
Fax: (+44) 1524 593608
Email: jam@comp.lancs.ac.uk

ISBN 0 901800 55 4
Lancaster University, 1994
This report is available via anonymous FTP from ftp.comp.lancs.ac.uk.
Further information on COMIC is available from http://www.lancs.ac.uk/computing/research/cseg/comic/

Table of Contents

Introduction

Steve Benford and Adrian Bullock
University of Nottingham

Tom Rodden
University of Lancaster

This introduction briefly summarises the major results of the strand four work during year 2, explains progress made in relation to the work-plan detailed in the technical annexe, suggests directions for further work and relates this work to that of other strands.

This deliverable forms a preliminary version of Deliverable 4.2 "Computer models and prototypes of interaction", the final version of which is due at month thirty. The input for this deliverable comes from two tasks within workpackage 4, Task 4.2 "*Computer modelling and prototyping of interaction within virtual computer spaces"* and Task 4.3 "*Modelling, prototyping and assessment of multi-user interaction with and through shared artifacts"*. Each of these tasks is addressed by a separate part of the deliverable, although this is not to say that they present unrelated work. To the contrary there has been significant cooperation between the partners involved in both of these tasks.

Task 4.2

The first portion of the deliverable focuses on the development and prototyping of a computational model of interaction within virtual computer spaces.

A number of topics are covered, ranging from prototyping the COMIC spatial model of interaction and Populated Information Terrains through to extensions to the spatial model, an approach to access control in shared spaces and requirements for user embodiment. The work described therefore addresses both prototyping and conceptual issues.

Task 4.3

The second portion the deliverable presents the work undertaken within task 4.3 of the COMIC project. The work described in this section of the deliverable focuses on the sharing of objects across a community of users.

The work described here builds upon the requirements outlined in the first year of the COMIC project. Considerable emphasis has been placed on development, experimentation and demonstration of the ideas and concepts outlined in Deliverable 4.1. As a result, a strong prototyping theme dominates the work described in this section of the deliverable. Each of the main chapters is supported by a prototype which demonstrates the core computational concepts uncovered by this work.

To complement this report, and give a clearer understanding of the prototypes developed, a video is attached which shows each of the prototypes in action.

1. Results From the First Year of COMIC

During the first year of the COMIC project, the work on strand 4 resulted in a number of key successes. The work reported in this deliverable builds upon these results and extends them.

Considering interaction in virtual spaces, three main areas of the year one work have acted as stimuli for the further activity:

- The identification of a set of requirements for interaction in large virtual spaces, together with a review of previous work.
- The most significant piece of work to come out of the first year was the specification of a spatial model of interaction based on the concepts of aura, awareness, focus, nimbus and adapters. This initial model seemed to offer an interesting route to constructing large-scale virtual environments with explicit support for cooperative work.
- An initial exploration of the concept of Populated Information Terrains. The concept of a PIT was introduced as an environment for visualising and manipulating data of many kinds whilst interacting with other users who were doing the same.

The second year work has concentrated on prototyping work concerning the spatial model and extensions to the conceptual work carried out in the first year. A number of prototypes demonstrating both the spatial model and approaches for representing large information spaces have been developed and progress has been made in developing the theoretical background as well.

The shared object work builds upon:

- The development of a set of requirements for a shared object services which supports the cooperative sharing of information. These requirements were formulated through a series of ethnographic studies of the use of documents within real world settings.
- The identification of the need to support shared interfaces to promote cooperative interaction with shared objects.
- The outlining of a lightweight activity model based upon shared objects and the notion of adapters that present objects in a context sensitive manner.
- The development of a set of awareness models applicable to a collection of shared objects.

The work of the second year focuses on the population of the shared object service as identified in Deliverable 4.1 and a clearer formulation of the associated shared interface service. Many aspects of the shared object service are developed in a range of related prototypes and contrasted with ongoing initiatives in a range of related areas of computing.

2. Goals and Achievements

A number of goals were proposed for the second year of the project in this strand. Many of these were initially outlined in the technical annexe and amended during the first year of the project. This section examines what was promised in the technical annexe, what was promised at the end of the first year in Deliverable 4.1 and what has been achieved in year two.

2.1 Work Promised in the Technical Annexe

The goal of this workpackage is to develop scalable metaphors and models of interaction applicable to larger groups. It is worth noting that the technical annexe for the project outlines task 4.2 as a running from month 7 until month 36, and task 4.3 as a running from month 13 until month 36. In effect, this deliverable is presenting work at an intermediate stage.

2.1.1 Task 4.2

The overall aim of task 4.2 was described as:

"• The aim of this task is to develop a computational model of multi-person interaction in virtual spaces based on the requirements and metaphors developed in task 4.1. This model will then be assessed through the construction of prototype demonstrators which embody different aspects of the model."

And its outputs were intended to be:

"• A computable model of interaction within virtual computer spaces

• Prototypes which demonstrate the application of this model

• An assessment of this model including estimated limitations of scalability and discussions of usability."

This work involved a number of more specific activities. We begin with conceptual work:

"• Developing a computational model of multi-person interaction in virtual spaces. The model will adopt an object-oriented approach providing detailed specifications of the objects involved in the spatial metaphor and will define appropriate mechanisms for modelling interaction using these objects"

Extensions have been made to the spatial model over the course of the year and these are dealt with in Chapter 5. Some problems with the model, as were stated in Deliverable 4.1, are discussed in terms of focus, nimbus, aura and adapters together with the initial extensions to try and resolve them. Extensions suggested during the course of the second year to introduce non-spatial factors into the model and apply the model to non-spatial scenarios are also considered. Finally the issues of movement between spatial, non-spatial, and locally created ad hoc worlds with orthogonal dimensionalities are considered in Chapter 6.

The issue of embodying users within collaborative virtual environments has also been examined. This involves identifying design issues for providing users

with *appropriate* body images in co-operative settings and describing mechanisms for supporting each of them. A number of issues pertinent to embodiment are considered together with descriptions of how the current prototype spatial model implementations address these issues. The longer term aim of this work is to provide designers with a "body builders work-out" for designing useful embodiments.

A model to control access in collaborative spatial environments based on boundaries has been developed which considers space to be segmented up into different regions with boundaries between each of these regions. The model enforces checking at boundaries to see if objects and users have sufficient credentials to enter a given region. This way access is governed according to where an object or person is in space, somewhat like real life (putting objects in safe places).

The technical annexe also identified prototyping work to be carried out.

> "• The development of user interfaces to support the spatial model will also be addressed through the application of both two and three dimensional visualisation techniques. This will be based on on-going work at both SICS and the University of Nottingham."

> "• In order to assess the viability of the model, a number of prototypes will be produced. These will focus on interaction within simple and complex spaces, including the relationship to wider structures (e.g. virtual organisations) and support for navigation."

There have been two implementations of the spatial model of interaction and these are presented in Chapter 1. The first implementation of the model has been made at Nottingham and is a system called Massive (Model, Architecture and System for Spatial Interaction in Virtual Environments) which provides a VR teleconferencing system which supports conferencing over combinations of audio, graphical and textual media, all governed by the spatial model. This represents a specialised implementation of the model with its own distributed architecture. The second of these is known as Mr Nimbus and has been implemented at SICS using DIVE. Mr Nimbus is a virtual creature who reacts to different awareness states between himself and a user as generated by a simple instantiation of the spatial model. Although simple, this approach suggests how the model might be realised in a range of existing VR systems using only commonly available services such as collision detection.

Prototyping work on Populated Information Terrains, as presented in Chapter 2, examines three dimensional visualisation techniques for data browsing and manipulation. Three PITS prototypes have been developed at Nottingham and Lancaster using DIVE. These prototypes each demonstrate different visualisation techniques for presenting large collections of data to the end user. VR-VIBE is a document browsing and filtering tool which is particularly apt for visualising bibliographic information. The Mapper is a tool for visualising hierarchical information structures such as filesystem and directory information. Q-PIT is a more general tool able to visualise information from databases based on the approach proposed by Michael Benedikt.

Finally, the technical annexe promised evaluation activities:

"• Theoretical and empirical assessment of the models and prototypes. Theoretical work will involve assessment of the general applicability of the model, especially with respect to limitations on scalability and the technical requirements for wide-scale implementation of the model. Limited empirical work will be conducted in which the prototypes will be used to gain an initial assessment of usability of the models"

Some initial experiences from using one of the spatial model prototypes in a laboratory setting are given in Chapter 1. However, assessment remains a major task for the final year of the project. Consequently, the future Deliverable 4.3 will deal with assessment more thoroughly.

2.1.2 Task 4.3

The outputs from task 4.3 were described as:

"• A computational model of multi-user interaction related to computer artifacts

• A user interface development kit based on user interface techniques for realising this model

• A prototype application which demonstrates both the model and the prototyping kit."

During the last year we have made considerable progress and have already achieved all of the outputs listed above to some degree. Specific contributions to these outputs worth highlighting include:

"• A computational model of multi-user interaction related to computer artifacts"

A set of computational models of multi-user interaction for computer artifacts are realised in the development of the shared object service (Deliverable 4.1) and the associated shared interface service (Chapter 8). These models combine work from Lancaster, KTH, GMD and Nottingham and are realised in a series of prototype demonstrators. The work on developing conceptual demonstrators has been undertaken with a view to developing a number of concepts and theories. Conceptual results of particular note which have been achieved during this year of the project include:

- The outlining of a generic architecture for the shared object service which supports the features outlined in demonstrator prototypes.
- The development of a generic model of space which acts a theoretical bridge between the notions of awareness uncovered in the work on shared objects and concepts of awareness arising from the work on shared virtual worlds.
- The uncovering of mechanisms to promote awareness based on event propagation and sets of models of event distribution which use these.
- The outlining of a shared interface and the structuring of interface was based on the principles of the interface service

These conceptual results underpinning the work of the project and outline a number of significant avenues of future work. The most significant area of investigation is an uncovering of the relationship between the supporting

mechanisms in shared user interfaces and interactive properties they exhibit to users.

"• A user interface development kit based on user interface techniques for realising this model"

The core model described in the shared interface service outlined in Chapter 8 provides the basis for toolkit development across the project. In particular, Lancaster has produced a user interface toolkit based on a model of shared access (Chapter 9) and KTH have constructed a toolkit using their MultiGossip system (Chapter 12).

"• A prototype application which demonstrates both the model and the prototyping kit."

In line with the strongly empirical nature of the project we have invested considerable focus on the development of applications which exploit our toolkits and systems. In particular it is worth highlighting the collaborative desktop developed at KTH (Chapter 11), the Codesk system at GMD (Deliverable 4.1, Chapter 5) and the use of applications in the development of supporting platforms and toolkits at Lancaster (Chapter 9, Chapter 10).

Achieving the goals of this task ahead of our perceived schedule has allowed us to broaden the work to consider emerging and new challenges. In particular, we have focused on the "deeper" distributed systems issues associated with sharing objects in cooperative application. The work at UPC (Chapter 14) has focused on the implications of our work for distributed platforms. This has been complemented by work at GMD examining the relationship between the services we outline and those proposed by the object management group's CORBA (Chapter 15).

A final theme of work over the last year has been one of reconciliation and integration. This theme is represented by the final two chapters of this section of the deliverable. We see the need for a general understanding of the nature of cooperative applications to be vital to CSCW. In Chapter 16, we seek to develop an understanding of the generic architecture of cooperative applications and outline a common reference framework based on our experiences over the last two years. The final chapter in this section presents a generic model of space and awareness which can be applied equally to the spatial systems and those based on the shared objects

The technical annexe also outlined the need for assessment and consultation with user groups as a means of verifying results. This was expressed in the original technical annexe as:

" • Assessing the models through discussions with representatives of potential user-groups and organisations who will be invited to try out and comment on the suitability of the developing prototype."

To meet this objective evaluation and assessment of systems based on shared objects has been undertaken as part of the COMIC project. Most notable in the case of air traffic control (Deliverable 2.2.). However, the assessment of cooperative systems poses many challenging methodological questions on the

nature of evaluation and its role in cooperative systems development. This represents a significant point of integration for the project and the work on assessment is presented within Deliverable 2.2.

2.2 Work Promised in Deliverable 4.1

In addition to our original goals listed in the technical annexe, a number of additional objectives were set for the project in Deliverable 4.1. In this section we wish to briefly review these and highlight how they have been met in the work of the second year of the project.

2.2.1 Task 4.2

Four main work areas were identified at the end of Deliverable 4.1, these being:

"• Problems with and extensions to the COMIC spatial model of interaction"

There has been much work on the spatial model, Chapters 1 and 5 presenting prototype implementations and model extensions respectively.

"• Research issues in the design of collaborative virtual spaces"

The ability to tailor virtual environments is necessary. Chapter 3 has investigated the issues surrounding user embodiment in virtual environments. Chapter 6 explores issues of divergence and communication between multiple worlds generated by tailoring the attribute to dimension mappings according to local need.

"• Prototyping and demonstration work"

Chapter 2 presents three prototypes which demonstrate different approaches to Populated Information Terrains. The spatial model has also been implemented, see Chapter 1 for details.

"• Resolving broader tensions between COMIC approach and multi-media and media-space systems"

To be completed in year three.

2.2.2 Task 4.3

The following additional goals were identified:

"• Presentation of Objects to Users and outlining a shared interface service"

During the second year we have examined the development of mechanisms to present shared objects to users in a number of ways:

- Using the collaborative desktop approaches developed at GMD and KTH
- Using user interface toolkits developed at Lancaster and KTH
- Using the notions of a resource broker and object adapters developed at UPC and Lancaster

"• Prototyping portions of a shared object service"

This prototyping has been undertaken at a number of sites in terms of a number of prototypes

- At Lancaster an activity platform based on the mechanisms of the SOS has been developed
- At UPC a resource Trader which provides a basic mechanism of the SOS has been realised
- At GMD a mapping to the facilities provided by CORBA has been outlined
- At KTH an object distribution mechanism has been developed to support multi-user interfaces

All partners have worked together to develop a generic architecture for the SOS based on the experiences of building these prototypes.

"• Cooperative activities represented as shared objects"

A number of partners have contributed to the development of mechanisms to represent activities.

- An activity model and platform has been developed at Lancaster
- At GMD, KTH and Lancaster notions of awareness have been developed to publicise ongoing activity across shared objects.
- At UPC the resource Trader makes use of activities to support brokering

"• The relationship between the spatial and shared object model based on awareness."

The relationship between the spatial and shared object work has provided a significant focus of integration. This integration has taken place at both a theoretical and practical level.

- At a theoretical level a general model of space which can be used to represent the notions of awareness in the spatial model and the use of awareness in shared object systems have been developed
- At a practical level a range of prototype populated information terrains have been realised which allow objects to be shared within a virtual space.

3. Publicity and Dissemination

There has been considerable success in the external dissemination of the results from the second year of the work in Strand 4.

3.1 Publications

Much of the work presented in this deliverable has been accepted for publication in journals, conferences or workshops over the course of the year.

Considering the work of Task 4.2, there have been two journal papers accepted, by Presence and by the Computer Journal, which discuss the spatial model of interaction and the prototyping work which has been carried out. Conferences where papers have been accepted include the ACM Virtual Reality

Software and Technology (VRST 94) conference in Singapore, the International Conference on Artificial Reality and Tele-Existence (ICAT 94) in Japan, the Interactive Database Systems (IDS) conference in Lancaster, UK and HCI 94 at Glasgow, UK. Much of the work was also presented at the Sixth European Research Consortium on Informatics and Mathematics (ERCIM) workshops in Sweden. There is also a chapter of the forthcoming BCS 'Virtual Reality Applications' book concerning the PITS work.

The work on shared objects within strand 4, Task 4.3, has been widely published and disseminated at a variety of conferences and journals. This work has been presented in the book chapters, conferences and journal publications detailed in the COMIC publication list.

3.2 Demonstrations

Some of the early prototypes from Task 4.2 were demonstrated at the European Information Technology Conference (EITC) exhibition in Brussels in June. These demonstrations showed the DIVE implementation of the spatial model of interaction complete with audio interaction and the VR-VIBE Populated Information Terrain.

Significant demonstrations of the COMIC Task 4.3 work have been carried out at workshops and major international conferences within the distributed systems and database communities. Perhaps, the most notable of these has been the demonstration of the work of CoDesk at CHI'94 in Boston, USA.

Part 1

Spatial Model and Prototypes

Chapter 1

Implementing the Spatial Model of Interaction

Steve Benford,
Chris Greenhalgh and
Adrian Bullock

University of
Nottingham

Lennart E Fahlén

Swedish Institute of
Computer Science

This chapter presents two implementations of the COMIC spatial model of interaction. The first, MASSIVE, has been realised on top of its own specialised distributed architecture. The second has been realised as an application of the DIVE system. Between them, they demonstrate complementary approaches towards using the model. More specifically, MASSIVE explores its distributed systems implications and the DIVE work shows how a light-weight version of the model might be realised as an application of existing VR platforms. The chapter also discusses plans for extending and testing these implementations in the final year of the project.

1.1 Introduction

The definition of the spatial model of interaction and its associated concepts of aura, awareness, focus, nimbus, adapters and boundaries was at the heart of the year one COMIC work on collaborative virtual environments. This chapter describes the implementation of this model as carried out in year two. In fact, it presents two complementary implementations based on quite different underlying approaches. The first of these, MASSIVE, is a multi-user virtual reality system specially intended as a test-bed for spatial model concepts and based on its own specially tailored distributed architecture. MASSIVE allows users to employ graphics, audio and text interfaces in arbitrary combinations, thus achieving scalability through heterogeneity. The second demonstrator has been implemented as an application of the DIVE multi-user VR system. This shows how the spatial model might be implemented as an application of existing VR platforms, using only commonly available facilities such as the ability for objects to respond to collision events.

1.2 MASSIVE

MASSIVE (Model, Architecture and System for Spatial Interaction in Virtual Environments) is a prototype implementation of the spatial model. The main goals of MASSIVE are scale (i.e. supporting as many simultaneous users as possible) and heterogeneity (supporting interaction between users whose equipment has different capabilities, who employ radically different styles of user interface and who communicate over an ad-hoc mixture of media).

1.2.1 MASSIVE Functionality

We begin with an overview of the functionality of MASSIVE. The system has been driven by two key requirements:

- Scale - supporting as many simultaneous users as possible;
- Heterogeneity - supporting interaction between users whose equipment has different capabilities and who employ radically different styles of user interface.

MASSIVE supports multiple virtual worlds connected via portals. Each world may be inhabited by several concurrent users who can interact over ad-hoc combinations of graphics, audio and text interfaces. The graphics interface renders objects visible in a 3-D space and allows users to navigate this space with a full six degrees of freedom. The audio interface allows users to hear objects and supports both real-time conversation and playback of sound files. The text interface provides a MUD-like view of the world via a window (or map) which looks down onto an infinite 2-D plane across which users move. Text users are embodied using a few text characters and may interact by typing text messages to one another or by "emoting" (e.g. smiling, grimacing, etc.).

A key feature of MASSIVE is that these interfaces may be arbitrarily combined according to the capabilities of a user's terminal equipment. Thus, at one extreme, the user of a sophisticated graphics workstation may simultaneously run the graphics, audio and text clients (the latter providing a map facility and allowing interaction with non-audio users). At the other, the user of a dumb terminal (e.g. a VT-100) may run the text client alone with a "dumb" graphics embodiment. By using a dynamic brokering mechanism (described below), MASSIVE matches compatible interfaces between objects whenever they meet in space (i.e. on aura collision). The net effect is that users of radically different equipment may interact, albeit in a limited way, within a common virtual world.

All of the above interfaces are driven by the spatial model. Interaction in a given medium is not possible until aura-collision occurs in that medium, thus an object cannot be seen until graphics auras collide and cannot be heard until audio auras collide. Subsequent interaction is controlled by awareness level and hence by focus and nimbus. For example, audio awareness level is mapped onto the volume of the channel, and is sensitive to the distance between and orientations of the objects or users involved.

In the current implementation, users may explicitly manipulate awareness by choosing between three settings for focus and nimbus - normal, narrow (intended for private conversations) and wide (intended for browsing). Also, two adapter objects are provided:

- A podium which extends the aura and nimbus of its user.
- A conference table which replaces its users' normal auras, foci and nimbi with new sets which span the table.

Each user may specify their own graphics embodiment via a configuration file. In addition, we provide some default embodiments, intended to convey the communication capabilities of the users they represent (an important issue in a heterogeneous environment). Thus, an audio user has ears, a non-immersive (and hence monoscopic) user has a single eye and a text user has the letter "T" embossed on their head. The aim of such embodiments is to provide other users with the necessary basic communication cues to decide how to address them. The basic shape of embodiments is also intended to convey general orientation in a simple and efficient manner.

We complete this overview of MASSIVE's functionality with two screen-shots. Figure 1.1 shows a typical view belonging to a graphics user. The figure is taken from a scenario with five people sharing a space (we are one of them). Both text-only and audio-graphical interface users can be identified. Figure 1.2 shows a typical view belonging to a text user. Note the use of simple characters to represent other users, the conference table and walls, and also the display of all currently known objects along with mutual awareness levels in the right hand column.

Having briefly described the functionality of MASSIVE, the following section focuses on the distributed architecture that has made this possible.

1.2.2 Architecture

This section describes the MASSIVE distributed architecture and its realisation of the spatial model. In particular, we focus on the following key techniques which have been introduced in order to support scalability and heterogeneity:

- A communications architecture based on typed peer-peer connections which utilise a combination of RPCs, shared attributes and streams.
- A spatial trading mechanism to dynamically broker interfaces following aura collisions.
- Negotiation of mutual awareness across a common peer protocol and the implementation of focus and nimbus as mathematical functions.
- Use of a separate adapter medium to trigger adapter objects which are realised by parameterising focus and nimbus functions.

We begin with a brief review of MASSIVE's communications architecture.

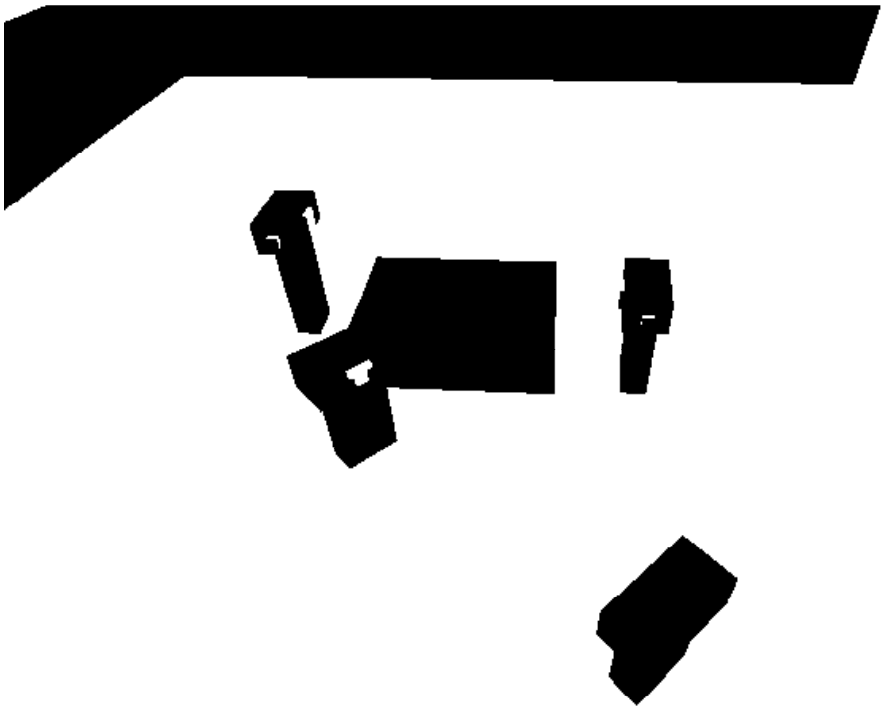


Figure 1.1: Graphical interface

Communications architecture

In contrast to distributed VR systems based on a shared database approach (e.g. dVS [Grimsdale, 91] and DIVE [Carlsson, 93]), the distribution model used in MASSIVE is of independent computational processes communicating over typed peer-to-peer connections (running over standard internet transport protocols). Each computational process may have any number of interfaces, each of which is characterised by a combination of remote procedure calls (RPCs), attributes and streams. We argue that this heterogeneous approach is required to support a mixture of distributed VR functions such as collision detection, distributed system functions such as trading, and continuous media.

Unlike many distributed systems, in order to bind to an interface an object must have a local interface of an appropriate type (i.e. each connection is characterised by the type of both the remote and the local interfaces). Having a defined local interface for each connection allows a natural integration of confirmation, reply and notification messages relating to the remote interface - handling of these messages is folded into the local interface type.

While some connections provide client-server style services, others are strictly peer-to-peer. For example, the connections for the different media in the spatial model have the same interface on each end of the connection - the link is symmetrical. The only asymmetry is in the creation of the link (one object must

make the initial request for a connection) but this asymmetry is hidden to a large extent by the system's software design.

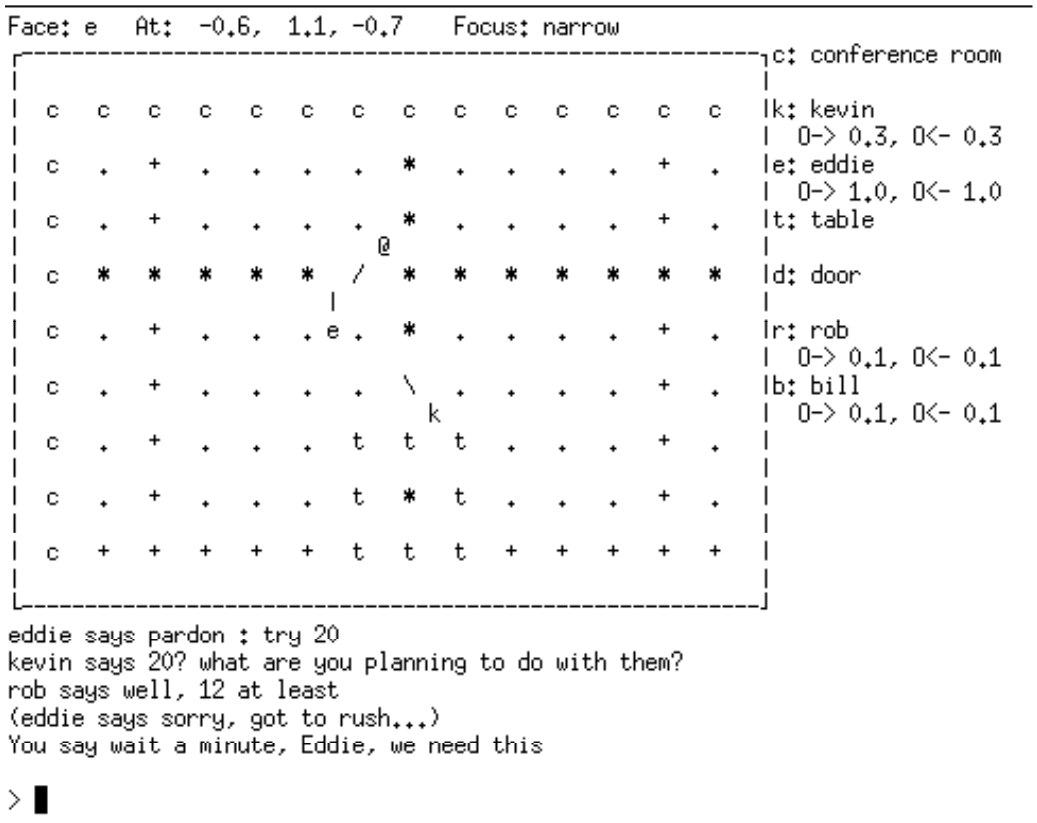


Figure 1.2: Textual interface

Aura and spatial trading

Interaction between objects in the virtual world only becomes possible when two conditions are met: first, it must be established that the objects involved possess some compatible interfaces; and second, the objects must become sufficiently proximate, as determined by their auras (remember that there is a separate aura for each interface). Both of these conditions are reflected in the concept of spatial trading. Spatial trading combines the virtual reality technique of collision detection with the distributed systems concept of trading (or request brokering). Trading, as proposed by recent work on Open Distributed Processing (ODP) and ANSA [Van Der Linden, 92], is intended to promote distribution transparency. A key system service known as the Trader provides a brokering facility between service offers, which are exported by supplier objects and service requests which are made by consumer objects.

To explain how spatial trading operates, we consider two objects in a MASSIVE virtual world. On entering a world an object contacts the master Trader and declares its interfaces as service offers, each with its own aura. At present, one master Trader handles all worlds and all media, and the address of

this Trader is the only information that an object requires in order to enter any world. The master Trader then redirects the object to the aura collision managers which are responsible for the declared interface types in the desired world. If these do not already exist then the master Trader spawns new aura collision managers. A second object subsequently entering the same world will go through the same process. Each aura collision manager monitors the relevant auras of all of the objects known to it. Upon detecting an aura collision it passes out mutual interface references to the objects involved which enables them to establish a peer connection. This situation is illustrated in Figure 1.3.

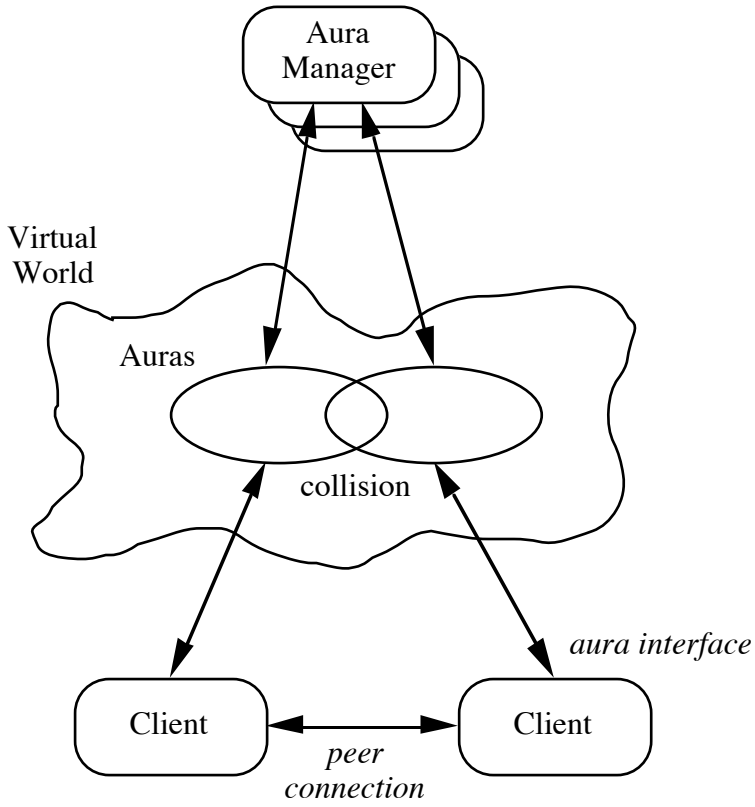


Figure 1.3: Spatial Trading

Heterogeneity is realised through the master Trader effectively registering all interface types currently available in all worlds and maintaining a corresponding set of aura collision managers. This enables MASSIVE to dynamically broker hitherto unseen types of interface. Scalability is supported by distributing the responsibility for detecting aura collisions between multiple aura collision managers.

Plans for future work include a further distribution of responsibility for aura collision detection in densely populated worlds between a federation or hierarchy of collision manager processes, each of which would be responsible for some region of space.

Implementation of focus and nimbus

Once connected, the calculation of mutual awareness levels using focus and nimbus is the responsibility of the peer objects.

In their most general sense focus and nimbus can be described by mathematical functions which map attributes of the communicating objects into focus and nimbus values. These are then combined into mutual awareness levels through a further awareness function. Focus and nimbus will typically involve spatial attributes (e.g. position and orientation), and may additionally take account of other non-spatial attributes. In the current implementation, objects are described solely by a sphere - a point location in space and a radius - and focus and nimbus are described by scalar fields over space, which are sampled at the location of a peer object to give focus and nimbus values. The awareness function is multiplicative (i.e. focus and nimbus are multiplied to give awareness). This gives equal control to the observer and the observed, and is subtractive in nature - i.e. either party can force zero (no) awareness, but neither party can force awareness against the others "wishes."

Focus, nimbus and awareness values are normalised in the range 0-1, where 0 corresponds to no awareness, focus or nimbus, and 1 corresponds to full awareness, focus or nimbus. Current focus and nimbus functions describe conical volumes projecting forwards from an object and dropping off with distance. The operation of focus and nimbus may be controlled by a few key parameters such as conical angle, maximum radius, background level and transition rate, allowing variation from spherical to tightly focused regions. It is this ability to describe focus and nimbus parametrically that allows us to provide the user with a variety of foci and nimbi and to implement adapters.

Objects must exchange information (e.g. position and nimbus) since each object is responsible for calculating how much it is aware of other objects. This is the role of the "peer protocol," with which medium specific peer interfaces, e.g. audio, graphics and text, are compatible.

Integration of client programs

MASSIVE currently supports three client interfaces, one for each of the audio, graphics and text media, which may be arbitrarily combined by the user. Each client protocol is compatible with the peer protocol mentioned above, and may define additional attributes, RPCs and steams as required. For example, the protocol between graphics clients includes attributes describing object orientation and geometry. Note that an object may export a graphical embodiment via a graphics interface even if it cannot display it itself. It is this feature that allows text clients to interact in the graphics medium, and vice versa.

In the case where a user runs multiple clients simultaneously, only one of them will act as a master client controlling navigation and the others will act as slave clients, tracking the position of the object from the master and updating their "views" accordingly. To coordinate these clients a further Trader process has been

created in the style of the ANSA Trader [Van Der Linden, 92]. Rather than matching offers and requests (auras and interfaces) on a spatial basis, this Trader allows offers of and requests for interfaces to be matched on the basis of a keyword (which encodes a common identifier for the processes). This trading is generally local and small scale so it does not pose an obstacle to creating large and scalable systems.

Adapters

Implementing adapters has proved to be quite difficult, and MASSIVE currently offers only a limited solution. There are two issues to be dealt with when implementing adapters: how to trigger the use of an adapter and how to realise its effect on aura, focus, nimbus and awareness. Both of these issues are addressed through the introduction of a separate adapter medium. Adapters exist in their own medium, complete with aura, focus and nimbus. Any object wishing to use an adapter must therefore support an interface for this medium, then as the object moves about it will connect to adapters as a result of an aura collision in the adapter medium. When the object's awareness of the adapter crosses some threshold level the adapter is triggered.

When triggered by an object, an adapter passes some new parameters back to the object via the adapter medium. These new parameters replace the object's current aura, focus and nimbus parameters. Thus, an adapter may extend the range of focus or nimbus, may change their shape (i.e. conical angle) or may alter the way in which they fade to a background level. When the object moves away from the adapter the object restores its original parameters. So as objects move about they naturally experience the effects of adapter objects.

There are several limitations in the current implementation of adapters which will be addressed by future work. Firstly, a single adapter medium is used, so adapters apply simultaneously and identically to all media. Secondly, the parameter substitution approach only allows one adapter to be used at a time; where more than one is triggered some random selection is made. Thirdly, the technique of adjusting focus and nimbus parameters does not readily support adapters which naturally sit in the information flow between two objects (e.g. boundaries).

Portals

Portals are gateways between different worlds, or between different locations within the same world. Portals are implemented in a similar manner to adapters. Each portal is described by a portal interface. One interface is created for each portal and is placed in a special-purpose medium ("portal"). Each object which can be affected by a portal inserts a null interface into that medium. Then when an object becomes aware of a portal interface as determined by specialised focus and nimbus functions it jumps to the world, position and orientation specified by the portal interface.

This concludes our discussion of the MASSIVE distributed architecture. The following section discusses our early experiences with the system and identifies a number of issues for further work.

1.2.3 Preliminary Analysis and Further Work

Up to now, MASSIVE has only been used in the laboratory setting by the researchers who designed and implemented it. Consequently, we cannot yet claim any systematic or objective evaluation of either the spatial model or this particular implementation. However, we are able to present some initial observations from early use which will be guiding our future work and which other researchers in the field may find useful. We divide these observations into two categories:

- usability - is the system useful to end users?
- architecture - advantages and limitations of our approach.

However, we first briefly describe our laboratory testbed as this has undoubtedly made an impact in both of these areas. We have run MASSIVE on a network of SUN and SGI workstations. Our most powerful graphics engines have been an SGI Indigo 2 Extreme and a SUN 10/51 ZX, both of which can run combined graphics, text and audio clients with adequate performance (but no texturing). Other less powerful machines have been used for text clients with occasional added audio and (slow) graphics. The network has been a standard Ethernet running TCP and UDP. Up to now we have managed sessions of up to six simultaneous users interacting over a mixture of graphics, audio and text. We are planning to test and develop the system for use by many more simultaneous users.

Usability issues

Our first observation is a positive one - the combination of real-time graphics and audio is very natural. Indeed, it actually becomes useful and enjoyable. Most of our real interaction has involved configuring the system while using it and it has been sufficiently useful that we have not had to keep running between adjoining rooms. However, the quality of the audio is variable and this can be frustrating. Our main conclusion here is do not attempt serious VR cooperation without decent audio! [Tang 1993]

Our current limited embodiments have caused problems. Although we can distinguish between different participants and can even guess at their capabilities from their bodies, determining their identity has been problematic. Although in theory everyone could craft their own body, few people have bothered to do this and, even if they did, it would probably take some time before we could recognise individuals on sight. This fits with everyday experience, but in general, better support is needed in the VR area of personal identification.

A more surprising observation concerns inter-working between 3-D graphics users and 2-D text users. Although they are mutually visible within a common

space, their conception of that space seems quite different. In particular, the "texties" seem to lack any notion of personal space and tend to stand directly in front of others or even walk straight through them. In contrast, graphics users tend to maintain a reasonable distance. The problem may be that the graphics field of view is much more limited than the textual one (which is 360 degrees) and so the graphics users are forced to stand back in order to obtain a decent view. On the other hand, it may be that the graphics view is sufficiently rich that people can more easily associate the embodiments they see with other people and so feel compelled to behave in a socially polite manner. Either way, this observation would appear to point towards some deeper issues concerning interaction between users with radically different interfaces in a common space.

Architectural issues

The interface approach to communications seems to work well here. In particular, the combinations of RPCs, attributes (with asynchronous notifications) and streams is very expressive. We argue that a distributed database alone would have difficulty recreating the functionality of this implementation such as the individual treatment of clients/peers and support for continuous media. On the other hand, RPCs alone would introduce extra complications in the handling of attributes, which provide a very natural way of expressing many of the items to be communicated in a VR environment.

We believe the introduction of spatial trading as a means of brokering heterogeneous interfaces to be an important feature of MASSIVE. Indeed, the distributed VR community could benefit considerably from previous work on trading in distributed systems.

Generally, it seems that shared Ethernet is not the best physical medium for this system and that TCP/IP is far from the best networking protocol. A switched medium with built in support for quality of service negotiation and hardware multicasts to specific destinations would be ideal.

This concludes our discussion of the MASSIVE implementation of the spatial model. The following section describes a complementary implementation as an application of the DIVE system.

1.3 DIVE and Mr. Nimbus

Although MASSIVE offers the most general and scalable realisation of the spatial model, it has the disadvantage of requiring significant alterations to whatever VR platform is chosen as its basis. This section describes how a simplified discrete version of the model may be used as the basis for a lightweight implementation at the VR application level. This is a less general approach, but does mean that the model may be relatively easily introduced on top of existing VR platforms. In our case, the existing platform is DIVE.

1.3.1 DIVE

Virtual reality research at the Swedish Institute of Computer Science has concentrated on supporting multi-user virtual environments over local- and wide-area computer networks, and the use of VR as a basis for collaborative work. As part of this work, the DIVE (Distributed Interactive Virtual Environment) system has been developed to enable experimentation and evaluation of research results. The DIVE system is a tool kit for building distributed VR applications in a heterogeneous network environment. In particular, DIVE allows a number of users and applications to share a virtual environment, where they can interact and communicate in real-time. Audio and video functionality makes it possible to build distributed video-conferencing environments enriched by various services and tools.

DIVE users control their own viewpoints and can navigate unencumbered in the environment. Self-representations (embodiments) allow participants to be aware of each other's presence and actions. These representations may range from simple block-like figures up to complex body simulations of real world humans.

The DIVE run-time environment consists of a set of communicating processes, running on nodes distributed within a local area network (LAN) or wide-area network (WAN). The processes, representing either human users or autonomous applications, have access to a number of databases which they update concurrently. Each database contains a number of abstracted descriptions of graphical objects that together constitute a virtual world. Associated with each world is a process group, consisting of all processes that are members of that world. Multi-cast protocols are used for the communication within such a process group [Birman, 91]. Figure 1.4 shows a typical DIVE architecture where several processes, running over a LAN, are members of process groups corresponding to DIVE virtual worlds. These processes are manipulating the world data and presenting it to users through a head-mounted display (HMD) and an ordinary computer screen. The LAN's are connected through a wide-area network (WAN).

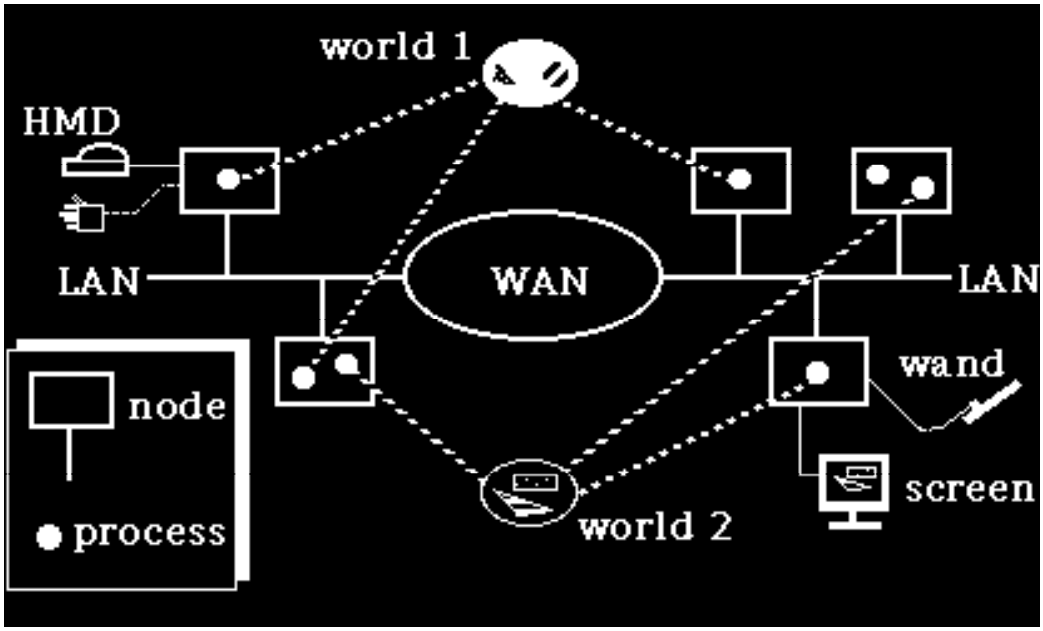


Figure 1.4: Typical DIVE Architecture

DIVE's distribution architecture is based on a peer to peer scheme and consequently there is no central server. Instead, each member of a process group has a complete copy of the world database and when a process joins a certain group, it copies its world data from another member of that group. The information in this replicated database is kept consistent by way of distributed locking mechanisms and reliable Multi-cast protocols [Hagsand, 91]. If there are no other members in the process group then the world state is read from a file. Also, when the last group member exits a world the state is lost unless it has (explicitly) been saved to a file. A process may enter and leave groups dynamically but at a given time it will only be a member of one process group. In VR-terms this means that an object (for example a user-representation) can freely travel between different worlds but it will at any given time only exist in one world.

It is possible to the distinguish between several different types of process in the DIVE environment. We will now briefly discuss the most important ones. For a more exhaustive treatise please refer to [Carlsson, 93] and also the DIVE reference manual.

A *user process* is a process that interfaces directly to a human user, and is therefore often responsible for managing the interaction between the user and the virtual environment.

The *visualizer* is a user process that is responsible for a large part of what a DIVE user encounters in the interface, being the manager of the display devices (immersive and non-immersive), the different input devices, navigational aids and so on.

The visualizer supports various so called *vehicles*. A vehicle is responsible for the mapping of input devices to actions inside the environment (e.g. navigation

and manipulation of artifacts). By having a well defined interface between the vehicles and the rest of DIVE it is possible for the application builder to maintain a certain amount of independence in regard to the hardware and interface capability of their target platforms. Several different vehicles have been developed in DIVE. Presently the two most sophisticated ones are the mouse vehicle and the head mounted display (HMD) vehicle. The HMD vehicle supports a head mounted display and a 3 button "flying mouse", both equipped with 6 degrees of freedom magnetic trackers. New vehicles are continuously being developed and experimented with. Recent examples are a terrain-hugging vehicle and a vehicle for the control of an industrial robot.

Finally, another type of user process is the auralizer [Pope, 93], which is responsible for generating audio output based on spatial localised sound sources.

Application processes on the other hand are typically used for the introduction and subsequent management of objects into the environment. Example tasks are collision detection, animated object behaviours, interfaces to external database services and collaborative applications. Application processes can introduce visible and user accessible objects into the distributed world database or can carry out behind the scenes operations. An important feature of the DIVE system is that it is possible to run application processes on *any* machine available over the network. The machine in question does not have to have any graphical capabilities at all as long as it can access the relevant DIVE world database. This capability makes it possible to build powerful and sophisticated multi-processor applications.

DIVE includes a number of applications that provide general support for cooperative work. For example, Mdraw [Ståhl, 92] is a DIVE application that creates a whiteboard inside a virtual world. A number of users can use this collaborative tool to create and modify two-dimensional graphics material such as text and drawings. When a user moves up to the whiteboard they will find a number of different drawing and manipulation tools available for their disposal. White boards can be used to create *documents*, which are small-sized drawing tools that can be picked up and carried around by users. Documents are basically miniature single user whiteboards. A document can be transferred between users either by explicitly "handing it over" or by some mechanism provided by the environment (for example a conference table). Each whiteboard or document contains a number of "paper" sheets, of which only one is visible. The visible sheet may be turned either forwards or backwards, making another sheet visible. When a document is "extracted" from a whiteboard, the top sheet from the stack is replicated onto the document. From that point on the two sheets will be copies of each other, i.e. any operations performed on one of them will be replicated on the other. This linkage between the two objects can, of course, be broken at will.

The Mdraw whiteboard is a simple but illustrative example of a VR tool for cooperative work. All the participants can easily get a good understanding and overview of the actual work situation, who is present and who is active. Furthermore, the whiteboard application shows the use of soft non-intrusive

access control to a shared resource, i.e. you wait around until a drawing pen (and space in front of the whiteboard) is available.

A second example of general support for cooperative work in DIVE is the inclusion of VR-based video-conferencing. The DIVE video-conference application is intended to combine virtual reality technology with ordinary video and audio technology as well as support for existing conventional computer based tele-conference tools. The audio and video facility makes it possible to associate streams of audio and video data with DIVE objects. Audio data is generated by microphones (typically), and output via speakers/headphones. Video information is handled by the DIVE renderer for integration with the virtual environment as continuously updated texture maps. (An interesting consequence of this is that any object can become a "video-screen".) Thus, in a conference setting participants can participate via a range of different hardware configurations, from sophisticated interfaces like head mounted displays and gloves through more conventional ones like the familiar screen and mouse based interfaces down to more mundane tele/video-phones.

1.3.2 Mr Nimbus

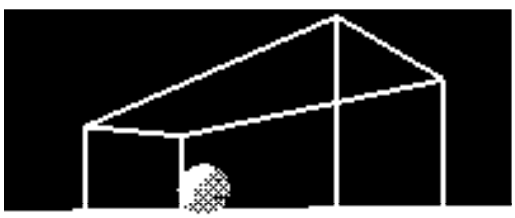
Beyond specific collaborative applications, DIVE also has some built-in support for the spatial model. Aura, focus and nimbus fields are implemented as (usually invisible) graphical objects surrounding users-representations as a kind of outer shell. Interceptions between aura fields are detected by the underlying system using the normal collision detection facility and appropriate event signals are sent to the objects involved (DIVE provides a general mechanism for generating event notifications and passing them onto relevant objects). From that point on, the awareness level between the two objects involved is continuously calculated based on the relative positions of their respective focus and nimbus fields. More specifically, DIVE monitors for collisions between the focus of one object and the nimbus of another and notifies the objects when this happens. On receipt of these event notifications, pairs of objects are able to move between different discrete mutual awareness states. There are four such states corresponding to different arrangements of focus and nimbus: full mutual awareness, no mutual awareness and two asymmetric states. In turn, these mutual awareness states can be used to control how the objects perceive one another across different media. All changes in focus and nimbus collisions generate event signals until the auras are no longer in contact. The following table summarises the four different mutual awareness states that can exist between two objects in DIVE's implementation of the spatial model.

relation between foci and nimbi	Awareness State
A's focus overlaps with B's nimbus and B's focus overlaps with A's nimbus	Full mutual awareness
A's focus overlaps with B's nimbus but B's focus does not overlap with A's nimbus	A aware of B but B unaware of A
A's focus does not overlap with B's nimbus but B's focus overlaps with A's nimbus	A unaware of B but B aware of A
A's focus does not overlap with B's nimbus and B's focus does not overlap with A's nimbus	No mutual awareness

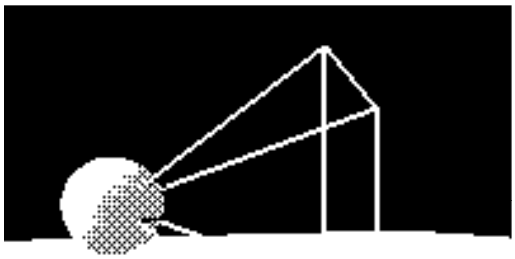
Table 1.1: The different states of mutual awareness

A demonstrator application which exhibits the awareness properties discussed above has been developed which allows a user to experience the effects of aura focus and nimbus through interaction with an imaginary creature called Mr. Nimbus (created by Olov Ståhl at the Swedish Institute of Computer Science). In Figure 1.5 the basic features of Mr Nimbus can be seen. More specifically, there is a face, a pair of ears and eyes and a mouth. Both the user and Mr. Nimbus are equipped with simple graphical representations of aura, focus and nimbus and these are used to trigger and control the volume of an audio signal which is emitted by Mr. Nimbus (examples of these aura, focus and nimbus objects are also shown in Figure 1.5). The awareness function in this demonstrator detects collisions between focus and nimbus objects, resulting in four possible awareness states (full mutual awareness, no mutual awareness and two asymmetric cases).

Example aura



Example nimbus



Example focus

Figure 1.5: Mr. Nimbus, equipped with aura, focus and nimbus

A further interesting feature of Mr. Nimbus is the manner in which his embodiment reflects different awareness states across both the visual and audio media. More specifically, visual cues such as flapping ears showing his audio awareness of us; high-lighted mouth showing our audio awareness of him; widening eyes showing his visual awareness of us and blushing showing our visual awareness of him are used to convey transitions between different awareness states. Figure 1.6 presents some examples of such embodiments.

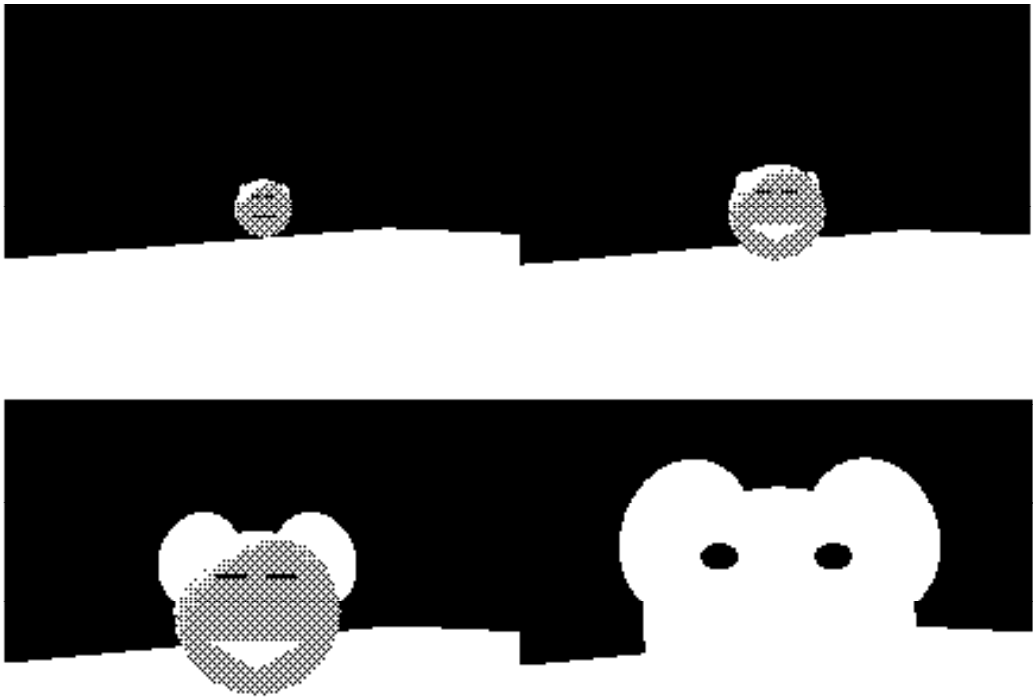


Figure 1.6: The embodiment of Mr. Nimbus provides awareness cues

The DIVE approach to the spatial model differs from that of MASSIVE in two major ways. First, the use of simple graphical objects to represent aura, focus and nimbus allows DIVE to support a simplified version of the spatial model (based on the notion of awareness states) using only standard system facilities such as collision detection and event notification. Given that similar mechanisms can be found in many current VR systems, this suggests a general way of realising the spatial model in such systems without the need for a specialised distributed architecture. Second, the awareness function which combines focus and nimbus in DIVE adopts an "overlapping approach" (i.e. awareness in DIVE is a result of overlapping focus and nimbus) whereas MASSIVE is concerned with relationships between focus and nimbus fields and object embodiments.

Several limitations exist in the present DIVE implementation of the spatial model. The two major ones are (1) each awareness computation involves only two objects and (2) the possibility of only four discrete awareness levels between these two objects, does not take into consideration the amount of overlap between the focus and nimbus objects. How to support a more complete version of the spatial model in a general purpose distributed virtual reality system with potentially a large number of users is currently an issue of debate at the different sites being involved in this research endeavour. The MASSIVE system described in a previous section is a case in point. Still, the functionality offered by the present implementation in DIVE has been instrumental in building interesting and powerful virtual environments.

1.4 Conclusions and Further Work

We have described two implementations of the spatial model. The first, MASSIVE, is based on its own specialised distributed architecture. The second has been implemented as an application of the DIVE system. We believe that both approaches are useful. There are already a number of existing VR platforms in use including academic tools such as DIVE and commercial products such as DIVISION's dVS. It is therefore important to show a route by which spatial model concepts can be integrated into existing systems so as to provide more flexible support for cooperative work. The lightweight route based on awareness states as demonstrated in DIVE achieves this. On the other hand, in order to take full advantage of the support for scalability included in the spatial model, it will be important to develop new distributed architectures for VR - an area in which MASSIVE aims to contribute.

In terms of future prototyping of the spatial model within COMIC (i.e. year three work), both implementations can be improved:

- The DIVE implementation requires extension towards an operational multi-user implementation with integrated audio.
- MASSIVE requires extension to support greater scalability and also needs to consider how its communications model can be integrated with the shared database models used by systems such as DIVE and dVS.

Our experience of implementing the model also indicates some conceptual areas where further work is needed. The most obvious of these is the notion of adapters, how they relate to boundaries and groups, and how they might operate in combination with one another.

Year three of COMIC should also involve some testing and evaluation work. One possibility is to carry out some distributed tests between various COMIC partners. Indeed, a few steps have already been taken in this direction by getting DIVE to operate between Lancaster, Nottingham and Sweden and also in testing out MASSIVE's audio library over wide area links. An interesting possibility might be to hold a strand four meeting in a wide-area collaborative virtual environment by the end of the project - a goal that should be achievable given current progress.

1.5 References

[Birman, 91] Birman, K., Cooper, R., and Gleeson, B., "Programming with process groups: Group and multicast semantics", Technical Report TR-91-1185, Department of Computer Sciences, University of Cornell, January 1991.

[Carlsson, 93] Carlsson, C., and Hagsand, O., DIVE - A Platform for Multi-User Virtual Environments, Computer & Graphics, Vol. 17, No. 6, 1993, pp. 663-669.

[Grimsdale, 91] Grimsdale, C., dVS - Distributed Virtual Environment System, in Proc. Computer Graphics '91, London, ISBN 0 86353 282 9.

[Hagsand, 91] Hagsand, O., “Consistency and Concurrency Control in Virtual Worlds”, Proceedings of the 2nd MultiG Workshop, Stockholm, 1991.

[Pope, 93] Pope, S., and Fahlén, L. E., “The use of 3D audio in a synthetic environment”, Proceedings of VRAIS 1993, Seattle, Washington, September 1993.

[Ståhl, 92] Stahl, O., “Mdraw - A Tool for Cooperative Work in the MultiG Telepresence Environment”, Technical Report T92:05, SICS 1992.

[Tang, 93] Tang, J. and Isaacs E. 1993. Why Do Users Like Video? Studies of Multimedia-Supported Collaboration. *Computer Supported Cooperative Work (CSCW). An International Journal* 1 (3).

[Van Der Linden, 92] Van Der Linden, R. J., and Sventek, J. S., The ANSA Trading Service, in IEEE Distributed Processing Technical Committee Newsletter, Vol. 14, No 1, (Special Issue on Naming Facilities in Internet Environments and Distributed Systems), pp 28-34, June 1992.

Chapter 2

PITS -- Populated Information Terrains

John Mariani
Tom Rodden and
Andy Colebourne
University of Lancaster

Steve Benford
Adrian Bullock and
Dave Snowdon
University of Nottingham

This chapter describes prototyping work carried out for Populated Information Terrains (PITs). We begin by briefly reviewing the concept of PITs arising from the year one COMIC work. Following this, we discuss three prototype DIVE applications which demonstrate different approaches to PITs. The prototypes are VR-VIBE, a statistical clustering based technique which supports browsing and filtering of document collections, VR-Mapper which visualises hierarchical structures of nodes and links, and Q-PIT which builds on the Benediktine approach. Finally some uses of these prototypes in real-world situations are discussed along with directions for year three work.

2.1 Introduction

The notion of PITs -- Populated Information Terrains -- emerged from the year one COMIC work as a natural integration of several disparate areas of research. If we consider the Information Terrain alone, this draws from work on database management systems, information retrieval, data visualisation and virtual reality. An Information Terrain can be constructed by analysing a body of data and applying appropriate mappings from the data and the domains of its values to create a three dimensional graphical representation. Once such a visualisation has been accomplished, the user is free to browse the data, manipulating it in a number of ways. In particular, the terrain should provide visual clues as to data "hot spots" or points of interest which would not otherwise be easy to locate using conventional databases or information retrieval systems. The basic point is that by providing awareness and communication, we can establish collaboration.

Next, we add the concept of a user population (hence *Populated* Information Terrains), so that an information terrain becomes inhabited by multiple users who can interact with each other as well as with the data. This interaction may range from explicitly cooperative activity (e.g. co-ordinated searching) to peripheral awareness of the activities of others (the latter enabling chance encounters and possibly allowing social negotiation of access to the data). This brings us to a point of contention with traditional database management systems. Although such systems have been for several decades multi-user systems, their overriding

concern has been the development of techniques for ensuring data consistency through concepts such as serialisable transactions and locking mechanisms. The upshot of this work has been to deliberately isolate users from one another, giving each the illusion of being the sole user of the data. This isolation is diametrically opposed to the concept of collaboration; how can one collaborate with another when one is ignorant of the other's existence and activities?

The rest of this chapter is structured as follows: having stated the general motivation for PITs, we then consider how to construct an Information Terrain. Four basic ways of doing so are described in the next section. Next, we describe three prototype realisations of the techniques discussed in Section 2.2: namely VR-VIBE, VR-Mapper and Q-PIT. To give examples of the applicability of PITs in real-world collaborative scenarios, we consider a CSCW bibliography visualised in VR-VIBE and Q-PIT, and an examiners' meeting held within Q-PIT, where the data consists of student records. Finally, we end with some early observations and indications of further work.

2.2 Means of Constructing an Information Terrain

In this section, we examine different techniques for generating information terrains from bodies of data. We then compare each technique and discuss its applicability.

2.2.1 Constructing Information Terrains

First we consider the problem of constructing the base information terrains. This involves processing data to arrive at a spatial configuration where the properties of and relationships between data are intuitively obvious from their position and presentation. We identify four broad approaches to this problem and subsequently argue that these are applicable to different types of data

"Benediktine" Cyberspace and TripleSpace: In his work on the structure of Cyberspace, Michael Benedikt argues that the attributes of an object may be mapped onto *extrinsic* and *intrinsic* spatial dimensions [Benedikt, 91]. Extrinsic dimensions specify location in space (e.g. x, y, z co-ordinates). Intrinsic dimensions determine characteristics of the resulting point in space such as colour, shape, size, spin, texture, vibration and sound quality. Using Benedikt's approach, we might extend database schema notations to specify which attributes map to which intrinsic and extrinsic dimensions. An example of an early "Benediktine" cyberspace, TripleSpace [Mariani, 92], has already been built on top of a triple store or binary relational database.

Statistical clustering and proximity measures: Statistical methods have been used to analyse collections of data (often documents) in an attempt to group objects together according to some measure of semantic "closeness" (i.e. do they logically belong together). The resulting proximity measures are typically scaled

and returned as numerical values that are then used to cluster the objects in a data space. Systems adopting this and similar approaches include VIBE [Olsen, 93] and BEAD [Chalmers, 92] (both for visualising collections of documents).

Hyper-structures: Some databases support the notion of explicit relations or links between objects (e.g. schema based on entity-relationship models or hyper-media and Network Information Retrieval systems such as Gopher, ARCHIE, WAIS and World Wide Web). The resulting structures might be visualised by applying three dimensional graph drawing techniques. In turn these might be extended through visualisation techniques such as fish-eye views, cone-trees and perspective walls [Card, 91; Mackinlay, 91].

Human centred approaches. This final approach relies on users or system implementors designing appropriate visualisations. The first approach is to use real-world metaphors. Examples might include fly-through *library* interface for a document store; *cities, buildings* and *rooms* for organisational information in Directory services; and *maps* of the physical world for geographical information. A second approach is to allow humans to construct and organise the information terrain themselves on an ad-hoc basis (effectively how files are organised under the desktop metaphor). This approach might also extend previous work on rooms metaphors in user interfaces [Henderson, 85].

It may be possible to combine the "Benediktine" and human centred aspects by allowing the user to choose the mapping from attribute to dimensions, following the approach taken by Gray et al. in the Iconographer project [Gray, 90].

2.2.2 Comparison and Applicability

The major difference between these approaches is in how they determine the spatial location of data. The Benediktine approach determines location according to the values of three attributes. In the statistical approach, location is determined by some weighted combination of possibly many attribute values in relation to those of other objects. The hyper-structure approach relies on the presence of pre-existing (authored or computed) links which indicate which data objects are neighbours to which others. Finally, users get to chose locations under the human centred approach.

Early experiences suggest that each approach is suited to a different kind of database.

The Benediktine approach relates best to data which is naturally ordered and scaled. Numeric attributes work well. Text may be ordered alphabetically, but this is only useful if this ordering has a semantic meaning to the user. For example, an ordered list of names in a register might be useful for browsing whereas alphabetic ordering of document keywords is unlikely to aid browsing (it does not help that "zebra" is next to "yellow"). The Benediktine approach also has problems with multi-valued attributes. In short, this approach may be most applicable to well structured, ordered and often numeric data collections such as are often managed by relational DBMS.

In contrast, the statistical approach is more suited to data that is less well structured and often highly textual. In particular, this approach aims to uncover important semantic relations between objects that are not known in advance and have therefore not been incorporated into some pre-defined database schema.

Hypermedia structures fall somewhere between the previous two approaches in that they require that relations between objects are made explicit (and therefore at least a basic notion of schema). On the other hand, these relations may be between large often textual data objects. Good examples of suitable applications might be visualising hypertext or filestores.

Human centred approaches are the most radical. They call for the development of tools and techniques for tailoring that are likely to be valuable in the other three approaches. The niche applications for which special worlds are developed will be limited, but the number and diversity of niches is likely to be very large. Development of special tailored worlds with different mappings of attribute to dimensionality lead directly to issues of communication, movement, and linkages between different, diverse worlds that have only been approached conceptually so far (see Chapter 6).

2.3 PIT Prototypes

In this section, we examine three automated approaches to generating Information Terrains. The first, VR-VIBE, implements the statistical approach. The second, VR-Mapper, implements the hyper-structure approach. The third, Q-PIT, implements the Benediktine approach. All three prototypes have been realised as applications of the DIVE system (see Chapter 1).

2.3.1 VR-VIBE

Introduction

VR-VIBE, an application which supports cooperative browsing and filtering of large document collections, uses visualisation techniques to represent data in order to overcome the difficulties and shortcomings of traditional document retrieval systems. VR-VIBE is based on the original VIBE [Olsen, 93], a system which considered two dimensional visualisations in a flat plane. VR-VIBE implements this 2D representation in a 3D space and then goes on to implement the ideas in the third dimension.

The VIBE system was developed at the University of Pittsburgh to support document retrieval from a large collection of documents with the emphasis on being able to provide users with an overview of documents they are interested in with respect to the rest of the document space. A set of queries are specified by defining a number of “Points of Interest” (POIs) which contain keywords. A full text search then takes place on the document space and the keywords are

compared to individual documents producing a relevance score for each document and point of interest. This determines how the space will be laid out; a vector is defined giving the position of each document in space in relation to the POIs. An example VIBE visualisation can be seen below.

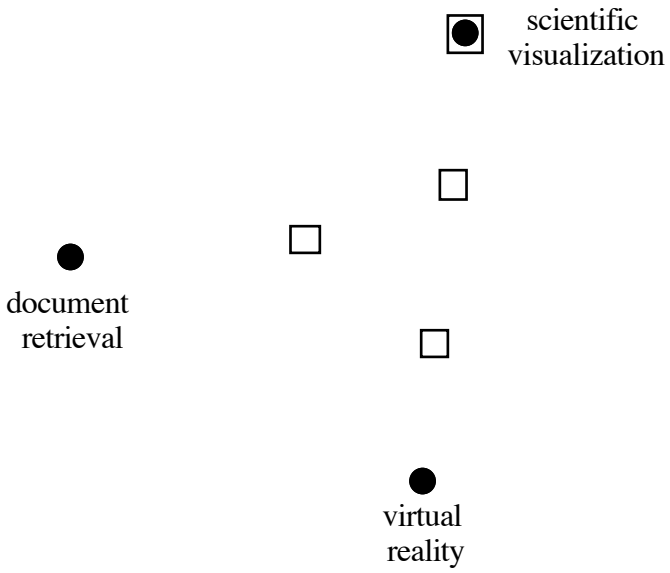


Figure 2.1: A simple VIBE display

In this trivial example three POIs have been specified (the dots) and there are four documents in the document space (the squares). One document is relevant to all three of the POIs and so is placed centrally, the other documents have no relevance to document retrieval but are relevant to scientific visualisation and virtual reality and so appear to the right of the display

Whilst this gives an overview of the relevance of documents to the POIs for small numbers of documents, the display can become very cluttered if a large document space is being examined. VR-VIBE approaches this problem through the use of the third dimension and through the use of filtering mechanisms.

Implementation details

The current version of VR-VIBE takes two files as its input. The first is the data-set to be visualised; currently a bibliography file defined in BibTeX format (and hence also UNIX Refer format as there are converters between the two). The second is a mapping file in which the user defines their set of POIs. The user can define as many POIs as they wish consisting of an arbitrary number of keywords. Individual weightings may be applied to each keyword and each of the BibTeX fields to be searched. VR-VIBE then constructs the visualisation and allows the user to navigate through it, selecting and inspecting individual BibTeX entries. At the end of the session, those entries that remain selected are saved to a separate output file.

The visualisation is based on two key measures: the relative attraction of each document to each POI and the overall relevance of each document to all POIs. The following pseudo-code summarises the algorithms used by VR-VIBE (adapted from those used by VIBE). First, the algorithm for calculating the relative attraction of each document to each POI. We start by calculating a score which represents the number of matches between keywords in the POI and fields in the document, taking weightings into account.

```
for each document, d, in the data-set {
    for each POI, p, in the mapping file {
        score_dp = 0
        for each keyword, k, in POI p {
            for each field, f, in document d {
                score_dp = score_dp +
                    ( number of matches of keyword k in field f *
                     weighting factor for keyword k *
                     weighting factor for field f )
            }
        }
    }
}
```

The relative attraction of a document to a given POI is then the result of normalising this score by dividing by the sum of scores across all POIs.

```
relative attraction of document d to POI p =
                    score_dp / sum of score_dp across all POIs, p
```

The overall relevance of a document is the result of summing its scores across all POIs.

```
overall relevance of document d =
                    sum of score_dp across all POIs, p
```

These results are then used to define a vector position for each document in the visualisation. These vector positions define the display.

There are two possible layouts available in VR-VIBE. The first is a simple extension of the VIBE system with documents residing in a 2D plane, the difference being that the person interrogating the documents has the ability to move around the document space in three dimensions. The second layout has the documents occupying three dimensions in space and it is possible to fly through the document space. Below we see an example of this with a POI near the centre of Figure 2.2, and documents fanning away from it at differing heights. In this

three dimensional representation properties of the icons representing documents have to be used in order to distinguish between documents and to show selection.

In the simple 2D case when a document is selected it suffices to raise or lower the document icon relative to the plane which all the documents occupy. In the 3D case this is not so simple - moving the position of the object any great distance will give a false impression of its relevance to the POIs. Also moving documents in the third dimension does not uniquely identify them as it does with the 2-D case. What actually happens is that colour is used and when a document is selected it moves position slightly and is highlighted. The combination of movement and colour change should be sufficient to catch the attention of the user.

Figure 2.2: Flying through the document space

VR-VIBE permits the user to store views onto the document space and then flip between different views at will. This permits comparison of different views onto the same document space and aids in the task of navigating around the documents (if you get “lost” you can always return to your starting point).

Having briefly introduced the VIBE concept and the VR-VIBE system which extends this concept, some example document spaces will be given with descriptions of which parts of the system they demonstrate.

Examples

In this first screen shot (Figure 2.3) we see a flat two dimensional document space. The documents are represented by rectangular block icons while the POIs are tetrahedral in shape. A query with five POIs has been defined and the documents placed in the space according to the scores held in the internal tables relating documents to POIs.

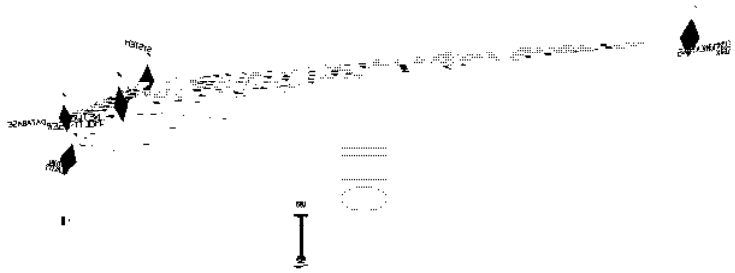


Figure 2.3: A query with five POIs in three dimensions - flat view

Just glancing at the picture shows that the majority of documents are relevant to the query points on the left side of the screen. While it is possible to move around this space and concentrate on the areas which we are most interested in this 2D view is still limited.

It is difficult to tell which POIs some of the documents relate to, for instance a document icon might be positioned between two POIs, 1 and 2, but also incidentally be between POIs 5 and 4 as well. Which pair does it relate to ? Obviously choosing where to position the POIs is a great importance, but by introducing a third dimension to the display it becomes easier to identify which documents are relevant to which POIs with less uncertainty.

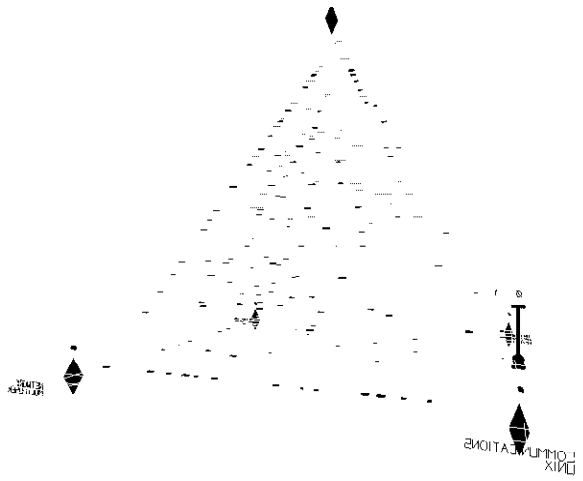


Figure 2.4: A query with five POIs in three dimensions - using the third dimension

Figure 2.4 shows the same document space as in Figure 2.3 but this time one of the POIs has been placed vertically above the other four forming a three dimensional pyramid shape. This use of the third dimension simplifies and structures the display. You only have to glance at the two images to see which is

the least cluttered. Again, the choice of which POI goes where will effect the positioning of the icons on the display.

There is still a large amount of information on the screen and the user may well only be interested in the most relevant documents to their query and are not too interested in the less relevant ones initially. This is where filtering comes in; filtering techniques allow for just the most relevant documents to be displayed. This simplifies the display and filtering can be tailored by specifying the degree of relevance required before a document is displayed. The user is able to tell the system to only display documents which most closely match certain search criteria.

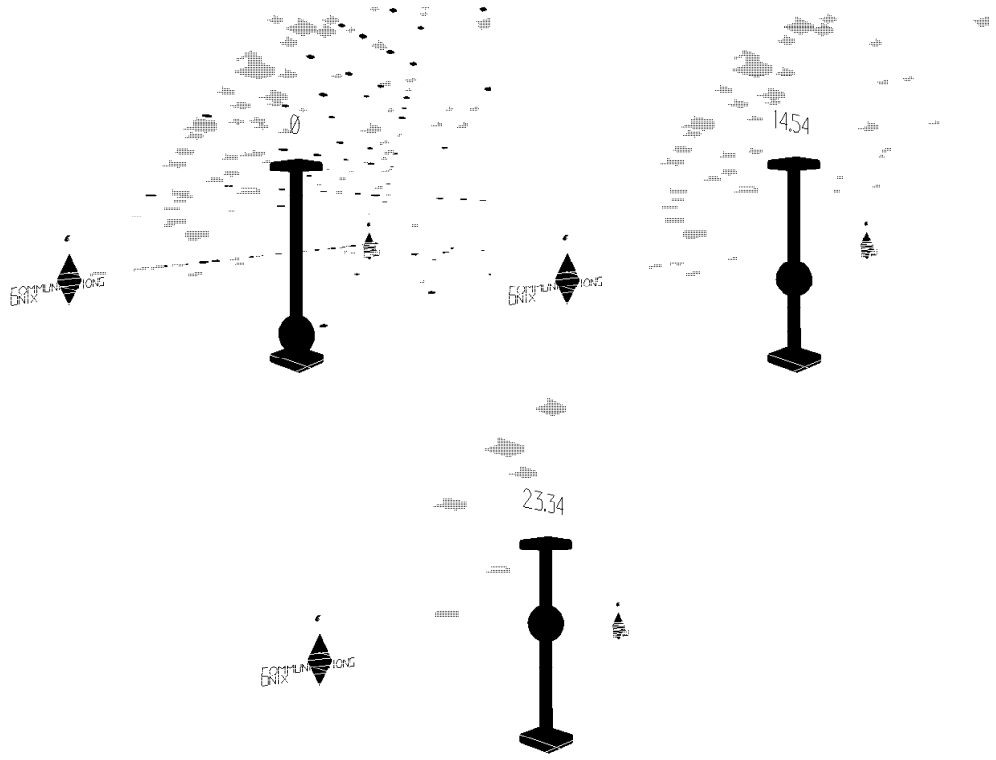


Figure 2.5: Filtering in VR-VIBE

Figure 2.5 shows three images from the same corner of the display in Figure 2.4. A widget controls how much information is displayed by allowing the user to set the relevance score; only documents of greater relevance than the score are displayed. In the first picture all documents in the space are displayed, whilst in the second only those whose relevance is greater than 14.54 are displayed - clearly a smaller subset. Finally the third image shows only those documents which are highly relevant to the search criteria. This powerful filtering mechanism permits fast searching of document collections for relevant information.

A user is directly embodied as a DIVE “blockie”, an embodiment which is used in the other prototypes also. Figure 2.6 shows a document store with four users (including ourselves). Two of the users are to the right of the pyramid while the third user is situated a small distance from the bottom left side of the pyramid.

Through each user's telepointer, a device which shows where a person's attention is and is also used for interacting with other objects, it is possible to observe not only where people are situated in space but also what they are doing. A sense of their activity is conveyed by the direction of their attention and the information they are interacting with. These telepointers are not plainly obvious in the image below, but manifest themselves as thin white lines emanating from the head of the "blockie".

Activity is also conveyed by changes in the document space as it is browsed. Icons will change colour or position as they are selected and these changes are available to everyone in the same document space. Hence, a combination of awareness of users and awareness of the document space itself conveys a sense of the ongoing activity with the document collection.

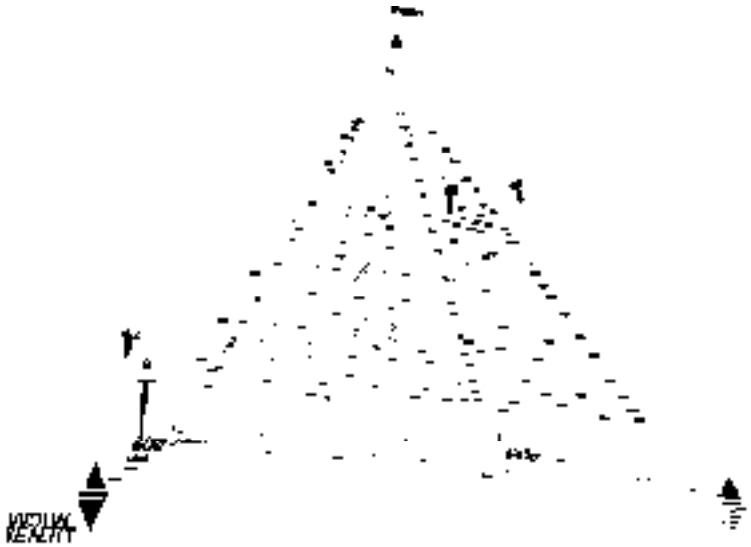


Figure 2.6: Embodiment within VR-VIBE

2.3.2 VR-Mapper

Introduction

The VR-Mapper tool originates from a conferencing system called CyCo [Benford, 93a] which is based upon a rooms metaphor. While this metaphor provided a good conferencing setting, the task of navigating around the conference space was not simple. Therefore the VR-Mapper was born, to provide a visual representation of a conferencing space and also provide information about the space.

Each conferencing world is described by a data file which enumerates all rooms and connections between rooms. The early VR-Mapper took this data file as input and displayed the information in the form of a spanning tree drawn in a

flat 2D window. This 2D representation was fine when small data files were to be visualised, but for larger spaces the visualisation could get very complicated and confused very quickly. An initial extension was to format the output of VR-Mapper to consist of concentric circles of nodes centred on the parent node for each subtree. While this improved matters slightly there was still a problem in visualising large data spaces. Thus the 3D VR-Mapper was developed which could not only deal with the complexity of large conferencing spaces, but which could also be employed to represent other data and not just the conferencing data for which it was developed in the first place.

Implementation details

The implementation consists of three modules, a data input module which currently reads in CyCo data files and UNIX filestore information, a map making module which uses algorithms to determine how the data will be laid out spatially, and a visualisation module which first of all generates the virtual world information and then displays this information in a user interface. The system should be easily extendible as each of these modules is independent of the others so to add further functionality to the system it is just a case of replacing or expanding the relevant module.

The VR-Mapper can make use of three different algorithms when laying out data. The first is the standard tree structure (Figure 2.8) in two dimensions. A variation on this algorithm is where a spanning map is drawn and the nodes are placed on concentric circles centred on the root node. All nodes which would have occurred on the first level of the tree are placed on the innermost circle, the second level on the second circle out and so on (Figure 2.7). The third algorithm utilises the advantages of using the third dimension. It is similar to the concentric circles algorithm except that each parent node appears raised in height with respect to its children (Figures 2.7 & 2.9). This algorithm is known as the cone tree algorithm.

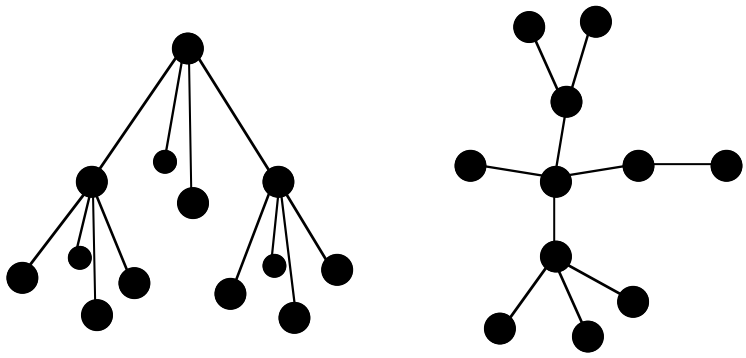


Figure 2.7: Layouts used by VR-Mapper

Any of the nodes in the visualisation are available for selection by the user. When a node is selected first of all the iconic representation in the display begins to rotate, showing that the node has been selected. Next the information which the node represents is displayed to the user. This can take many forms dependent upon what information is actually being displayed by VR-Mapper. In the case of Conferencing data files then the information displayed might be a description of the selected room together with a list of applications available in that room and which other rooms can be reached from it.

When the 3D cone tree layout algorithm is used, arbitrarily large data spaces can be displayed. Spaces with upwards of 10,000 nodes have been displayed using VR-Mapper and indeed the biggest drawback in displaying large data spaces is the computational power needed to drive the display with that much information on the screen. Presently this is a cost issue more than anything else.

A number of convenience functions exist in VR-Mapper. First it is possible to set up viewpoints on the data being observed. This way when you get to one point of interest, want to reference it, but also want to examine other parts of the data then you can simply set a view point and return to it whenever required. It is also possible to rotate the data space around a central axis so that rather than the user having to go to the far side of the data, the data comes to the user. This allows for different perspectives on the same piece of data to be had.

Some example images from VR-Mapper offer the best way of describing the results available using VR-Mapper, and the advantages it offers in the way it can structure data in different ways.

Examples

The following examples demonstrate the features afforded by VR-Mapper when it is used to represent information. All of the screen shots have been taken while VR-Mapper is visualising UNIX filestore information.

The first screen shot shows the traditional 2D tree structure approach for representing data. The information shown is a small section of a user’s home filespace. The parent directory is the root node of the tree and there are a number of sub-directories and files below the parent. The display is quite spread out laterally and it is very difficult to get an overview of the entire directory structure from close up.

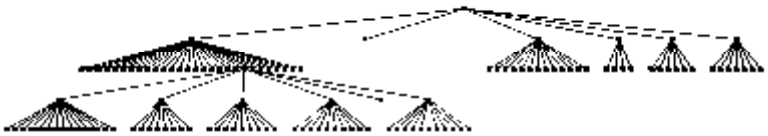


Figure 2.8: A Tree representation of an area of the UNIX filestore

The second picture (Figure 2.9) shows the same information as in Figure 2.8, but uses the three dimensional cone tree algorithm to represent the information. The information is displayed in a much more compact way than in Figure 2.8. Rotating the tree gives us different views and we can move up and down the hierarchy in 3D space.

When a node is found which is of interest then it is possible to select the node by clicking on it with the mouse pointer. This causes the selected node to rotate and in the case of UNIX filestore information the selected file is opened using an editor. Figure 2.10 shows VR-Mapper with a node having been selected (in the top leftmost cluster of data), the node being source code for an application, and the file is displayed in an emacs editor window. It is now possible to make any changes to the file as required, provided of course that you have the requisite permission to do so.

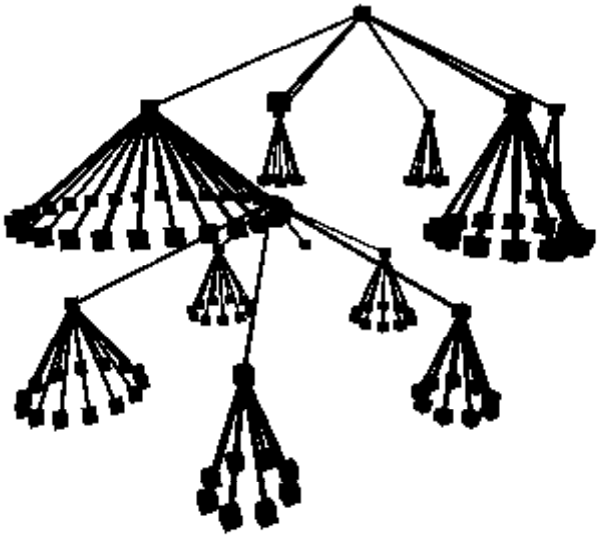


Figure 2.9: The same filestore data represented using a cone tree

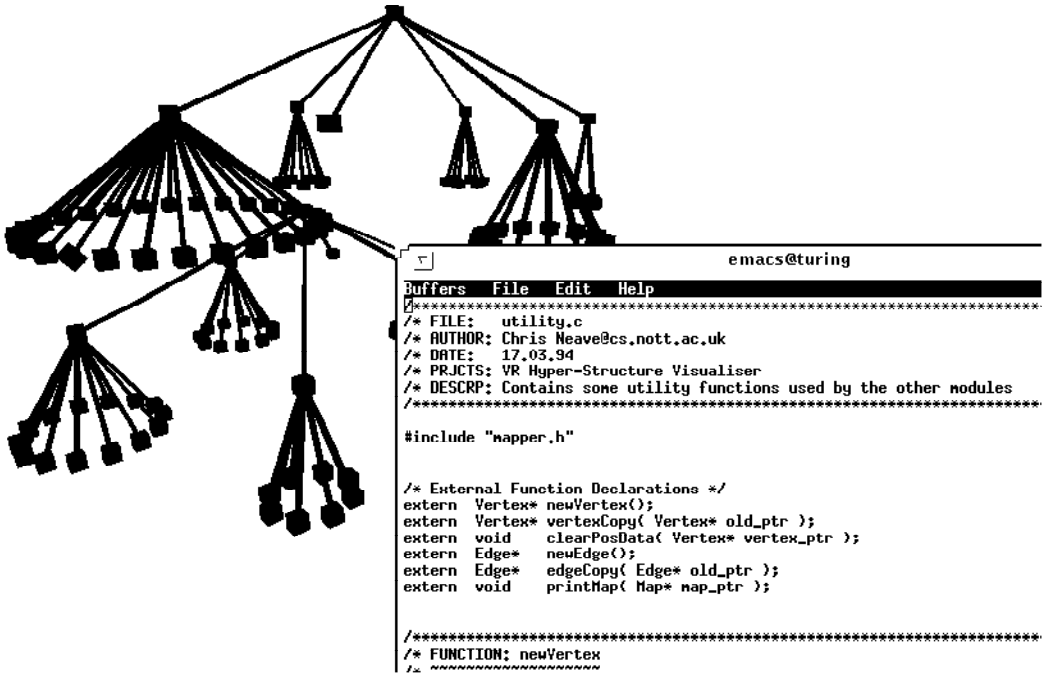


Figure 2.10: Examining one of the files

A text editor was used to open the text file, but it should be possible to automatically fire up the appropriate application to view each node as required.

As mentioned previously it is possible to move through the data being visualised by VR-Mapper. Figure 2.11 shows the user having moved to being inside the data space being examined.

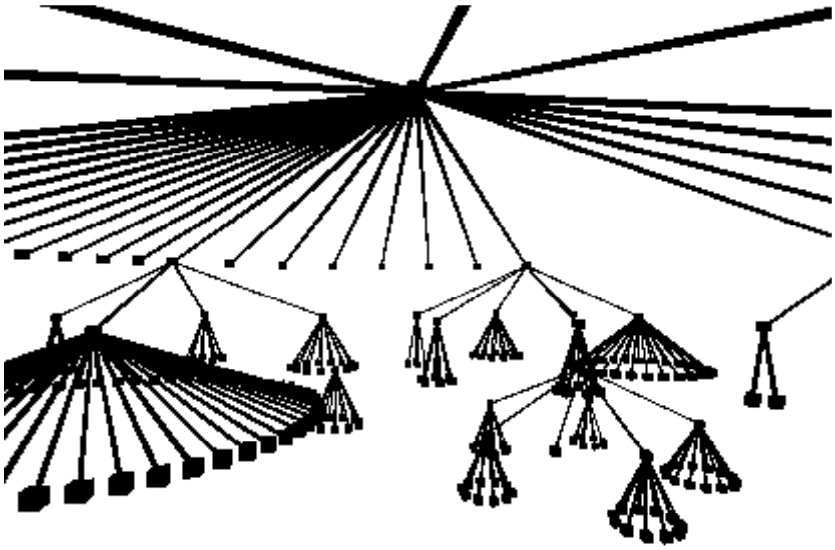


Figure 2.11: Inside the directory, the root node at the top of the picture

Finally to suggest that this approach is scalable and capable of dealing with complex information spaces a visualisation of a 5000 node directory structure is presented (Figure 2.12). This represents the system manual pages for a Silicon Graphics IRIX 4.0.5H system. Two areas where large amounts of information is stored are easily identifiable and these can be mapped onto two directories under */usr/catman*, the *a_man/cat1* directory and the *u_man/cat1* directory. These contain the manual pages for user commands which are the most common type of command documented on the system.

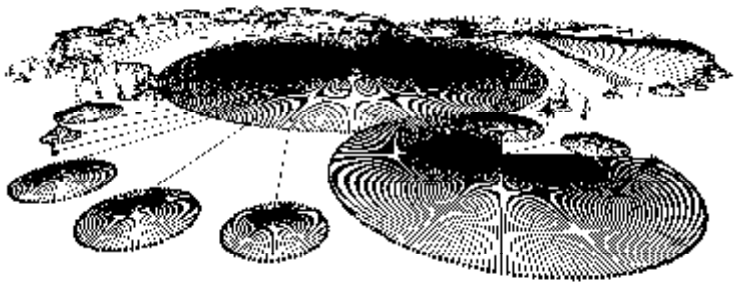


Figure 2.12: Part of the UNIX filestore with 5000 nodes

2.3.3 Q-PIT

Implementation

The “Benediktine” demonstrator, Q-PIT, was originally implemented on a SUN Sparc 10 workstation using the WorldToolKit virtual reality library. It has recently been ported to DIVE which will be the platform for further development.

Currently, Q-PIT can process a simple database containing a number of named tuples, display them within a three dimensional space and then allow this space to be shared and manipulated by multiple users.

Attribute mapping and schema in Q-PIT

Q-PIT maps tuples onto graphical objects according to an extended schema notation. As an example, we will consider a database which contains tuples representing people with the following domains: (name, age, gender, location, occupation). For each domain in the database, we build and maintain a list of values appearing within that domain. Each domain list is then sorted into alphabetical order. An attribute mapping file might then be created containing the following additional schema information:

extrinsic	location	x
extrinsic	occupation	y
extrinsic	name	z
intrinsic	age	spin_speed
intrinsic	gender	shape male cube
		female cone

We can see the mappings from the attributes location, occupation and name onto the extrinsic (x, y, z) axes. At the moment, these mappings are linear, being based on the ordering of the domain list. So, for example, if the name list is (Adrian, Bruce, Gordon...), then the tuples containing “Bruce” as the value of their name field are mapped onto co-ordinate 2 of the z axis. The intrinsic mappings currently supported are **shape**, **height** and **spin_speed**. Current shapes include cubes, spheres, diamonds, pyramids, cones, cylinders and hemispheres. We can associate a field and a field value with a shape; this dictates the shape of the graphical object which represents the tuple. In our simple example, we have associated the field “gender” with the shape mappings, followed by a list of (field value, shape) pairs. If a field value does not appear in the mapping, the shape defaults to a cube. It is possible to have as many (field value, shape) pairs as required, with duplication of shape mappings if necessary. If the tuple has the value “male” for field “gender”, then the matching graphical object has a cube shape. Similarly, if the tuple is “female”, the object is cone shaped.

Spin_speed dictates whether the object should rotate, and if so, how quickly. In the example, we map the integer value of a person’s age onto the speed. Similarly,

height dictates the height of an object. We could, for example, map age onto height; thus, the taller the object, the older it is.

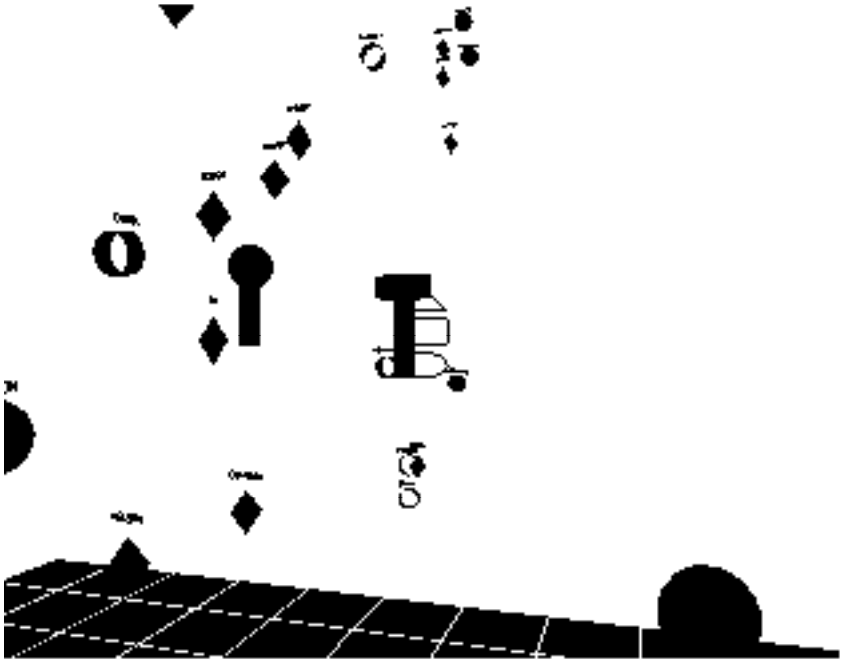


Figure 2.13: A Q-Space of people

Data manipulation in Q-PIT

Data may be manipulated in a number of ways. First, by selecting an object via the mouse, the underlying tuple is displayed. Second, by issuing a query, all objects "hit" are automatically selected and highlighted. Thirdly, a user can update the values of the underlying tuple by engaging in an interactive sequence of textual instructions. Once an update has been completed, not only is the underlying tuple updated, but so is the corresponding graphical object. This may entail alterations to both intrinsic and extrinsic attributes, the latter taking the form of a *smooth animation* of the object moving to the new position. We believe that the display of smoothly animated changes in position as opposed to instant repositioning to be particularly effective at providing people with peripheral awareness of changes to nearby data. Lastly, again through textual interaction, a brand new tuple can be added to the data. This is matched by the creation of a new graphical object.

Populating Q-PIT

A user is directly embodied as a DIVE "blockie" (a large, thin cuboid). Associated with each user is a unique colour, currently specified in their "user.profile" file. When two or more users are sharing a Q-space (the space defined by a Q-PIT), they see the others as differently coloured "blockies". Whenever a Q-PIT session is commenced, the users start in the same location, but

are subsequently allowed to change position freely and independently of each other, so that everyone will eventually have a distinct view of the shared Q-space. Users may either use a Spaceball (an input device for movement with six degrees of freedom) or the more conventional mouse to navigate. By selecting a "blockie", a user can obtain information about the other user. Direct communication currently has to be achieved though external applications including text and audio conferencing.

This simple visualisation of fellow users immediately provides an awareness of presence which is not normally available in traditional databases. Moreover, by their positioning within the Q-space, it is possible to identify the areas or clusters of data in which other people are currently interested. To provide awareness of activity, whenever the user selects an object, the object changes to the selecting user's colour. This allows other users to see what objects are currently in use and by whom.

2.4 Scenarios

In this section, we give a brief description of two scenarios explored thus far with the prototype PITS; these include a CSCW bibliography and an examiner's meeting.

2.4.1 A CSCW Bibliography

In the earlier section on VR-VIBE, a CSCW bibliography was used as an example. It seems readily apparent that, by giving the user the possibility of positioning their POIs in three dimensions rather than two, the results of a scan of the bibliography can be much more understandable. In a populated VR-VIBE, it is easy to see how multiple users might cluster round POIs and enter into cooperation regarding existing documents within the bibliography.

By way of comparison, the same bibliography was used as the basis of a Q-PIT visualisation. Clearly, we envisage that the different visualisation techniques will be suitable for certain application areas (as discussed earlier in this chapter), and the results of the experiment appear to back us up.

In order to prepare the data for suitable use with Q-PIT, some pre-processing of the BibTeX format file was required, in order to produce simple tuple representations of each document. For example, the extensive abstract information was discarded (this is a property of the very simple "tuple store" currently being used to support Q-PIT). Next, for our positional mappings, we chose journal title, authors, and year of publication. As year of publication (in the CSCW field) is a relatively short domain, this resulted in a very narrow, almost 2-D Q-Space. Moreover, authors gave us a large domain with a great deal of uniqueness. For example, if a CSCW author is called J.T. Kirk, but writes papers with combinations of M. Scott and L. McCoy, this results in distinct authorship of each

paper (there is no recognition of multiple authors). This could be addressed with further pre-processing of the data.

The point remains that at first glance VR-VIBE (as one might expect) is more readily suitable for the visualisation of a bibliography. With much more effort, we might be able to produce a more powerful pre-processing and subsequent mapping that could allow Q-PIT to produce a more competitive visualisation.

2.4.2 Examiners' Meeting

A much more suitable scenario for Q-PIT is that of an examiners' meeting. The data involved is based on exam marks and degrees awarded. In this example, shape was dictated by degree awarded. The positional attributes were based on the results of the final year project, the database exam, and type of major. In the example data, there were only three types of major so this again produced an almost 2-D display.

This example highlights the power of different mappings. For example, we could have used three exams for all three axis and achieved a wider spread. We could use degree awarded as one positional axis and shape to reflect type of major. It seems clear that we should improve the potential for dynamic mapping and also for user-specified mapping functions (i.e. take the average mark for a group of related exams and use that as a positional attribute).

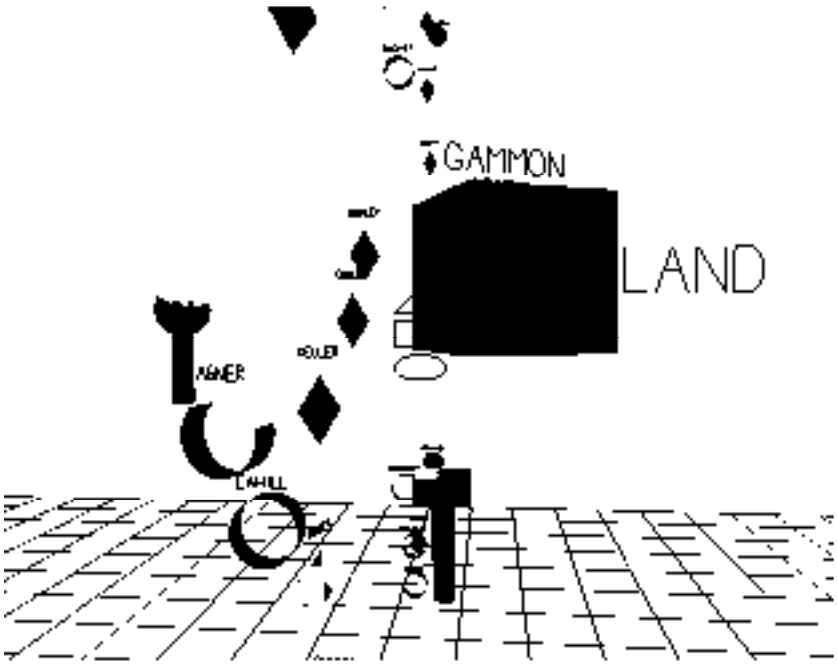


Figure 2.14: Examiners in Q-Space meeting

We can readily imagine a number of examiners co-habiting this Q-Space to analyse, discuss, and modify marks. The external examiner could indeed be operating from their home university.

2.5 Further Work

2.5.1 Evaluation and Assessment

First and foremost, we aim to carry out an evaluation of PITS as a means of supporting data visualisation and sharing. This will involve comparing the three prototypes when they are presenting different information structures. This has been done to a small degree with Q-PIT and VR-VIBE visualising the same CSCW bibliography file to differing success levels. Q-PIT has also been used to examine an examiners meeting scenario. A more thorough assessment is required however which can provide input into the further development of revised versions of the current prototypes and new ones.

A series of information spaces, be they database records, document stores, Usenet News, etc. need to be identified. Each of the spaces can then be visualised using the different prototypes and the resulting presentations compared. The usefulness of each visualisation technique for different information spaces should become apparent.

2.5.2 Representing Query and Results

The information displayed by the PITs prototypes is formatted according to some query definition. In VR-VIBE this is the definition of the POIs which then affect where the documents will be positioned in the display. In Q-PIT object positions are defined by a triple of attributes which locate the object in space. The VR-Mapper displays information from a hierarchy definition.

Exactly what each of these queries are is not explicitly stated in the presented display and some form of visual query definition would be a useful feature. To some extent this has been addressed in the VR-VIBE system where it is possible to reposition POIs dynamically, the rest of the document space moving to incorporate the changes in real-time. It is the possibility of the results of a query being presented to a user in a real-time dynamic nature with the space restructuring itself that is of most interest here.

2.5.3 Populating Information Terrains

Much of the work to date has concentrated on the Information Terrain with less emphasis placed on the idea of Population. The population of information terrains with multiple users centres on the idea that users are as much a part of the database as is the data. Put another way users should be *directly visible* as objects within a shared database and should not be relegated to the status of external agents whose presence is merely implied as a side effect of their actions. Population raises a host of further issues.

Communication between users

First, PITS should support direct communication between their inhabitants, allowing them to discuss and negotiate access to data. Such communication should be both real-time, involving the possible use of audio, video and text channels, and asynchronous, such as the ability for inhabitants to annotate data with messages for other people (e.g. a kind of "post it" note facility). In turn, this requires techniques for initiating, managing and terminating communication. One approach for densely populated PITS is to initiate direct communications whenever two users enter close proximity to one another. This supports the notion of bumping into people who are working with the same data as yourself. Of course, it should also be possible to communicate with other more distant users if required.

Awareness

Beyond direct communication, we need to consider the notion of *awareness* [Gaver, 92]. Peripheral awareness of other people plays a vital and subtle supporting role in cooperative work, enabling ad-hoc interaction, promoting knowledge of the state of activity, and supporting important behaviours such as monitoring and overseeing. PITs should automatically and continuously provide awareness of the presence and actions of other inhabitants. Awareness of action involves showing what data inhabitants are accessing and also what kind of access is being made (thus, reading might be distinguished from deletion). In the long term, a range of more sophisticated CSCW awareness mechanisms such as [Benford, 93a; Benford, 93b] might be used to allow people to control their levels of awareness of others.

This notion of awareness should also apply to past action. Thus, not only should one be made aware of what changes have taken place in a database since last access, but one should also know who was responsible for these changes. Put more generally, in order to support cooperative data sharing, databases must notify users of the presence, location, ongoing activity and past activity of other users. This represents a direct challenge to the traditional view of database access transparency.

Embodiment

Awareness of presence raises the further issue of embodiment; that is, users need to be directly represented in the space alongside the data. This requires the provision of *appropriate* body images. The design of these virtual bodies may be constrained by a variety of factors, a few of which we list here:

- body images should convey presence, location and orientation.
- body images should convey identity.
- body images should convey activity and availability for communication.
- body images should support voluntary gesture and perhaps even involuntary expression.

- body images should be personalisable (i.e. tailorable by users).
- body images should reflect the capabilities of the users (e.g. “ears” suggest that they can receive audio messages).
- body images should be truthful - that is they should not lie about the above (especially identity).

Embodiment is dealt with in more detail in Chapter 3.

2.6 Summary and Conclusions

A Populated Information Terrain is an abstract data space that allows multiple inhabitants to visualise and jointly manipulate the contents of databases. This chapter has explored theoretical issues concerned with how PITs might be constructed and then populated for a range of database types. The aim has been to explore the notion of PITs both as a means of improving the way in which users browse and interact with data and also as a means of actively supporting cooperative data sharing. To begin with the concept of a PIT was introduced, what it is and why it is of importance. A number of different types of PIT were discussed, including Benediktine spaces, statistical structuring and hyper-structures, with the relative merits of each approach emphasised.

Next three prototype PIT visualisation systems were described in some detail, the VR-VIBE system for document browsing, which provides strong relevance and filtering mechanisms, VR-Mapper, which visualises hierarchical information and Q-PIT, a database front end. Experiences were given from the use of each of these prototypes to visualise different information structures. Two real world situations which the prototypes have been used for were also presented. While these early prototypes are sufficient to demonstrate the basic principles of PITS in a tangible form, further development is required before more meaningful evaluation can be made.

It is still very much early days as far as drawing any meaningful conclusions from the work undertaken so far on the prototypes. Certainly they present information in a different, visual form to users but how this presentation can be used to improve cooperative working is not clear. More development and a thorough evaluation of their use is required before such conclusions can be drawn about their prospective benefits and gains.

There are a number of improvements which can be made to move the prototypes forward and these include:

- addressing user embodiment, where currently crude embodiments are used which look like data from afar, and so introduce confusion into the display. A few simple steps would overcome some of these problems including the use of a slightly more humanoid shape (the DIVE distributed VR systems shows just how simple such a shape can be and yet still convey the impression of representing a human [Fahlén, 93]).

- connected with embodiment is the ability to allow users of a space to personalise their own embodiment, but care must be taken to ensure a truthful embodiment which can be supported by the application is chosen.
- improving the possibility for users to communicate with each other while they occupy the same data space. Currently you can obtain a description of the other people occupying the same space, but it is envisaged to provide simple direct communication using tools such as the text based UNIX talk in the first instance, and later providing more sophisticated connections such as audio or video.

To conclude, a number of prototypes have been developed and these demonstrate some of the different approaches available for Populated Information Terrains. It is too early to say anything conclusive about the success of these prototypes, evaluation is planned during the final year of the project. Desirable extensions and additions to the prototypes have been identified and these will be addressed in year three.

2.7 References

[Benedikt, 91] Benedikt, M., Cyberspace: Some Proposals, in Cyberspace: First Steps, Michael Benedikt (ed.), MIT Press, 1991, pp 273-302.

[Benford, 93a] Benford, S., Bullock, A., Cook, N., Harvey, P., Ingram, R. and Lee, O., From Rooms to Cyberspace: Models of Interaction in Large Virtual Computer Spaces, in Interacting with Computers, Vol. 5, no 2, pp 217-237, Butterworth-Heinemann, 1993.

[Benford, 93b] Benford, S. and Fahlén, L.E., A Spatial Model of Interaction in Large Virtual Environments, in Proc. Third European Conference on Computer Supported Cooperative Work (ECSCW'93), Kluwer, 1993.

[Card, 91] Card, S. A., Robertson, G.G., and Mackinlay, J.D., The Information Visualiser, an Information Workspace, in Proc. CHI'91, Human Factors in Computing Systems, ACM SIGCHI, May 1991, pp 181-188.

[Carlsson, 93] Carlsson, C. and Hagsand, O. (1993). Dive - a Platform for Multi-user Virtual Environments, *Computers and Graphics*, Vol. 17, No. 6, pp. 663-669.

[Chalmers, 92] Chalmers M. and Chitson, P., Bead: Explorations in Information Visualisation, in Proc. SIGIR'92, published as a special issue of SIGIR forum, ACM Press, pp. 330-337, June 1992.

[Fahlén, 93] Fahlén, L. E., Brown, C. G., Stahl, O., Carlsson, C., A Space Based Model for User Interaction in Shared Synthetic Environments, In Proc. InterCHI'93, ACM Press, 1993.

[Gaver, 92] Gaver, W., Moran, T., MacLean, A., Lovstrand, L., Dourish, P., Carter, K., and Buxton, W., Realising a Video Environment: EuroPARC's RAVE System, In Proc. CHI '92 Human Factors in Computing Systems, Monterey, Ca., USA, May 1992.

[Gray, 90] Gray, P.D., Waite, K.W. and Draper, S.W., "Do-It-Yourself Iconic Displays", Human-Computer Interaction -- INTERACT '90, D. Diaper et al (Eds)., Elsevier Science Publishers B.V. (North-Holland), pp. 639 - 644, IFIP, 1990

[Henderson, 85] Henderson, A.J., and Card, S.A., Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention, ACM Transactions on Graphics, Vol. 5, No. 3, July 1985.

[Lamport, 86] Lamport, L., *Latex, A document preparation system*, pp139-148, Addison-Wesley, (1986).

[Mackinlay, 91] Mackinlay, J.D., Robertson, G.G. and Card, S.K., Perspective Wall: Detail and Context Smoothly Integrated, in Proc. CHI'91, Human Factors in Computing Systems, ACM SICGHI, May 1991, pp 173-179.

[Mariani, 92] Mariani, J., and Lougher, R., TripleSpace: an Experiment in a 3D Graphical Interface to a Binary Relational Database, Interacting with Computers, Vol. 4, No. 2, 1992, pp 147-162.

[Mariani, 94] Mariani, J.A., "Design of a 2D PIT", COMIC Internal Report, Computing Department, Lancaster University, 1994

[Olsen, 93] Olsen, K.A., Korfhage, R. R., Sochats, K. M.., Spring, M. B. and Williams, J. G., Visualisation of a Document Collection: The VIBE System, Information Processing and Management, Vol. 29, No 1, pp. 69-81, Pergamon Press Ltd, 1993.

[Rodden, 92] Rodden, T., J.A. Mariani and G. Blair, Supporting Cooperative Applications, the International Journal of Computer Supported Cooperative Work (CSCW), Vol. 1, Nos. 1-2, 1992, Kluwer.

[WTK, 93] WorldToolKit 2.0 Reference Manual, Sense8 Corporation, 4000 Bridgeway #101 Sausalito CA 94965, 1993.

Chapter 3

Embodiment

Steve Benford and Chris Greenhalgh
University of Nottingham

Lennart Fahlén
Swedish Institute of Computer Science

John Bowers
University of Manchester

This chapter addresses the issue of embodying users within collaborative virtual environments. More specifically, this means identifying design issues for providing users with *appropriate* body images in cooperative settings and describing mechanisms for supporting each of them. The design issues uncovered include representation of presence, location, identity, activity, viewpoint, actionpoint, gesture, facial expression, voluntary and involuntary expression, capability, degree of presence and history of activity. They also include more general issues such as truthfulness, representation across multiple media and efficiency. The chapter then describes how our current prototype systems DIVE and MASSIVE address these issues and goes on to look at how embodiment is supported in other multi-user VR systems and other kinds of CSCW technology. The longer term aim of this work is to provide designers with a "body builders work-out" for designing useful embodiments.

3.1 Introduction

This chapter discusses the issue of embodiment of people within shared virtual spaces; in other words, providing people with *appropriate* body images so as to represent them to others and to themselves. We focus on embodiment of human users. However, embodiment of non-human objects may involve many of the same issues. The aim of the paper is to clarify key design issues and then to list and explore possible design choices.

The need for appropriate embodiment of users was identified in chapters two and three of the year one strand four deliverable [Comic, 93]. To recap, a problem with some existing CSCW technologies is that users are not always directly represented to each other. For example, in the case of video, problems may arise when one person can watch another without being seen themselves. The observer may be hidden behind a camera and their presence may not be known. Even if their presence is signalled (e.g. playing a door sound when initiating a glance mechanism in media-space), their identity may not be known. Even if their identity is signalled, their current degree of activity or presence may not be known or sufficiently "felt" by the person being observed. Or in the case of shared window systems, users may have a very limited embodiment in terms of a telepointer. However, no distinction is made between the point where they are

manipulating with this pointer and where they are attending. Also, this embodiment is extremely limited in terms of its representational capabilities. In short, we argue that the *inhabitants of shared spaces ought to be directly visible to themselves and to others through a process of direct and sufficiently rich embodiment.*

The need to directly embody users seems to have been more widely accepted for virtual environments. The issue with these systems is therefore not if, but how. In other words, are the body images provided appropriate to supporting collaboration? To go a step further, as opposed to merely discussing appearances of virtual bodies, we should focus on providing some sort of "marionette" with active autonomous behaviours together with a series of strings which the user is continuously pulling as smoothly as possible.

The topics discussed in the following sections constitute an overlapping and often conflicting set of issues. Designing an appropriate body image will therefore be a case of maintaining a sensible balance between them. Furthermore this balance will probably be application and user dependent and will no doubt be constrained by available computing resource. As a final introductory comment, perhaps the major issue underpinning these is the tension between realism (i.e. designing bodies that reflect our physical form) and abstraction (i.e. providing entirely new images, unconstrained by the properties of the real-world).

3.2 Requirements and Design Choices

The following list of issues represents a first attempt at classifying what this information might be. We suspect that the relevance of a given issue will be both application and user dependent and that designing a virtual body will therefore involve identifying the most important issues and trading them off against the available computing resource. Given more experience, it may eventually be possible to refine this list into some kind of "body builders work-out".

3.2.1 Presence and Location

The primary goal of a body image is to convey a sense of someone's presence in a virtual environment as well as their location (including both *position* and *orientation).*

3.2.2 Identity

Body images might convey identity at several distinct levels of recognition. First, it could be easy to recognise at a glance that the body is representing a human being as opposed to some other kind of object. Second, it might be possible to distinguish between different individuals in an interaction, even if you do not know who they are. Third, once you have learned someone's identity, you might

be able to recognise them again (this implies some kind of temporal stability). Fourth, you might be able to find out who someone is via their body image. Underpinning these distinctions is the time span over which a body will be used (one conversation, a few hours or permanently) and the potential number of inhabitants of the environment (from among how many people does an individual have to be recognised?). Allowing users to personalise body images is also likely to be important if collaborative virtual environments are to gain widespread acceptance

3.2.3 Activity, Focus, Nimbus and Availability

Body images might convey a sense of on-going activity. For example, position and orientation in a data space can indicate which data a given user is currently accessing. Related to the idea of conveying activity is the idea of showing availability for interaction. The aim here is to convey some sense of how busy and/or interruptable a person is. Bodies might also convey where in space people are currently focusing and projecting their nimbus so as to support the spatial model.

3.2.4 Gesture and Facial Expression

Gesture is an important part of conversation and ranges from almost sub-conscious accompaniment to speech to complete and well formed sign languages for the deaf. Support for gesture implies that we need to consider what kinds of "limbs" are present. Facial expression also plays a key role in human interaction as the most powerful external representation of emotion, either conscious or sub-conscious. Facial expression seems strongly related to gesture. However, the granularity of detail involved is much finer and the technical problems inherent in its capture and representation correspondingly more difficult. A crude, but possibly effective approach, might be to texture map video onto an appropriate facial surface of a body image (e.g. the "Talking Heads" at the Media Lab [Brand, 87]). Another approach involves capturing expression information from the human face using an array of sensors on the skin, modelling it and reproducing it on the body image (e.g. the work of ATR where they explicitly track the movement of a user's face and combine it with models of facial muscles and skin [Ohya, 93]).

This discussion of gesture and facial expression relates to a further issue, that of voluntary versus involuntary expression. Real bodies provide us with the ability to consciously express ourselves as a supplement or alternative to other forms of communication. Virtual bodies can support this by providing an appropriate set of limbs and strings with which to manipulate them. The more flexible the limbs; the richer the gestural language. Involuntary expression (i.e. that over which users have little control) is also important (looks of shock, anger, fear etc.). However, support is technically much harder as it requires automatic

capture of sufficiently rich data about the user. This is the real problem we are up against with the expression issue - how to capture involuntary expressions.

3.2.5 Degree of Presence

Virtual reality can introduce a strong separation between mind and body. One consequence of this separation is that a virtual body may be present in space when the mind behind it has gone out of the office for a few seconds (particularly with screen-based “desktop” virtual reality). This could cause a number of problems such as the social embarrassment and wasted effort involved in one person talking to an empty body for any significant amount of time. As a result, it may be important to convey the degree of actual presence in a virtual body (e.g. by increasing translucence or closing eyes on a face).

3.2.6 History of Activity

We also need consider historical awareness of presence and activity. In other words, conveying who has been present in the past and what they have done. Clearly we are extending the meaning of “body” beyond its normal use here. An example might be carving out trails and pathways through virtual space in much the same way as they are worn into the physical world.

3.2.7 Manipulating One’s View of Other People

To what extent can the observer control the representation of the observed? Can you make someone else invisible to yourself? If so, how do *they* know that you can not see them and what, if anything, can they do about it? Can you select from among a set of possible body images for another person the one that most suits your needs? A good example might be choosing the representation that best suits your style and capabilities of interface and equipment.

3.2.8 Representation Across Multiple Media

Up to now we have spoken mainly in terms of visual body images. However, body images will be required in all available communication media including audio and text. For example, audio body images might centre around voice tone and quality, be it that of the real-person or be it artificial.

3.2.9 Autonomous and Distributed Body Parts

We have discussed virtual bodies as if they are localised within some small region of space. We also need to consider cases where people are in several places at a time, either through multiple direct presence or though some kind of computer agent acting on their behalf (e.g. issuing a database query as in the previous chapter on Populated Information Terrains).

3.2.10 Efficiency

Body images should be efficient by conveying all of the above in as simple a way as possible as both processing power and bandwidth are likely to be limited resources. More specifically, we need to support “graceful degradation” so that users with less powerful hardware or simpler interfaces can obtain sufficiently useful information without being overloaded. This suggests prioritising the above issues in any given communication scenario.

3.2.11 Truthfulness

This final issue relates to nearly all of those raised above. It concerns the degree of truth of a body image. In essence, should a body image represent a person as they are in the physical world or should it be created entirely at the whim or fancy or its owner? We should understand the consequences of both alternatives, or indeed of anything in between. Examples include: truth about identity - can people pretend to be other people?; truth about facial expression - imagine a world full of perfect poker players; and truth about capabilities - this body has ears on, can they hear me? One the one hand, lying can be dangerous. On the other, constraining people to the brutal physical truth may be too limiting or boring. The solution may be to specify a *gradient* of body attributes that are increasingly difficult to modify. Those that are easy require relatively little resource. Those that are not require more. For example, changing virtual garments might be easy whereas changing size or face or voice might be difficult.

3.3 Early Prototypes

This section briefly considers how some of the above issues are reflected in our current prototype systems.

3.3.1 Embodiment in DIVE

A variety of bodies have been implemented within the DIVE system. The simplest are the “blockies” which are composed from a few basic graphics objects. The general shape of blockies is intended to be sufficient to convey presence, location and orientation. In terms of identity, simple static cartoon-like facial features suggest that a blockie represents a human and the ability for people to tailor their own body images supports some differentiation between individuals. A more advanced DIVE body for immersive use texture maps a static photograph onto the face of the body, thus providing greater support for identity.

The use of a line extending from a DIVE body to the point of manipulation in space represents actionpoint. Immersive blockies support a moving head which tracks the position of the users head in the real world. This is very effective at conveying viewpoint, general activity and degree of presence.

On the other hand, non-immersive blockies do not currently support any notion of degree of presence, which has led to some interesting experiences. One side-effect of DIVE is that occasional application and system crashes can leave corpses behind - i.e. uninhabited blockies. One of the conventions that has emerged from early experimental use is that, on encountering a possible corpse, one picks it up and gives it a shake. An angry reaction means that the body is still occupied!

In the DIVE PITS applications, each user may be associated with a different colour which is used to show which data they are accessing (selected objects change to this colour), thereby providing peripheral awareness of their activity.

Finally, video conferencing participants can be represented in DIVE through a video window.

Figure 3.1 shows a range of DIVE bodies including a blockie (centre), a video conferencing participant (centre right) and more humanoid forms with textured faces (extreme left and right).

Figure 3.1: Assorted embodiments attend a DIVE conference

3.3.2 Embodiment in MASSIVE

MASSIVE currently only provides simple blockies, although as with DIVE, the user can tailor their own body image [Greenhalgh, 94]. Given MASSIVE's heterogeneity, a major goal of these embodiments is to convey users' capabilities. For example, considering the graphics interface: an audio capable user has ears; a desk-top graphics user (monoscopic) has a single eye; an immersed stereo user would have two eyes and a text user ("textie") has the letter "T" embossed on their head.

In the text interface, users are embodied by a single character (typically the first letter of their chosen name) which shows position and may help identify

users in a limited way. An additional line (single character) points in the direction the user is currently facing. Thus, using only two characters, the MASSIVE text interface attempts to convey presence, location, orientation and identity.

Figure 3.2 shows the use of embodiment in MASSIVE to convey information about users' capabailities. We can see from their bodies that only two of them can receive audio (the two facing each other across the table). We can also tell at a glance that the person facing us is a text user.

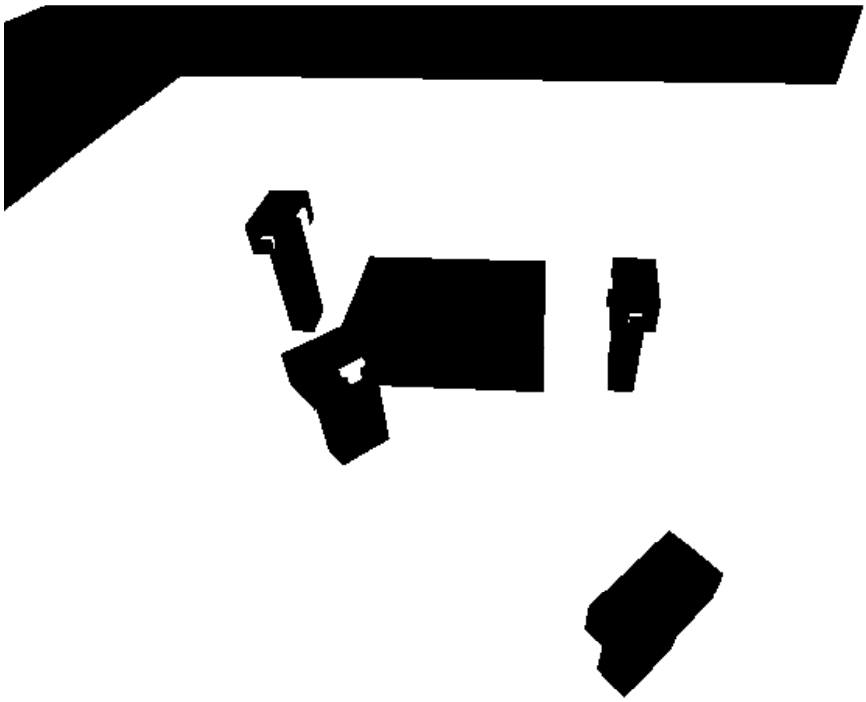


Figure 3.2: Embodiments show their capabilities at a MASSIVE conference

3.4 Embodiment and other CSCW Technologies

Next, we briefly examine the body images provided by a range of existing systems, matching them up to the criteria listed above. Our intention is that designers of future systems could perform a similar exercise and so gauge the likely effectiveness and limitations of their proposed body images for cooperative work.

The selected systems span a broad range of collaborative workspaces including other shared virtual reality systems, three dimensional design spaces, video conferencing systems and shared window systems and design surfaces. The last two of these discuss generalisations of a range of systems as opposed to a specific system.

3.4.1 dVS

dVS, from DIVISION Ltd, is one of better known and more powerful commercial virtual reality system on the market [Grimsdale, 93]. dVS supports multi-user networked virtual reality applications running on both DIVISION's own hardware and on Silicon Graphics machines. Users may operate in either immersive or desktop modes. The default embodiment in dVS is currently rather limited, involving only a disembodied head and a single limb (e.g. a 3-D telepointer).

Presence	users are directly represented.
Location	The use of head and hand tracking support some notion of general location and orientation although the lack of a body linking the two make this hard to discern at times.
Identity	no specific support.
Activity and availability	no specific support.
History of activity	no specific support.
Viewpoint	supported through head-tracking.
Actionpoint	supported through hand-tracking.
Gesture	supported through the tracked hand only (and the representation of the hand as a pointer severely limits this ability).
Facial expression	not supported.
Voluntary versus involuntary expression	not supported.
Degree of presence	not really a problem in immersive mode; not supported in desk-top mode.
Reflecting capabilities	not supported.
Physical properties	not supported.
Active bodies	not supported.
Time and change	not supported.
Manipulating your view of others	not supported.
Representation across multiple media	not supported.
Autonomous and distributed body parts	not supported.
Truthfulness	given sufficient programming ability, the user can define their own body as they wish.
Efficiency	not explicitly supported.

3.4.2 Collaborative Workspace

The ATR lab (part of Japan's NTT) have been exploring the use of virtual reality to support cooperative work for some years [Ohya, 93]. The main thrust of their research has been on supporting two-party teleconferencing and, in particular, on automatically capturing and reproducing facial expressions. Their collaborative workspace prototype achieves this by attaching a video camera to a head-mounted frame which also supports a position tracker. The use of small reflective disks attached to the user's face allows automatic analysis of their facial movements from the video image. This is then used to animate a texture mapped model of the user's face.

Presence	users are directly represented as humanoid looking forms.
Location	from the video recording we have seen, it appears that the user occupies a relatively fixed overall position (e.g. seated at a table).
Identity	the aim is to make the user look as much like themself as possible using a human head model onto which a photographic image of the user is textured and then animated.
Activity and availability	no specific support.
History of activity	no specific support.
Viewpoint	the users head position is tracked and represented as are the positions of their eyes. Thus, this system is one the very few to convey gaze direction at a very detailed level.
Actionpoint	the user wears a single data glove and the position of one hand is therefore tracked.
Gesture	supported through the tracked hand.
Facial expression	this appears to be the primary focus of this work and a reasonably sophisticated range of facial expressions are possible through the use of tracked mouth, eyebrows and eyes.
Voluntary versus involuntary expression	both are supported.
Degree of presence	this is not really a problem due to the use of immersive technology.
Reflecting capabilities	not supported.
Physical properties	unknown.
Active bodies	not supported.
Time and change	not supported.
Manipulating your view of others	not supported.
Representation across multiple media	the users own voice is used through the audio channel.
Autonomous and distributed body parts	not supported.
Truthfulness	unknown.
Efficiency	does not appear to be a key requirement of the project given the nature of the powerful graphics super-computers used.

Complimentary, and equally impressive, work on the capture and reproduction of facial expressions has been reported by Thalmann [Thalmann, 93]. In this case, the user is not constrained to wearing a head-mounted camera or any facial ‘jewellery’ or special make-up. The advantage of this is clearly a lack of intrusiveness. However, the disadvantage is the inability to combine facial expressions with head tracking. Also worth a mention is the recent TV star Ratz the Cat, an blue cat face that introduces UK children’s TV who is animated in real time by an actor wearing a complicated facial waldo.

3.4.3 Doom

Doom is a multi-user virtual reality game for networked PCs. Doom allows its users to navigate through a maze of corridors and rooms killing everything that they meet using a variety of weapons. The multi-user version can either be played in death-match mode (i.e. scoring points for killing each other) or, most interestingly, in cooperative mode (i.e. scoring points for killing other things together). The graphics in Doom are excellent, realising navigable texture mapped environments on a 486 platform. In order to achieve this level of graphics performance, the designers of Doom have placed some necessary constraints on their virtual worlds such as restricting them to use only perpendicular surfaces. Indeed, this is what makes the issue of embodiment in doom particularly interesting; here we have a system where efficiency is of very great importance.

Presence	users are directly represented as humanoid looking forms.
Location	each user has a location and a limited number of orientations. Doom portrays users using flat 2-D textures which face the observer. Swapping between several such textures showing the user from different angles (North, South, East and West) gives a rough sense of orientation.
Identity	other users (player characters in gaming terminology) are distinguished from computer generated monsters (non-player characters). Users are individually distinguished from each other through the use of colour.
Activity and availability	the activity of firing weapons is clearly shown.
History of activity	no specific support.
Viewpoint	only supported through rough orientation as described above.
Actionpoint	the impact point of weapons is shown, as is the trace of projectiles for some weapons (e.g. the rocket launcher).
Gesture	not supported.
Facial expression	this is not supported in other people. However, the user does see a separate self image which shows how healthy they are and which also grins disconcertingly whenever they pick up a powerful weapon.
Voluntary versus involuntary expression	neither supported.
Degree of presence	there is no mistaking a dead player in Doom.
Reflecting capabilities	not supported.
Physical properties	bodies are solid and cannot be picked up.
Active bodies	not supported.
Time and change	not supported except for the users self image where improvements in health are portrayed.
Manipulating your view of others	not supported.
Representation across multiple media	there are not multiple media.
Autonomous and distributed body parts	not supported.
Truthfulness	to the extent that people cannot alter their body images.
Efficiency	this is where Doom excels; the whole system is an exercise in achieving maximum possible functionality with extremely limited resources.

Another multi-user VR game worthy of note is the Legend Quest Dungeons and Dragons game from Virtuality. Legend Quest users are immersed and embodiment is based on the character type being played (e.g. warrior, sorcerer or thief). Thus, bodies show capabilities, position, orientation, viewpoint and

actionpoint. A particularly interesting feature of Legend Quest is its multi-media embodiment where users voices, as heard over the live audio, link are electronically altered to reflect the character they are playing. For a discussion of characters in Virtuality's VR games, see [Waldern, 93].

3.4.4 Teledesign

Shu and Flowers reported a prototype system for collaborative 3-D design at CSCW'92 [Shu 94]. The system provides a 3-D environment for multiple users to construct designs from basic graphical objects. As well as a navigable perspective view, the system allows designers to "snap" to pre-set orthogonal views. Early experiments with the system led to the central notion of needing to directly represent the viewpoints of other users in the shared design space.

Presence	users are directly represented as small pyramids.
Location	each pyramid has a specific location and the pyramid shape conveys general orientation. This is emphasised by drawing a line from the tip of the pyramid to the current position on the design object where a designer is attending (the viewpoint idea).
Identity	no support is described for identifying different users.
Activity and availability	activity is implicitly conveyed through the viewpoint.
History of activity	no specific support.
Viewpoint	explicitly supported (after all, this is where the concept originated).
Actionpoint	not supported.
Gesture	not supported.
Facial expression	not supported.
Voluntary versus involuntary expression	neither supported.
Degree of presence	not supported.
Reflecting capabilities	not really relevant.
Physical properties	unknown.
Active bodies	not supported.
Time and change	not supported.
Manipulating your view of others	not supported.
Representation across multiple media	there are not multiple media.
Autonomous and distributed body parts	not supported.
Truthfulness	is supported because people cannot easily alter their body images.
Efficiency	the current bodies are very simple.

3.4.5 Video

As opposed to considering any specific video conferencing system, we focus on the general nature of embodiment within video as a communication medium.

Presence	the presence of the person in front of the camera is clearly represented. However, in situations where there are one way connections (e.g. media space “glances” or surveillance cameras), the presence of the person behind the camera may not be.
Location	the physical location of a user may be shown to some degree. However, there is no real sense of a common location (i.e. you cannot place many people in relation to each other). The same is true of orientation. Other than knowing whether they are facing the camera or not, you cannot tell where someone is looking. First, if they are looking off camera, what are they looking at? Second, in groups of more than two people, who are they looking at if they peer into the camera?
Identity	is conveyed nearly as well as in the real world (subject to picture resolution problems). Personalisation requires altering your physical self (e.g. putting on a hat).
Activity and availability	It may be possible to tell whether someone is busy or not but not what they are doing. Several researchers have investigated techniques for displaying availability to make a video connection (e.g. metaphors such as “doors”).
History of activity	no specific support.
Viewpoint	not really supported, although you might be fooled otherwise (the orientation issue from above).
Actionpoint	not supported.
Gesture	supported as in the real world subject to field of view constraints.
Facial expression	supported!
Voluntary versus involuntary expression	supported via facial expression!
Degree of presence	not really an issue with video.
Reflecting capabilities	not so much an issue.
Physical properties	there is very little opportunity for any kind of physical interaction across the video connection.
Active bodies	not supported.
Time and change	only natural ageing.
Manipulating your view of others	not supported.
Representation across multiple media	not applicable.
Autonomous and distributed body parts	not supported.
Truthfulness	generally enforces the brutal truth as the is little chance to break away from the real person’s appearance. Some more advanced systems may allow some manipulation of video images.
Efficiency	debatable. Requires at least a reasonable bandwidth.

3.4.6 Shared Windows, Editors and Design Surfaces

There are a wide range of multi-user 2-D interfaces in existence including shared windowing systems, editors and design surfaces [Greenberg, 92]. Typically, such systems provide very limited embodiment in the form of a telepointer and perhaps the use of different colours to distinguish different users. The over-riding impression given by such systems is of the user being external to the shared space, non-embodied and looking down onto it through a window that is their computer screen.

3.5 Summary

This chapter has identified the following list of issues relevant to the embodiment of users in virtual environments: presence, location, identity, activity, availability, history of activity, viewpoint, actionpoint, gesture, facial expression, voluntary versus involuntary expression, degree of presence, reflecting capabilities, physical properties, active bodies, time and change, manipulating your view of others, representation across multiple media, autonomous and distributed body parts, truthfulness and efficiency. It has also reviewed how these issues are currently reflected in a range of technologies including multi-user VR (our own and other systems), 3-D design spaces, video and shared drawing surfaces.

We suspect that the importance of any given issue will be both application and user specific and that the art of virtual body building will involve identifying the important issues in each case and supporting them within the available computing resource. However, at the present time, this list remains merely a framework for the discussion and exploration of embodiment. Year three COMIC work should aim to support a larger number of these issues within our own DIVE and MASSIVE systems, gaining deeper insights into their importance and possible implementation. In the longer term, this work might even generate a cook book, allowing the choice of the most appropriate designs for the available equipment, application, users, scale and longevity of intended CSCW systems.

3.6 References

[Benford, 94] Benford, S., Bowers, J., Fahlén, L. E., and Greenhalgh, C., Managing Mutual Awareness in Collaborative Virtual Environments, *Proc. Virtual Reality Systems and Technology (VRST) '94*, August, 1994, Singapore.

[Brand, 87] Brand, S., The Medialab — Inventing the future at MIT, Viking Penguin, 1987, ISBN 0-670-81442-3, pp. 91-93.

[Comic, 93] Benford, S. and Mariani, J. (eds), Requirements and Metaphors of Shared Interaction, *COMIC Deliverable D-4.1*, Available from the Department of Computing, The University of Lancaster, Lancaster, LA1 4YR, UK, ISBN 0-901800-31-7.

[Fahlén 93] Lennart E. Fahlén, Charles Grant Brown, Olov Stahl, Christer Carlsson, *A Space Based Model for User Interaction in Shared Synthetic Environments*, in Proc.InterCHI'93, ACM Press, 1993.

[Greenberg, 92] Greenberg, S., Roseman, M., Webster, D., Bohnet, R. 'Human and Technical factors of distributed group drawing tools'. *Interacting with Computers,* vol.4, no. 3,1992, pp.364-395.

[Greenhalgh, 94], Greenhalgh, C. and Benford, S., MASSIVE: Model, Architecture and System for Spatial Interaction in Virtual Environments, Report available from Department of Computer Science, The University of Nottingham, Nottingham, NG7 2RD, UK.

[Grimsdale, 93] Grimsdale, C., Supervision - A Parallel Architecture for Virtual Reality, in *Virtual Reality Systems*, Earnshaw, R.A., Gigante, M.A and Jones, H. (eds), Academic Press, 1993, ISBN 0-12-227748-1.

[Ohya, 93] Ohya, J., Kitamura, Y., Takemura, H., Kishino, F., Terashima, N., Real-time Reproduction of 3D Human Images in Virtual Space Teleconferencing, *in Proc.VRAIS'93,* IEEE, Seattle Washington September, 1993, pp. 408-414.

[Shu 94] Shu, L., and Flowers, W., Teledesign: groupware user experiments in three-dimensional computer-aided design, Collaborative Computing, Vol. 1, Issue 1, Chapman & Hall, 1994.

[Takemura 92] Haruo Takemura and Fumio Kishino, *Cooperative Work Environment Using Virtual Workspace*, In Proc. CSCW'92, Toronto, Nov 1992, ACM Press.

[Thalmann, 93] Thalmann, D., Using Virtual Reality Techniques in the Animation Process, in *Virtual Reality Systems*, Earnshaw, R.A., Gigante, M.A and Jones, H. (eds), Academic Press, 1993, ISBN 0-12-227748-1.

[Waldern, 93] Waldern, J. D., A Note on Software Design of Virtual Team-Mates and Virtual Opponents, in Proc. London Virtual Reality Expo '94 (VR'94), February 1994, London, Mecklermedia, ISBN 0-88736-972-3.

Chapter 4

An Approach to Access Control for Spatial Systems

Adrian Bullock and Steve Benford

University of Nottingham

Security is a critical, but usually neglected, aspect of multi-user computer systems. This chapter proposes the basis of an approach to access control for spatial systems, with suggestions for further application areas at the end. Our approach combines two concepts; i) access control, and ii) the concept of **boundaries** [Bowers, 93], which segment space into regions. The result is a new approach to access control where access is governed according to the space within which subjects and objects reside and whether a subject can traverse space to get close to an object.

The idea of boundaries segmenting space and acting as barriers results in the notion of an **access graph** - an abstract representation of the possibilities of movement around a given spatial environment. Furthermore, the application of standard mathematical techniques to this graph promises to allow us to answer a variety of security related questions about the environment and also to predict the access control implications of changes to the environment.

We also discuss how our approach can support the idea of group access control by allowing boundaries to choose how they combine individual properties to arrive at a set of properties representative of the group. Finally an example instantiation of the model is described in some detail, the application of the clearance-classification access model to boundaries.

4.1 Introduction

Security is always a critical issue for any software system that will be used in the real-world. This is particularly true where the system is to be shared between multiple users.

In this chapter, we propose an approach to access control for spatial systems. This focus on access control means that we are interested in general techniques for managing individual and group access to resources and are ignoring, for the time being, related security issues such as authenticating user's identities and auditing their actions.

There are already a range of approaches to access control in use in current computer systems including access control lists, capabilities and the clearance-classification model. We briefly review these in section 4.2.

Section 4.3 summarises our proposed approach to access control for spatial systems, based on a combination of the notion of boundaries, attributes and properties associated with the boundaries and mathematical graph theory.

Section 4.4 revisits the concept of boundaries as the fundamental units in our model.

Section 4.5 then shows how we can analyse a virtual space defined by a combination of boundaries to arrive at the notion of an "access graph" which provides an abstract description of the possibilities for traversal of this space. This section also describes how standard mathematical techniques can then be used in conjunction with this access graph to answer key security related questions.

Section 4.6 extends our approach to include the idea of managing "group access" to resources in a cooperative environment.

Section 4.7 explores one possible instantiation of the general model outlined previously. The clearance-classification access model is used to provide property checking at boundaries and an example scenario is presented.

Finally, Section 4.8 considers future issues and summarises the work presented.

4.2 A Very Quick Tour of Access Control Models

We begin by briefly reviewing existing access control mechanisms for general purpose computer systems. At the most basic level, access control is concerned with managing the actions which different users can take on objects. Probably the best known access control model is the access matrix model [Lampson, 74] which has acted as the basis for many of the access models which have been developed since the early 1970s. Under the model, a matrix is constructed where the rows represent users, the columns objects and a given matrix cell the permitted actions of a given user on a given object.

This matrix can be sub-divided in two ways, by row or by column, resulting in two derived approaches. The first of these, capabilities, is where a row of the matrix is associated with each user in the system. This row, called a capability, acts as a ticket, enabling this user to access various objects. The second, is where each column is associated with the relevant object. This approach is called Access Control Lists and is probably the most widespread access control model in current use (a prime example is its use in the UNIX filestore).

Later models such as the clearance classification model [Bell, 73] have been developed to consider in more detail the semantics of the information being controlled. In essence, the clearance classification model attaches different values (classifications) to specific actions on objects, and corresponding values (clearances) to subjects. In order to undertake a given action, the clearance of the subject wishing to undertake the action must be at least the classification of the action itself.

4.3 Our Approach

Our approach to access control in spatial systems is fundamentally different to those described above. Instead of associating permissions with either subjects (as in capabilities) or objects (as in access control lists), we associate them with the spaces which subjects and objects inhabit. In other words, the access security of an object depends not so much on what the object is or who the subject is, but rather on where the object is currently located and whether the subject can get into this space*. Although at first sight this might seem a rather crude approach, we argue that it may actually lead to a high degree of subtlety and flexibility and is also particularly well suited to space-based environments. As further justification, we argue that our approach mirrors many aspects of access control in the real-world. For example, locking valuable objects in rooms, safes and boxes.

We now briefly consider the components that make up our model before proceeding to a more detailed description.

Boundaries - the fundamental component of the model is the boundary [Bowers, 93]. Boundaries control both traversal of and awareness across regions of space. A key aspect of the proposed model is the ability to specify a set of properties or criteria (attributes) which are held by the boundary. These must then be met at the boundary before authorised traversal can take place. The boundary is a fundamental tool in the model.

Access graph - if we combine the concept of a boundary with the criteria required to be satisfied in order to traverse the boundary in an authorised manner (credentials) then we arrive at the notion of an access graph. The graph represents all the different routes possible for traversal of a space together with requirements needed to make these traversals. If user credentials are mapped into numeric form then the application of standard mathematical techniques to this graph allow us to answer a range of key security questions for spatial environments such as, what credentials do I need to move from region A to region B? what is the minimum guaranteed security of a region relative to any other? and by what route could I move an object from region A to region B so as to maintain at all times its current level of access security?

Groups - a key aspect of the proposed model is that it explicitly deals with the notion of groups in relation to boundaries and hence access control. In essence, we extend the boundary concept to deal with situations where a group of individuals try to simultaneously cross the boundary by proposing how individuals' properties may be combined to arrive at a single group representation.

The following sections explore each of these components of the model in detail.

---

* This is not to say that the properties of who and what will be ignored - indeed these will provide valuable information for credential checking at boundaries. Rather the emphasis is being placed on the spatial aspects.

4.4 Boundaries

Previous work within the COMIC project has proposed the concept of boundaries as a way of structuring space and influencing both awareness and traversal [Bowers, 93]. To recap:

- a boundary may effect both traversal and awareness;
- effects may be of four sorts: obstructive, non-obstructive, conditionally-obstructive and transforming;

This description suggests the existence of sixteen fundamental types of boundary. However, we can now generalise in two further ways:

- the awareness effects of a boundary may behave differently with respect to different media;
- the behaviour of a boundary may vary according to the direction of traversal (in the simplest case we can think of a boundary as separating two regions of space and therefore being bi-directional).

Thus, a very wide range of specific boundary types may in fact exist in spatial environments. The exposition that follows makes several simplifying assumptions. First, we consider only one medium. Second, we will not consider transforming effects of boundaries. Third, for the time being, we will consider uni-directional boundaries.

Thus, three states for transit will be examined:

- No transit
- Conditional transit
- Full transit

As will three states for awareness:

- No awareness
- Partial awareness
- Full awareness

Given these constraints, we can informally identify some different boundary types. Taking combinations of each of the awareness and access states listed above gives the following nine table entries. For each entry a possible boundary analogy has been identified together with a description of the boundary type. This table (Table 4.1) represents the most general level of boundary identification.

Obviously not all of these boundaries are as easily traversed as each other. This difference is manifest in the model by way of the criteria defined at the boundary which must be met for authorised traversal. So we can say that a wall type boundary will have difficult criteria to be met in order to traverse, and conversely a door will have relatively easy criteria to satisfy. In fact the nine boundaries can be placed in a rough ordering showing what access security they offer.

Awareness	Access	Boundary Analogy		Description
None	None	Wall	1	A does not know what is on the other side of the boundary and cannot cross the boundary.
None	Conditional	Door (with lock)	2	Awareness as above but transit is possible with a "key" object.
None	Full	Door	3	Awareness as above and free transit is possible.
Partial	None	Frosted Glass Window	4	A knows vaguely what is on the other side of the boundary but cannot cross the boundary.
Partial	Conditional	Frosted Glass Door (with lock)	5	Awareness as above and transit is possible with a key object.
Partial	Full	Bead Curtain ‡	6	Awareness as above and free transit is possible.
Full	None	Window	7	A knows exactly what is on the other side of the boundary but cannot cross the boundary.
Full	Conditional	Glass Door (with lock)	8	Awareness as above and transit is possible with a key object.
Full	Full	Line on the Floor	9	Awareness as above and free transit is possible.

Table 4.1: Boundary types resulting from considering access and awareness

(Wall)
(Frosted Glass Window) (Window)
(Locked Door) (Locked Frosted Glass Door) (Locked Glass Door)
(Door) (Bead Curtain)
(Line)

This informal identification of boundary types suggests to us that boundaries may provide a powerful basis for a spatial access-control model. Furthermore, both traversal and awareness effects would seem to have implications for access security. The role of traversal is clear - the harder it is to get to an object, the more secure it is likely to be. The role of awareness is more ambiguous. On one hand, one could argue that the fewer people who are aware of an object's existence, the more secure it is. On the other, if all access to an object is publicly visible, it may in fact become more secure. Consequently, we propose that traversal is the fundamental boundary effect that can be used as a basis for access-control and that awareness effects then provide different flavours of access security (e.g. a publicly visible object which is hard to get to vs. a hidden object which is hard to get to).

Now we introduce bi-directional boundaries. Thus, each boundary may be associated with two quite separate classifications which govern traversal in two separate directions.

‡ Imagine the entrance to the kitchen at an Indian restaurant where string beads hang down from the door allowing staff to pass through easily, together with the smell of the food, but you can barely make out what activity is going on in the kitchen (sometimes a blessing!!).

The way in which it is proposed to undertake checks at a boundary is to have some *function* at the boundary which checks properties/attributes of the subjects or objects trying to traverse. In its most simple form this could be a function which has two possible values, 1 and 0, and depending on which value is returned authorised traversal may or may not take place. Applying this simple case to an entire space gives us a very simplistic access model - traversal is either possible or else it is not possible; sophistication is easily introduced by providing more complex functions at the boundaries.

Later on (section 4.7) an example is given where the clearance-classification access model is used to provide the function checking at boundaries. This is only one possible method and the model is generalisable to any attribute/property checking technique at boundaries.

In summary, we have seen how the fundamental concept of a spatial boundary can be a useful basic concept for controlling access to objects located in spatial environments. We have also seen how different criteria can be used to enforce checking of object properties at boundaries. The next critical step in our model is to relate the boundary concept to that of an access control model. The following section considers how many boundaries may be linked together to control access within a large, highly structured space.

4.5 Constructing an “Access Graph”

Boundaries are the fundamental atoms which make up our spatial access control model. We now consider how they are combined into a useful framework called an “access graph” which lets us easily manage access within a given space.

A large space may be partitioned into regions by the use of many boundaries. For example, Figure 4.1 shows a plan view of a simple space which has three types of boundary - there are walls, windows and doors; five regions make up the space, A B C D and E. Each region has a combination of boundaries, there being many possible ways to move from one region to the other.

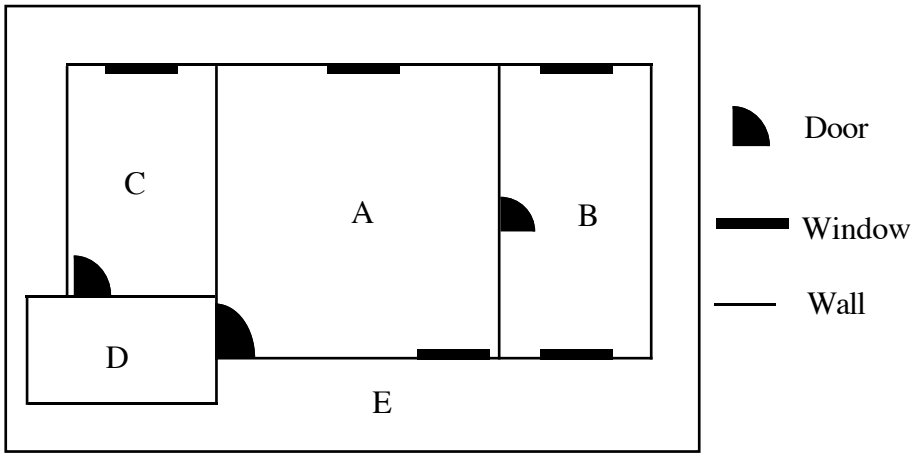


Figure 4.1: Plan view of a simple space

In Figure 4.2, we have created a graphical representation of the space from Figure 4.1. Each of the nodes of this graph represents a region and the directed arcs between the nodes represent a distinct boundary that must be traversed to move from one region to another.

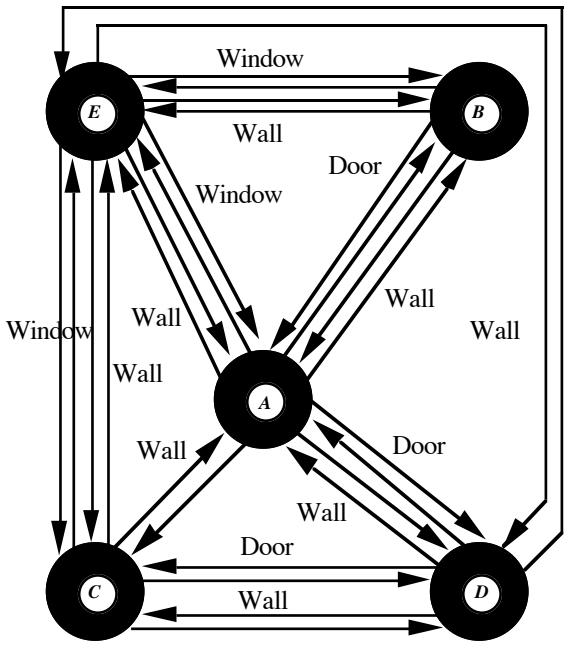


Figure 4.2: Graphical Representation of Figure 4.1

The next step is to try and construct a simpler version of this graph. First, we can recognise that the access constraints involved in moving between two adjacent regions is the minimum of all the boundary types that directly exist between them. In other words, a composite boundary has the same behaviour as

that of its weakest link. Thus, the complex graph of Figure 4.2 collapses onto the relatively simple one of Figure 4.3.

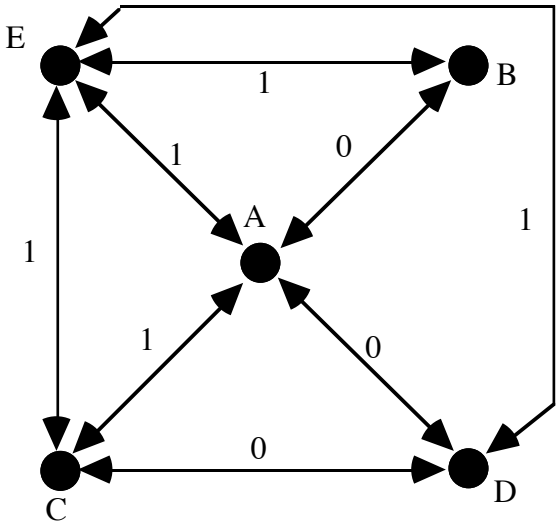


Figure 4.3: Simplified Graph

Instead of labelling the boundaries with their name, we label them according to the function which governs access. In this case we have a function which returns 0 if a boundary can be traversed and 1 if not. Clearly you can walk through a door, whereas the same is not true of walls or windows (not in normative space anyway).

In this particular case we end up with a very simple graph; indeed each boundary has the same effect from both directions. In the general case, we end up with some directed graph, and an associated set of "weightings" (which are the rules/conditions associated with each boundary).

This example may seem extremely trivial but it is equally possible to have an arbitrary function governing access (for example, if you have blue eyes, are over 6 feet tall and wear glasses then you may traverse the boundary). All that is happening is that attributes and properties of a user are being examined to provide security information. This is similar to the access matrix methods discussed in section 4.2.

The mapping of security credentials onto numeric values allows some sort of comparison between boundaries which may not be obvious unless the mapping is applied. Any mapping of abstract attributes onto concrete numbers is going to be approximate, further functionality will be required at the boundary to distinguish between apparently identical security clearances (e.g. Two very different attribute sets may map onto the same numeric value. While in general it would seem that the two sets have similar privileges, in practice different restraints may apply dependent upon the context and setting of the ongoing activity).

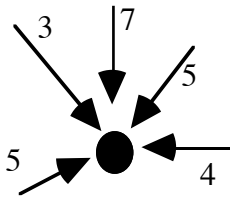
We call the above graph the **access graph** for the space. In essence, the access graph provides us with a concise summary of the possibilities for and constraints on movement around the spatial setting. The fact that we have produced a standard mathematical graph turns out to be very useful because we can now apply well known algorithms to answer a range of interesting questions about the access properties of the space. The kinds of questions we might want to ask include:

- Is it possible to move to region A from region E?
- What properties do I need to travel from region A to region B?
- How secure is region A relative to B?
- What effect does changing the boundary between regions E and C have on the security of the rest of the space?
- If I add a region and associated boundaries to the graph what properties do the new boundaries need to have to maintain the confidence in the rest of the graph?
- What is the effect of removing a region (node) from the graph? Do certain conditions have to be met for this to occur?
- Which is the most secure region in which to place an object?
- Which is the least secure region in which to place an object?
- Where can an object be moved from its current location so that its security rating is not lessened?
- What path can be taken in moving this object, making sure it is not compromised during the move?
- Who is permitted to alter boundaries and their properties?
- How are the properties of boundaries permitted to be changed?
- If I have security credentials x where can I go in a space and where am I unable to go?
- What properties do I require to be able to traverse the whole space? (Super User analogies and possibly one interpretation of absolute security)
- What is the effect of increasing an object's security (decreasing)? How is this manifest?

The following table explores the questions posed above and suggests possible algorithmic techniques which may be employed for a solution. First, we explore some techniques which will be used in answering these questions. When a user's clearance is mentioned here it is referring to the value onto which their credentials have been mapped. For simplicities sake we will initially assume that sets of credentials are mapped onto the same numeric values for all boundaries. This is restrictive in that it does not permit each boundary to interpret each users attributes dynamically, but it does allow cross-space comparisons to be carried out with a minimum of calculation.

We start be defining the notion of **absolute classification** to be the minimum weight of all the arcs which enter a given region. We know that in order to enter

this region, a person must have a clearance at least equal to this minimum absolute classification**.** To increase the absolute classification of a region, the arc with the minimum weight must be modified (see Figure 4.4).



absolute classification of the region is
min (5, 3, 7, 5, 4) = 3

To make the region more secure the
second arc from the left needs modifying.

Figure 4.4: absolute classification

The **relative classification** of a region (relative to another) is a more subtle concept. We are interested in the classification of one region relative to another region. This means the minimum clearance required to move to the former from the latter via any valid route. For example, considering Figure 4.3, we may be interested in the security of region D relative to region A. There are many possible routes between the two (e.g. A->D, A->E->D, A->C->D and A->B->E->C->D).

To be more precise, we can derive the relative classification one region to another through two steps.

The relative classification of a particular route between two regions is the maximum classification of the arcs traversed along the route.

The relative classification of one region to another is then the minimum of the relative classifications across all possible routes from the latter to the former.

Section 4.7, which deals with an instantiation of the access model using the clearance classification model for credential checking at boundaries, gives an example which demonstrates the use of relative classification. Now the table of questions and answers (Table 4.2).

Security Issue	Graph Theory Technique
Is it possible to get to region A from region E?	Use an Adjacency Matrix to check if a path exists
What is the security classification of B relative to A?	same as above
What effect does changing the boundary between regions A and C have on the security of the rest of the space?	Re-apply the security checking algorithm to the new graph and evaluate the result.
If I add a region and associated boundaries to the graph what properties do the new boundaries need to have to maintain the confidence in the rest of the graph?	Apply security checking algorithm. Compare results with previous values. Modify new boundary classifications as is necessary.
What is the effect of removing a region (node) from the graph ? Do certain conditions have to be met for this to occur?	Remove the node - recalculate the graph and observe results.
Which is the most secure region to place an object?	node with the maximum value for absolute classification
Which is the least secure region to place an object?	node with the minimum value for absolute classification
Where can an object be moved from its current location so that its security rating is not lessened?	return all nodes with absolute classification greater than or equal to current node
What path can be taken in moving this object, making sure it is not compromised during the move?	weight of all arcs in a given path must be greater than or equal to the desired security level
Who is permitted to alter boundaries and their properties?	management issue rather than graph theory
How are the properties of boundaries permitted to be changed?	up and down - again management
If I have a security clearance of x where can I go in a space and where am I unable to go?	all nodes with absolute classification less than or equal to x are possible, those with absolute classification => x are no go
What clearance do I require to be able to traverse the whole space? (Super User analogies and possibly one interpretation of absolute classification)	value of the node with maximum absolute classification

Table 4.2: Security Questions and Answers

We argue that it is the ability to apply standard mathematical techniques to access graphs in order to predict the consequences of alterations to specific boundaries that makes this approach to access control particularly interesting.

We raise one final issue in this section - that of the mode of transit adopted and, in particular, the issue of teleportation. It would seem that our model requires that in order to move between two regions a user follows some valid route, possibly passing through intermediate regions. This need not be true. There is no problem with direct teleportation from one region to another. However, the effects on the access control model need to be considered. To this end we offer the following proposal. It should be possible to teleport from one region to another provided that the users credentials would have allowed them to follow at least one valid route through the access graph. In other words, although teleportation may move you infinitely quickly, it should not circumvent the access control model.

4.6 Group Access

So far we have considered the use of multiple boundaries, represented as an access graph, as the basis of an access control model for spatial environments. However, we have focused solely on how this model may be used to control the access which an *individual* has to the environment. A collaborative spatial environment should also consider how such a model controls the access of groups of people who are acting together.

Let us assume that the environment provides some way for a collection of users to form an identifiable group (exactly how this is achieved is outside the scope of this initial work). We can consider what might happen when this group tries to cross a boundary. There are several possibilities for combining group members' properties which would seem to be valid in some circumstances:

- Take the person with the minimum credentials in the group to represent the group.
- Take the person with the maximum credentials in the group to represent the group.
- Take the sum of the credentials of all the people in the group.

The minimum case seems clear where no unauthorised person should be able to cross a boundary no matter who they are with. So does the maximum - this is where you find a sufficiently powerful friend who is allowed to take you across a boundary. The sum case is also interesting. One example might be that a group of people can gather sufficient strength between them to break down a boundary (perhaps any boundary in theory). This has a particularly democratic feel to it - indeed, it has a strongly real world feel to it! The sum approach might also apply to a second interesting case - where it is necessary to always have a specific number of people present in a region at a given moment in time. In this case, the properties of the boundary can be made the sum of a specific number of individual credentials so that only a group of the correct size can enter the region.

Perhaps the general solution is for each boundary to be associated with an appropriate combination function which it uses to determine a group's clearance. As potentially any function could be allowed this would support all three of the above cases, as well as possibly others.

4.7 Clearance Classification - An Instantiation of the Model

So far the model presented has assumed that boundaries will be used to partition space and that checking functions will be enforced at boundaries to compare each user's credentials against the requirements of each boundary. What this checking function is depends upon the requirements of the space being secured. For some spaces a simple "yes/no" function at the boundary which either permits or denies

access will be sufficient whilst other spaces will require much more subtle techniques relying on attribute checking and the like.

This section presents the Clearance-Classification access model [Bell, 1973] as a possible candidate for the checking function at a boundary. For certain application areas this model is very good at providing a secure environment. The model will be described and an example of its use given.

In it's simplest form the Bell and LaPadula clearance classification model can be summarised by two axioms [Bell, 73; Landwehr, 81]:

- No user may read information classified above his clearance level ("No read up");
- No user may lower the classification of information ("No write down").

Simply what this means to us is that users and objects are given clearance values and boundaries given classifications. No user is permitted to traverse a boundary unless their clearance is at least equal to that of the boundary's classification. This is axiom 1. We relax axiom 2 because it may be desirable to lower boundary classifications at some stage; we do not require to absolute provability of sensitive military systems.

We can associate each boundary with an integer number representing a classification level. In order to traverse this boundary the clearance of a user must be greater than or equal to this classification. A classification of 0 results in a non-obstructive boundary. An infinite or maximum classification results in an obstructive boundary. There might be any number of possible classification levels depending on specific applications. As one example, instantiating just seven classification levels allows us to rank the boundary types introduced in section 4.4 according to the level of access security they provide (Table 4.3).

Boundary	Classification
Totally Impassable Boundary	□
(Wall)	5
(Frosted Glass Window) (Window)	4
(Locked Door) (Locked Frosted Glass Door) (Locked Glass Door)	3
(Door) (Bead Curtain)	2
(Line)	1
No Boundary	0

Table 4.3: Possible Boundary Classifications

Now an example which illustrates the use of classifications and clearances, together with demonstrating the concept of relative classification of regions introduced in section 4.5.

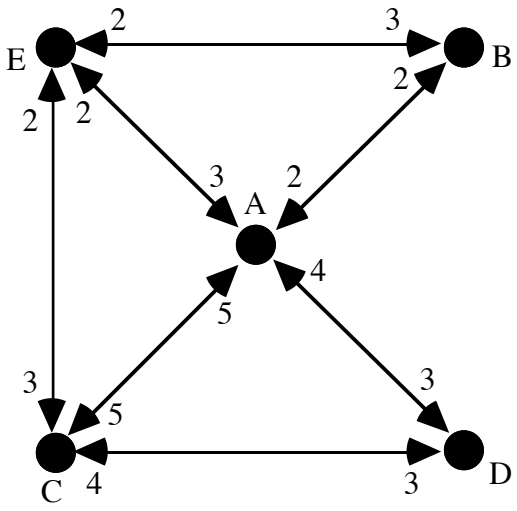


Figure 4.5: An example

In Figure 4.5 numerical classifications have been applied to each of the arcs in the graph. It is worth noting that, in this case, the boundaries are not symmetrical, that is they have different classifications dependent upon the direction of traversal. (By way of a real-world example consider a door with a Yale lock - you can pass through it from the inside very easily, but need a key to open it from the outside).

Imagine we are interested in finding out the clearance required to move from region A to region C. First of all we must consider all possible paths from A to C. There are 4 of them: AC, AEC, ADC, ABEC. Now consider the classifications attached to each boundary. We can re-write the paths in terms of classifications: AC = 5; AEC = 2, 3; ADC = 3, 4; ABEC = 2, 2, 3. Taking the maximums from each of these paths gives us four values: 5, 3, 4, 3 and the minimum of these values is clearly 3. Thus in order to be able to traverse from region A to region C a clearance of at least 3 is required, and there are two possible paths to take as there are two minimum values.

This represents one example of the kind of algorithmic approach we can adopt to answering security questions, given an access graph.

4.8 Summary and Further Work

Security is an important, but often neglected, aspect of computer systems. This paper has therefore proposed an approach to access control for spatial environments, in particular collaborative virtual environments. The essence of the model is that it combines the COMIC concept of boundaries with that of a capability based access model enforced at boundaries. The result is a new spatial model of access control which exhibits a number of interesting properties.

First, as opposed to associating access control information with either subjects or objects in an environment, the model associates it with the spaces which objects and subjects inhabit. Thus, in order to gain access to some object, one has to gain access to the space within which it currently resides.

Secondly, the notion of spatial boundaries having effects on traversal and awareness is clearly relevant to such an approach and it may be possible to classify boundaries according to their access security properties.

Thirdly, we can merge the notion of boundaries and the access model by associating a boundary with a given classification derived from a set of credentials. In the general case, there might also be a different classification for each direction of traversal.

Fourthly, by considering all the boundaries which define a space together, we can construct an access graph which represents the possibilities for transit between different regions.

Fifthly, by applying standard mathematical techniques to this graph, we can answer a wide range of access control related questions for a given space. We can also predict the impact on access security of given changes to the spatial topology. It is this predictive ability that we believe makes the proposed approach particularly interesting.

Sixthly, the approach explicitly deals with the notion of group access (one of the few access control models to consider this issue) by allowing to a boundary to specify how it will combine individual credentials into some group credentials.

At this stage, this paper constitutes a basic proposal for an approach to space-based access control. Much further work remains to be done and no doubt many more generalisations can be made (e.g. the replacement of simple numerical clearances and classifications with more general conditions). Below are three points for further elaboration which offer pointers to the direction of future work:

- A more generalisable model for spatial access needs to be developed which controls access based on properties of both objects in a space and boundaries. This general model can then be applied to a number of different scenarios.
- The work so far has concentrated on the clearance-classification approach to access control. In fact this is just one instantiation of a far more general spatial access model. Other possibilities are worth exploring and two other possible instantiations are i) analysing real world spaces for their security properties and ii) analysing network security, routing problems etc.
- An implementation of some of this work is intended, starting perhaps with a description of a space being stored in a formatted file, and then using this file to either generate visualisations of this space or to write tools that analyse the information stored in the file about the space and answer general security queries.

4.9 References

[Bell, 73] Bell, D. E. and LaPadula, L. J. “Secure computer systems: Mathematical foundations”, ESD-TR-73-278, vol.1, ESD/AFSC, Hanscom AFB, Bedford, Mass., Nov. 1973.

[Bowers, 93] Bowers, J. M. “Modelling Awareness and Interaction in Virtual Spaces”, in Proceedings of the 6th MultiG Workshop, Stockholm-Kista, May 1993.

[Feiertag, 77] Feiertag, R. J., Levitt, K. N. and Robinson, L. “Proving multilevel security of a system design”, in Proc. 6th ACM Symp. Operating Systems Principles, ACM SIGOPS Operating Syst. rev., 11, 5 (Nov. 1977), 57-65.

[Lampson, 74] Lampson, B. W., “Protection,” in Proc. Fifth Princeton University, March 1971, pp. 437-443, reprinted in Operating Systems Review, 8, 1, January 1974, pp. 18-24.

Chapter 5

Spatial Model Issues and Extensions

Steve Benford and Chris Greenhalgh

Nottingham University

This chapter ties together a number of COMIC spatial model issues that have arisen during the year two work. These issues are of two types: problems with and extensions to the model as it stood at the end of year one and new features and applications of the model that have come to light in year two.

Following a brief introduction, sections 5.2 through 5.4 address the first type of issue, focusing on each of the concepts of aura, focus and nimbus, and adapters in turn.

Sections 5.5 and 5.6 then address the second type, defining the new concept of groups, introducing non-spatial attributes into aura, focus and nimbus and finally, formulating an entirely non-spatial version of the model.

5.1 Introduction

One of the products of the year one COMIC work in strand four has been a spatial model of interaction which defines the concepts of aura, awareness, focus, nimbus, adapters and boundaries to support flexible interaction across multiple media within densely populated shared spaces. Chapter 1 of this deliverable described the implementation of this model in the MASSIVE and DIVE systems. This chapter discusses several spatial model issues and extensions that have arisen as a result of this work.

Section two briefly reflects on the aura concept and suggests the need for a more subtle approach than just detecting collision between bounding sub-spaces.

Section three discusses different interpretations of focus and nimbus and how they are be combined to calculate awareness.

Section four focuses on the adapter concept, generalising it to cover a wider range of functionality and discussing the relationship between adapters and boundaries.

Section five extends the spatial model to include the concept of a group, a specialised kind of adapter, with its own auras, foci and nimbi, which can act on behalf of a collection of individuals.

Section six considers how the spatial model can be generalised to take account of non-spatial factors. This involves two steps: introducing non-spatial attributes into aura, focus and nimbus and specifying an entirely non-spatial definition of the model applicable to a wider range of collaborative systems than just shared spaces.

5.2 Alternative Auras

Aura has been described above as the enabler of interaction between objects. An object's aura defines an upper limit on the region of interest of the object. Current auras are regions of space which define the object's region of interest (for transmitting and receiving information). Interaction is enabled when two auras collide. Aura is used to facilitate the scaling of the system to large numbers of objects; objects out of aura range are completely unaware of each other and make no demands (in terms of communications or computation) on each other. Monitoring processes (collision managers) monitor auras and notify objects when auras collide, so that the objects can begin to interact.

Considerations affecting aura are:

- simplicity - they should be understandable and intuitive.
- efficiency - of communication and computation.
- accuracy - to accurately reflect an object's interests in order to reduce unnecessary interactions.
- consistency - should not change too rapidly or abruptly in normal use.
- sufficiency - must give rise to all desired interactions.

To satisfy all of these considerations in a diverse system with a range of adapters will require a flexible and powerful model.

Where focus and nimbus include non-spatial factors (see Section 5.6), these could be reflected in the auras. This would make the auras more accurate (eliminating irrelevant interactions) but would increase their complexity and would increase the computation and communication required to support auras.

At present, the same aura is used in all collision tests. I.e. the same region of interest is applied independently of the other object's aura. This is not the case in the real world. Consider communication between a person with a megaphone (the transmitter) and a person with a microphone (the receiver). In the real world, the megaphone might make the person able to be heard at five times the distance at which a normal person can be heard. So the transmitter's aura might be extended to five times its normal size. Also the microphone may allow the person to hear things four times as far away as normal, so the receiver's aura might be extended to four times its normal size. In the real world, the person with the microphone would hear the person with the megaphone at twenty times the normal distance (four times five), but in the virtual world, with aura collisions, the distance would only be nine times normal (four plus five).

A similar argument can be applied to the visual medium - an object which is twice as large may be visible from twice the distance, while an object with twice the normal resolution (e.g. desktop vs. immersive) may wish to render objects which are twice as distant as normal. Using aura collisions would make large objects comparatively less visible to sensitive renderers - effectively penalising better renderers.

To give rise to these kinds of interactions, the objects' auras must be continuous functions in space. The transmitter's aura will fall off with distance, the value at a given distance indicating how "easy" the object is to interact with (how "loud" or "large" it is). The receiver's aura will define the minimum useful values of the transmitter's aura. A "collision" is then deemed to have occur when the transmitter's aura exceeds the receiver's threshold level at any point. Realising this approach will require separate aspects of aura for transmitters and for receivers, to accurately reflect the disparate requirements in each direction (inwards and outwards).

5.3 Interpretations of Focus and Nimbus

The second year of COMIC has seen some debate over the interpretation of the spatial model; in particular, about how focus and nimbus are combined to calculate awareness. This section briefly summarises this debate.

Adopting an information theoretic point of view, an object's focus is the means by which it filters and selects the information that it receives. I.e. focus articulates the receiver's control over a uni-directional (incoming) channel. Equivalently, an object's nimbus is the means by which it filters and controls the information that it gives out. I.e. nimbus articulates the transmitter's control over a uni-directional (outgoing) channel. The overall state of the channel from transmitter to receiver depends on both the transmitter's nimbus and receiver's focus. This combined state is the receiver's awareness of the transmitter, and may describe the importance, quality or nature of the channel. The state of the channel may be subject to other constraints or controls (e.g. an intervening wall).

For the time being the discussion is limited to a single transmitter and a single receiver having complete (joint) control over their connection. Environments with more objects can be handled by repeated application of this model to each distinct transmitter-receiver pair (alternatively one could consider the notion of groups as proposed in Section 5.5 below).

There are currently two slightly different formulations of the interaction between focus and nimbus. The approach being developed by SICS has no concept of an object when considering the use of focus and nimbus to generate connection awareness levels. I.e. all relevant information for determining awareness is included in the focus (for a receiver) or nimbus (for a transmitter). This is the approach embodied in the Mr. Nimbus demonstrator described in Chapter 1. The approach being developed at Nottingham University makes use of the objects themselves (as well as focus and nimbus) for determining connection awareness levels. This is the approach embodied in the MASSIVE system. This distinction between a "without objects" and "with objects" view of nimbus and focus is shown diagrammatically in Figure 5.1.

Both approaches can be used to achieve similar functionality, but they give different interpretations of focus and nimbus and what they include. In the with-

objects model, an object's foci and nimbi may be considered as attributes associated with the objects which describe its desired control. Focus and nimbus would then be expressed in terms of attributes of the other object (e.g. position, geometry, orientation, foci and nimbi).

In the without-objects model, an object's focus or nimbus has two components. Firstly there must be an expression of its desired control of the connection (as for with-objects model, above). Additionally, the focus or nimbus must include any information which the other object's nimbus or focus (respectively) requires. So if an object's focus is expressed in terms of the other object's position, then the other object's nimbus must include its position.

The main drawback of the without-objects model is that it may confuse the focus and nimbus concepts, because each includes not only the expression of the object's control, but also the supporting information required by the other.

The drawback of the with-objects model is that it introduces the additional concept of an object with attributes, of which foci and nimbi are examples. This also imposes a constraint on an object's representation: an object will normally appear the same (consistent attribute values) to both the incoming and the outgoing interaction on a single connection. I.e. a peer object whose focus and nimbus include a common attribute will normally use the same value for this attribute in both calculations, whereas in the without-objects model different values could be given in the focus and the nimbus. However if each connection is used in one direction only, then the model has the same flexibility as the without-model object (i.e. communication in each direction is completely independent).

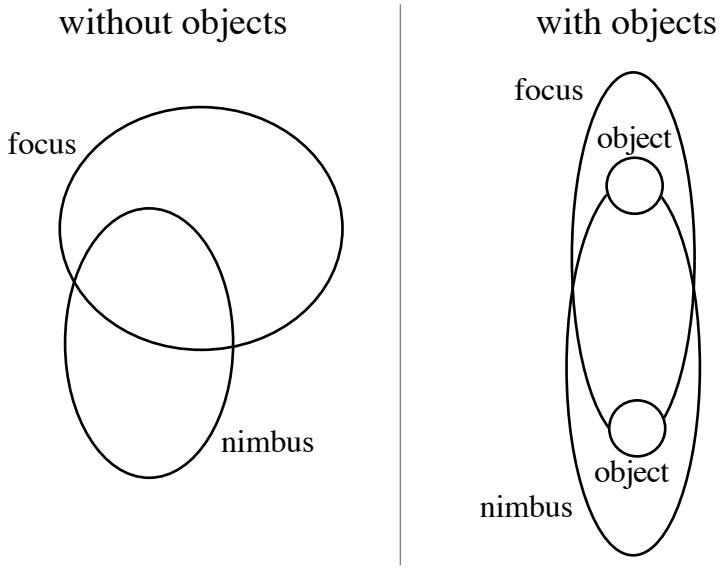


Figure 5.1: Awareness models with and without objects

Each model may be most appropriate in different situations. A highly connected, highly parallel system may in some cases fit the without-objects model

better, while current computing platforms and methods (e.g. object oriented methods) seem to fit the with-objects model better. It has the advantage of defining focus and nimbus more tightly (they are defined as the object's control, separate from its general state as represented by the sum of its attributes).

5.4 Generalising Adapters

The related concepts of adapters, boundaries and groups have posed some interesting problems. This section therefore explores these concepts a little further.

Adapters and boundaries are entities which can modify the awareness objects have of each other. Adapters are viewed as modifying ("adapting") an object's focus, nimbus and aura. Boundaries lie somewhere between two objects (spatially and conceptually) and may modify the aura, focus and nimbus of one object or both objects, or may be an independent factor in the calculation of awareness. Boundaries may also affect movement, but this aspect of their function is not considered in this chapter. Adapters and the awareness-affecting aspects of boundaries will be considered together in this section.

Several reasons for having varying levels of awareness are outlined below:

- believability - "fading" objects into awareness at the boundary of an object's interest space may reinforce the impression of a window on a larger space and reduce distracting edge effects (e.g. flickering objects).
- security - object's only receive information which they are allowed to receive (however that may be decided).
- communication - to indicate whether information is directed at a particular object or is simply "overheard," and to guide conversation and turn-taking.
- interest - an object may wish to emphasise more "interesting" objects and attenuate less interesting ones (this corresponds to targeting information from sources and selection information for sinks).
- avoiding confusion - in a crowded space the total available information may be overwhelming, and so some additional selectivity must be used to limit the information to a useful and comprehensible level.
- hardware limits - the potential information (available or demanded) may overwhelm the hardware, as well as the user, so some mechanism is needed to limit communication to sustainable levels.

The likely influences of these factors in designing foci, nimbi and boundaries are shown in the following table. Boundaries can be used to create rooms with particular purposes and so can be motivated by most of the above factors. E.g. a space set aside for people with a particular interest is indicating a common purpose or interest among the occupants and may also reduce outside noise, thereby reducing confusion and information loads for the occupants (and those outside).

Factor	Focus	Nimbus	Boundary
believability	yes	no	no
security	no	yes	yes
communication	yes	yes	yes
interest	yes	yes	yes
avoiding confusion	yes	no	yes
hardware limits	yes	yes	yes

Listed below are some real-world equivalents of adapters and boundaries and how they are used:

megaphone	amplify a source to reach a larger audience.
microphone	make quiet or distant sources audible (also telescope, microscope).
ear plugs	reduce or eliminate external noise.
headphones	provide access to an “alternate world” of sound (also VR headsets).
telephone	communicate with a specific distant person.
video relay	see a distant area.
television/radio	broadcast identically to a large dispersed audience.
thermal camera	translate from one medium to another (oscilloscope, hi-fi).
solid wall	reduce or eliminate cross-talk.
window	attenuate some media only (sound, not vision).
frosted window	allow only limited awareness.
one-way mirror	asymmetric attenuation.
private office	prevent overhearing and reduce interruptions (also phone box).
conference room	enhance communication within a group and reduce outside noise.
lecture room	deliver information to a specific audience.
coffee room	encourage social and spontaneous or opportunist interaction.
stadium	allow common access to information for many people, promote mutual awareness among observers and between the observers and those observed.

It is worth noting initially that there is quite a wide range of adapters, but they all have similar scope: to increase or decrease awareness in one region, often with a compensating change of awareness in another region. The thermal camera and related adapters are an exception to this because they bridge the gap between media rather than between regions.

Real-world adapters may be viewed as (and in many cases are) independent sources, which present to the observer a reinterpretation of the world in place of the observer’s normal view. I.e. they obscure the observer’s direct input from the world but provide new input in that region based on their own direct input from the world. This is possible because in the real world the information is carried by forces and fields through effectively continuous media, and the adapter can be placed in the medium between the source and the observer and thereby affect the information transferred. However in most computer systems the information is

carried directly from the source to the observer and there is no general facility for physically inserting an adapter into the information stream. The generally sequential nature of computation as distinct from the massively parallel way that information moves in the physical world would also make such a facility a serious problem with regard to throughput and latency. With massively parallel world simulation hardware which simulates transmission through a medium it may be possible to realise adapters in this way, but latency and through-put are likely to still pose difficulties.

The alternative, which is best suited to most current and foreseeable systems, is to incorporate the consideration of adapters and boundaries into the process of creating and controlling inter-object connections. I.e. very few adapters will actually sit in the information stream and modify it; most adapters will influence the transmitting and receiving objects to yield the effects of adaptation over the normal (direct) connections. Cross-medium adapters, and some group-related adapters may have to sit in the information stream, but these should be kept to a minimum because they are likely to be computational expensive and will introduce additional delays into the data stream.

So the normal way in which a megaphone would work would be to extend the object's nimbus (and aura, if necessary), while a microphone would extent the object's focus (and aura, if necessary). Considering again the factors which may influence awareness, adapters that relate to security or hardware limitations will have to operate, at least in part, on the information source (nimbus), limiting the information transferred. But the other factors may be handled by the source (nimbus) or the sink (focus) or both, whichever is most appropriate. Consequently, the following questions need to be addressed in the final year of COMIC:

- how should adapters be realised?
- how do adapters interact with objects and connections?
- what control do objects have over adapters?
- what is the effect of combining adapters?
- what influence do the objects retain over their nimbus and focus in the presence of adapters?
- what form should focus and nimbus take so that the full range of adapters can be realised?

5.5 Groups - a New Kind of Adapter

At present, the spatial model considers interaction between dyads - i.e. two people at a time. Of course, there may be many such dyads in a crowded space. We now consider how the explicit notion of a "group"[1] might be introduced into the model

[1]OK, so it's an overloaded term. Of course, 'group' in this document is really short for 'spatial model group'.

in order to provide greater flexibility and also possibly reduce computational load in implementations.

For our purposes, a group is considered to be a collection of entities which may be treated as a single entity for some purposes and as a set of individuals for others. There might be several advantages to introducing an explicit notion of a group into the spatial model:

- to ensure that a collection of individuals is perceived in a common and consistent way by the outside world.
- to ensure that a collection of individuals receives common information from the outside world.
- to enhance communication between group members (e.g. by providing a broadcast channel) and perhaps to reduce external distractions.
- to simplify the number of focus, nimbus and awareness calculations involved in implementing the model (e.g. through a notion of "group distancing").

5.5.1 Definitions and Types of Group

We now expand our definition of a group.

A *group* in the spatial model is an entity which has its own auras, foci and nimbi and which contains a set of further member entities. A group may undertake interaction with other entities as a single unit using its auras, foci and nimbi. When it does so, it can be thought of as acting on behalf of its members.

From a more technical point of view, a group acts as an intermediary entity which transforms incoming information from a set of entities and transmits the results to a further set (see Figure 5.2). This transformation may be any arbitrary function including combinations of multiplexing, attenuation, amplification and even substitution of information Any entity may have an awareness relationship with the group entity such that:

- the awareness that an individual has of the group determines what information they receive from the group as whole.
- the awareness which the group has of an entity determines to what extent the entity contributes to the group.

There are two major factors which determine how a given group operates:

- The nature of the transformation of information performed by the group.
- The degree to which entities interact with the group as a whole instead of with its individual members and the circumstances under which this happens. In other words, under what conditions and to what extent do group members give their identity over to that of the group as a whole.

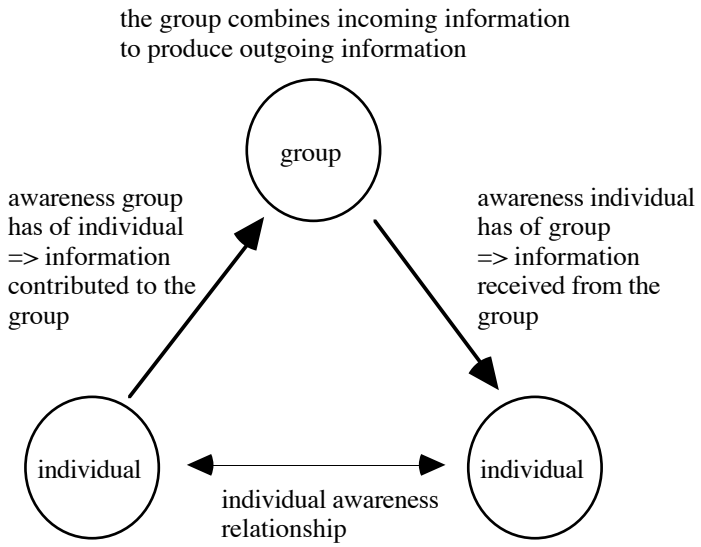


Figure 5.2: The fundamental role of a group

By varying these factors, many styles of group with differing uses are possible. For example, consider the following three kinds of group.

A semi-private conversation group

In this first example, a group is used to form a semi-private subspace for its members. In essence, group members interact with each other on an individual basis whereas all interaction with non-members is through the group entity (see Figure 5.3). Thus, this group entity might attenuate external distractions for group members and can also hide the details of conversation from the outside world (i.e. perhaps audio is attenuated although a visual representation is still provided). Such a group might be represented through the metaphor of a conference table - thus, the act of sitting at the table would make you a member of the group and the act of standing up again would dissociate you from the group.

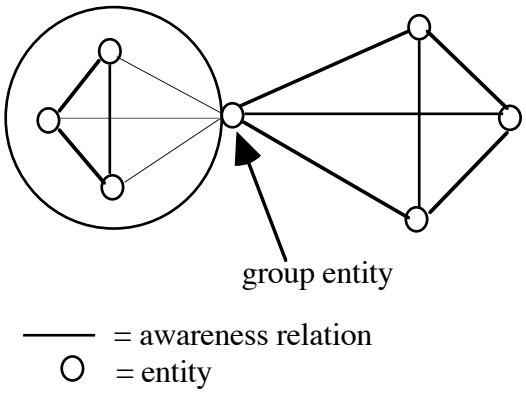


Figure 5.3: group separating members from non-members

Distancing

The second example employs a group to achieve a distancing effect (see Figure 5.4). Thus from a distance all interaction is with the group entity which provides some kind of overview of group activity (e.g. replaces individual embodiments with some kind of collective body such as a cloud). Closer in, individual members become visible as awareness relations transfer from the group entity to individual members.

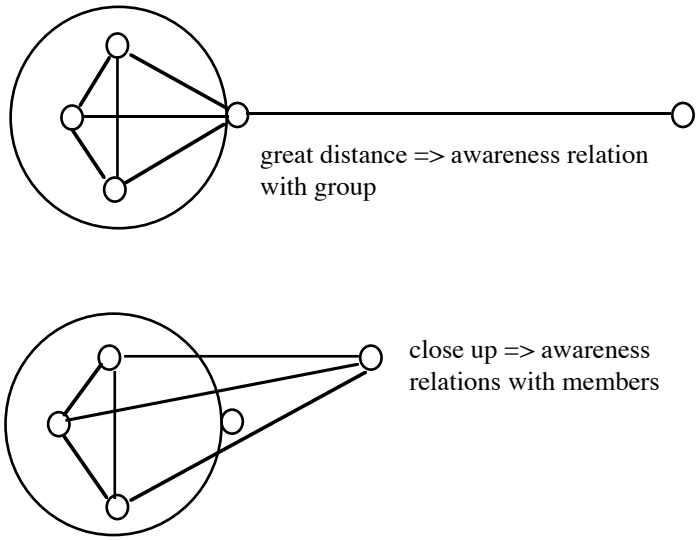


Figure 5.4: A group used for distancing

Broadcast channels

Group aura, focus and nimbus need not replace those of members, but instead, could provide an additional broadcast channel (see Figure 5.5). Thus, group members could interact both via the group and also on an individual basis. We might even provide the group entity with some control mechanism for managing the broadcast channel (e.g. integrate those horrible floor-control mechanisms from conferencing systems into the spatial model).

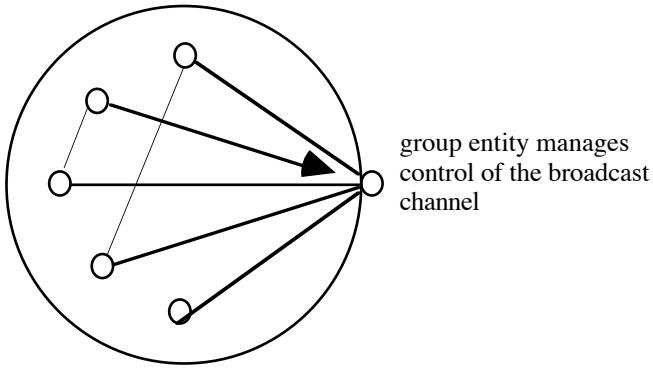


Figure 5.5: A group broadcast channel

These three examples show different uses to which the general notion of a spatial model group might be put.

Of course, the member entities of a group may themselves be groups and no fundamental constraints should be placed on the membership, size, spatial distribution or temporal nature of a group.

5.5.2 Managing Group Membership

The temporal nature of groups may range from highly dynamic to quite static. However, we do need to consider possible mechanisms through which groups might come into existence and through which membership might be managed. Both explicit and implicit mechanisms are possible.

An explicit mechanism is where a group of people decide to form/disband a group and where the group or individuals make explicit membership decisions. In relatively static forms, group members may be listed or membership criteria may be defined. In more dynamic forms, groups might come into existence via a spatial model adapter object. For example, sitting at a “conference table” may cause an individual’s aura, foci and nimbi to be temporally replaced with those of the table (i.e. group). The same effect might be triggered on entering some bounded region of space such as a room.

An implicit mechanism is where the underlying system automatically recognises that a group has naturally formed. This will typically involve applying some density measure or clustering technique to a combination of individual locations, auras, foci or nimbi. Such groups may be highly dynamic in nature and users may not even be aware of their existence.

The difference between these explicit and implicit mechanisms reflects our different goals for introducing groups into the model. In the explicit case, a group of people decide to work together and subsequently require some change to their awareness of others (e.g. heighten internal awareness at the expense of external). In the implicit case, the system may be recognising that a group has effectively

formed and using this insight to reduce the computational overhead in carrying out awareness computations (e.g. the automatic application of distancing).

5.6 Non-spatial Definitions of Aura, Focus and Nimbus

The original formulation of the spatial model defined aura, focus and nimbus as spatial fields for managing interaction. In particular, all three were specified in terms of the attributes of spatial position and orientation. It is interesting to consider the introduction of non-spatial factors into these concepts and even formulate a more general non-spatial definition of the model as a whole.

5.6.1 Extending the Spatial Model with Non-spatial Attributes

First, aura focus and nimbus could be extended to take account of additional non-spatial attributes beyond position and orientation. These might be attributes of either communicating party or of the external world. Some examples follow:

- extending aura to include an attribute which indicates whether an object is available for interaction. In this way, users could easily switch on and off all communications. This could be extended to take account of the time of day or scheduled events (e.g. turn off aura collisions between 10 and 11 so that I cannot be interrupted).
- extending focus and nimbus to include attributes describing specific objects. For example, focusing on all objects of a certain type or characteristic.
- including the names of specific individuals in aura, focus and nimbus. These might be people with whom you specifically do or do not want to interact. For example, “I want to be more aware of X” or “I want Y to be more aware of me”.
- associating with each user a description of their interests (e.g. a set of keywords) and using these in aura, focus and nimbus.

5.6.2 A More General Non-spatial Formulation of the Model

Although potentially useful in themselves, these examples suggest a more general definition of the model based on arbitrary attributes of communicating objects and their environments, of which the ‘spatial’ version would be a special case. Our aim is that this more general definition will support a broader range of applications than just shared spaces.

We begin by re-stating the goals of the model. Information overload will be a key problem for large scale collaborative systems. More specifically, given limitations on a user’s local cognitive and computational ability, mechanisms are required to ensure that they are presented with the most relevant information available. Our aim is therefore to enable flexible control in determining the

manner in which other objects, both human and computational, are perceived in collaborative systems. The key requirements of our control mechanisms are as follows:

- allowing highly dynamic control, including dynamic load-balancing;
- being applicable to a wide range of applications and situations;
- controlling perception across multiple simultaneous media;
- scaling to very large (i.e. densely populated) systems;
- providing various levels of perception from no awareness, through peripheral awareness to full awareness;
- enabling a power balance such that both the observer and observed can influence how an object is perceived;
- being equally applicable to both people and computational objects (either as observers or observed).

The provision of such a control mechanism would have implications on nearly all aspects of system design from the display of data at the user interface down to managing the quality of service across communication links.

Definition of aura, focus and nimbus

Our approach is based upon the idea that objects perceive one another in a variety of media (e.g. video, audio, graphics, and text) and that several such media may often be used simultaneously. We propose that a key factor in controlling perception in these media should be awareness. In essence, the awareness that one object has of another in a given medium is a measurable quantity ranging across an arbitrary set of values (continuous or discrete). Awareness is medium specific; an observer can have different levels of awareness of an object in different media. Awareness need not be mutually symmetrical; I can be more aware of you in a given medium than you are of me.

A measurable notion of awareness might be used in many places in a collaborative system. In the interface, it might control the display of information (e.g. level of detail of graphics, resolution and size of video or volume of audio). In terms of behaviour, it might be used to trigger or permit certain events (e.g. a voice-driven object might react to your speech-input only when its awareness of you passes a specific threshold). In terms of communications link, levels of awareness might be directly mapped onto quality-of-service.

One object's awareness of another might depend on many factors including properties of the observing object (e.g. interests and attention), properties of the observed (e.g. importance and activity) and general properties of the system, application and current working context (e.g. time of day or general system load). Two particularly important factors in awareness are likely to be the observer's ability to select objects of interest and the observed's ability to select objects that ought to be aware of it. These two factors are embodied in the concepts of *focus* and *nimbus*.

- Focus describes the degree to which an object wants to be made aware of other objects. Focus may be expressed in terms of arbitrary combinations of attribute values of the observing and observed objects and the working context.
- Nimbus describes the degree to which an object wants other objects to be aware of it. Again, nimbus may be expressed in terms of arbitrary combinations of attribute values of the observing and observed objects and the working context.

It is the definition of both focus (observer's control) and nimbus (observed's control) that support our desired power balance in interaction and which is central to the model.

Awareness between an object and all other objects in a system will have to be re-calculated whenever one of its relevant attributes (i.e. those used in focus and nimbus) changes its value. The cost of this calculation is of the order n^2 where n is the number of objects in the system which soon becomes unmanageable in a densely populated system. Consequently, some mechanism is needed for limiting the scope of awareness calculations. This is where aura comes in. An aura scopes the presence of an object in the system (again, on a medium specific basis). It is assumed that awareness between objects who are outside of each other's aura is negligible and so does not require calculating. In other words, only when objects are sufficiently within each others auras do we need to bother to calculate awareness on the basis of focus and nimbus. Aura may involve the same attributes as focus and nimbus[1] or it may involve quite separate attributes as in the case of some additional context or domain. In either case, aura provides us with an initial scoping mechanism for awareness and can therefore be though of as a fundamental enabler of communication.

Examples

We now quickly explore two examples of how this model might be used.

Trading

We can identify a mapping between the spatial model of interaction and the notion of trading proposed by the Open Distributed Processing community. Trading is the process by which client objects (i.e. consumers of services) are put in contact with server objects (i.e. providers of services). The kinds of objects envisaged are typically encapsulated pieces of code comprising some larger system. However, the model might have implications for other kinds of system. Under the trading model, service providers export service offers to a Trader object. Service offers identify and describe a specific service and may include

[1] We can conceive of the attributes involved in focus and nimbus as defining a set of dimensions in which objects exist. An object's position in these dimensions is therefore defined by the current values of these attributes. Any change in these attribute values represents a change in this position. This ties us back to the original spatial version of the model which used the attributes position and orientation in focus and nimbus and which therefore defined aura to be a bounding volume using these same attributes.

parameters such as service name, service type and quality of service offered. Sometime later, a service consumer may contact the Trader and request information about a service of interest. The Trader searches its local knowledge and, if it finds an appropriate service offer, returns a reference to this service to the consumer (a reference is a way of invoking or accessing the service - like an address).

Thinking in terms of the spatial model, a service offer looks somewhat like a nimbus; it says, "objects who are interested in service X ought to be made aware of me". Correspondingly, a service request might be likened to a focus; it says, "I want to be made aware of objects who offer service X".

In a global system, matching of service offers to service requests (i.e. awareness calculations involving focus and nimbus) might potentially involve vast numbers of objects and Traders. Consequently, some mechanism is needed for controlling the scope of these calculations. Two possibilities spring to mind: limit matching to the single local Trader (i.e. no cooperation between Traders) and limit cooperation to some 'trading domain', possibly involving several cooperating Traders. Both approaches equate to a notion of aura. In the first case, an objects aura is defined to be the current Traders it is registered with or bound to. In the second, it is its current trading domain.

USENET News

Here is just one of many possible applications of the model to a system like USENET news. Imagine that we wanted to extend USENET news to support real-time interaction between its users. In other words, one would be aware of other people reading the news and would be able to interact with them. Given the potentially vast numbers of news users present at any one time, there would seem to be scope for employing our model of interaction to control awareness and communication. Let us define our aim as to provide an additional audio channel between news users. It would then seem sensible to establish awareness between users based on some notion of common interest. We might consider describing interest in terms of the following attributes of a user (some existing and some new):

- topics of expertise - keywords describing topics this user knows about.
- topics interest - keywords describing topics that this user wants to find out about.
- subscription list - news groups to which this user subscribes.
- currently/recently active newsgroups - newsgroups which this user is currently reading or which they have recently read (i.e. in their current session).

Focus and nimbus could be expressed in terms of values of these attributes. For example, focus might specify "I wish to be made aware of users who are experts on the following topics" or "... who subscribe to the following newsgroups" (perhaps the same ones as myself) or "... who are currently active in the following newsgroups (or even at all)". Similarly, nimbus might involve specification such

as “users who want to know about the following topics ought to be aware of me” or “users who subscribe to the following newsgroups ought to be aware of me”. Information about topics of expertise and interest as well as level of activity in different newsgroups might even be gleaned automatically from the news service itself. Using these foci and nimbi, the system could calculate awareness values between users. For example:

- A in B’s focus and B in A’s nimbus => B fully aware of A;
- A in B’s focus Xor (exclusive) B in A’s nimbus => B peripherally aware of A;
- A not in B’s focus and B not in A’s nimbus => B minimally aware of A.

In turn, these awareness levels might result in different levels of embodiment and/or audio interaction between news users. For example:

- Minimal awareness => no embodiment or audio channel;
- Peripheral awareness => basic embodiment (e.g. showing name/icon of the other user);
- Full awareness => show name/icon and fire up an audio channel.

We might also provide users with the ability to explicitly increase or decrease awareness of others (e.g. on seeing that someone else was present a user might explicitly request to open an audio channel to them). This involves including the names of individuals as attributes used in focus and nimbus.

As USENET news is a global system, the role of aura is to scope these awareness calculations. Here are two examples:

- Limiting awareness to people who were currently active in the same newsgroup (i.e. one’s aura would be one’s current newsgroup).
- Limiting calculations to some geographic/telecommunications domain in order to keep the costs of audio communications within some sensible limit (USENET already includes the concept of different localities or management domains to facilitate scaling of management and distribution functions).

Now, this example is rather vague and many details would have to be worked out before implementation. However, we hope that it conveys the general idea of defining focus and nimbus (in this case based on some expression of interest) and hence calculating awareness in a non-spatial collaborative system. It also shows the general role of aura in scoping awareness calculations. One goal of the year three COMIC work should be to explore this non-spatial version of the model further, addressing, and possibly implementing, a number of applications

5.7 Conclusion

This chapter has explored a number of spatial model related issues including those related to the existing concepts of aura, focus, nimbus and adapters as well as

extensions to the model in the areas of groups and consideration for non-spatial attributes. The main conclusions arising from this work are:

The notion of aura as a simple containment space may be too limiting. Instead, auras might be defined as continuous functions in space and it may also be necessary to associate different 'transmitting auras' and 'receiving auras' with an object.

There are at least two views as to how focus and nimbus might combine to yield values of awareness. Under the without objects approach, all relevant information is contained in the focus and nimbus themselves. A's awareness of B is then some function of the overlap between A's focus and B's nimbus. In the with objects model, focus and nimbus are evaluated with respect to attributes of explicit objects. A's awareness of B is then some function of A's focus on B and B's nimbus on A.

There is a wide range of possible adapters including boundaries and communication tools. Adapters may support many possible uses including security, managing objects interest, increasing believability of virtual worlds, dealing with resource limitations and even cross medium translation of information.

An explicit notion of group might be introduced into the spatial model to support greater flexibility and scalability. A group is an object with its own auras, foci and nimbi which might, under certain circumstances, act on behalf of its members. Possible uses of the group concept include enhancing privacy within a group, enabling group distancing effects and the provision of broadcast channels to group members.

Finally, we can easily consider the introduction of non-spatial attributes into aura, focus and nimbus make them sensitive to a broader range of factors then just spatial position and orientation. Such attributes might include the identities of particular objects, object type and other characteristics and also general world properties such as time of day. To go a stage further, it is possible to make an entirely non-spatial definition of the model such that focus, nimbus and aura are defined in terms of general attributes of objects, thus making the model applicable to a broader range of collaborative systems than just shared workspaces.

Further exploration and implementation of these issues will form the basis of much of the year three COMIC work on the spatial model of interaction.

Chapter 6

Conceiving Odysseus: Social Dimensions & Multiple Populated Worlds

Mike Robinson

Sageforce

Considerations of experience with COMIC "spatial model" prototypes, and of general social and organisational activities, conflicts, and contexts lead to two suggestions for computer support utilising space-based representational techniques of Virtual Reality and "Cyberspace". First, users will need powerful tools to create, configure, and furnish their worlds and their interaction with others within those worlds. Second, multiple worlds with connections and transition paths between them will need to be supported.

6.1 Introduction

This chapter is intended to pick up on the generalisations of the spatial model, embodied in prototypes such as Q-PIT, VR-VIBE, & VR-Mapper (Chapter 2). It also takes up the questions of non-spatial models raised in the last part of Chapter 5. It is centred on the notion raised in Chapter 2 ("Human Centred Approaches) that people will wish to organise their own information terrains. This means that people collectively, in organisations, networks, or ad hoc groupings, will be making their own mappings of attributes to dimensions. If such self organisation is to be possible, then there is no doubt that powerful tools and techniques will be needed to support the creation of "domain specific" information terrains, just as spreadsheet software supports the creation of "domain specific" applications [Nardi, 93]. Once multiple information terrains have been enabled and created, large conceptual and practical issues arise about how they are to be linked — or whether they should be linked. Modes and problems of linkages between multiple worlds are discussed with the aid of current and past CSCW research, recent discussions, and a scenario based on the International Classification of Diseases [Bowker, 91].

This chapter also assumes a very general n-dimensional model onto which attributes can be mapped. The Benediktine [Benedikt, 92] spatial model is regarded as a special case, and its distinction between "extrinsic" and "intrinsic" dimensions is not important in terms of the general model.

6.1.1 Old Problems

Sisyphus, son of Aelus, owned a fine herd of cattle on the Isthmus of Corinth. His neighbour, Autolycus, was a past master in theft. Hermes had given him the power of metamorphosing any beasts he stole. He could change them from black to white, from horned to unhorned, and so on. Sisyphus noticed that his herds grew steadily smaller, while Autolycus's grew larger, but there was never any proof of theft. So one day he engraved the inside of all his cattle's hooves with a SS[1]. That night, Autolycus helped himself as usual. At dawn, the hoof prints along the road provided Sisyphus with enough evidence to call his neighbours to witness the theft. They all went to Autolycus' stable, and the marked hooves provided the proof. While the witnesses remonstrated and argued with the thief, Sisyphus hurried round to the back of the house, got in by the portal, and seduced Autolycus's daughter, Anticlea. She bore him Odysseus, the manner of whose conception accounted for the cunning he habitually showed, and for his nickname "Hypsipylon". (Adapted from [Graves, 62]).

The Sisyphus story illustrates the strange and problematic mix of symbol, ingenuity, action, and shifting social context. In many ways, virtual reality is not much of an addition to the complexity of our existing, subtle and precarious social frames [Goffman, 75]. Organisation, procedure, and orderliness are not given per se but have to be constantly recreated [Sheil, 83; Suchman, 83; Goodwin, 93]. This is not a recipe for regarding all organisational activity as *ad hoc*. Coherent interactions are usually mediated by artifacts that can act as placeholders in the flux of discourse [Robinson, 91]. Procedures can be viewed as such placeholders. They may be quite transient, but, when embodied in some material form, and functioning as boundary objects between large numbers of people, procedures may be hard to change [Star, 92]. Procedures and proofs, as in the case of Sisyphus, may also be a distraction from the real issue.

6.1.2 New Problems

Virtual reality provides a *medium* of interaction, constrains *modes* of interaction, and can provide the *grounds*, the artifacts and procedures, around and through which interaction takes place. Moulding virtual reality to any specific organisational context is ultimately the business of those who act, and are experts in that organisational context [Nardi, 93]. Nevertheless, the systems provided must lend themselves to such moulding [Henderson, 91]. To this end, it is worth examining general social and organisational activities, conflicts, and contexts with an eye to teasing out those aspects most relevant to the design of space-based many-user systems. The first part of this paper attempts to do this.

---

[1]Some say even this is not to be believed, since SS in early greek is ↀ , which was an icon for cloven hooved animals, and is pretty much like a simple hoofprint.

Considerations of social uses of CSCW applications fall into several categories.

First, those extensively documented CSCW usages, in relatively small groups of people, where cooperation is emphasised in theory and practices. These are summarised in the concept of common artifact.

Secondly, the less well documented[1] larger scale uses (inter-departmental, organisational, inter-organisational, large distributed populations, etc.) Here the questions of scalability and diversity need to be tackled. In this arena questions arise on how to regard the inevitable conflicts arising from many different sources connected with scale, division of labour, differentiation of practices, and coordination. The notion of boundary object will prove useful.

Thirdly, experiences documented in a recent survey of an old, well-used, but rudimentary cooperative tool — the "finger" command in UNIX — will be considered from the viewpoint of scale and event distribution.

Lastly, the broad conclusions of a recent workshop on "Social Dimensions of Populated Information Spaces" will be utilised as a framework for contextualising all these considerations. These differentiate between "internal" questions of usage and formalism, and "external" questions of organisational purpose, constraint, and infrastructure.

The second part of the paper looks more closely at some suggested design consequences for computer support utilising space-based representational techniques of Virtual Reality and "Cyberspace". First, the importance of powerful tools to create, configure, and furnish representational worlds and interaction with others within those worlds is stressed. These are termed "Nardi tools" in recognition of the fact that they will be formalisms whose decisive criterion is not ease of use, but power to represent the work domains of those using them. Second, there is a discussion of the need for multiple worlds with connections and transition paths between them. Multiple worlds can provide a structure that matches the complexity of the dynamics of interacting sets of activities, contexts, and conflicts and facilitates non-mechanistic coordination of organisations, and of large scale, socially constructed enterprises in general.

The third part of the paper explores the multiple worlds hypothesis through a scenario drawn from [Bowker, 91] account of the International Classification of Diseases (ICD). This is an international enterprise with maximally differentiated participants: individual professionals and experts; interest groups; professional associations; firms, corporations, and multi-nationals; local and national public bodies; nation states; and international organisations. The enterprise is data centred — its raison d'être is to construct a definitive list — and it demands "a series of dynamic compromises between a wide range of players in a number of dimensions". It seems there is a good fit between a multiple world conceptualisation and the needs arising in the scenario.

---

[1] in terminology and methods useful to CSCW design

It is concluded that the multiple world hypothesis is a good candidate for further scenario testing, and for some space-based representational system prototyping.

6.2 Likely Social Uses

6.2.1 Social Dimensions

The report on the "Social Dimensions" workshop suggests a distinction between the internal and external[1] dimensions of populated information terrains (worlds). External dimensions are those that are independent — more or less outside the control of the user and her uses: for instance, organisational rules and constraints; technical (hardware and software) infrastructure. Internal dimensions are those that can and are — and arguably should be — influenced or controlled directly by the users. These are such aspects as locally configurable formalisms to reflect and augment the target domain in which the user is the expert [Nardi, 93; Robinson, 93a]; and as specific situated actions in the context of organisational constraint, technical infrastructure, and domain representation. The latter is close to the notion of "articulation work" defined by Star[2] as the "real time management of complexity".

These four aspects and two dimensions can be visualised as a single interacting field in which the four poles can be, and usually are, distinguished for design and implementation purposes.

[1]The report uses the terms "intrinsic" and "extrinsic" but these are replaced in this text by "internal" and "external" to avoid confusion with the usage of the former by (Benedikt, 1992)
[2]Oksnoen Symposium, 1994

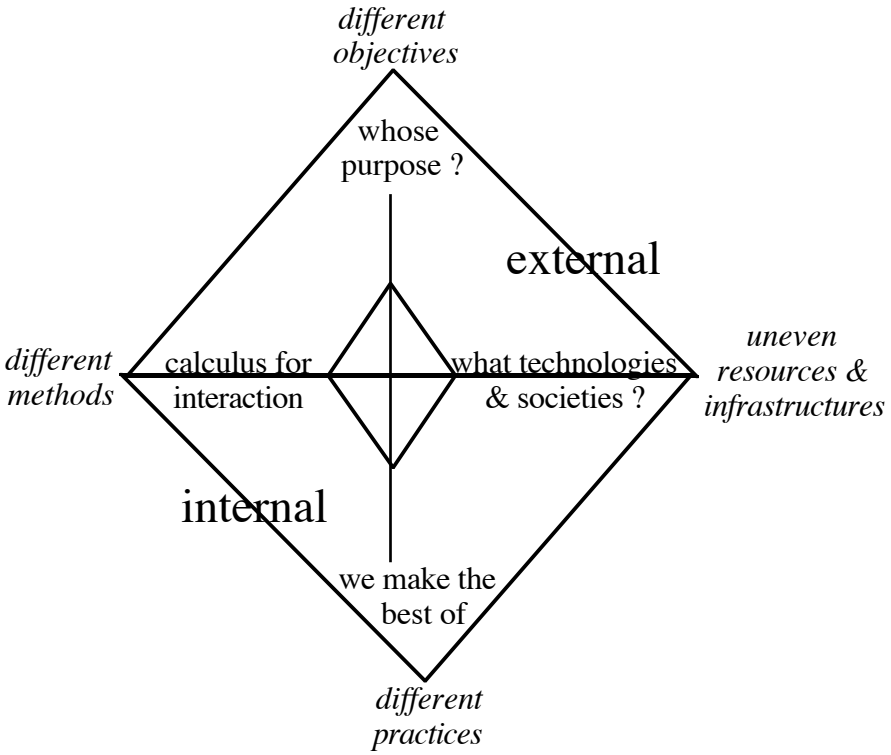


Figure 6.1 Dimensions of populated information terrains

Utilising this framework, it is pretty clear that most CSCW studies to date have been located on the internal dimension. They have been concerned variously:

- with providing appropriate support for interaction.
- with studies of working practices as a preliminary to providing support.
- with the interaction between support and practice.

Appropriate support is essayed by such applications as COLAB [Stefik, 87a; Stefik, 87b]; ASPECTS [Technologies, 90]; GROVE [Ellis, 91]; GroupKit [Roseman, 92]; Information Lens [Malone, 87]; Semi structured audio [Hindus, 92]; Meeting environments [Mantei, 88; Nunamaker, 88]; and very many others [Malm, 93] In the terms used here, these primarily provide a computer based formalism[1] that (in most cases) can be locally configured by the users.

---

[1]"formalism" is used here in its general sense that they provide "a set of symbols for which there are formation and transformation rules, [and hence] they support conventionalised implicit communication" (Robinson, 1993b).
A similar view is expressed by (Nardi, 1993) who says:
"Visual formalisms are formal in that, within an implementation, each visual formalism has a clear set of rules governing its editing and its form" (p.95)
and
"Visual formalisms can be formally represented for computational purposes; as Harel (1987) said, visual formalisms are formal in that 'they are to be manipulated, maintained, and analysed by computers' " (p.96)

Studies of working practices are exemplified by [Hughes, 1991; Suchman, 91; Heath, 92; Goodwin, 93].

Interaction between support and practice is dominated by the Participatory Design ("Scandinavian") school [Greenbaum, 91] with offshoots and influences in the US and Europe.

It is readily apparent that, overall, this body of work is concerned in computer terms with experiments and prototypes, and in sociological terms with the artful practices that appear as orderliness in the work of relatively cohesive work groupings. These experiments, prototypes, and studies are usually careless[1] of wider organisational purposes and regulation, and of existing infrastructures. One cost of such a stance is that it will focus away from conflicts that arise in the structures, from the external dimensions of constraints, multiplying objectives, and uneven infrastructural resources, will de-emphasise systemic clashes between the local ecologies that are its subject matter. Several other studies have shown that scaling applications into organisational constraints and technical infrastructure, while maintaining usability can be very problematic. e.g. [Perin, 91; Egger, 92; Orlikowski, 92b; Orlikowski, 92a]

6.2.2 Appropriate Support for Small Scale Interaction

Numerous studies already cited indicate a constellation of factors at work in the successful adoption and use of smaller scale systems. These have been summarised and collected together under the label "common artifact" [Robinson, 93b]. This is a device, or set of devices, whose use provides overview, peripheral awareness, implicit and explicit communication (together forming a "double level language"), and may perform other functions too, such as template.

In other words, common artifacts play a significant role in the process of interaction — they are not merely its record, outcome, or prescription.

In addition to the affordances of common artifacts, two further desiderata of local use can be noted. First, the large topic of tailorability [Greenbaum, 91] and specific support for tailorability, as discussed for instance in [Henderson, 91] who says:

> "If an artifact is created with change in mind, even if the specific change is not anticipated by the designer, the creation of change can be greatly simplified, regularised, and most importantly, supported." (p.239)

Second, deeply related to both implicit communication and tailorability, the need for powerful domain specific formalisms [Nardi, 93]

[1] "careless" is used positively here..... It means that the prototypes can concentrate on local adaption, integration, appropriateness, and the studies do not have to accept "organisational logic" in generating an account of local practice. In these meanings, such work is not independent of organisational constraint and infrastructure, but needs to be "careless" of them.

6.2.3 Scalability

These is little doubt that scalable applications (especially those involving spatial models) will need to support the above features and activities. The reasoning here is that successful groupware needs to build on social habits and computer applications people are already using routinely and productively, rather than substitute in entirely new habits (even if it were known how to do this) and new systems [Grudin, 94]. A second line of reasoning is that scalability is sometimes posed in a misleading way. Namely, there is an assumption that a group is fixed, something that can have a circle drawn round it, and it then needs to be connected to other groups. On this view an organisation is the set of connected groups, and a single (orgware?) application is needed that can mirror the connections, and that everyone can interact with. This may be true in some specifiable and limited cases. In general, it is necessary to observe that groups are not fixed in this way, but have fluid boundaries and changing members [Bannon, 89; Star, 92]. In the same way, it can be pointed out that much organisational cohesion, liaison, communication, decision making, etc. is done by groups that overlap other groups, by people with multiple memberships. Organisational "connections" do not have an independent existence, but are enacted and re-enacted according to the flavours and colours of current contexts, conflicts, and uncertainties. In this sense CSCW systems may be said to be scalable if:

- An application (artifact, boundary object) used by a single group supports both its work and its relations with other parts of the organisation[1];
- An application (artifact, boundary object) is used by several groups and provides common ground between them.
- An application is used by an overlap group.

In these cases, it is appropriate to regard an organisation as a mobile nexus, whose functioning lies in the interactional dynamics of contextualised, situated activities — where activities, "getting the job done", are locally prior to coordination. However, the scalability of small applications is not solely, or even mainly dependent on the suitability of the application. The application must work within the given organisational and technical frameworks. It is located in and must be compatible with these 'external dimensions'. But, following [Grudin, 94], it appears that such compatibility is a result of building onto local applications rather than replacing or coordinating by fiat: see also [Button, 93]. Fortunately it is not necessary to make a simple choice — at least in principle — between useful local applications without global consistency/compatibility/interoperability and consistent (etc.) global applications that are locally awkward or unsuitable. Local applications that include, generate, or utilise boundary objects [Star, 89; King, 93] will build on and facilitate the useful-local while also supporting global-organisational coherences.

---

[1] e.g. the completed airport ground control "complex sheets" in (Suchman and Trigg, 1991)

Boundary objects are objects which are both plastic enough to adapt to local needs and constraints of the several parties employing them, yet robust enough to maintain a common identity across sites. They are weakly structured in common use, and become strongly structured in individual-site use.

A boundary object "sits in the middle" of a group of actors with divergent viewpoints. Crucially, however, there are different types of boundary objects depending on the characteristics of the heterogeneous information being joined to create them. For example, a combination of different time horizons produces one kind of boundary object; joining concrete and abstract representations of the same data produces another. [Star, 92]

Thus scalability builds on existing practices and applications. Local plasticity and a robust global identity are two sides of the same coin. Heterogeneity — the sine qua non of organisation — is an orthogonal category to, and does not signal incompatibility. Organisational coherence and coordination (scalability) can be achieved by building on interacting local dynamics, by supporting the enactment and re-enactment of the real organisational nexus — the one that always eludes explicit specification and reification. The beauty of it is that boundary objects, in their global manifestation, are simple, straightforward mundane entities, like repositories, standardised forms, manuals/textbooks, and prototypes. The mystery is that there is very little mystery.

6.2.4 Lessons from "finger"

An e-mail survey of 40+ indicative users of the "finger" command in UNIX, an old and well-used, but rudimentary cooperative tool was recently completed. It provides evidence for the following limited propositions about event distribution — information about people's current status and activity — and about different needs according to different contexts and scales.

- People distinguish naturally and spontaneously between local and wider contexts, and wish to be able to configure their applications and tools to take account of such contextual differences.
- Many express a desire to be able to control what information about them is available in different contexts.
- Some express a desire to have the *option* of a log of who has accessed information about them.
- There was very strong evidence that asymmetrical use of finger[1] was valuable and valued. It other words, there is no absolute requirement for symmetry (I see you if you see me). There are however different levels and types of asymmetry that have different influences on the acceptability of an application (or the suspicion with which it is regarded). For instance, full symmetry for finger was unanimously rejected: it would be a nuisance and a

[1] i.e. x can access information on whether y is "online" etc. without y being aware of it.

distraction to have an explicit notification every time a finger query was made. At the same time, full asymmetry was regarded by many as worrisome. Apart from the survey, this is evidenced by one of the "most frequently asked questions" in UNIX: how to construct a log of "finger" calls. It should also be said that many others were quite happy with full asymmetry, thinking of "finger" information like a phone directory.

Generalising from this, it seems likely that any successful system, including cyberspaces, that distributes "events" (information on ongoing activity) will enable users to filter/configure the information that becomes available about them according to the context in which they are working (or playing). Enforcing absolutes of symmetry or asymmetry would probably be a mistaken path. Support for filtering/configuring would need to be tailorable for different needs of different users, and the different contexts and distinctions between contexts made by particular users.

6.2.5 Conclusions Concerning Likely Social Uses

Local configurability of virtual working environments, and the patent undesirability of sudden, inexplicable changes in the working environment/working material of others, means that the configurability should be limited in scope — it should (generally) only affect those that enact it. So:

- *since* it may be desirable to an individual or working group to have a special perspective on the whole cyber-world, to have their own configuration;
- *and* individuals or working groups in the same location may need different configurations,
- *then* it is not possible to satisfy the need for local configurability by spatial separation or localisation (i.e. changes only effect a limited area, as with a classic rooms/buildings metaphor). Instead it is necessary to postulate parallel worlds that can be inhabited separately by different populations, and where changes in configuration in one world do not precipitate changes in the configurations of the parallel worlds.

Configurability of each parallel world, and the ability to construct appropriate furniture, needs to be supported by powerful tools, as discussed by [Henderson, 91; Nardi, 93]

The "interior" of any parallel world should support the affordances summarised by the notion of common artifact (overview, peripheral awareness, double level language, etc.) However, the "exterior" of any parallel world should not support these affordances, or should only support them under the conditions below.

It is likely that users will:

- Appreciate or demand the ability of configure/control what information from their working environment spills over into other environments — i.e. the ability to configure event distribution.

- Appreciate or demand the ability to monitor how information that spills out from their working environment is accessed — i.e. the ability to monitor their "audience".

The utility of parallel worlds will depend on how far they support "the organisation as a mobile nexus" — and how far they do this better than other conceptual frameworks. Discrete, disjoint, purely local worlds may be useful, but do not address the questions of scalability and organisation. If cyberspace applications, and specifically Populated Information Terrains (PITS) are to be effectively scalable, the parallel worlds need to be augmented by the implementation of concepts to support and facilitate coupling and decoupling, overlaps, linkages and transitions. How for instance, will people move between "normal" spatial worlds, special worlds, such as those supported by Q-PIT, VR-VIBE, & VR-Mapper, and others that may have been constructed on the fly? Computer support for transitions and linkages needs to be geared to riding, without trying to anticipate, the specifics of the inevitable recurrent tensions and conflicts that characterise organisation.

6.3 Multiple Worlds

6.3.1 The Need for Multiple Worlds

The material on social dimensions just reviewed used terms like context, local plasticity, local adaptability, etc. and set them in a framework of uneven resources and different methods, objectives, and practices. The implication is that any workgroup or individual may (and probably will) wish to configure their part of "cyberspace" in such a way that it is largely incomprehensible to outsiders. This of course, is not so very different to what happens in any medium to large company, where the operations of the finance, computing, and productions departments (for instance) are only comprehensible in general terms to outsiders. To understand them in detail usually requires a lot of knowledge, and to understand the meaning of some specific action often requires direct participation, as well as knowledge and experience. Such different localities, based on different objectives, different experiences in different domains, different methods, and different resources mean that we need to deal with multiple worlds. A single, uniform, non-localisable cyberspace is likely to remain uninhabited as a work domain.

Multiple worlds is used in a strong sense. Objects and actions in one world may not (or even cannot) exist in another. Although Benedikt's notion of the primacy of "extrinsic" (of the three spatial) dimensions, is not shared by this chapter, it can provide a good example of such difference. In World A, height, breadth, and depth of objects and landscapes are mapped onto the 3 spatial dimensions of the virtual world. The result is an easily recognisable scene. In World B, a file store is mapped onto the 3 spatial dimensions (e.g. size of

file=height; age of file=depth; number of users currently accessing file=breadth). A and B are different worlds. Actions in one do not make sense in the other. The different dimensional mapping mean that objects that occur in one cannot occur in any similar way in another.

Similar ideas arise in [Benford, 94] where considerations of different global perspectives are outlined: the Benediktine; the statistical; the linked (hypertext or hypermedia); and the user centred. The framework for these constructive visualisations is one of Benediktine extrinsic and intrinsic dimensions. The central problem is not (apart from technically) one of configuring a PIT. Rather, consistent with the multiple world perspective taken here, the issue is how to conceptualise, visualise, and implement the consequences of my reconfiguration of my perspective (my world) on your perspective (your world).

All these are strong arguments for decoupling working contexts and locally convenient perspectives. In effect, allowing the creation of multiple worlds. But, as was said in the workshop, it is simply not acceptable that one person manipulates (reconfigures) the social dimensions of another[1]. Nevertheless, multiple worlds will inevitably need to be related, linked, mutually accessed. The issue is thus to settle on a conceptual framework that will allow both generation of, and relationships between multiple worlds in ways that are consistent with the social considerations of the earlier sections.

6.3.2 When are n Worlds the Same?

The question could be posed in an abstract, or a philosophical manner, and could be the subject of interminable debate. Are two MUD's running on the same machine and using the same software really the same world? Is a two-dimensional projection of the same set of objects the same world as a three-dimensional projection? Is a projection of n objects the same world as the same projection of n-1 of the objects? Is a projection of n objects and m users the same world as the same projection of n objects without the users? And so on. Rather than get philosophically investigative [Wittgenstein, 67], I will opt for a definitional resolution intended to be consistent with [Benford, 94] and the social considerations discussed in the first part.

Two (or more) projections of a class of objects are the same world if and only if they map the same set of properties onto the same dimensions.

This is a general solution, in the sense that there is no restriction on the number of dimensions, nor are any subset of dimensions prioritised over the others.

In practice, for purposes of implementation, it may be necessary to limit the number of dimensions, thus departing from a completely general solution. However, it does not seem necessary to restrict the number of primary dimensions

[1]Sometimes this will be wanted and desirable, in the same way that TIMBUCTO is sometimes useful and desirable. But most of the time one simply does not want one's view of a document tied to the view of another.

to 3 or 4, as Benedikt suggests. It terms of the system provided, and its underlying model, there is no a priori reason to prioritise any subset of dimensions over others. This general approach is discussed in earlier chapters, and exemplified by Q-PIT, VR-VIBE, & VR-Mapper. Of course, in any particular use, some dimensions, most probably the spatial dimensions, will be set up and seen as more significant than others. But this does not seem a good reason for building restricted dimensionality into the underlying model.

The above definition is very strict. Two worlds that shared 99 out of 100 dimensions would *not* be the same. This is a way of saying, or recognising, that *indexicality*[1] in two such worlds is less reliable than usual. If I point at or indicate something or a group of somethings, you may see something else. The non-common dimension, differently mapped or absent, may result in a confusion over and above the confusions of normal, everyday interaction. A very simple example of such confusion originating in loss of indexicality arose in COLAB from the ability of participants to independently resize and reposition their windows on a common document. A was not able to understand B because what was pointed at (e.g. top right of the screen) was different for each [Lauwers, 90; Tatar, 91].

Nevertheless, some way needs to be found to deal with, allow for worlds that are nearly the same, quite similar, overlap a bit, and so on. The next section will deal with the easiest case. When are n worlds different, independent, completely unrelated. Before passing on to that, it may be useful to note a consequence of the above definition.

Some divergences between virtual and "real" worlds arise. Namely, two (or more) projections of the properties of two disjoint, non-overlapping *sets* of objects of the *same class* would count as the same world. To be literal with the comparison, Mars and Venus, under this definition would be the same world in cyberspace, but are clearly different worlds in the real universe. The difference is a matter of emphasis that is common to both universes of discourse: difficulty of movement[2]. In the "real" world it is hard (in the extreme!) to move between Mars and Venus, while in the virtual world it is easy.

Similarly, in the "real" universe, a temperature gradient map of Mars would be the same world as a physical map (since both are projections of the same object set). In the virtual universe the physical and temperature gradient maps of "Mars" would be different worlds: since different sets of object properties are differently dimensionally mapped it is not possible to *move* between the two directly. This systematic difference of interpretation of "same world" is another natural consequence of different emphases. In the "real" world, one looks at and moves around representations while in the virtual world one inhabits and moves within representations.

---

[1]Indexicality is probably our most important resource in constructing common actions and meanings. It is the ability to give meaning by pointing or indicating in some way: a mode whereby an object, context, or situation can stand in for itself in an utterance.

[2]akin to Benedict's freedom of movement

6.3.3 When are n Worlds Independent?

The common sense answer is: when nothing that happens in one is visible, or has any consequences for the other. This needs to be unpacked a little. The visibility and consequences are potential not actual. Anything that happens in one *cannot* be visible and *cannot* have consequences in the other.

Two or more worlds are independent if there is no common class of objects with properties that can be mapped onto any available dimensions.

In other words, it is not enough that two projections do not share *any members* of a class of objects, nor is it enough that two projections do not share *any mapping* of properties onto dimensions. It should be noted that under this definition, it is almost inconceivable that those working in related subject areas would inhabit independent worlds (e.g. the geology of oil distribution and the economics of oil production) — since any non-empty common class (e.g. size of deposits) would destroy independence.

One intuitively correct consequence of this definition is that two or more worlds are not independent if a change to an object in one of them results in a change to an object in another.

Sameness and independence, as defined above, deal with a tiny subset of worlds that are likely to be generated. They are important because they are limiting cases, not because they are likely to be found frequently. The next step is to examine the variants of overlap and linkage.

6.3.4 Enclosure

Enclosure is probably the simplest, limiting case of overlap.

One projection of a class of objects a encloses another b if and only if all the mappings of properties onto dimensions in the second b occur in the first a, but at least one mapping is found in a that is not found in b.

An obvious example of this is the three and two dimensional versions of MASSIVE. The three dimensional version enclosed the two dimensional, since all the 2D mappings are found in the 3D version, but the reverse is not true.

It is predictable, on grounds discussed earlier, that there will be a loss of indexicality. Empirical results so far indicate that there may also be a loss of ability to negotiate social conventions that support indexicality[1].

The first part of this paper noted, and the Aarhus workshop discussed in detail the issue of uneven infrastructure, as illustrated by e.g. [Star, 94]. A promising approach to this would be to provide variable dimensionality (e.g. 2 & 3D) according to the local limitations of computing facilities and support. If the above predictions of loss of indexicality and loss of ability to form conventions between the enclosing and the enclosed world are correct, then, in practice, resolving the

[1] "Flatlanders" were reported by 3Ders to behave in an extremely rude way, walking through tables, being the wrong way up, and so on.

issue of disparate infrastructure will call for more support and effort than the simple provision of versions (worlds) with different dimensionalities.

An additional issue for enclosure is the poltergeist phenomenon. Roughly this means that objects may change, move, appear, disappear, etc. under the influence of unseen and incomprehensible forces. People (or their embodiments) may be doing quite normal things in their (enclosing) world. But, since some of the dimensions in which they move are not available in the enclosed world, the effects will appear out of the blue, driven by an alien, invisible, and incomprehensible logic. In terms of the ability of people to works together, there will be a loss of, or serious impairment to peripheral awareness. This can be expected to compound difficulties of loss of indexicality and loss of ability to form conventions.

6.3.5 Overlap

In common sense terms, two or more worlds overlap if they share some dimensions but not others. More strictly, two or more projections of a class of objects overlap if a common subset of mappings of properties onto dimensions is found in each, but each contains at least one mapping that is not found in the others.

The same problems of loss of indexicality, loss of peripheral awareness (poltergeists), and loss of ability to form conventions predicted for enclosure are likely to be found under conditions of overlap. It appears that the losses or impairments will be greater the fewer the shared dimensions, but this is an open question. It may be more catastrophic to mutual understanding to share many dimensions, than to experience a breakdown in mutual comprehensibility.

The question of sharing worlds with overlapping and non overlapping dimensions is an important one. Under the current multiple-worlds conceptualisation, it is likely to be very common. Communication and interaction between worlds is beset by at least the difficulties noted above. How and whether these can be managed in practice is an open question, as is the provision of appropriate support. It is not a truism or an empty incantation to say that further research is needed.

One obvious problem where communication and interaction will be necessary, where the parties cannot just ignore each other, is when activities that take place in one world (e.g. deleting a file) have consequences in another. It may be possible to provide some support for peripheral awareness — the ability to notice that something may be about to happen — e.g. objects that appear simultaneously in multiple worlds could be flagged as “unstable” (pulsating or something) to show they are liable to unpredictable changes. It might also be possible to provide some path to the other world, or it its inhabitants. Under such circumstances, it would almost certainly be necessary to have procedures to prevent sudden unexpected disruptions.

6.3.6 Transitions and Linkages Between Worlds

In situations where an individual or group wishes to move into another world, it seems necessary to allow both transitions and linkages.

Transitions would be an explicit re-mapping of categories/phenomena to dimensions. This could be done dimension by dimension with an appropriate tool, or via formalisms of the sort suggested by [Nardi, 93]. These are essentially user constructable macros (by analogy with spreadsheet macros) to configure the dimensionalities. Such a formalism has the advantage that it can be work domain specific, is re-usable, and is transferable (does not necessarily have to be created by the person using it).

Linkages would be implicit remappings. The user(s) would move to a different world without necessarily knowing its dimensional mappings, or having access (as in a macro) to a statement of them. This could be done by "portals" (see the account of MASSIVE in Chapter 1), by "unfolding" (Benedikt), or in traditional terms, by "opening" an object, and moving into it. In the different worlds of temperature gradient and physical maps of "Mars" (discussed earlier) several people could be in the same world (W1) looking at the maps, but if anyone "unfolded" a map and moved into that world (assuming distinct dimensional mappings) they would disappear from W1.

Transitions are thus explicit, and in some sense external to the virtual world one currently inhabits. Linkages are implicit, and facilitated by objects (hyperspace gateways) within the world.

6.3.7 Embodiments, Nimbus & Focus

The most important aspect of a (cyber)spatial metaphor, its distinguishing feature, is the appearance of people within the data, currently concretised in terms of embodiment, focus, and nimbus. How do these concepts and their current implementations work under a multiple world framework?

How and when do embodiments (with focus and nimbus) from one world appear in another? Obviously in the same world they appear normally (!), and in independent worlds they do not appear at all. The first and simplest suggestion is that in overlapping worlds they should appear as far as the common dimensionality allows. More sophisticated versions are possible, and may well be desirable. Again this is an empirical question.

6.3.8 Summarising

A framework of multiple worlds with potential enclosure, overlap, transitions and linkages appears consistent with the desiderata for a scalable application that will support the likely social uses and demands sketched in the opening sections. In particular it is consistent with a mobile organisational nexus in which we find multiple, ambiguous, and inconsistent purposes; uneven and possibly dislocated

infrastructures and resources; divergent formalisms; and multitudinous, heterogeneous local activities.

6.4 Scenarios

Drawing on the methodology developed in EuroCODE [Bødker, 93] for dealing with large scale system development, it seems a good idea to explore the many worlds framework via a scenario. One scenario that suggests itself is the International Classification of Diseases (ICD). The ICD seems appropriate for several reasons. It is unquestionably a *very* large system, with a dynamic evolving relationship to computerisation. It is (effectively) a very large database, fed by, distributed over and partially replicated in a highly variable set of infrastructures (some of which are not computerised). There are serious problems of data consistency, compounded by major political problems of data definition — and hence major problems of interpretation and analysis. Operation and usage modes and difficulties are documented in [Bowker, 91; Bowker, 93]. This work has been explicitly related to a spatial conception of computational representation in [Kaplan, 94]. Lastly the ICD case utilises the notion of boundary object that is an important ingredient of scalability.

6.4.1 The ICD[1]

The International Classification of Diseases (ICD) is a list of the world-wide causes of death and disease administered by the World Health Organisation. The ICD is distributed in book form to statistical bureaux, public health offices, etc., and is used in planning and implementing epidemic control, reduction of infant mortality, etc. It is classified as a list rather than a database (p.74)

Tensions and conflicts inherent in the ICD are exemplified with reference to the power and resources that flow from the adoption of one classification rather than another. Typical interest groups are *medical specialists* claiming the value of new sorts of treatment, *public health officials* claiming the value of sanitation in cities, and *economists* claiming the beneficial effects of a rise in the standard of living.

Other conflicts are present in the way information is collected and coded, and in variations of the speed with which it is returned. Different countries typically have different reporting methods and practices at individual and national levels. Two of the many national examples given are: a period when the USSR did not record causes of death in areas with less than 10,000 inhabitants; “the curious case of Japan’s low rate of fatal heart attacks” — stemming from cultural bias towards “strokes” as a sign of an overworked brain, and against “heart attacks” as a sign of demeaning physical labour. Different individual (reflecting national) practices are

[1]Page numbers in this section all refer to (Bowker and Star, 1991) unless otherwise stated.

illustrated by very different reporting practices on abortion, suicide, and stillbirth/miscarriage. The tensions produced by different values, needs, and practices of doctors, epidemiologists, and statisticians have already been mentioned. These are compounded by the conflicting purposes of insurance companies, industrial firms, and pharmaceutical companies, etc. who also have roles in the ICD. Other conflicts arise from national desires for (the status of) political control over the ICD, and between highly detailed information suitable for developed and computerised countries and basic trend information more suited to the problems of developing countries.

The whole thing, as Bowker & Star aptly remark

> "is not so much a list of causes of death as a series of dynamic compromises between a wide range of players in a number of dimensions" (p.76)

6.4.2 Comment: the Appearance of Non-orthogonal Dimensions

It is fairly self evident that the ICD is structured by multiple, heterogeneous worlds connected by the project of the ICD itself. The intention of the World Heath Organisation was to produce a positivist, standardised list — a single homogeneous world. Different purposes, methods, practices, and infrastructures in the "worlds" that contributed to the list, the "worlds" that coordinated and constructed the list, and the "worlds" that used the list, meant the objective of standardisation could only be met by a sophisticated battery of techniques and diverse special categories to reconcile the heterogeneous and orthogonal inputs and pressures. In effect, each constituent world had its own perspectives, ways of viewing the material and the list itself.

This description generates an interesting set of issues for VR representations. The general considerations of multiple worlds in Part 2 considered issues like sameness, independence, overlap, and so on. The ICD scenario points to an additional complexity. Namely, that, in at least one real case, it is not possible to construct an unique global set of categories that map neatly onto orthogonal dimensions in the worlds considered The general case of overlap could, for instance, be illustrated by 10 mappings of categories to dimensions, as follows

In world X and world Y	**In world X only**	**In world Y only**
a —> D1	g —> D7	h —> D8
b —> D2	i —> D9	j —> D9
c —> D3		
d —> □		
e —> D5		
f —> D6		

So 6 mappings are common, 2 (g&h) are unrelated and different, and 2 (i&j) are mappings of different categories onto the same dimensions.

The ICD scenario complicates this rather over simplified picture. The same mapping — e.g. (a —> D1) might be the same mapping. But it might be that the same category label is interpreted/counted differently in different places — e.g.

what appears to be a common (a —> D1) is in fact a case of (i —> D9 and j —> D9). It might be a case that was not considered — a case of *non-orthogonal dimensions*. For example, what appears to be a common (a —> D1) might be a case of (i —> D9 and i+j —> D9). And this is an illustrative simplification of the sort of thing that could happen.

This abstract account can be exemplified by the category of *disease incidence*. Both words are problematic in the ICD. I will leave aside the problems of different ways of recognising, testing for, classifying, etc. a disease, and concentrate on *incidence*. Depending on the nature of the reporters (doctors, statisticians, morticians, hospitals, etc.) this might be an approximation to overall incidence (how common is the disease?); or it might be that what gets counted is economically circumscribed incidence (how common is this to the class of paying patients?); or the statistical base might be politically circumscribed incidence (how common is this to the socio-economic group defined as a statistical and medical priority by the government?); or it might be a calculation of a new emergency interest in the rate of increase of incidence (is this an epidemic?); or it might be curability; or it might be counting cost of curability (by hi-tech or lo-tech methods?); etc. In the set of worlds (countries, regions) taken together as the ICD, it appears that most of these primary classifications might be used.

The complexities and issues that arise from different groups of people using the same words to mean different things, and different words to mean the same thing cannot be resolved by discussions of system design. Nor can they be resolved simply by rules, prescriptions and norms so that such ambiguities never reach the system (since this is one of the things the WHO tried to do).

The issue of non-orthogonal dimensions arising in the cases where worlds overlap is an issue for system design, and needs to be considered along with the more social issues of overlapping worlds.

[Bowker, 91] note various techniques and special categories as “solutions” to these issues of different motives, perspectives, classification methods, quantification methods in the construction of the ICD. These are: Distributed Residual Categories; Heterogeneous Lists; Parallel Different Lists; Full Complementary Localisation; Convergent Bureaucracy; Computerisation; and Standardised Forms. (pps.77-78) Since these categories were not generated with an eye to compatibility with scenarios of spatial representation, they form a good test case, and each will be discussed in turn.

Distributed residual categories

These are defined as “that array of categories where things get put that you do not know what to do with — the ubiquitous ‘other’”. An international commission specified three general causes for putting entries in ‘undefined diseases’: “ either because there was not enough information, or because the disease was badly characterised, or finally because the doctor failed to formulate a complete diagnosis”. What went into this residual category was a matter of continuing dispute. For instance, the “vague term, ‘haemorrhage’” was kept separate, with a

view to ‘not over-inflating the figures concerning badly defined diseases.” Similarly, ‘other diseases’ were distinguished from ‘unknown or badly defined diseases’ on the grounds that the latter indicated possible omissions in the overall set of categories.

Comment 1

From this account, Distributed Residual Categories appear to be a logical join of heterogeneous categories. In terms of many-worlds representation, the same effect could be achieve by a join of the relevant mappings. This would generate a new world, different from, and *enclosing* all the originals.

For instance, the categories/dimensions (from different worlds) of politically unimportant to diagnose, incomplete diagnosis, and lack of information might be combined as a single dimension. This would be a funny[1] sort of dimension by any usual standards, but, technically, location on the dimension could be simply determined by the total number of instances. This in not technically problematic.

Comment 2

The issue of a join raises some interesting questions. Is there a relation of dependence between the worlds originating the data, and the world constituted by aggregating it (the join)? If so, should this be tracked, or made visible to users? A positive decision on the latter would mean (at least) that updating and disaggregation would become possible[2]. Such a relation of dependence would, in design terms, mean that there should be a linkage that makes transitions possible. But it should also be noted that, in the case of the ICD, uncontrolled updating — or even worse, category modification —would be a disaster. The uses of the list depends in many ways on stability between well defined (decennial) update points.

Heterogeneous lists

Heterogeneous lists appear to be a homogeneous statement of disease incidence, but it is noted that at least four classificatory principles are involved:

> “*topographical*, the seat of the disease, which part of the body it manifests in; *etiological*, the origin of the disease (genetic, viral, bacterial, etc.); *operational*, the responses to certain tests;

---

[1]It could even in practice be meaningless, incomprehensible, or nonsensical. Such can be the result of logical joins! Some “residual categories” may be worthless, but this is the business of people (in the ICD case, committees) to sort out. It is not self evidently something that should be prevented by software or hardware, since in many other cases it is desirable, necessary, or useful.

[2]In the ICD scenario, it is not in general possible to disaggregate joined categories. Detail is lost. A computer based many worlds (with dependence) scenario would enable the component detail to be retrieved, at least from those sites that were computerised, and internet accessible. ((Bowker and Star, 1991) also note that computerization offers many ways of passing through a data set, and forces fewer decisions than permanent inscription by pen and paper methods.

> and *ethical/political*..... [as in the definitions of stillbirth, abortion, suicide, iatrogenesis and euthanasia]" (p.77)

It is not entirely clear from the text whether the original classifications are lost in aggregation or not. However, from the example[1] is appears the principles and classifications they give rise to remain distinct. Thus a systemic difference between residual categories and heterogeneous lists may lie in the preservation of distinctions.

This makes it more difficult to regard heterogeneous lists in the same way as residual categories: as a logical join between local mappings. In the former case, technically, all that is needed is to count the number of instances in the combined categories, and the position on an extrinsic (spatial) dimension is fixed. In the latter case, where distinctions are to be preserved, this technical solution is inappropriate. The category of heterogeneous lists presents similar difficulties to mapping a set of heterogeneous shapes or sounds (as opposed to unit area or decibel level) onto a spatial dimension. This is difficult enough, but it may also be that the categories and dimensions joined are not orthogonal. Apart from this, the same issues of dimensional join, transitions, dependence, stability, and enclosure apply as in the consideration of Distributed Residual Categories.

Parallel different lists

[Bowker, 91] say that "Different groups have found that the list just did not serve their purposes, and so they have modified it.[2]" Examples of groups producing modified lists are African Countries (tropical diseases were not covered in the early ICD classification); researchers interested in finer granularity; insurance companies.

> "Rather than lose control of this whole process, the ICD committee has chosen to issue rules for how the list is to be modified. This gives them a level of control at the second level that they have lost at the primary one. The critical advantage of this secondary control is that it gives an algorithm for working back from the modified list to the ICD itself." (p.78)

From the VR point of view this can be conceptualised as an interesting case of transitions. These, as discussed earlier, are *explicit* re-mappings of categories/phenomena to dimensions. It can be done dimension by dimension with an appropriate tool, or via formalisms (that are essentially macros to re-configure the dimensionalities as a whole). The category set that is common to the ICD as an organisation and the local group is also a good example of a constructed boundary object: it is both robust and stable in general use, yet locally is plastic and malleable to local purposes. The VR possibility of "visitors" to the local world seems to present possibilities that are not available in the current ICD.

---

[1] "There is no necessary one to one correspondence between test results and a given topographical or etiological feature — though in general one or the other is asserted"

[2] It is not entirely clear from the text whether this applies to recording information for direct local use, or to using the lists the ICD produces. I assume the former.....

Full complementary localisation

Some participants in the ICD have questioned whether such an aggregation has any validity at all, or whether it simply imposes an order which masks the inherent vagueness of the diagnoses — and is a waste of time. They argue for total localisation with no global claims. In VR terminology, this is an example of seeing every world as independent in the strict sense, or collections of worlds as beset by intractable problems of aggregating non-orthogonal dimensions. [Bowker, 91] say that such explicit tensions between the local and the global serve to strengthen the ICD as boundary object. They continually test it and reset its limits. The ongoing list is constrained by usefulness — its existence is rightly threatened if it does not live up to at least this function. Similar remarks could be made about a VR implementation. It is, and will be difficult and problematic to enclose, overlap, and make transitions between worlds — and the problems will be especially tricky when non-orthogonality is a significant issue. As in the ICD itself, the usefulness of such implementations can be expected to be constantly challenged and tested, new limits found, and old ones overtaken.

Convergent bureaucracy

Bureaucracies responsible for the ICD are becoming more similar. This makes the job of the ICD easier ("It is much less likely anywhere that it is the village priest who determines the cause of death"). To some extent this provides a counter-balance to the centrifugal tendency of "full complementary localisation" where diversity of data and methods constantly increases. It is noted, though, that increasing similarity of practice and category does not reduce the historical contingency of both. Common sense says that a growing acceptance and utilisation of VR techniques in the ICD can be expected to increase this centralising tendency. Increased ease of communication — in the virtual presence of the objects and structured being communicated about — should result in transfer, imitation, exchange etc. of methods. Nevertheless, it needs to be stressed that convergent bureaucracy and divergent local methods and data types are not incompatible. Given the potential relations between local and global already discussed, it is to be hoped that both will increase.

Computerisation

It is noted (p.78) that the history of the ICD is interwoven with the history of computing from the earliest days of punch cards. The chief advantage of computing today is that it

> "keeps uncertainty at the level of closure on analysis. Computers, that is, offer many ways of 'passing through' a data set"

I take this to mean that when data is *printed*, many decisions have to be made ("forced") about the viewpoint, modes of aggregation, classes to be aggregated, etc. Moreover, these decisions, and the classes and categories underlying them, are not generally available to anyone using the data. Thus any doubts about the

validity of the data, any attempts to check it, or to view it in a more appropriate way, are beset with difficulties and demand large amounts of extra work. Thus uncertainty is suspended in the very solidity of the printed figures.

Conversely, when data is held in a computer, there is no (or, at any rate, less) need to make global decisions on viewpoint, modes of aggregation, classes to be aggregated, and data sets to be suppressed or lost. The "uncertainty" now shifts from the data to the people using it — at what point do they regard their data as adequate, as sufficient for their purpose.

Computerisation is visualised as a way of making analysis less rigid, more sophisticated:

> ".... more axes can be added to disease descriptions encoded by computers, and more comparability is possible due to added complexity." (p.78)

While agreeing with this, and while trying to avoid technophilia and machine-centred optimism, is seems that the changes may be more far reaching. VR techniques in multiple worlds should allow and encourage much fruitful interaction between disparate perspectives, in the virtual presence of, and facilitated by the many ways of passing through and comparing disparate data sets.

Standardised forms

The authors note that standard forms are "essential for the ICD to work" and that they "cannot be over-precise or the different players will not be able to use them" (p.78). They conclude that

> "Standardisation procedures must be tailored to the degree of granularity that can be realistically achieved."

Implicit in the usefulness of standard forms there is another aspect that the authors do not consider at any length: temporality, or timeliness. Put simply this means that input forms (as a specific example) need to be stable and super-predictable[1] over long periods of time. Busy doctors etc. do not have time to keep getting the hang of new forms. Constantly changing forms would make any data collection process more or less meaningless, since any attempt to detect trends would become problematic. This means that the stable temporality needs to be achieved in a similar way to that utilised by the ICD — infrequent but periodic revisions of categories and methods.

Similar considerations of temporality and stability attach to the ICD output: "the book". This standard format, fixed perspective printed item is universally available and widely used. Its format can be learned, and locally modified in derivatives. It is, like a textbook [King, 93], a boundary object.

Such issues of formal procedure, standard forms, interchange formats, etc., and timeliness all underpin usefulness with stability, and will need careful

[1] i.e. predictable without any effort whatsoever, since the form is so routinized, so mundane, that it can be taken entirely for granted.

consideration and support. VR representations can enhance both global standardisation, local heterogeneity, and comprehensible relations between the two. They also enhance the potential for conflict and confusion.

Conclusions from the ICD scenario

Current (database) technology allows many ways of passing through data, and this must be a gain — at least for those able to utilise it. However, currently developing technology allows far more than this. In addition to multiple ways of passing through data, it allows multiple ways of *representing* data, and *interaction* between people trying to use the data regardless of geographical, temporal, and even conceptual, contextual, and linguistic distances.[1]

These new affordances seem especially important in view of the (documented) social and inter-social nature of the ICD, and, critically, seem to allow for radical developments on the basis of existing ICD methods of data collection, aggregation, and political compromise. It seems possible that a useful, but dislocated and elephantine bureaucracy may be assisted in transforming itself into a heterogeneous, distributed, community of practice.

6.5 Concluding Note

The notion that domain experts, in particular people who use VR systems in their work, will need powerful tools to configure and furnish their own environments has been ubiquitously stated throughout the paper. These environments can be expected to form multiple worlds, as considered, and defined. The issues of linking and overlapping multiple worlds are important questions of infrastructure design — and of understanding complex organisation. The ICD scenario threw up the difficult case of non-orthogonal dimensions that was absent from the original abstract formulation of relations between multiple worlds. Other such difficult social and technical questions are certain to arise with further research into VR modelling and the ICD data, and with other cases and scenarios. Two things seem fairly certain:

- The cunning Hypsipylon, or Odysseus, will always be with us, switching contexts when we least expect it.
- The social, the abstract, and the scenario based arguments for supporting multiple worlds all point to the lesson that scalability is not uniformity.

[1] At least, in principle.....

6.6 References

[Bannon, 89] Bannon, L., and Schmidt, K., CSCW: Four Characters in Search of a Context. In *EC-CSCW '89. Proceedings of the First European Conference on Computer Supported Cooperative Work*, Gatwick, London, 13-15 September, 1989. - Reprinted in *Studies in Computer Supported Cooperative Work. Theory, Practice and Design*, J. M. Bowers and S. D. Benford, Eds. North-Holland, Amsterdam, 1991, pp. 3-16.

[Benedikt, 92] Benedikt, M., *Cyberspace, First Steps*. Cambridge, MA: MIT Press, 1992.

[Benford, 94] Benford, S. and Mariani, J., *Populated Information Terrains: Virtual Environments for Sharing Data*. COMIC NOTT-4-12 (Draft), 1994.

[Bødker, 93] Bødker, S. et al. *The EuroCODE Conceptual Framework*. ESPRIT Project 6155, Empirica, Oxfordstraße 2, D-5300 Bonn 1, Germany, 1993.

[Bowker, 91] Bowker, G. and Star, S. L., Situations vs. Standards in Long-Term, Wide-Scale Decision-Making: The Case of the International Classification of Diseases. In *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, Kauai, Hawaii, January 7-11, 1991, ed. Jay F. Nunamaker Jr. and Ralph H. Sprague Jr. IEEE Computer Society Press, Vol. IV.

[Bowker, 93] Bowker, G. and Star, S. L., Knowledge and Infrastructure in International Management: Problems of Classification and Coding. In *Information Acumen: the Understanding and Use of Knowledge in Modern Business*., ed. L. Budd, London: Routledge, 1993.

[Button, 93] Button, G. and Harper R.H.R., Taking the organisation into accounts. In *Technology in Working Order: Studies of Work, Interaction, and Technology*, ed. G. Button, London & New York: Routledge, 1993.

[Egger, 92] Egger, E. and Wagner, I., Time-Management: A Case for CSCW. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, 1992, 249-256.

[Ellis, 91] Ellis, C. A., Gibbs S. J., and Rein, G. L., Groupware: Some issues and experiences. *Communications of the ACM* 34 (1): 38-58, 1991.

[Goffman, 75] Goffman, E., *Frame Analysis*. Middx. England: Penguin Books, 1975.

[Goodwin, 93] Goodwin, C. and Goodwin, M., Formulating Planes: Seeing as a situated activity. In *Communication and Cognition at Work*, ed. Y. Engestrom & D. Middleton, N.Y.: Cambridge University Press (in press).

[Graves, 62] Graves, R., *The Greek Myths*. Harmondsworth: Penguin, 1962.

[Greenbaum, 91] Greenbaum, J. and Kyng, M. ed., *Design at Work: Cooperative Design of Computer Systems*. London and New Jersey.: Lawrence Erlbaum, 1991.

[Grudin, 94] Grudin, J., Groupware and Social Dynamics: Eight Challenges for the Developers. *Communications of the ACM* 37 (1): 92-105, 1994.

[Heath, 92] Heath, C. and Luff, P., Collaboration and Control: Crisis Management and Multimedia Technology in London Underground Line Control Rooms. *Computer Supported Cooperative Work* 1 (1-2): 69-94, 1992.

[Henderson, 91] Henderson, A. and Kyng, M., "There's No Place like Home: Continuing Design in Use". In *Design at Work: Cooperative Design of Computer Systems*, ed. J. Greenbaum & M. Kyng, 219-240. London and New Jersey.: Lawrence Erlbaum, 1991.

[Hindus, 92] Hindus, D. and Schmandt, C., Ubiquitous Audio: Capturing Spontaneous Collaboration. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, 1992, 210-217.

[Hughes, 91] Hughes, P. T., *Going off the Rails: Understanding Conflict in Practice*. BNR Europe Limited, 1991.

[Kaplan, 94] Kaplan, S. M., Star S. L., Tolone W. J., and Bignoli, C., *Grounding the Metaphor of Space in CSCW: Meta-Structures and Boundary Objects*. DRAFT MS, 1994.

[King, 93] King, J. L. and Star S. L., Organisational decision making. *Organisational Science* (forthcoming).

[Lauwers, 90] Lauwers, J. C. and Lantz K. A., Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems. In *Proceedings of CHI '90*, Seattle, Washington, April 1-5, 1990, 303-311. ACM.

[Malm, 93] Malm, P. S., *The unOfficial Yellow Pages of CSCW: Groupware, Prototypes, and Projects*. Cand. Scient. in Informatics Appendix, University of Tromsø, 1993.

[Malone, 87] Malone, T. W., Grant, K. R., Lai, K. -Y., Rao, R. and Rosenblitt, D., Semistructured messages are surprisingly useful for computer-supported coordination. *ACM Transactions on Office Information Systems* 5 (2): 115-131, 1987.

[Mantei, 88] Mantei, M., Capturing the Capture Lab Concepts: A Case Study in the Design of Computer Supported Meeting Environments" In *CSCW '88. Proceedings of the Conference on Computer-Supported Cooperative Work*, September 26-28, 1988, ACM.

[Nardi, 93] Nardi, B. A., *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA: MIT Press, 1993.

[Nunamaker, 88] Nunamaker, J.F., Konsynski, B.R. and Applegate, L.M., Computer Aided Deliberation: Model Management and Group Decision Support. *Operational Research* (Nov/Dec) 1988.

[Orlikowski, 92a] Orlikowski, W. J., The Duality of Technology: Rethinking the Concept of Technology in Organisations. *Organisation Science* 3 (3): pp. 398-427, 1992.

[Orlikowski, 92b] Orlikowski, W. J., Learning from Notes: Organisational Issues in Groupware Implementation. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, 1992, 362-369.

[Perin, 91] Perin, C., Electronic Social Fields in Bureaucracies. *Communications of the ACM* 34 (12): 75-82, 1992.

[Robinson, 91] Robinson, M., Double-Level Languages and Cooperative Working. *AI & Society* 5: 34-60, 1991.

[Robinson, 93a] Robinson, M., Computer support for meetings: the role of formalism in local control. *European Journal of Information Systems* (forthcoming).

[Robinson, 93b] Robinson, M., Design for unanticipated use.... In *ECSCW '93 (3rd. European Conference on Computer Supported Cooperative Work)*, Milan, Italy. Kluwer, 1993.

[Roseman, 92] Roseman, M. and Greenberg, S., GROUPKIT: A Groupware Toolkit for Building Real-Time Conferencing Applications. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, 1992, 43-50.

[Sheil, 83] Sheil, B., Coping with complexity. *Office: Technology and People* 1, 1983.

[Star, 89] Star, S. L. and Griesemer, J. R., Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science* 19: 387-420, 1989.

[Star, 92] Star, S. L., The Trojan Door: Organisations, Work, and the 'Open Black Box'. *Systems Practice* - forthcoming.

[Star, 94] Star, S. L. and Ruhleder, K., Steps towards an Ecology of Infrastructure. In *Proceedings of CSCW '94*, Chapel Hill, N. Carolina, USA. ACM, 1994.

[Stefik, 87a] Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., and Tatar, D., WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems* 5 (2): 147-167, 1987.

[Stefik, 87b] Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S., and Suchman, L., Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM* 30 (1): 32-47, 1987.

[Suchman, 83] Suchman, L. A., Office Procedures as Practical Action: Models of Work and System Design. *ACM Transactions on Office Information Systems* 1 (4): 320-328, 1983.

[Suchman, 91] Suchman, L. A. and Trigg, R. H., "Understanding Practice: Video as a Medium for Reflection and Design". In *Design at Work: Cooperative Design of Computer Systems*, ed. J. Greenbaum & M. Kyng, 65-89. London and New Jersey: Lawrence Erlbaum, 1991.

[Tatar, 91] Tatar, D., Foster, G. & Bobrow, D., Design for conversation: Lessons from Cognoter. *International Journal of Man-Machine Studies* (34): 185-209, 1991.

[Technologies, 90], Group Technologies, *ASPECTS Manual*. Group Technologies Inc., 800 North Taylor Street, Suite 204, Arlington, Virginia 22203, US, 1990.

[Wittgenstein, 67] Wittgenstein, L., *Philosophical Investigations*. Translated by G.E.M. Anscombe. London: Blackwell, 1967.

Part 2

The Shared Object Service and Prototypes

Chapter 7

Developing the Shared Object Service

John A. Mariani

Lancaster University

7.1 Populating the Shared Object Service

In Deliverable 4.1 (Requirements and Metaphors of Shared Interaction), the requirements for a shared object service were identified and enumerated. These lead to a set of services which we believed a general shared object service would be required to provide and support. Of the COMIC partners, a subset of the total services reflected particular interests, both within COMIC and within other on-going relevant projects.

Within the operational parameters of the COMIC effort, it appeared sensible to allow partners to select a number of these services that they felt they could best investigate and, in most cases, implement prototypical versions. The Shared Object Service is one of the aspects of COMIC nearest to existing, established technology -- with important differences -- and lends itself readily to this investigative approach to assess the feasibility of the provision of such services.

Most of this portion of the deliverable concentrates on the definition, description, implementation and experience with the prototypes. The purpose of this introductory chapter is to recall the services required, list the prototypes developed, and briefly describe which services are addressed (and how) by each prototype.

The identified services:

Service Element	Description
Access	Facilities which control users' ability to access different aspects of the service.
Association (linking; relationships)	Facilities which allow objects to be interrelated; to form potentially complex objects structures.
Awareness	A set of facilities which allow objects to manage the level of awareness they provide to different users of the service.
Events, Filtering and Subscription	A set of service facilities which allow the handling and management of events within the service.
External References	Facilities provided by the shared object service to allow some form of bridging between electronic objects within the service and other information objects outside the service.
Filtering and Subscription	A set of facilities which allow different foci of interests to be defined and managed for different groups of users.
History	A set of history mechanisms which promote asynchronous cooperation by recording the history of object use.
Locking	A set of facilities to support coordinating concurrent access to shared objects by cooperating users.
Object Interaction and Invocation	Facilities which allow objects to interact with each other and indicate how objects are to be invoked.
Object Presentation	The presentation of different objects and aspects of the service to end users. This is achieved through a specialised user agent which determines a set of appropriate presentations for each user.
Persistence	Facilities which allow objects to survive beyond user sessions.
Queries and Views	Facilities to allow the definition of different views of objects within the service and the use of these
Trading	Facilities for locating objects which best match a user's requirements.
Versioning	Facilities to support and manage the recording and maintenance of versions in cooperative setting.

The prototypes:

CoBoard	Developed at KTH
CoDesk	Developed at KTH
COLA	Developed at Lancaster
GroupDesk	Developed at GMD
Interface Builder (IB)	Developed at KTH
Resource Manager (RM)	Developed at UPC
SOL	Developed at Lancaster
Trader	Developed at UPC

The initial results of the prototyping effort are summarised in the matrix below. Notice that some entries are marked "not yet"; the services have not yet been prototyped but there is a strong indication that they are required as part of the prototype's overall function.

The matrix:

	SOL	COLA	GroupDesk	CoDesk	CoBoard	IB	RM	Trader
Access	yes	yes	not yet	yes (UNIX)	yes	not yet	Yes	Yes
Association (linking; relationships)	yes	yes	yes	yes	yes			
Awareness	yes	yes	yes	yes	partly	not yet		
Events	yes	yes	yes	used via broadcasts	partly	yes	Used	Used
External References	yes		not yet	yes	yes			
Filtering and Subscription		yes	not yet	yes	not yet	not yet		
History			yes	not yet	not yet	not yet		
Locking	yes	yes	not yet	not yet	not yet	yes		
Object Interaction and Invocation	yes	yes	yes	yes	yes	yes		
Object Presentation	yes	yes	yes	yes	yes	yes		
Persistence	yes		not yet	yes	yes	yes	Yes	Yes
Queries and Views			not yet	partly via assoc.	not yet	yes		
Trading	•	•	•				Yes	Yes
Versioning			•	not yet				

As we briefly examine each system, we present a “local” portion of the above matrix, in the format shown below.

Acc	Assoc	Aware	Event	Ext	Filter	Hist	Lock	Inter	Pres	Pers	Query	Trade	Vers

Please note that further details of the prototypes can be found in individual chapters in the rest of this deliverable.

7.2 SOL

Acc	Assoc	Aware	Event	Ext	Filter	Hist	Lock	Inter	Pres	Pers	Query	Trade	Vers
yes	yes	yes	yes	yes			yes	yes	yes	yes			

Access: Access is used to configure each user interface object. Each user has an access permission for each interface object. The permission describes how that user may interact with that object, and how that interaction is shared among users.

Association (linking; relationships): The shared interface objects are linked to each other in the interface hierarchy. They also link to the application/SOS/other interface objects using a simple message/token passing mechanism.

Awareness: One user’s actions may be immediately broadcast to a subset of other users, informing them of some action, whether it is moving an object, resizing it, selecting a button, entering text, selecting text, etc.

Events: Events from the X-Windows interface invoke callback/event handlers to be called within the application. Using the Policy node, these events may be used to file other events such as passing a Token to the SOS.

External References: To specify policies we may use a “user profile” which is a simple file holding information about users, similar to the environment variables held within a csh in UNIX. The access specifications, role information and policy information/specifications are all held within an external database, which is referenced by the server.

Locking: When one user interacts with an object in their display, this object may be locked for others; for example if one user moves a button around, none of the other users may move it at the same time. This is also the case for resizing and selections of interface objects. The level of locking is configurable.

Object Interaction and Invocation: Interface objects may invoke actions on other interface objects using the policy node. Users interact directly with the interface objects. This interaction may cause some kind of invocation (application call, SOS call etc.).

Object Presentation: Each interface object uses the access information to configure its representation for each user. It maintains consistency for any shared methods on the object, i.e. if the movement method is shared between two users, both their images will move synchronously if moved by either one.

Persistence: All the access specification, user panel and policy information is held within a database, and persists when an application or even the server terminates. While a session is running, any number of users may leave and then rejoin, and they will see the application in a state as it would be had they not left. The session will hold its state even when no users are interacting with it, but when the session ends it is up to the application to save its state.

7.3 COLA

Acc	Assoc	Aware	Event	Ext	Filter	Hist	Lock	Inter	Pres	Pers	Query	Trade	Vers
yes	yes	yes	yes		yes		yes	yes	yes				

The COLA platform is a realisation of a lightweight model designed to augment and assist CSCW applications through the provision of semantically “light” or free cooperative mechanisms. These mechanisms bridge the existing void between distributed systems, which tend to be highly abstract in their support for CSCW, and CSCW environments, many of which can be classified as heavyweight with constraining cooperative semantics. The two main areas of the platform are its *lightweight activity model* and a *shared object service*. Interaction between users, objects, applications and services with the platform components occur through libraries or a number of graphical browsers.

The relationships of people to the platform, and of objects and people to each other, is provided through the lightweight activity model. In a lightweight activity,

groups of people and objects participate in some common task, or activity, and may be grouped into "roles" within the activity. No semantic attachment is made to the notion of a role or what may be required in an activity. This grouping of people and objects into activities and roles forms unique <user, role, activity> triples, called a user's cooperative "context" which is presented by objects and users whenever they interact with COLA.

In order to provide some explicit awareness of the current state of the cooperative activities, as well as a general mechanism for passing information around, the COLA platform uses an event service. Events may be sent to and from users, objects and activities with reference to their cooperative contexts. In order to allow these users to control the amount of information which they receive, each activity has its own event manager where users can register events, which are subsequently used to filter out unwanted events, or to subscribe to others which would not normally be received.

The COLA shared object service uses and supports the lightweight activity model by building upon any available distributed system services. At the core of the shared object service are special objects called object adapters. Adapters are activity dependent intermediary objects which reside between the clients of one or more shared objects and the objects themselves. These objects are said to be "adapted". The only way a client may invoke an operation on an adapted object is through the adapting objects interface. This interface is a combination of: operations which only apply to the adapter; interface operations from the underlying objects; and adapter operations which map down onto underlying object interface operations. The ability for adapters to include their own independent functionality into the presented interface enables the adapted objects to be extended with new "cooperative" or "shared" functionality. For example, the adaptation of an existing simple file object allows the introduction of several *locking* schemes which control consistency with multi-user access to the file, even though the file remains unaware of the multiple clients.

The adapter interface itself is presented differently to different clients in order to reflect the current state of the cooperation the objects are in, and also the cooperative context the client has. This flexible and dynamic presentation forms a basis for access control over the adapter operations, and therefore over all the adapted object operations. Access to shared objects within the COLA platform can change according to the current state of the lightweight activities, the client's context within the activities, and which operation the client actually wishes to invoke.

7.4 GroupDesk

Acc	Assoc	Aware	Event	Ext	Filter	Hist	Lock	Inter	Pres	Pers	Query	Trade	Vers
not yet	yes	yes	yes	not yet	not yet	yes	not yet	yes	yes	not yet	not yet		

The GroupDesk system supports the cooperation of people working at different times in offices that may be located at different sites. It allows people to structure their different workspaces and to share them with other persons. The GroupDesk system presents itself as a number of desks (each of them displayed by a window) each representing a workspace that may be shared with other members of a working group or representing the personal workspace which is private. The system wants to make one aware of other persons working in the same workspace, of the activities going on within the workspace and the whole environment. With our aim of using events in particular to provide awareness, the evaluation of events that occurred and a suitable presentation of the events were important during the first stage of the implementation process.

The implementation of the GroupDesk system provides a general event class that contains the following attributes:

- a unique id
- a source which is a reference to the object that represents the person who has caused the creation of the event through interaction with the system
- a target that is a reference to the object that represents the real entity which is the target of the modification or activity (for instance a document that is moved by someone)
- the time when the event was generated and
- the kind of event: modification or activity.

Two specialised event classes are derived from the abstract one: they represent modification and activity events. Whereas modification events describe modifications of objects (changes of the contents of a workspace that are valid for “a longer time”, modification of a document), activity events describe short-living changes to objects (the opening and closing of a document without saving). Activity events contain an attribute specifying the type of event (what has happened exactly). In addition, activity events have a corresponding event: for example a close-document event corresponds to an open-document event (when source and target are the same in both cases).

At the moment these events are explicitly exchanged by the objects involved. Further plans are to propagate these events via associations that constitute a semantic net together with the objects. This will be done when the underlying implementation of the Common Object Request Broker Architecture (CORBA) provides the association service.

The awareness service provides functions to process generated events. Here we try to reduce the amount of generated events and to relate corresponding events to each other (identification of corresponding events). This information may be used

to sort out events that do not have to be recorded in a history (as an open-document event and a close-document event without any modify-document event in between).

Object interaction and invocation is achieved through the facilities the Object Request Broker provides. The first time a client object wants to invoke a server object, it calls a special object used for administration purposes (which knows about all existing objects and provides a naming service) and gets back a handle to the server object. All further invocations are forwarded by the broker.

The GroupDesk system distinguishes semantic objects and objects that present the object and the state of the object to the user. The latter contain more information about the graphical representation as the corresponding icon.

Until now, the implementation of CORBA used does not include any object services that are defined by the Object Management Group (OMG) as, for example, the persistent storage manager, object versioning or object security (access control). We plan to integrate these into the current system as they mature.

GroupDesk currently supports certain types of relations. The supported relation types are currently structural and operational relations. Unsupported relations are semantic and interest relations.

7.5 Resource Manager and Trader

Acc	Assoc	Aware	Event	Ext	Filter	Hist	Lock	Inter	Pres	Pers	Query	Trade	Vers
yes			used							yes		yes	

The prototype presents the Resource Manager and the Trader with a unique view, so it is presented to users with a unique interface and therefore, in the matrix, both have the same set of properties or services.

The services that our prototype supports or will support are:

Access control: In a system where there is competition to use resources we need to add instruments that assure orderly access to them. Access rights and load balancing are instruments to provide resource management, and it includes concepts like access control, accounting, failure detection and recovery. In order to enforce these controls we need to apply *policies*. We understand as policies the set of rules that govern access to resources or objects. But policies are not only the restrictions, policies are also the way in which the authority applies these restrictions and how the authority interprets all the policies.

Events: Some kinds of events are supported in order to get information about objects and adapters. This information is gathered to generate the dynamic information needed when anyone works with an object via adapters. For instance, the performance information consists of data about the current quality of the services. The resource manager obtains this information from events.

Persistence: Our prototype supports a kind of fault tolerance. It means that the system will attach a new object to a user if their current object goes down. We

have named this service as dynamic reconfiguration, and it could be seen like a kind of persistence service (not still supported).

Trading: In a large distributed environment a user cannot know all the available services. Also, it is not possible that this user can know the access point of each service (its interface reference). In this kind of environment the services could dynamically change their access point and services offered. Therefore, to be able to profit from the multitude of services available in a distributed environment, there is the need for a means of finding the resource that best fits our needs. *Trading* is the mechanism that allows users and applications to find these resources and isolates from changes of name and location and even from destruction or creation of new resources. Therefore, the trading service permits users to move inside an environment that could change.

7.6. CoDesk

Acc	Assoc	Aware	Event	Ext	Filter	Hist	Lock	Inter	Pres	Pers	Query	Trade	Vers
yes	yes	yes	used via broad cast	yes	yes	not yet	not yet	yes	yes	yes	partly via assoc.	no	not yet

The Collaborative Desktop (CoDesk) is an attempt to make collaboration a natural part of daily computer use. Our way to achieve this is to put the user in the centre of the computing experience in a similar way that applications and documents are defined and visualised in the desktop metaphor. We have extended the traditional desktop metaphor with a few new objects that enable cooperative work.

All Codesk objects are per default persistent and, as files in the standard filesystem, could be public or protected through access permission rights. Codesk objects are currently stored in a database but could be linked to external UNIX files as well as being used as references to internal links.

As in most desktop systems operations are performed with direct manipulative actions. Objects present themselves as icons in a window based environment. However as a user starts to interact with the objects (the icons) different views of the objects could be presented, e.g. as an info dialogue. All kinds of objects could be invoked by simple drag-and-drop operations. E.g. by double-clicking on a document an editor will start to enable editing of that document.

CoDesk is a basic environment for CSCW. Without limitation to a specific model of cooperation each user could tailor or form her desktop to their individual need for cooperation and communication. In CoDesk it should be as easy to look for your colleagues as for shared or individual working material. Central to CoDesk is support for groups or teams to form cooperative settings. By e.g. being aware of the presence of other users and their access to shared objects.

Different forms of cooperative awareness are reflected in CoDesk in graphical forms as highlighting of icons, change in position, naming and size. An active object indicates that it is used, e.g., a user that has logged in to the CoDesk. An object can also be notified, this provides a mechanism to trigger colleague's attention to certain objects. A notified object expires after a certain time and becomes a passive object -- this is the default awareness mode.

Primarily CoDesk provides mechanisms that extend the network from a computer network to also be a user network by integrating the essence in communication and collaboration through different tools and media. Events are used in CoDesk for a wide range of purposes. For communicational use events are used for broadcasts to initialise a call or conference session.

We are currently building prototypes to test how some of the CoDesk objects, such as documents, could be extended to have multiple versions and a history. This functionality is currently in the CoDesk SOS but we have not yet created an easy to use graphical interface to the versioning and history functionality.

The rest of this deliverable is structured as follows:

8	Rodden and Mariani	Requirements for the COMIC Shared Interface Service In this chapter, the requirements for the SIS, the necessity for which was established in Deliverable 4.1, are listed.
9	Smith and Rodden	Using interface objects to support cooperation In this chapter, aspects of the SOL system are described and discussed. Many of the techniques used to support cooperative interfaces match the requirements for the SIS, as discussed in chapter 8.
10	Trevor et al	The use of adapters to support cooperative sharing Adapters were identified as an important mechanism for realising some of the requirements for the COMIC SOS. This chapter further explores these issues, through the design, use and discussion of the COLA system.
11	Fuchs and Prinz	Supporting User Awareness with Local Event Mechanisms This chapter explores the provision of awareness using event mechanisms, as prototyped in the GroupDesk system.
12	Eiderback and Hagglund	Interface Builders and Bulletin Boards: Techniques and Requirements on SOS/SIS A system for generating collaborative interfaces is presented, and its use within a collaborative notice board (CoBoard) is described. This work also explores the use of MultiG as the distributed platform, and addresses some of the issues of the SIS.
13	Sundblad and Tollmar	The Collaborative Desktop - Experience from designing and building an environment for CSCW This chapter describes the CoDesk system and reports on the experience gathered from the system. It also considers how CoDesk might be supported by the COMIC SOS.
14	Rodriguez	The Design of the Resource manager and the Trader This chapter considers the needs of the resource manager and Trader, and the users of such SOS components, and describes a design for the components.
15	Syri	Realisation of the COMIC Shared Object Server on the basis of CORBA This chapter takes a detailed look at how the facilities of the SOS can be layered on a CORBA compliant platform.
16	Fuchs et al	A Reference Framework for the COMIC Shared Object Service This chapter analyses the material generated throughout this phase of the Strand Four work concentrating on the SOS and SIS -- much of which has been reported in this deliverable -- and attempts to synthesise the findings of the various paper and prototyping explorations into a framework for the SOS.
17	Rodden	Objects in space, the spatial model and shared graphs This chapter considers a mathematical foundation for representing the spatial model and shows its applicability in existing non-spatial shared object applications within COMIC.

Chapter 8

Requirements for the COMIC Shared Interface Service

Tom Rodden and John Mariani

Lancaster University

8.1 Introduction

This chapter briefly outlines the requirements for a shared interface service that complements the facilities provided by the COMIC shared object service. The aim of the shared interface service is to provide an appropriate set of mechanisms to allow user interfaces to be shared in a manner which encourages cooperation. To enable this, the shared interface service provides mechanisms to support the development of cooperative interfaces and a set of 'run-time' support facilities. An object oriented approach to user interfaces is adopted and interface objects in the shared interface service promote equivalencies between the shared interface service and the shared object service.

A variety of different forms of user interface have been exploited in CSCW exhibiting different properties. These interfaces have varied considerably in the interaction techniques and technology they exploit. As in the case of single user systems early interfaces were predominantly text based and tended to support cooperative interaction in an asynchronous time independent manner. More recent graphical interfaces support concurrent real-time interaction. A number of shared interfaces have been developed which exploit shared virtual reality.

To promote correspondence with the shared object service we wish to consider these different forms of interface in terms of the way they exploit shared information. In our outlining of the shared object service three significant levels of abstraction were identified for sharing information.

The windowing or environment level

This level of sharing determines the style of interface adopted. For example, a number of CSCW systems have been developed which support linked graphics screens. Similarly, systems have been developed which support shared virtual environments.

The presentation level

Presentation level sharing allows a number of cooperating users to exploit different views of information to support cooperation. This presentation can be tied to the needs and role of each of the users in the cooperative activities. A number of development environments have provided facilities to define and support this form of sharing.

The object level

This level of sharing focuses on the use of shared information as a means of mediating cooperation. This form of facility could be provided by the shared object service. This arrangement allows radically different interface styles to co-exist. For example, the same shared object could be accessed through a text interface, a graphics display and from within a shared VR environment.

The aim of a shared interface service is to provide part of a platform for cooperative systems. In particular, our outlining of the shared interface service assumes the existence of a closely related companion shared object service. The functional relationship between these services and the user applications which use them are shown in a Figure 8.1. The diagram is reproduced from our earlier outlining of the shared object service and is intended only to illustrate the functional relationships between the services rather than to suggest any particular architectural structure or separation for the service.

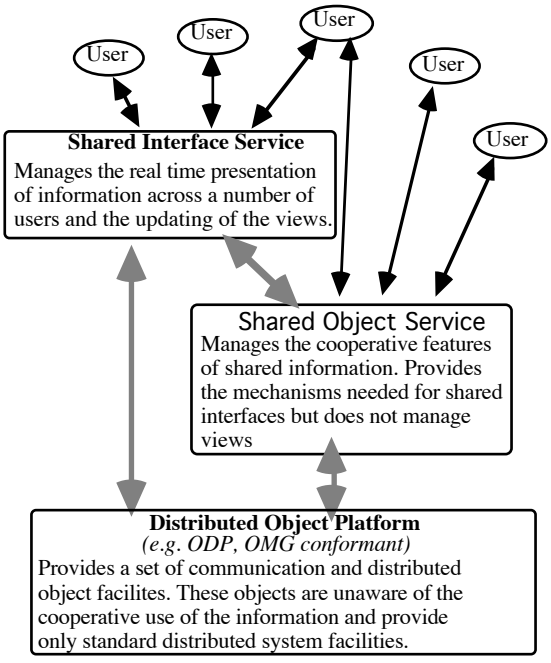


Figure 8.1: Relationships between the shared interface and object services

The explicit identification of the shared interface service highlights the close relationship between a shared user interface and the form of cooperative support provided. For example, one form of shared interface service could be workstation

based visualisation facilities such as those provided by the MEAD system [Bentley, 94]. Other alternatives include the shared interaction provided by TheKnowledgeNet [Marmolin, 91] or the spatial representation and interaction provided by the COMIC spatial model [Benford, 93].

Each of these approaches to support cooperative interaction could in themselves constitute a specific instance of a shared object service. Our task in outlining a shared interface service is to consider the development of a set of services which can be widely applied to support a range of applications. To this end we have chosen to focus on the provision of a shared interface service for workstation based environments. This focus complements the approach of the spatial model which examines the use of novel interaction techniques to support cooperation. Obviously, the facilities provided by the shared object service can be exploited by applications independently of the form of interaction techniques they exploit. We would also expect shared objects and the facilities provided by the shared object service to be used to support cooperation across heterogeneous forms interface.

8.2 Shared Interfaces Within the Service

The provision of facilities within the shared interface service depends on a particular object oriented model of the user interface. Interfaces within the shared interface service are constructed from a set of shared interface objects. These interface objects are analogous to the use of interface widgets in most modern user interfaces toolkits. As in the case of their single user counterparts, shared interface objects embody particular styles of interaction. In addition, interface objects supported by the shared interface are aware of the shared nature of their context and provide a number of additional cooperative features.

Shared interface objects are intended to provide a generic set of cooperative facilities which sit between the community of user interacting with an application and the particular semantics of the application. This model of interaction builds upon the separation between surface and deep interaction outlined by Took [Took, 90] and generalises a range of previous approaches to the construction of cooperative application including Rendezvous [Patterson, 90], Liza [Gibbs, 89], MEAD [Bentley, 94]. Each shared interface object:

- Is responsible for its presentation across a range of user displays
- Provides mechanisms to generate and interpret events
- Can invoke procedural behaviour within the application
- Provides facilities to manage interaction across the different displays involved.

Each of the interface objects supported by the shared interface service distinguish between management interaction which allows users to dynamically alter the cooperative aspects of the shared interface and application interaction

which invokes particular behaviour within the application being shared. The general role of interface objects in the service is shown in Figure 8.2.

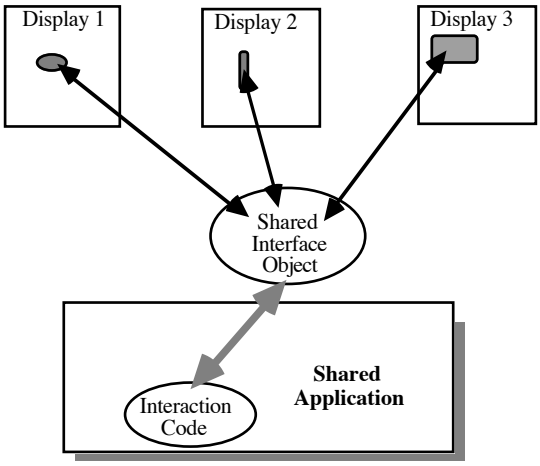


Figure 8.2: The role of shared interface objects

Each interface supported by the shared interface service is in fact constructed from a collection of shared interface objects. Each of the interface objects within an interface are aware of the different displays accessing them and how they are presented on each display. In addition the shared interface service provides facilities to manage the composition of objects to construct application interfaces and the detailed run time facilities required to support this model of shared interaction.

Consider the portion of a simple interface consisting of a text field and two buttons shown in Figure 8.3. As in the case of most single user interfaces this shared interface is constructed from four separate interface objects with an associated set of composition rules. These composition rules record the physical and logical arrangement of the visual components of the interface. In this case the objects are arranged in a hierarchy with the placement of interface objects recorded as offset positions.

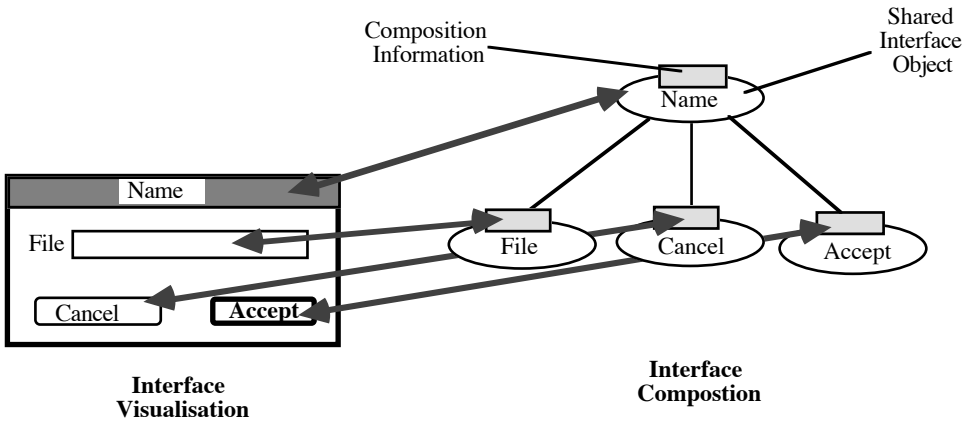


Figure 8.3: The composition of an interface

A crucial issue in cooperative interfaces is the extent to which the interfaces presented on different displays by the application are linked together or coupled. A number of researchers have investigated the coupling of interface and have suggested that the WYSIWIS approach originally advocated by Stefik *et al.* [Stefik, 87a] needs to be relaxed allowing local private views [Stefik, 87b; Greenberg, 91; Dewan, 91]. These private views are often referred to as being decoupled from the shared public view. The behaviour of the shared interfaces and the level of coupling between the interfaces is determined by the extent to which the different aspects of the state information associated with the interface are shared or replicated. In the example shown in Figure 8.3 five different forms of state information are evident:

- The state information associated with the underlying application. This may be stored and shared using the shared object service.
- The state information within the shared interface object. This state information focuses on the cooperative aspects of interaction and is managed by the shared object service.
- The state information associated with the visualisation of the shared object. In effect, the way in which the shared interface object is shown to the user.
- The state information recording the composition of the shared interface composed from the collection of shared objects. In addition to defining the composition of a particular application this compositional information can be used by windowing systems to represent the different arrangements of a series of shared applications within a set of displays.
- The state information associated with the interaction device used to manipulate the application. In most single user cases this is displayed as a cursor which can change shape to indicate different interaction modes. In the multi-user case this is also associated with the actions of a particular user.

The decision to allow users to have shared copies of this state information rather than enforce an acceptance of a single shared state determines whether the effects are private or public. By merely deciding if users have the ability to alter and manipulate copies of different state information we can effectively determine a considerable amount of the cooperative behaviour associated with the shared application. The implications of state information being shared or replicated on the cooperative interaction properties of an application are briefly summarised in Table 8.1 shown below.

State information	**Shared**	**Decoupled**
Application	Consistent application state with limited race conditions. Poor semantic feedback. Limited ability to scale up.	Complex management needed for consistency. Good semantic feedback. Scales up with limited effect.
Interface object	Consistent co-ordination of cooperative action.	Limited control over co-ordination.
Visualisation	Same view of interaction presented.	Alternative views of interaction elements allowed.
Composition	Consistent view of application presented.	Similar view of interaction presented.
Interaction	Users can see the interaction of others using some form of telepointing.	The interaction of users is not propagation although the effects of interaction may be.

Table 8.1: Different forms of state information and the effects of sharing

The representation and management of this state information is a core aspect of the services provided by the shared interface service. In addition, the service needs to provide facilities to manage the initiation, connection and termination of shared cooperative applications. This form of rudimentary session management is one of the most basic facilities required from a shared interface service. In fact, a range of different aspects of the shared interface service can be readily identified.

8.3 Aspects of the Service

The rest of this chapter considers the different functionality required of the shared interface service. As in the case of the SOS the focus of this examination is in identifying how various aspects of the service may be realised as part of the service. The initial set of service elements considered in this document are outlined in the following table with a brief description of the particular responsibilities of each service element.

Service Element	Description	Section
Distribution and Communication		
Registration and session management	This section of the service manages the connection and registration of users displays.	Section 8.4.1
Events, Filtering and Subscription	The use of window level events is mapped down to semantic events for shared objects.	Section 8.4.2
Sharing Interaction		
Highlighting and interaction propagation	The ability to select areas of the screen and interface elements and propagate the selection of these interfaces.	Section 8.5.2
Application interaction and Call-back	The cooperative invocation of application code is managed through the use of shared information and policy statements	Section 8.5.3
Telepointing	This section manages the use of pointers and telepointers in a distributed fashion	Section 8.5.1
Shared Display Management		
Window/Environment Management	Local arrangement and the sharing of different parts of the environment is managed by this section of the service	Section 8.6.1

Table 8.2 : Different Aspects of the shared interface service.

8.3.1 Architectural Implications

Any realisation of the shared interface service will be constructed from a number of components. Each of these different components determine different interaction properties and the architectural arrangement of the shared interface service is significant. Architectures for multi-user interfaces originate from distributed systems research and must address problems such as network delay, loading and latency. It is possible to identify two extremes of pure *centralisation* and *replication*, between which lie a continuum of hybrid architectures.

Centralised architectures

In a pure centralised (or *client-server*) architecture a central server program handles all user input and displays output events which are routed by way of local client programs. Local workstations act as graphical terminals and window servers. A variant is the *master-slave* architecture where one client is merged with the server and all other nodes run clients.

The primary advantage of the client-server approach is simplicity; application and all data are held centrally, simplifying access management and data consistency. Implementation is easier still with a networked window system such as X Windows. This approach is used in shared window systems and is widely adopted by computer conferencing systems. It is relatively easy to support presentation level sharing as the server can replicate display directives to all clients.

It is also possible to support view level sharing using the client-server approach. For example, Rendezvous [Patterson, 90] is based on a client-server

architecture with all user interaction and display management handled centrally. Each user has an associated *view process* which interprets input events and display directives. The sharing policy is detached from the information being shared, as each view process can interpret events and display directives differently, supporting alternative information representations.

The embedding of the sharing policy in the central server means that the system developer is responsible for interface tailoring. Rendezvous, for example, does not make the sharing policy visible, severely limiting end-user tailoring. The centralised architecture is also vulnerable to failure of the central node (or the network connections to it), and delayed feedback as all events must travel over a network.

Replicated architectures

At the other extreme, replicated architectures maintain exact copies or *replicas* of the application on each workstation. Each replica handles screen management and feedback locally and broadcasts any change in application data to all other replicas to maintain consistency. Local display management means that different views are easily supported. End-user interface tailoring is relatively easy to provide, as each replica can adapt its sharing policy to the user's preferences.

The major difficulties with replicated architectures concern synchronisation and data consistency. Users can perform actions simultaneously which are executed locally before being broadcast to other machines. If these actions conflict - for example one user deletes the selected object in a group drawing program at the same time as another user changes the selection to a different object - inconsistent interfaces can result due to events arriving in a different order at each machine.

To prevent such *race conditions* requires complex synchronisation algorithms. The standard distributed systems solution is to use a global clock to timestamp each event and then *rollback* should inconsistency arise, replaying events in temporal order. This is unacceptable for multi-user interfaces where screens may have been updated, and alternatives based on transforming updates to prevent rollback have been developed [Ellis, 89].

A further problem occurs when users wish to join a group session after it has started. This *dynamic registration* is straightforward with a centralised approach as new clients need only contact the central server. The server can then broadcast the current state of the application to bring the new client up to date. Using a replicated approach, however, a new replica must contact all other replicas to tell them that it needs to receive updates. This means that new replicas must know or can find out the locations of all other replicas.

Arrangement of components in the service

Both centralised and replicated architectures offer benefits and limitations. As neither of these architectures fully meets multi-user interface requirements, a

hybrid solution is required where different components of the interface service are centralised or replicated depending on the requirements of the facility they provide. The following table summarises the different aspects of the service identified and their distribution arrangement within the service.

Service Element	**Arrangement**
Registration and session management	Centralised
Telepointing	Centralised
Highlighting and interaction propagation	Centralised
Application interaction and Call-back	Replicated
Events, Filtering and Subscription	Replicated
Display Management	Replicated
Window support	Hybrid

Table 8.3 : The architectural arrangement of different aspects of the service

The following sections of this chapter outline each of these different aspects of the service in turn before considering the ways in which the interface service can be used to support different applications. We begin by considering the need to provide low-level support and management for the communication involved in real time shared applications.

8.4 Distribution and Communication

Our first consideration within the service is the form of support provided for the distribution and communication aspects of the service. Cooperative real-time interfaces are distributed in that the interface is presented across a range of displays and interaction must be managed from a number of machines connected by some form of communication medium. The interactive properties of a shared interface are closely tied to the properties of the underlying communication medium. One result of this strong dependency is that the shared interface service needs to provide users with a set of facilities to manage the supporting communication. In particular, users need to be able to control the starting up and closing down of shared application and the propagation of effects across displays.

8.4.1 Registration and Session Management

A central part of multi-user interface support is the low level management of the sessions needed to support a number of users when they interact with a shared application in real time. It is important that some form of facility is provided which supports this session management. Session management in this sense will include:

- Authentication of users
- A directory of currently available applications

- Specification of available facilities and the display needs required for applications
- Initial connection with application and set up of communication channels
- The addition and removal of participants during a session
- Location and naming management for user displays and applications.

These facilities require the establishment of a number of communication mechanisms within the service. The majority of these services involve directory and location work and consequently require centralised management. Issues of importance in the establishment of these communication mechanisms include:-

- A central register of applications and the facilities needed to access applications. An approach analogous to the use of trading within ODP and OMG would be appropriate.
- A protocol for establishing connection of displays to applications in an interface independent manner. This could well build upon the features of network based user interface systems such as X-windows.
- A central name server to allow logical access to named windows independently of the display on which they are being shown. This allows window references to be maintained in a manner which allows interfaces and applications to be readily migrated across machine boundaries.

These components represent the supporting infrastructure of the shared interface service and provide the initial point of contact with a shared interface service and will also provide the basic facilities for managing identified distributed user interfaces.

8.4.2 Events and the Shared Interface Service

It is essential that users of a shared cooperative application are aware of the actions of other members of the community of users sharing the application. This awareness requires support from a low-level lightweight communication mechanism that abstracts away from the particular forms of interaction evident within displays. This suggests that the shared interface service needs to provide some form of event mechanism. The event mechanism used by the shared interface service is exactly the same as that used by the shared object service. Consequently, the event mechanism represents the most significant point of contact between the shared interface service and the shared object service.

The shared interface service uses the same mechanisms to define and transmit events and may exploit the event subscription services and transformation facilities to handle events between shared interface objects. There are a range of events occurring within cooperative settings. Some are particular to the presentation and display and include things like click and double-click on the mouse button(s). We will not address those actions here as they are managed locally by display managers on each workstation. These kinds of actions are

translated into more abstract actions, such as icon select. Interaction events at this level would include:

- Pointer movement
- Drag-and-drop
- Select
- Open
- Close
- Unselect

To help us uncover which aspect of the service is responsible for handling particular events we can classify the events observed in the cooperative use of application. In particular we wish to focus on three classes of events.

Personal events

These events are initiated by a user and cause some graphics update, but the effects of that update only occur on the user's own screen. For example, if a user relocates a window on his desktop. The window will only appear to move on his desktop, and not on the desktop of any other user.

Shared events

Shared events are similar to personal events, except that their effect is transmitted to all sharing users. If a desktop item is shared, then movement of a shared window by one user will cause the image of the window to move on all participant users' windows.

A simpler example is a screen pointer. If a user has exclusive access to a drawing tool, then pointer movement is classified as a personal event. If it is a shared drawing tool, then pointer movement is classified as a shared event; when a user moves his (identified) pointer, it must appear to move on all participating screens. Notice that we can begin to see how tools can be tailored as single-user and multi-user by simply changing the classification of an event.

Persistent effect events

These are events whose actions result in a "permanent" change of state of the underlying application objects. For example, when the salary field in an object representing a person, Ludwin, is updated, this may raise personal or shared events (i.e. transforming the iconic / 3D representation of the object, raising of the form for the change dialogue, etc.) but once the update is committed, this becomes a persistent event with the underlying application state being amended.

This classification is analogous to a separation between deep and surface interaction where we classify events depending on their context. Personal events have only a surface or presentation effect associated with them, while shared events have public effects and persistent events have more permanent effects on

the underlying application. This arrangement is outlined in Figure 8.4 in terms of how deep events reach within the application.

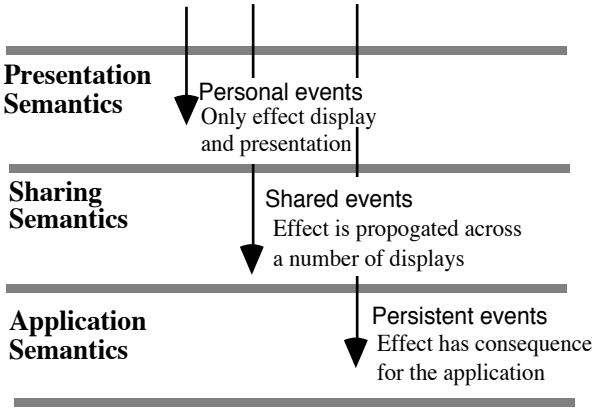


Figure 8.4: Different events in the shared interface service

Now that we have classified the events in this way, we can allocate responsibility for their effects to the different components of the overall service. In particular, we can distinguish between the responsibility of the shared interface service and the associated shared object service.

Personal events are the responsibility of a local display manager involved in realising the presentation of shared interfaces. When a graphic manipulation occurs, it only occurs on the user's own screen. There is no need for its effects to be propagated to other users. As a result the handling of this class of event is exactly the same as handling graphical updates in the single user case.

Shared events are the principle responsibility of the shared interface service. For example, when a user selects a shared object, then a "shared_object_select" event is raised. The local display indicates a graphic selection on the user's own station. The "s_o_s" event is communicated to the shared interface object in the SIS and from there dispatched to other users' interfaces as appropriate. Each user's local display interprets the event and updates their own displays accordingly.

Note that the propagation of shared events raises a number of potentially complex issues. To begin at the simplest level, when two (or more) users are cooperating via an application, if one user selects an object, then the information transmitted with the "s_o_s" event will include the name of the selected object and the name of the selecting user. A receiving user will get this event and the attached information. His process must be able to map the object id onto the matching graphical representation, to identify the other user and to present this information in an appropriate way. If we assume a system-wide user directory, then recognising the user should be relatively simple.

The responsibility for translating the "s_o_s" event into a graphical representation must belong to each individual display manager. The SIS's task is to distribute event messages across the cooperating tools. The translation of these

events to local graphical updates may also make use of particular display semantics. For example, one form of display manager may show the local effect as altering colour in a 2-D wimp display. Alternatively, the effect could be shown within a 3-D visualisation as a change of colour. More specialised display managers may exploit application semantics to open folders and windows to show the effect of events.

Lastly, we have persistent events. Events such as pointer movement and even "s_o_s" events are not persistent events. This means they are solely within the province of the SIS. If, however, we want to consider awareness and history of action, then the "s_o_s" event should be recorded in the SOS. Awareness not withstanding, any object (data or meta-data) updates must of course be recorded in the SOS. Update actions will generate "object_update" events, which will have repercussions for the SIS as it ensures the event is propagated across displays, and also, of course, for the SOS as it carries the information required to effect the change.

8.5 Sharing Interaction

The previous section considered the forms of support provided by the shared object services for the underlying communication essential to distributed shared interfaces. In addition to these "low level" mechanisms, the shared interface service provides a set of facilities to manage high level shared interaction. This form of service focuses on the benefits to be gained from sharing abstract interaction objects across a number of displays. Since the abstract interaction object is aware of the displays on which it is being shared it can provide facilities to manage and control the propagation of interaction between users. The following sections outline some of the specialised services provided by different shared interface objects.

8.5.1 Telepointing

One key aspect of shared interfaces is the need to manage a number of distinct pointing devices. Although these pointing devices may be controlled from a number of different displays they are rendered and managed on a local display. The use of telepointing is most notable in the case of shared drawing applications where a number of user input devices need to be simultaneously tracked. The shared interface service needs to provide explicit support for the management of simultaneous pointing devices. The provision of this support will require two major components. The first component adopts a replicated strategy and manages local renderings of the pointing devices. This component is responsible for managing conflicts and interaction from pointing devices. In addition, a centralised update component is responsible for determining the propagation of hints from the events generated by a pointing device.

In the shared interface service telepointers are specialised classes of shared interface objects. They are responsible for ensuring that the position of a broadcasting cursor on a local display is mapped to other displays (Figure 8.5). This will normally require the actual position of remote cursors to directly reflect the exact position of the controlling cursor. However, the way in which the cursor is presented in each display may be different.

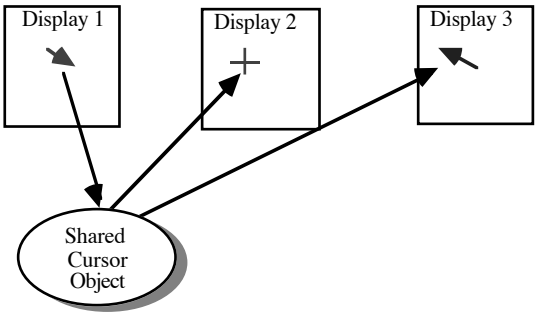


Figure 8.5 : The role of the shared cursor object

The shared cursor object is special in that it does not communicate further with any underlying application, rather it broadcasts updates across a number of connected displays. When the configuration information associated with the root display is shared then the position of the cursor is reflected directly on remote displays. However, when only portions of a display are shared the position of the cursor relative to the origin of the shared portion it is on top of is broadcast. The remote telepointers are updated using this relative position to ensure correspondence relative to the shared portion of the display. The translation of position information (or cursor warping) ensure a correspondence across shared portions of the display even when these shared portions are re-arranged locally by users.

8.5.2 Highlighting and Interaction Propagation

A significant feature of cooperative interface development is the role of selection and highlighting and the need to allow highlighting to be propagated to other users. Consider for example two users accessing a piece of text. Each could highlight two different sections which may be:

- Equal or equivalent.
- Overlapping.
- Distinct with no overlapping sections.

In the shared object service the resolution of highlighting is managed locally by the shared interface objects. The interface object maintains sufficient state information to allow selection and highlighting to be handled locally. In addition, the shared information object will allow different highlighting and selection

strategies to be defined. For example, each shared interface object may have distinct highlight propagation strategies reflecting the use of the interface object.

The ability of shared interface objects to locally manage the cooperative aspects of highlighting is dependent on a number of features:

- A common highlighting protocol for each user display agent where selections of different components by different users is represented. The level of specification for selection will vary depending on the details of the interaction component being selected.
- The centralised component shared interface objects can be used to determine the propagation of selection information based on given policies stored within the shared interface object.
- The set of replicated display components which transform the particular selection information to local window display events.

These separate components allow selection and highlighting to occur in a relatively display independent manner while resolving the different consistency issues locally for each portion of the display managed by a shared interface object. Consider for example, a text field which allows a number of users to edit a string. The shared interface object would be responsible for the locking and selection policy associated with the text field. Thus it would recognise when users had selected different portions of the text string, broadcast these selections across a number of displays and allow local display managers to interpret the particular selection details.

8.5.3 Application Interaction

Our final consideration of shared interaction in the shared interface service focuses on the manner with which a number of users cooperatively interact with a shared application. This form of interaction requires the provision of facilities which co-ordinate users to ensure that interaction with the application is meaningful and consistent. In the shared interface service this is achieved by the shared interface object interpreting user interaction with different visualisations of the object prior to invoking behaviour within the shared application.

As an example, consider a button managed by a shared interface object and displayed on three different screens. Each of the users may “press” the button at any given time by interacting with the local visualisation. Once the users have interacted with their local representations the underlying shared interface object may determine a number of different actions. It may use the interaction information to:

- Select a particular behaviour based on who has “pressed” the button by calling a particular piece of code.
- Highlight or disable the other presentations associated with button.

- Record the button as having been pressed and wait for a number of users to press the button before calling a particular piece of code within the application.
- Inform other shared interface objects that an interaction has occurred using the shared interface services event mechanisms.

Each of these courses of action and others is open to the button, and more than one may be followed by the shared interface object. Each shared interface object needs to provide a range of different policy specification techniques which allow users to dynamically amend the cooperative behaviour of the shared interface. This policy specification can take the form of:

- *Selection statements* which match different users to particular behaviours within the application.
- *Event generation mechanisms* which raise events when a particular template is matched.
- *Local state information* which records interactions allowing patterns of consensus to be measured.

The particular technique used is dependent on the particular shared interface object supported by the service. However, you would imagine a set of consistent interface objects being provided for developers to allow the construction of shared interfaces. One such interface object set is described in chapter 9 which outlines the SOL system.

8.6 Sharing Management

The final set of facilities provided by the shared interface service focus on the management of shared interfaces. The assumption is that in addition to providing shared interface objects, the shared interface service makes the configuration information detailing the composition of interfaces publicly available and shared between users. This allows the management of displays to be shared.

8.6.1 Distributed Display Management

The sharing of interface configuration information allows the management of displays to be a cooperative endeavour. The effects of reconfiguring interfaces are propagated across the community of users as the shared state information recording this information is amended. As users reposition windows within the display or alter the colour of buttons, the font of text fields, for example, these changes are propagated to all displays. It is essential that appropriate management facilities are provided to promote management as a cooperative endeavour. These facilities should include:

- An access model which controls the ability of users to access and alter shared configuration information.

- An appropriate model to control the propagation of changes to configuration information across the collection of shared interface object which make up the distributed display.
- The development of a specialised shared window management application which manipulates the configuration details associated with shared information objects.

The provision of this management information allows users to define the limits to which they can reconfigure the representation of cooperative application interfaces. By locking configuration information we can determine if portions of the application interface are shared and whether the effects are public or private. We can also specify whether the interface can be reconfigured or not and if so which users can reconfigure the display.

8.7 Using the Facilities of the Shared Interface Service

The previous sections of this chapter have considered the different aspects of the shared object service and the mechanisms required to support the shared interface service. This final section considers how the shared interface may be used to support previously defined cooperative interfaces. The intent is to provide an indication of how the shared interface may be used in practice.

To allow us to consider the use of the shared interface service we have chosen to focus on the use of the shared interface service within a developed populated information terrain. In addition to exploiting the links between the work on shared objects and VR, our intent in using this example is to demonstrate how the interface service may be used to support heterogeneous interface techniques.

Consider a number of users browsing and updating a shared information base (for example, the shared object service). The shared information base has two distinct interfaces associated with it. Both interfaces provide the same functionality (as far as object browsing and manipulating is concerned) but exploit radically different representations.

Firstly, we have a 3-D “Benediktine” Q-PIT. In this system, objects are given a position in space according to their extrinsic attribute values. Their appearance and behaviour is dictated by their intrinsic attribute values. Users are (currently) represented as monoliths which move on the display as the corresponding users change their view point (and hence position). User actions on objects are (currently) represented by colour changes to the objects being acted upon.

Secondly, we have a 2-D “Mac-style” system called NFNY. In this system, object collections (including classes) are represented as folders. When a folder is opened, it presents a window containing objects as icons and further folders if appropriate. Open objects are represented as forms and any collections within an object (i.e. a person object having children) are again represented as folders. Users familiar with the “Mac-style” will readily be able to move through an object store quickly and easily. Manipulation of objects and collections are achieved through

simple “drag-and-drop” operations. The emphasis of the NFNY system is to provide data manipulation, but it can readily be seen how schema evolution could also be supported using the “Mac-style” metaphor.

The presence of users within NFNY is indicated by iconic representations which move according to the “position” of the user within the database. Operations on objects are indicated by an icon of the user carrying out the action, and an icon which represents the user’s action.

As a simple scenario of use in both these interfaces, consider selecting a PERSON icon representing LUDWIN and updating his salary attribute. The LUDWINObject represent a person “Ludwin Fuchs” who works for GMD on the COMIC project. The following sections describe this scenario for both systems, from the point of view of the user carrying out the actions, and of a “bystander” user who is observing the actions.

8.7.1 NFNY

Acting User

Beginning at the desktop, TOM will have a display of folders. If PERSON is near the root, then it should be one of the initial folders presented. If not, TOM will have to descend through a number of folders until he arrives at the folder containing the PERSON object. Alternatively, he could engage in a query and arrive within the folder.

Now, he can select the icon that represents LUDWIN. The LUDWIN icon will be tagged with a smaller version of TOM’s icon. When TOM opens the icon, he will obtain a new window containing a form which represents the attributes of the LUDWIN object and their current values. He selects the value of the salary field and types in the new value. He then closes the form window and it collapses into the LUDWIN icon, which is now deselected. The TOM icon which tagged LUDWIN disappears.

Observer User

If the observer has the folder with the PERSON object in view, he will see a greyed-out icon of TOM tagged next to it. This shows that TOM is present within the folder. If the observer has an open window on the PERSON folder, he will see a full-sized icon of TOM present within the folder. When TOM selects LUDWIN, the observer will see a mini-TOM appear next to LUDWIN. When TOM opens LUDWIN for update, yet another mini-icon will appear next to mini-TOM, to indicate the on-going update action. When TOM completes his update and closes the form window, the mini-update icon disappears, followed by the disappearance of the mini-TOM. If TOM now leaves the PERSON folder, the full-sized TOM icon will also disappear.

Using the Shared Interface Service

In this scenario the local presentation and representation of the information as icons is the responsibility of local display agents. These display agents contain the definition of the icon representations and know the mappings from various shared interface objects, their selection states and icons. The shared interface service manages the set up of appropriate processes and the connection on new displays. In our example scenario, shared interface objects would exist within the SIS which represented:

- The PERSON object
- Folders
- The desktop
- The update form for amending salary
- Users.

Each of these shared objects manages the local presentation depending on the particular details associated with them. For example, the shared interface object for a folder would know when to grey out user icons tagged to the folder. Similarly the object in the shared interface service would know to alter the representation of the person object to reflect the actions of an editing user as different icons. The manner in which these are presented is the responsibility of the local display manager. A significant part of the development process in using the shared interface service is in defining these mapping from the abstract state representation of the shared interface service to the actual presentation of the object. Given the use of the “Mac-style” interface we may also wish to include support for telepointing. This would be achieved in the interface service by defining a telepointer shared interface object for each user. These objects would be responsible for sending update events to local display.

8.7.2 Q-PIT

Acting user

The first problem is to locate the LUDWIN object which represents the person “Ludwin Fuchs” who works on COMIC at GMD. This could be done by engaging in a text-based query exchange with the system, causing the LUDWIN object to be highlighted (i.e. colour change). The user can then move towards that object. (A possibility would be to allow the user to directly teleport to a position adjacent to LUDWIN).

Depending on how we have mapped object attributes to the Q-PIT, it may be possible to visually browse the Q-space to locate LUDWIN. If names form part of the extrinsic attributes, it may be possible to follow an alphabetic ordering of names until “Fuchs, Ludwin” is found. If the “works_for” attribute has been mapped onto an intrinsic, then people who work for GMD would have a specified shape and now objects of that shape would form the focus of visual browsing.

Once the user has located the object, they can select the object. This will cause the object to change colour to mirror the selecting user. If the selecting user is TOM (the project manager for COMIC) whose colour is red, then LUDWIN will take on a scarlet hue. If the user wants to update an attribute of LUDWIN, by indicating this intention, a form representation of LUDWIN will appear on their screen.

The LUDWIN object, meanwhile, will now have a two-tone colour scheme; red, to continue indication that TOM has selected it, and blue (say) to indicate that TOM is now updating the object.

The user, interacting via a form, updates the "salary" field, and closes the form. The update is made to the underlying database. The LUDWIN object changes from two-tone back to pure red. If the "salary" field is a participating attribute in either extrinsic or intrinsic attributes, this is not the last change in its spatial behaviour.

If "salary" is an extrinsic, then it is one of three attributes which dictate LUDWIN's position in Q-space. Having updated the "salary" of the underlying object, LUDWIN's position must be recalculated and his position moved. In the current system, this takes the form of a smooth animation as LUDWIN travels through Q-space until taking up his new position.

If "salary" is an intrinsic, then LUDWIN's behaviour may change. For example, if salary is related to spin-speed, then if the salary has gone up the speed will increase. If the salary has decreased, then so will the speed. Finally, we would expect the user to deselect LUDWIN once this is done (by again clicking on LUDWIN), and the object regains its original colour.

Observer user

This assumes the observer has a view of Q-space that contains at least LUDWIN and perhaps TOM. If TOM carries out a query, he will suddenly seem to appear in the vicinity of LUDWIN. If he browses, the observer will see TOM move about Q-space (and probably accessing various other objects) before settling down in the vicinity. When TOM (represented as a red monolith) initially selects LUDWIN, LUDWIN will change colour to red. When TOM initiates the update, LUDWIN takes on a two-toned hue, scarlet and blue. When TOM completes the update, LUDWIN returns to red. Depending on whether an intrinsic/extrinsic attribute has been changed, LUDWIN will either move smoothly to a new location (which may be out-of-sight for the observer) or his behaviour will change (i.e. speed up or slow down).

Using the Shared Interface Service

The extensive use of 3D VR technology in this interface highlights the separation between the abstract management of shared interface objects within the interface service and their presentation using the 3D browser. While the shared interface objects would contain information concerning their state and configuration, the

decision as to how they are presented is the responsibility of some display agent or visualiser. In this case the configuration agent associated with each object has been extended to include a third dimension. The shared interface objects are also responsible for the particular presentation states in an abstract manner. The mapping of these states to particular presentations is the responsibility of the local display agents.

In the case of the Q-PIT we have chosen to share the configuration information for each interface object across all display agents. Thus the position of users within the abstract world is consistently held across the displays. In addition, each interface object records whether it has been selected, its particular shape and colour and its links to underlying shared objects. To support the Q-PIT interface objects are managed by the interface service which represent:

- Objects in the Q-PIT.
- Users in the Q-PIT.
- The update form for amending salary.

The session management facilities associated with the SIS would be used to update the local displays of objects using the service and the services event mechanisms would be exploited to propagate object selection and movement.

8.8 Summary

In this chapter we have outlined the requirements of a shared interface service and how these requirements can be reflected in different aspects of a common shared interface service. The shared interface service complements an associated shared object service which manages the cooperative sharing of information. The shared interface service focuses on making explicit the cooperative context within which interfaces are used and providing facilities which manage the cooperative aspects of shared real time interaction. The intent is to identify generally useful mechanisms and techniques which can be used across a range of applications.

We have chosen to illustrate the applicability of this approach by briefly describing how the shared interface service would be used to support two different styles of cooperative interface to a shared object service. In addition, we have undertaken a number of separate prototyping activities with a view to uncovering particular interaction techniques for the shared interface service. These different detailed investigations and the associated prototyping work are presented in other chapters of this deliverable. One focus of particular note has been the development of toolkits to aid the construction of shared interfaces. A principle aim of these toolkits has been to outline an initial set of shared interface objects which a shared interface service could provide as a standard set.

8.9 References

[Benford, 93] Benford, S. and Fahlén, L. (1993), A spatial model of cooperation in large virtual environments, *Proceedings of ECSCW93*, Milan, September 1993. Kluwer.

[Bentley, 94] Bentley R, Rodden T., Sommerville I. and Sawyer P., Architectural Support for Cooperative Multiuser Interfaces, *IEEE Computer*, May 1994, Volume 27 No 5, pp 37-48, 1994.

[Dewan, 91] Dewan P. and Choudhary R., Flexible User Interface Coupling in a Collaborative System". *Proceedings of Computer Human Interactions (CHI) 1991.* ACM Press, pp 41-48

[Ellis, 89] Ellis, C. A. and Gibbs, S. J., Concurrency control in groupware systems, *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, 1989, Portland, Oregon, ACM Press, pp 399-407

[Gibbs, 89] Gibbs, S. J., LIZA: An Extensible Groupware Toolkit." *CHI '89 Proceedings*, ACM Press, 1989, pp 29-35.

[Greenberg, 91] Greenberg, S., 'Personalisable Groupware: Accommodating Individual Roles and Group Differences', in *Proceedings of ECSCW '91,* Bannon, L., Robinson, M. and Schmidt, K. (eds), Sept 25-27, 1991, Kluwer, pp 17-31.

[Marmolin, 91] Marmolin H., Sundblad Y., Tollmar K., Avatare A. and Eriksson H., CoDesk - An interface to TheKnowledgeNet, Design and Implementation. *Proceedings of the 4th MultiG Workshop,* Stockholm, May 1992

[Patterson, 90] Patterson, J. F., Hill, R. D., Rohall, S. L. and Meeks, W. S., Rendezvous: An architecture for synchronous multi-user applications, *Proceedings of CSCW 1990*, October 7-10, Los Angeles, Ca., ACM, 1990, pp.317-328.

[Stefik, 87a] Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., and Tatar, D., WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems* 5 (2): 147-167, 1987.

[Stefik, 87b] Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S., and Suchman, L., Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM* 30 (1): 32-47, 1987.

[Took, 90] Took, R., Surface Interaction: A Paradigm and Model for Separating Application and Interface. *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*. ACM Press, 1990, pp 35-42.

Chapter 9

SOL: A Shared Object Toolkit for Cooperative Interfaces

Gareth Smith and Tom Rodden

Lancaster University

The chapter presents a user interface toolkit to support the construction of cooperative multi-user interfaces. The toolkit is based on the configuration of shared interface objects to construct cooperative interfaces. A principle focus of the toolkit is on the provision of accessible facilities to manage interface configuration and tailoring. Most existing facilities to manage multi-user interfaces tend to be application specific and provide only limited tailorability for purpose built cooperative applications.

In addition, the current structure of most cooperative applications fails to separate the semantics of applications from the cooperation specific semantics. In this chapter we present a multi-user interface toolkit that provides management facilities in a manner that it separates appropriate features of cooperative use from application semantics. This is achieved by allowing multi-user interfaces to be derived from a common shared interface constructed from shared interface objects. We would suggest that the separation of semantics in this form represents an initial identification of the re-usable cooperative interface components.

9.1 Introduction

The development of applications that support a number of interfaces across a community of users has played a prominent role in CSCW. Different researchers have investigated techniques and architectures to support the real time management of these interfaces. Research has focused on the development of facilities to co-ordinate and manage interaction across the different interfaces. In general, one of two alternative strategies has been adopted for the construction of multi-user interfaces: cooperative applications have been characterised as either *collaboration transparent* or *collaboration aware* [Lauwers, 90].

Collaboration transparent interfaces focus on allowing the large body of existing single-user applications to be used by a group of users without modification. This offers significant advantages in migrating application to a cooperative setting. However, this is achieved at the cost of limiting the amount of control users have in managing the cooperative properties of the interface. In contrast, collaboration aware applications provide extensive facilities to allow users to both configure and control interfaces. The cost of this increased management is the need for the cooperative applications to be directly responsible for the facilities provided. This incurs a significant development cost and facilities developed tend to be specific to an application and little consideration is given to

providing management and tailoring facilities across a range of applications. This is somewhat in contrast to the dominant trend in single user interface development which focuses on the identification of generally applicable user interface components (or widgets) which can provide common interface facilities across a range of applications.

In this chapter we present an approach to multi-user interface construction within a toolkit called SOL that allows the rapid construction of application interfaces while also providing facilities to manage the cooperative aspects of the interface. The approach we have adopted is to consider the different interfaces presented to members of the user community by a cooperative application as being projected from a common shared application interface. This allows the management of the cooperative features to be achieved by providing facilities that control the derivation of the projected interfaces [Smith, 93]. As these cooperative facilities are external to detailed application behaviour we encourage the adoption of general facilities across a range of applications. We have realised the SOL environment through a set of interface libraries, an associated run time environment and a user interface toolkit that augments the existing X-window system.

9.2 Background and Motivation

Our aim is to provide a set of facilities that allow the realisation of effective cooperative interfaces. Central to the development of these facilities is the provision of appropriate mechanisms for the management and control of multi-user interfaces. Our approach is strongly influenced by the need to find a compromise between what is currently viewed as two disparate approaches to the provision of multi-user interfaces. The development of special purpose applications that are *collaboration aware*; and the adapting of existing single user applications to provide *collaboration transparent* shared applications.

Collaboration transparency offers the significant advantage of allowing a wide range of already developed single-user applications to be used without modification. The majority of transparent systems use an agent external to the application to multiplex output from applications onto a number of display screens and to route input from each display to the appropriate applications. In this approach an application does not know that it is being shared between users. Consequently each user is presented with identical views of the interface and only strict WYSIWIS is supported prohibiting end-user tailorability [Greenberg, 91a]. Systems that exploit collaboration transparency include Rapport [Ahuja, 88], SharedX [Gust, 88], and MMConf [Crowley, 90].

This approach also relies on an assumed model of cooperation based on many readers but only one writer. The responsibility for managing the interaction involved lies solely with an external agent that uses a floor control policy to co-ordinate interaction. This control policy normally applies to the application

interface as a whole with each user having permission to interact with the application in turn by being given the floor (Figure 9.1).

The combined result of the simple replication of output and the assumed model of interaction is that it is difficult for users to effectively manage collaboration transparent systems. Any facilities that exist have concentrated on the development of techniques to support different forms of turn taking protocols [Roseman, 93; Greenberg, 91a].

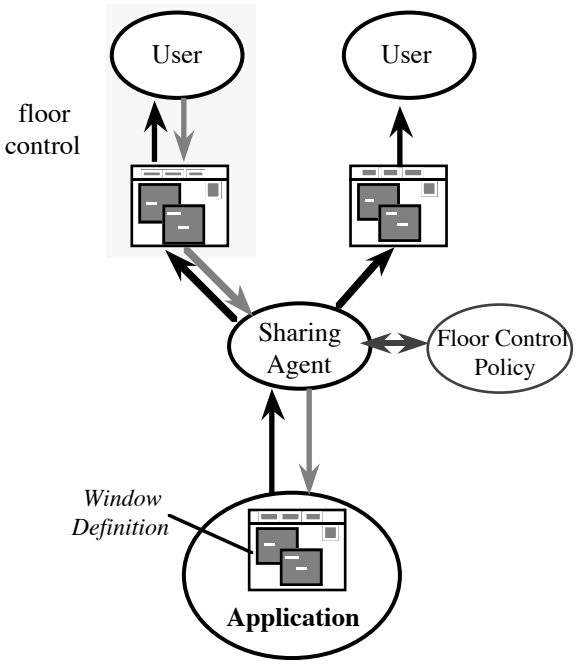


Figure 9.1: A Collaboration Transparent Arrangement

In contrast to the transparent approach outlined above, collaboration aware solutions provide facilities to explicitly manage the sharing of displays between different users. This approach has been adopted by a range of applications including Cognoter [Stefik, 87], Grove [Ellis, 88] and rIBIS [Rein, 91]. Under this arrangement it is the responsibility of the application to manage the different user interfaces presented, (Figure 9.2). Consequently, users are only supplied with the set of management facilities deemed necessary by the developer. This results in each application adopting a particular set of cooperation facilities unique to that application.

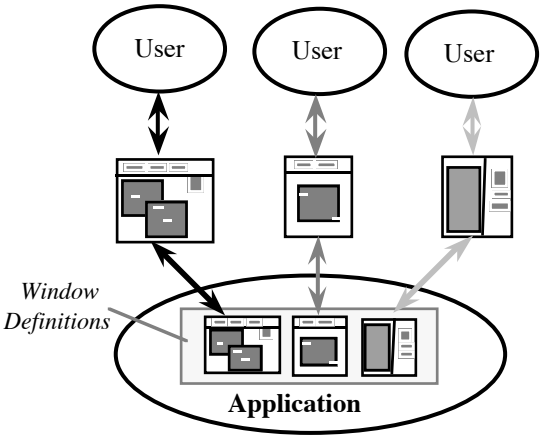


Figure 9.2: A Collaboration Aware Arrangement

Often the facilities predicted by developers of collaborative applications are inappropriate for users [Grudin, 89] and a significant proportion of reconfiguration and tailoring is needed to meet the actual demands of the application. A number of more recent collaboration aware toolkits have considered how this should be provided by the development of architectures that explicitly support tailorability [Patterson, 90; Bentley, 92]. While increasing the availability to users of cooperative management facilities, these toolkits require applications to be developed within their particular programming model rather than within a general framework.

9.2.1 Our Approach

We have developed an environment based on the X-Windowing system, which allows user interfaces to cooperative applications to be defined, constructed and managed. The aim of our work is to provide a set of cooperative interface facilities which promote user involvement in the tailoring and management of multi-user interfaces. We also wish to provide these facilities in a manner which promotes the establishment of generally applicable cooperative user interface components which can be used across a range of applications. Consequently, we wish to examine the provision of cooperative interface facilities which exist outside the application and are independent of its detailed semantics. Our intent is to provide facilities which:

- Allow multi-user interfaces to be constructed from a set of generally applicable cooperative interface components;
- Maintain a clear separation between the behavioural characteristics of the application and the cooperative aspects of the user interface;
- Provide users with a means of configuring the cooperative behaviour of applications to meet their needs;
- Provide a set of simple and consistent management facilities which can be applied across a range of multi-user applications.

The basis of our approach is to consider the user interface as a collection of different shared user interface objects and to focus on managing this sharing. Rather than manage each object and its associated views independently, we view each user interface as being derived from a common application interface constructed from a collection of shared interface objects. The use of collections of objects to construct user interfaces is not in itself novel and has been widely applied by a number of researchers. However, our user interface objects (UIOs) have been specifically developed to be shared between users and represent an initial set of cooperative interface widgets. Each of these interface objects is itself collaboration aware and can present itself to a number of users' displays in different ways. Each object also provides external facilities to manage cooperative interaction independent of the application. A selection of these objects has already been defined and constructed which mimics the facilities provided by the Motif widget set. The choice of starting with the Motif widget set allows us to port a range of existing applications with minimum changes to the source code.

Managing the user interface is achieved through the manipulation of access to this common application interface objects. One reason for choosing access as the basis for managing shared user interfaces is the familiarity and simplicity of the concepts associated with access. Access models are both widespread and well understood within multi-user computing systems. Access models have been successfully used for many years to "tailor" a user's control over information in a number of multi-user systems, such as operating systems and databases.

Our general approach is evolutionary in nature, applications are constructed in a similar manner to client programs in the X Window system. Structurally the application is similar to a normal X client, with the addition of a Display Manager (Figure 9.3). As in the case of single user applications a separation is maintained between the deeper semantic behaviour of the application and the interaction properties of the interface objects. The Display Manager is used to set up the different user interfaces, and contains an abstract definition of a common interface provided by the application. Three principle concepts are used to manage the cooperative features of the interface:

- The common abstract interface definition is used by an access client to derive alternative interfaces for different users;
- A set of mappings is provided to the display manager between shared interaction objects and the callback routines within the application;
- An interaction criteria is used to control the *extent* of collaboration required to invoke these callback routines.

End-users and application developers are not directly concerned with the display manager as it provides the mapping between users and the interface objects of the application.

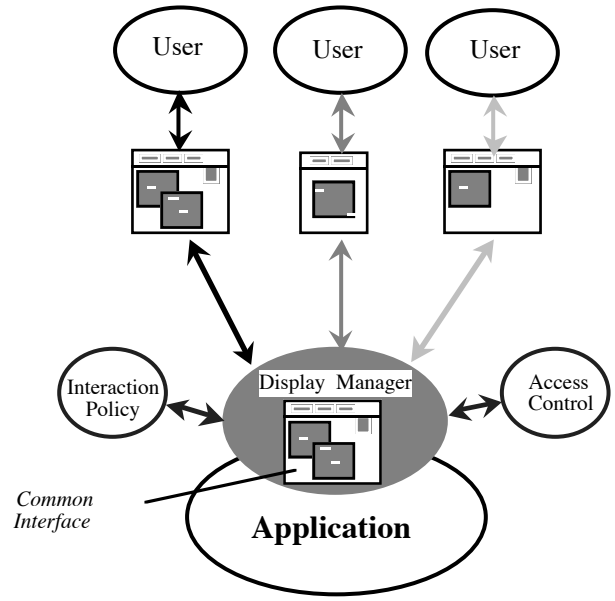


Figure 9.3: The use of a common interface definition

The use of different views of shared elements has emerged as a definite theme within real time cooperative interfaces. Private or personal views are supported in a number of systems such as Suite [Dewan, 90], Liza [Gibbs, 89] and CES [Seliger, 85]. A number of architectures also exist [Patterson, 90; Bentley, 92] which support these views through the use of an underlying object model. The model maintains a clear logical separation between a shared object and its views, and separate individual views are derived from the underlying object. Provision of such views permits a control over sharing of information. In this respect we agree with Patterson [Patterson, 91] when he states "*The essence of multi-user applications is sharing and control over sharing*". However, in contrast to the constraint based approach exploited within Rendezvous(Patterson 1990) we have chosen to extend existing multi-user control techniques to incorporate user interfaces.

The following sections consider the provision of facilities to manage the cooperative aspects of the user interface provided by our shared interface objects. We begin by considering the use of an access model to control both the presentation and crude interaction features of interface objects. This is followed by a consideration of how callback mappings can be used to provide finer grained control of the effects of interaction with these cooperative interfaces.

9.3 Configuring Presentation Using Access

The configuration of individual user interfaces within a multi-user application are specified in the SOL environment using an access control system. Each shared interface object controls a number of derived images of that object and any associated interaction. Shared interface objects may exhibit a core set of

configuration and interaction properties; that is, they may be *used*, *moved* or *resized*. Interface objects know how to present themselves to particular users by referring to an access model within the system. Interface objects have a varying number of permissions allowing users access to different interaction properties particular to the object.

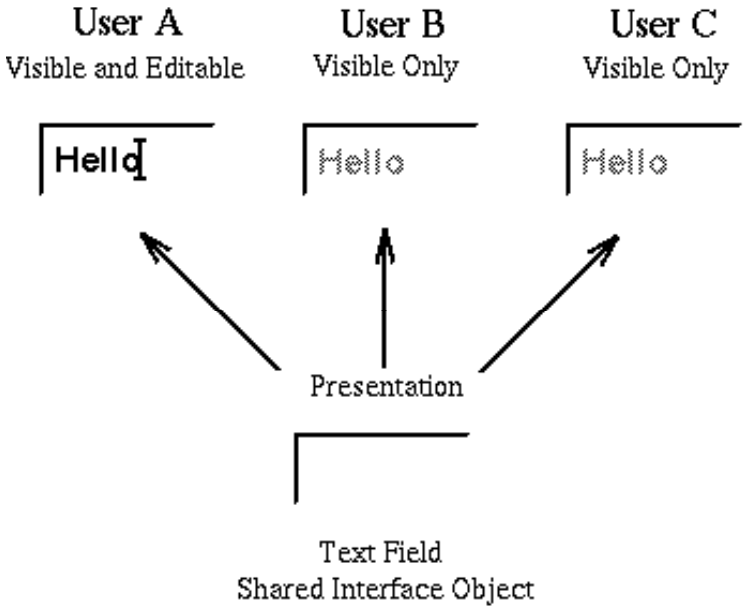


Figure 9.4: A text field shared interface object and three representations

For example, a text field User Interface Object(UIO) may show itself to three users in varying ways. A user of the text field shared interface object could have “Visible” and “Editable” permissions allowing the user to see and alter the text object. One user may see and edit the text, while the other two may *only* view it, as in Figure 9.4.

9.4 Access Models

Access control is widely used within computing systems and many access models exist [Goscinski, 91], most of which are derived from the Access Matrix Model [Graham, 72]. In this general model there exists a set of passive Objects which are the resources to be protected, and a set of active Subjects who wish to access these objects. Each Subject, Object pair (S, O) is associated with an access right(s). An access right is a privilege that a subject (S) has over an object (O). A Subject may access an object in a way specified by the access right(s) held by the pair (S, O). A number of more specific models are derived from the Access Matrix including:

- Capabilities, which divide the matrix into columns, each user holds a list of accessible objects. The addition of a new object requires each user’s access list to be updated.

- An Access control list, in which the matrix is divided into rows, and each object holds a list of users who may interact with it. It is difficult for a user to tell which objects they may use. Addition of a new user requires all objects to update their lists, and the list of users able to access an object may become large.

Although the access matrix defines the principal model for the control over a user's (or subject's) access to an object it has been used in only a few collaborative systems, such as RTCAL [Greif, 87], Suite [Dewan, 90] and Grove [Ellis, 88]. In fact, Shen and Dewan [Shen, 92] make the point that current access models were not designed for collaborative information, and are inappropriate for direct inclusion in collaborative systems. They highlight the need for access control in collaborative systems in general and argues that "most collaborative systems give all collaborators the same rights to all objects and expect access to be controlled by some social protocol". They also highlight a number of specific problems with the existing models derived from the access matrix:

- They do not facilitate the use of roles or groups; individual users must be used.
- They do not facilitate easy specification of access rights for users.
- They often do not offer a general mechanism for specification of the access matrix, or for access checking

Shen and Dewan forward an access control model to solve some of these problems. This model provides us with a number of useful innovations. However, the model they define does not directly support our needs. The semantics of the rights are embedded within the model, and they are not readily extendible. The suggested model is also heavily associated with the Suite framework. We have developed a simple access control model, that includes many of the features mentioned by Shen and Dewan, but we believe with a more generic approach. We have included this model into the SOL environment, to control individual user interfaces. In this way it is similar to Shen and Dewan's Ctool example, but we take a different approach based upon mapping different interaction properties of objects to cooperating users.

9.4.1 Our Access Model

Given a user id or a role (which represents a group of users) and an interface object id, our model returns a permission specifying how that user or role may access that object. The returned permission must not only specify how a user may access an object (in detail), but also how that object is shared among users. To specify such a, possibly complex, permission our permissions are contracted from an extendible list of simple flags.

In our case, each element in the list specifies a particular method that may be invoked on that user interface object, and how each method is shared among users. For example, a push button user interface object may have the methods:

- Usable (for buttons ‘Pressable’; for text ‘Editable’)
- Moveable
- Resizable

So a returned permission for a push-button UIO will contain at least six flags: three specifying if a user may use the button, move the button and resize the button, and three specifying whether the effect of pressing, moving or resizing the button should be broadcast to other interested parties. When the effects of methods are broadcast, all users who have also configured their interface object to broadcast that method, see the effect of that method. If a user has chosen not to broadcast a method, then any effect is local to that user’s display. This allows users to control the degree of coupling between screens. To this end, we have a 2 by 3 *permission matrix*. for the button (Figure 9.5)

Methods	Use	Sharing
Usable or ‘Pressable’		
Moveable		
Resizable		

Figure 9.5: A permission Matrix for a Button

The value each flag may assume in our implementation of the model is one of four possible states. Each state denotes whether the access is granted or denied, and whether this setting may be altered by the end-user. The four states are represented by the letters {T, F, t, f}, as denoted in Figure 9.6:

	Access granted	Access denied
Not end-user tailorable	**T**	**F**
End-user tailorable	**t**	**f**

Figure 9.6: Possible values each flag entry may assume

So, for example, an access specification for a button may be [T, t, t, F, f, t], one value for each entry in the matrix. The specification of permissions for all the users of an object may be an arduous management task. Consider for example, a complex user interface containing one hundred user interface objects, used in a domain of one hundred users, in this case ten thousand specifications must be entered. To ease this problem we adopt the use of *roles* to refer to a number of users. A role may be given an access permission to represent the permission for all the users in that role. For each interface object then, we may have a list of **role/user_id: permission** pairs. Roles are purely a shorthand for describing

access and no implication is made between roles and the behaviour associated with users in that role.

If an interface object holds a number of *role*: *permission* entries within its list, and if each role were expanded out, any one particular user may have duplicate entries. For example, an interface object may have the associated permissions:

Students	**T**	**T**	**f**	**f**	**T**	**T**
SEgroup	**F**	**t**	**t**	**f**	**f**	**F**

Users who are a member of both these roles, have two access specifications for the object, apparently in contention with each other. We resolve this problem by a simple *permission resolution* strategy.

9.4.2 Permission Resolution

A user's permissions for a particular interface object will normally be derived from one or more roles. So a particular user's individual permissions may be affected by any of the roles of which they are a member of. To allow these permissions to be consistently resolved, specifications of an interface object's permissions are ordered in a list. The top most role in the list has priority over those proceeding. Two simple rules are used to derive a user's permissions from the list (moving downwards), these rules are:

- The first occurrence of a 'T' or 'F' within a column remains, overriding any current 't' or 'f' value.
- An unspecified field (i.e. one that does not contain a 'T' or an 'F' after the list is traversed) assumes the users own preference (which will be a 't' or 'f'). If the user is not present within the list, the first occurrence of 't' or 'f' for that field is used.

So for example, some of the possible permissions (for simplicity, we have not included the sharable aspect) for a UIO may be:

	Usable	Moveable	Resizable
students	**T**	**t**	**f**
SEgroup	**F**	**F**	**t**

Figure 9.7: A UIO's access specification for two roles

If a user is a member of both the above roles, their access permission can be derived using the two rules above. In the *usable* column the first entry 'T' remains the permission for that user (using rule 1). The same rule applies for the second field, where the 'F' overrides the 't'. The second rule applies for the third field as no 'T' or 'F' has be selected, which returns the first occurrence of any value, which in this case is 'f' from the *students* role. So then the returned permission for such a user will be:

T	F	f

Figure 9.8: Resulting permission for member of both roles in Figure 9.7

9.4.3 User interface Permission Inheritance

So far, we have described the derivation of a user's permissions for *one* user interface object. Here all objects are isolated from each other, and any common behaviours must be respecified for each one. To simplify the process of UIO permission specification, we also provide a means for permissions of a parent UIO to alter those specified in the child. For example, this may permit one container object to prevent all its children from being moved, without the user having to manually alter each UIOs permissions.

We use one *permission rule* for each container object to dictate the level of influence a parent has over its children. A simple scenario is shown in Figure 9.9, where a container has a single child UIO:

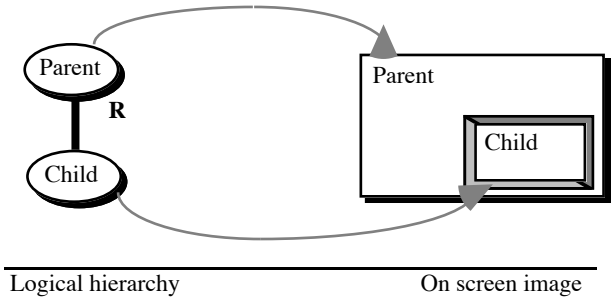


Figure 9.9: Where a permission rule (**R**) is held

The permissions *Childp* of the child UIO may be derived from those of *Parentp* in a way defined by **R**. So far we have defined two permission policies (**R**), and in accordance with the rest of the access model, these policies are extendible. We define **R** as one of:

Rule	Effect
None	*Childp* remains as specified, regardless of those specified in *Parentp*
Override	*Childp* inherits exactly the permissions held within *Parentp*, regardless of its original specification

Figure 9.10 : Two permission inheritance rules

For example, consider a menu container which may not move or be resized. We could now give that container an *Override* permission rule. This would not permit any of its children to resize or move within it, whatever their own

permissions, but makes no difference to the usable (or any other) fields. In this way we have a *permission filter*.

To derive a child UIO's permissions, we not only consult its parent UIO's permission file and permissions, but also its parent's parent, and so on. This allows us to specify restrictions over a users access to objects from a higher level of abstraction in the interface. For example, giving a container an 'F' value in the movable permission and using the *override* rule, any children of the container cannot be moved. Furthermore, if one child was also a container, none of its children may be moved either. If the case was such that the children of the second container needed to be moved, we could use the *Override rule* in the second container to grant that permission. Thus containers with override rules nearest to a UIO in the interface hierarchy have priority over those higher up.

9.5 The Access System in Use

The access model is realised as a central part of our developed system. To illustrate the access model in use, consider a simple application designed to assist lecturers. The application interface consists of a text field and two push buttons. The text field represents a pad of paper, and the buttons (*Next Slide* and *Previous Slide*) move between different 'sheets' on the 'pad'. Therefore a lecturer may enter text into the first slide, select the 'next slide' button, then enter text into the next slide and so on, see Figure 9.11..

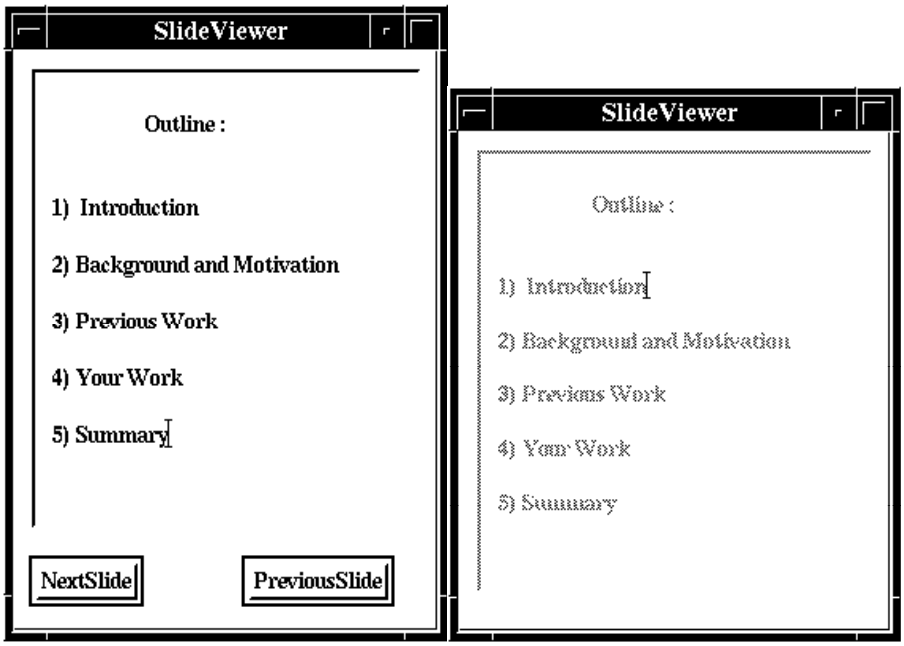


Figure 9.11: Two screen dumps of the SlideViewer application, for members of staff (left) and students

The application originally existed as a single user program for previewing slides and was constructed using a user interface development toolkit based on the Motif Widget set for X-Windows. The application was ported to our environment by amending the source code to replace the Motif widget set with our developed interface objects. This allowed us to reason about the tool being used cooperatively. We set the access permissions for the application using the roles *Staff* and *Students*, *so that both groups can see the slides, but only staff may edit and move between them.* They may not see the buttons, and therefore cannot move between slides.

We may continue to tailor the user interface of the application by adding a further role “assistants”, which is based on the students access specification, but users who are a member of this role may edit the text field. Such users may refine or edit the current slide, thus assisting the member(s) of staff but not move between the slides.

Finally, we can allow staff to alter the permission on the button to allow it to be moved around the screen. To allow this direct tailoring of the interface by users, each interface object has a management interface which shows the access permissions for that UIO. The re-arranged screen and a management interface for a button is shown below. As the results of this move are not broadcast the effect of alteration is local, allowing the user a private view of the application.

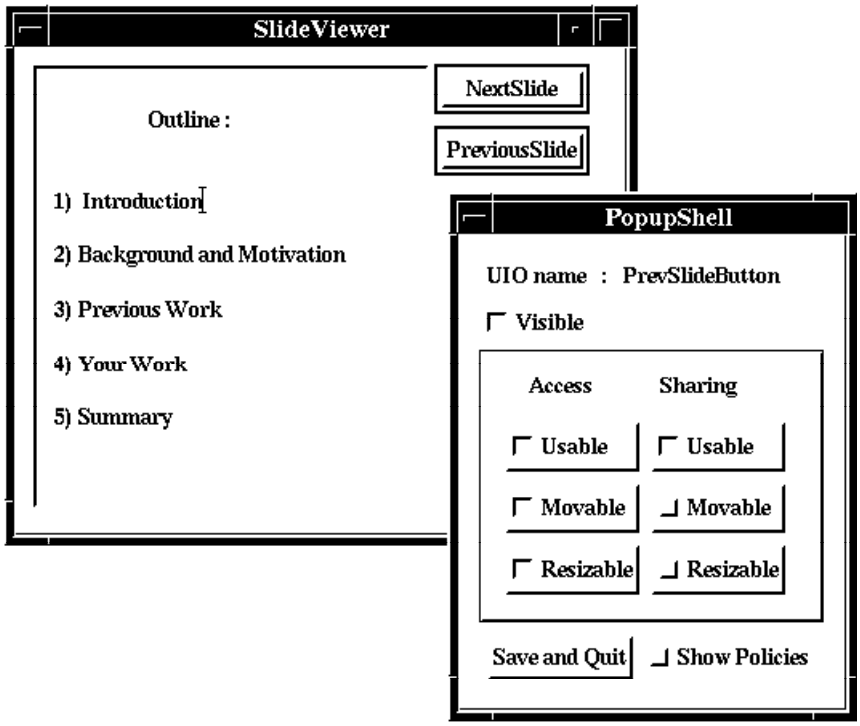


Figure 9.12: The re-arranged application and the management interface for a button.

9.6 Controlling Application Behaviour

The use of an access model to define presentation and sharing of user interface objects allows multiple users to interact with the application through differing user interfaces. For example, three users may see and use a button, although the position and size of this button may be different for each user, similarly each may or may not wish to know when anyone else has pressed it. The use of the access model within our system allows us to specify this behaviour externally to the application. Essentially, our access model makes the application's interface aware of its cooperative setting by allowing users to specify:

- The presentation and layout of interface objects on different users' displays;
- The ability for users to interact with different interface objects and for propagation of interaction across different users.

However, *input* from each user is delivered to the shared application without reference to the cooperative setting. For example, the effect of any user pressing a button is identical, in terms of the semantic behaviour of the application. This results from the way in which user interaction is handled by event driven windowing systems (such as X windows, Macintosh, Microsoft Windows). These share a common model where a procedure is called on the occurrence of a particular event. This procedure is known as a "callback routine" in the X windowing system. We use the term more generally to represent the portion of the application called from an interface object. If we choose to use the access model alone to manage shared interface objects, then interaction from any user is handled by executing the same piece of callback code (this arrangement is shown in Figure 9.13).

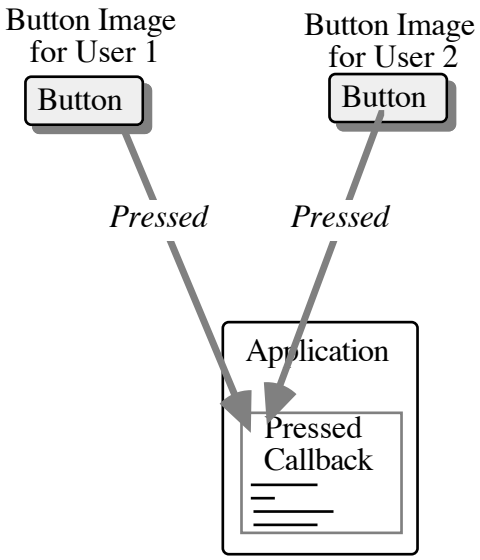


Figure 9.13: Two buttons invoking a callback independently

Often cooperative applications wish to behave differently depending on who is interacting with them. This requires the cooperative setting to have deeper affects on the application. The simple approach outlined in Figure 9.13 severely limits the possibility of providing facilities to manage the deeper semantic behaviour of the application.

9.6.1 Policy Node

In addition to our access model we provide a means of managing collaborative aware interaction through the use of a *policy node* attached to user interface objects. The policy node intercepts input from users and depending on its particular policy, may undertake some cooperative behaviour and then decide whether the callback should be executed (Figure 9.14).

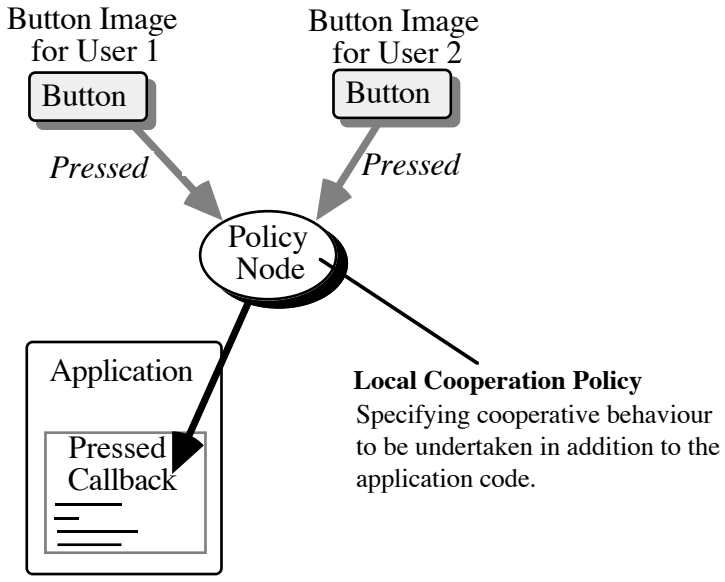


Figure 9.14: Location of the Policy node

The policy node allows cooperative behaviour to be reasoned about independently of the behaviour of the application. For example, in the arrangement shown in Figure 9.14 we could state that the callback code should only be called when both users have pressed their button. In this way we provide a means of specifying user coupling [Dewan, 91], allowing users to collaboratively communicate with the application. The use of policy nodes ensures a clear separation between the behavioural semantics of the application and the cooperative setting within which it is placed to be maintained. This separation promotes the provision of external management facilities and the migration of single user applications to a multi-user setting.

The policy node associated with each UIO receives inputs from each user when they interact with their representation of the UIO. The policy node recognises and reacts to particular forms of input to undertake different cooperative behaviour

prior to the invocation of the callback code. The policy node receives all input events from the different images of the UIO and produces an output token. The node contains a set of criteria that determine which output token will be produced for a particular set of input events. This arrangement is shown in Figure 9.15.

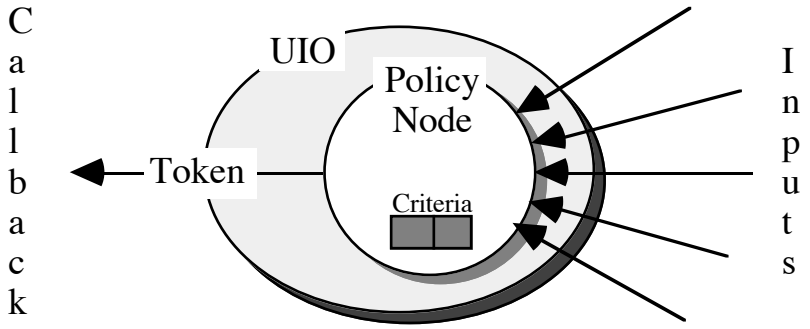


Figure 9.15: The policy node model

The criteria in the policy allows the UIO to react to the different patterns of input in different ways. Rather than simply executing the callback code, the use of criteria tells the UIO *or* the callback code that a certain condition has been met. For each recognised criteria, the policy object produces a different output "token" (Figure 9.15). The token may be either:

- *UIO specific*, in which case it commands the UIO to do something, or
- *Application dependent*, where the token is passed to the callback code, and the *application* decides the nature of the cooperation.

Tokens are simply strings, UIO specific tokens are prefixed with "UIO"; for example, *UIO_LOCK_WORD* will cause a text widget to lock the word under the user's cursor; it is not passed to any callback routine. Most UIO specific tokens facilitate semantic feedback to other users, for example other users may see that the locked word has actually been locked. Different types of UIOs have an associated set of output tokens defined for them. In developing our set of UIOs we have defined an initial set of output tokens for a selection of UIOs. Some of these and their associated behaviour are described in Figure 9.16. In contrast, application dependant tokens are ignored by the UIO and passed straight to the callback routine. These application specific tokens may be any string and are interpreted by the application.

UIO type	UIO specific token	UIO Behaviour
Text Field	UIO_LOCK_WORD	Lock the word that the user's cursor is within, (when he/she starts typing); free when the cursor moves away from that word
	UIO_LOCK_SENTENCE	Lock all the chars between the previous and next end of line chars. Free when cursor moves away from the sentence
	UIO_LOCK_ALL	Lock all the text within the text field, free when user's pointer moves out of UIO
	UIO_OVERRIDE	Ignore all locks
Scrollbars (Slider)	UIO_FORCE_MOVE_ALL	Override the access constraints and synchronise all representations of this scrollbar

Figure 9.16: A selection of output tokens associated with UIOs

The criteria determining the policy are represented in the form of a table, with one side listing the required context (type of input), and the other the token (or tokens) to be sent. A simple *if <input = context> then <token>* rule is used to interpret the policy.

We have two types of context; ***selective*** and ***consensus***. *Selective* behaviour is available for all interaction UIOs (buttons, text fields, etc.). This type of behaviour queries the status of the user interacting with the UIO. *Consensus* behaviour is exhibited within a specialised type of button. In these buttons (which may also exploit selective behaviour), the number of users who have selected their button is used as the criteria. We have two types of consensus buttons, *accumulated* and *total*. These compute the number of selected buttons over a specified sampling period.

Accumulated buttons count the number of users who are currently holding down their buttons. The sampling period is temporal (specified by the end-user, using the policy node). That is, the UIO waits for a specified time after the first user pressed their button. After this time the UIO counts the number of buttons *currently* held down. This value is then used to determine the correct token to be passed.

Total buttons operate in a similar fashion, but this time the sampling period is controlled by a user. For each button of this type, two buttons may appear; one is the consensus button itself, the other is the controlling button to *end* the sampling period. The access control system is used to constrain the availability of this second button. This type of button does not count the number of buttons *currently* held; instead, it counts the number of users who have clicked their button *at any time* during the sampling period (one per user). The sampling period is terminated when the control button is selected.

We have two types of expressions (or *criteria*) associated with the number of users who have pressed their buttons:

- An explicit number and
- A percentage value.

For example, a specification of when to execute the callback code may be "TOTAL> 5" or "TOTAL> 50%". The former executes the callback when over five users have pressed the buttons, the latter when over half the users have pressed their buttons.

As a simple example, we may wish to reflect a button's cooperative setting by using some form of consensus to determine if the behaviour associated with the button is to be undertaken. An appropriate criteria for the UIO is shown in Figure 9.17. This criteria ensures that if over fifty per cent have pressed the button the, callback code is executed and the "marginal" token is passed; if over sixty per cent of the users have selected it the token, "Overwhelming" is passed.

If the latter case is true, for example if 75% of the users have their buttons pressed, then the former case is also true (i.e. 75% is also greater than 50%). But only the token "Overwhelming" is passed, because the policy node automatically places lower bounds on each context entry. Thus in Figure 9.17, we have the ranges 0 to 50; 51 to 60 and 61 to 100.

Any explicit values in a policy node, for example "ACCUMULATED > 10" (more than ten users), are regarded as percentage values (at run time) by the policy node. These percentage values are treated as normal percentage values, and thus used to calculate lower bounds. The two tokens here are purely application specific, and are passed to the callback code. The callback code may choose to ignore these tokens, or use them to process some information in a certain way.

Context	Output Token
ACCUMULATED > 50%	Marginal
ACCUMULATED > 60 %	Overwhelming

Figure 9.17: A simple set of criteria

Simple policy nodes of this form can modify the behaviour of interaction objects within a multi-user interface in a limited way. A simple consensus button fits this model, however we are severely limited if the number of users is our only input type. To overcome this limitation we use the *selective* criteria within policy nodes. Selections can be made on information held within a *user's profile*. This allows us to also exploit information about a particular user to determine policy. For example, we can specify that we only want the button to execute the callback code if more than five users have pressed it **or** i*f a member of staff has pressed it*.

9.6.2 The User Profile

Multi-user applications which use shared interface objects are executed in a specialised run time environment [Smith, 93]. This environment includes a central daemon which all the applications communicate with. This system daemon maintains a *user profile* for each user permitted to use the system. The profile is represented as an extendible set of resources associated with each user. These may

be general in nature or application specific. Application specific resources only make sense to policy nodes associated with a particular application. More general resources such as user name, role and title may be used across a number of applications.

The policy node may use any of the resources held within a user's profile in the input context of a criteria. As an example, consider a policy node associated with a button in a shared application used within an academic department. The application is run within an environment which records user activity. A criteria for the policy node using profile information is shown in Figure 9.18.

Input Context	Output Token
PROFILE.Activity=Examiner	examiner
PROFILE.Role=Staff	staff_override

Figure 9.18: A set of criteria using a user profile

This criteria will execute the callback code when a user in the *Examiner* activity presses the button, which passes the token "examiner" to the callback code. The tokens used in this example are not UIO specific; that is, they are ignored by the UIO. The second entry in the above example operates in a similar manner, but this time passes the token "staff_override" to the callback code. The policy node using this criteria makes the shared application sensitive to activities external to the application.

The use of application specific tokens provide a powerful way to tailor cooperative interaction. It provides us with a single point of amendment to allow the migration of single user application to cooperation sensitive ones. The application callback code can be written to undertake a specific action if the token "staff_override" is passed to it and another when "examiner" is passed. The external statement of policy allows us to tailor the criteria table at run time to allow other users to emulate particular actions. For example, we can allow a particular user to undertake the behaviour associated with "staff_override" by adding the line:

PROFILE.Name="Gareth Smith"	staff_override

Figure 9.19: Amending use through an additional criteria line

We may also remove this line when it is no longer needed. This use of application tokens relies on amending the callback code within the shared application. However, when migrating applications, access to the callback code will often be limited. To deal with this situation we use UIO tokens to amend the behaviour of shared interface objects. As mentioned earlier, the UIO interprets these tokens and may alter any user's representation. In this way we allow collaboration awareness independently of the application. That is, the semantics

of the user interface defined within the application are unchanged, but the behaviour of the collaboration is defined externally, within each policy node. We have defined a number of UIO specific tokens for a number of types UIOs (Figure 9.16).

To illustrate the use of UIO tokens, consider a text field within an application set in an academic department. We define the criteria of a text UIO as shown below in Figure 9.20. In this case, students lock only the words they are editing (one at a time), whereas members of staff lock the entire sentence they are working on. If a user is acting as a spell checker they are unrestricted by any locks, and may change any word.

Context	Output Token
PROFILE.Role=student	UIO_LOCK_WORD
PROFILE.Role=Staff	UIO_LOCK_ALL
PROFILE.Activity=SpellChecker	UIO_OVERRIDE

Figure 9.20: The criteria for a text UIO

In a similar manner, we can also extend the slide viewer application introduced in our consideration of access, where a member of staff may lock all the text, and an assistant just one word at a time. We can also more finely modify the collaboration semantics for the application. For example, if we wish students to see and use the “next slide” button, but for it only to fire if the majority of buttons are pressed, and also allow a *member of staff* to move to the next slide immediately, we would use the criteria shown in Figure 9.21.

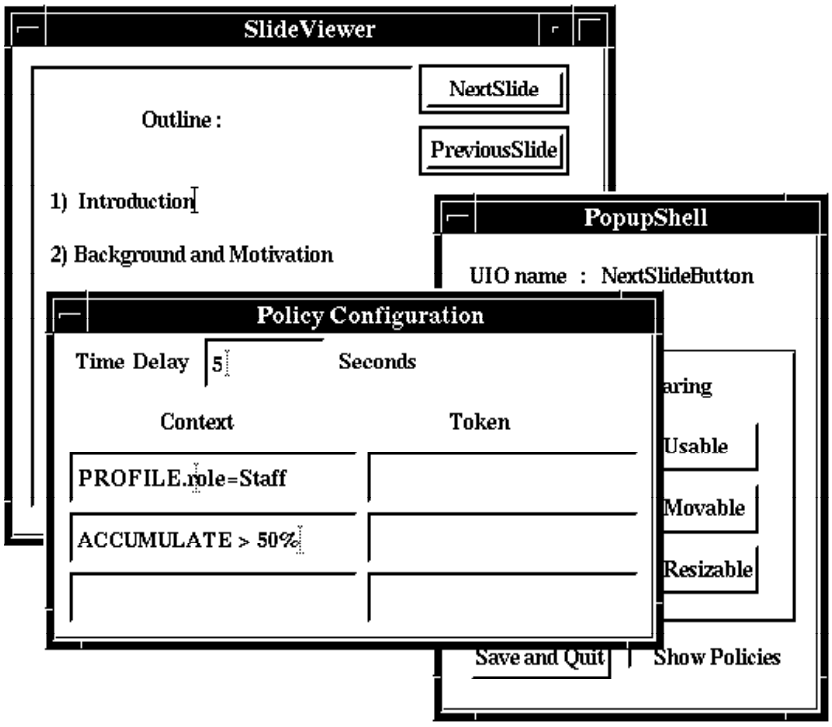


Figure 9.21: The use of policy in the slide viewer application

When we mix consensus and selective contexts, each press of a user's button is matched (during the sampling period) to any selective contexts. If a selective context is matched, then the correct token is passed (or executed), the sampling period then terminates and all the buttons are reset. If none of the inputs match any selective contexts during the sampling period, the consensus contexts are used as normal.

In this case, as buttons are pressed, the node checks if their origin is from a member of staff, and if so, the callback is executed. Otherwise the node counts the number of buttons held down *after* five seconds of the first user selecting it. If this result is greater than half the current number of users, then the callback will also fire.

The policy for the button displayed in Figure 9.21 uses the management interface associated with the shared interface object. These management interfaces are available to user during run-time to allow detailed tailoring of the shared user interface.

9.6.3 Node Linking

Multi-user interfaces in our environment are all constructed from objects. We can use the policy node to not only send a token to the callback code or the UIO itself, but also to another UIO. This allows us to reflect cooperative use externally from the application. For example, we may require a title of a label to be modified at run time, depending on whether a member of staff or over half of the users have selected it. A suitable criteria is shown in Figure 9.22.

Context	Output Token
PROFILE.Role=Staff	call “MainTitle” <Title ‘Staff ‘>
ACCUMULATED >50%	call “MainTitle” < Title ‘Most’>

Figure 9.22: Passing tokens to other UIOs

In the figure above, we can see the button reacts to either a member of staff pressing it or at least half the users pressing it. In the former case the label field of the “MainTitle” object will change its title to “Staff”. Likewise its title will change to “Most” when over 50% of the users have pressed the button.

When we wish to communicate with other UIOs in the user interface, the format of the output token is:

[**Token**] [**call** “UIOname” < Attribute Value>]

A call is made to a UIO of the name “UIOname” and passed a value attribute pair for the UIO. In this case the title of the UIO is altered however we could amend other properties of the user interface by sending different attribute value pairs (e.g. <Size 20x20> or <Coords 200x20>).

Updating of other UIOs via the call with a token is restricted by the access mechanism. That is, a call made by a token to update another UIO’s size would only take place if that user has the resizable permission for *that* UIO. Further, if the user has specified that the resizable method was shared, this change may also be reflected in other users’ displays.

9.7 The Toolkit

We have developed a number of tools to assist with the creation of cooperative interfaces within the SOL environment. A user interface builder, written with the MOG toolkit [Colebourne, 93], allows a user interface to be constructed by “drawing” it using a number of user interface objects (see Figure 9.23). The output of this tool is the C++ source code of the interface using UIOs. This code may then be compiled and used immediately within our environment.

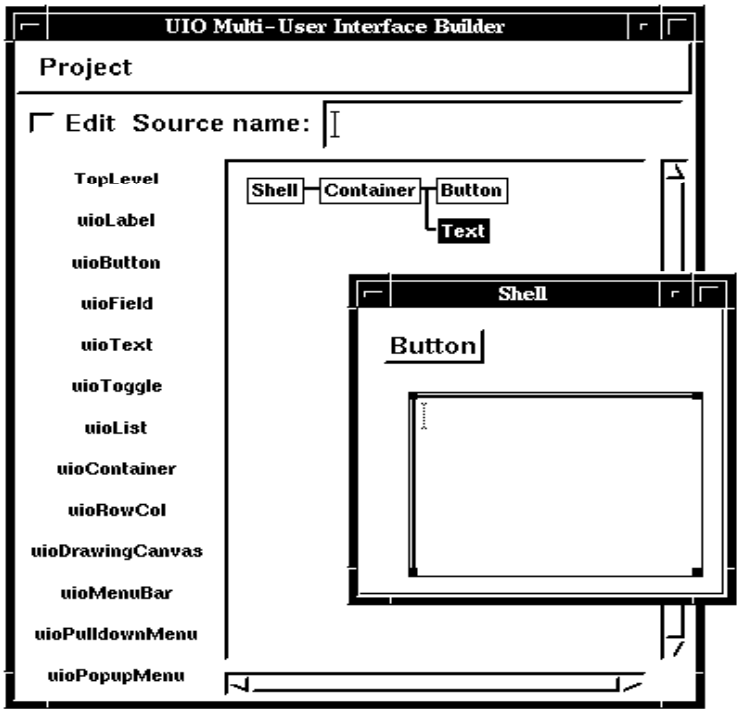


Figure 9.23: The SOL user interface constructor

The user interface builder constructs the common user user-interface for an application. The interface for the slide viewer application used in earlier examples in this chapter was created in this way, using a canvas, a text field and two buttons.. The user interface construction tool was originally a single user application. However by replacing the existing “single user” interface widgets we have converted the previous single-user application to one which is collaborative aware in our environment. The tool runs within our environment allowing the user interface builder to be used collaboratively. We have used the builder to construct a range of applications including, a shared brainstorming systems, a minute taker and a cooperative network management facility.

The access control and interaction policies are specified after the application interface is built from UIOs. This may either be done by modifying access and policy on a per object basis through their management interfaces. Alternatively, the UIO can be configured from an initial set-up by an authorised user. At any time in the application’s lifetime, end users may tailor their interfaces using the management menus associated with the interface object.

9.8 Future Enhancements

A desired addition to our environment is support for inter-application communication. Initially this may take the form of a messaging bus, but a shared memory approach to application interaction may be preferable, for the reasons of efficiency outlined by Koshizuka and Sakamura [Koshizuka, 93]. We also wish to

investigate some distributed issues. Our environment is distributed, but coarsely grained. Each application, user interface and the system daemon may execute on different machines, but we still have a central node of failure, in the system daemon.

Moreover if we scale up our domain from local area, to wider area networks, the delays incurred may make the environment unusable. Semantic feedback may be sluggish, and delayed events may render the timed *consensus* buttons useless. We would therefore desire our environment to support more fine grain distribution, including a replicated federated system daemon to be used in such a scenario.

We wish to explore support for an activity model [Araujo, 88; Danielson, 88], by using the *user profile* to store activity information. We are also investigating the possibility of extending the UIO linking call, to support an attribute value pair containing an *access specification* for the new UIO. This would allow a great deal of flexibility in automatically manipulating multi-user interfaces at run time, as we may turn other UIOs "off" and "on" (visible / invisible or usable / greyed out) through other interactions. Using the messaging system, we may extend the node linking feature to call UIOs held within *another* application, to further enhance inter-application interface communication.

So far, we have implemented little to support telepointing. Current telepointers, in systems such as GroupSketch [Greenberg, 91b], are used in WYSIWIS areas of each user's display. In a non-WYSIWIS environment, we cannot directly map the coordinates of one telepointer to another, because the item being pointed to may not be at the same coordinates for all user interfaces. We therefore wish to experiment with techniques to support non-WYSIWIS telepointing in our toolkit.

9.9 Conclusions and Summary

We have provided a means to tailor collaborative aware user interfaces within our environment using an access control system. Using this system we can accurately control any particular user's access to any user interface object, and specify how that object is shared. It allows us to use tightly coupled collaborative aware user interfaces and (at the same time) collaboration transparent user interfaces.

The use of a policy node provides collaboration aware or collaboration transparent *application input*. That is, the application may receive input from users individually, or from a group of users acting cooperatively. The node provides us with context sensitive input. For example, one user's interaction with an interface object may result in a different action to that of another user. Also, the action of a group interacting cooperatively with an interface object may differ depending on the number of users.

We have developed a number of cooperative applications using the SOL environment and toolset. These include a simple brainstorming generator tool, a minute taking facility and a network administration tool. In addition, we have also

used our environment to produce cooperative aware multi-user tools from existing single user applications. An example of this is the simple slide viewing tool, this simple application was converted into a powerful cooperative tool with ease.

The mechanisms that support these collaborative features are external to the application code. Separating application behaviour and cooperative use in this way allows us to outline an initial set of generally applicable cooperative features. We belief that the development of cooperative interface objects in SOL and their associated access and policy nodes represents an initial outlining of a set of cooperative interface widgets. We believe that the development of general purpose building blocks of this form represents a significant research goal for cooperative application developers. In addition, our use of shared objects allows existing single user applications to be readily ported to collaborative settings. In essence we require the use of existing motif widgets to be replaced by SOL's interface objects.

The collaborative mechanisms of SOL interface objects are dynamic in nature. They allow the collaborative behaviour of an entire, or any part of, a user interface to be changed at run-time (by any subset of users). These mechanisms within interface objects may also be used to provide "automatic" features, such as changing another UIO's label, size, etc. when a specific input is received. Since these mechanisms are associated with general user objects rather than particular applications, shared user interface objects also promote orthogonality of cooperative facilities across different applications.

9.10 References

[Ahuja, 88] Ahuja, S. R., Ensor, J. R. and Horn, D. N., 'The Rapport Multimedia Conferencing system', in *Proceedings of the Conference on Office Information Systems (COIS88)*, Allen R. B. (ed.), March 23-25, Palo Alto, Ca., 1988.

[Araujo, 88] Araujo, R.B., Coulouris, F., Onions, J.P. and Smith, H.T. (1988): *The Architecture of the Prototype COSMOS Messaging System*, Elsevier Science Publishers B.V., North-Holland.

[Bentley, 92] Bentley, R., Rodden T., Sawyer P. and Sommerville I., An Architecture for Tailoring Cooperative Multi-User Displays, in *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*. 1992, p. 187-194.

[Colebourne, 93] Colebourne A., Sawyer P. and Sommerville I., "MOG user interface builder: a mechanism for integrating application and user interface" *Interacting with Computers,* Volume 5 No. 3. Pages 315-331, 1993.

[Crowley, 90] Crowley, T., Milazzo, P., Baker, E., Forsdick H. and Tomlinson R., 'MMConf: An infrastructure for building shared multimedia applications', in *Proceedings of CSCW '90*, October 7-10, Los Angeles, Ca., ACM, 1990, pp 329-342.

[Danielson, 88] Danielson, T. and Pankoke-Babatz, U. (1988): "The Amigo Activity Model.", in R.Speth (ed.):*Research into Networks and Distributed Applications*. Elsevier Science Publishers B.V., North Holland, pp.227-241.

[Dewan, 90] Dewan P., "A tour of the Suite User Interface Software." *Proc. of the 3rd ACM SIGGRAPH Symposium on User Interface Software and Technology*, October 1990, pp 57-65.

[Dewan, 91] Dewan P. and Choudhary R., "Flexible User Interface Coupling in a Collaborative System". *Proc Computer Human Interactions (CHI)* 1991. pp 41-48

[Ellis, 88] Ellis, C., Gibbs, S. J. and Rein, G., 'Design and use of a group editor', Technical report STP-263-88, MCC, Austin, Texas, September 1988.

[Gibbs, 89] Gibbs, S.J., "LIZA: An Extensible Groupware Toolkit." *CHI '89 Proceedings*, ACM Press, 1989, Pages: 29-35.

[Goscinski, 91] Goscinski, A., *Distributed Operating Systems - The Logical Design* Addison-Wesley, 1991. Chapter 11 Resource Protection. Pages: 585-647

[Graham, 72] Graham G. S. and Denning P. J., "Protection: Principles and Practices." *Proc of the AFIPS Spring Joint Computer Conference* 1972, Pages: 417-429

[Greenberg, 91a] Greenberg, S., 'Personalisable Groupware: Accommodating Individual Roles and Group Differences', in *Proceedings of ECSCW '91,* Bannon, L., Robinson, M. and Schmidt, K. (eds), Sept 25-27, Kluwer, 1991, pp 17-31.

[Greenberg, 91b] Greenberg, S. and Bohnet, R., "GroupSketch: A multi-user sketchpad for geographically-distributed small groups" *Proc. Graphics Interface* 1991. pp 207-215

[Greif, 87] Greif, I. and Sarin, S., "Data sharing in group work." *ACM Transactions on Office Information Systems*, 5(2) pp: 187-211, 1987.

[Grudin, 89] Grudin, J., Why groupware applications fail: Problems in design and evaluation. *Office: Technology and People*, 1989. 4(3): p. 245-264.

[Gust, 88] Gust, P., 'Shared X: X in a distributed group work environment', presented at the *2nd Annual X conference*, MIT, Boston, January 1988.

[Koshizuka, 93] Koshizuka N. and Sakamura K., "Window Real-Objects: A distributed shared memory for distributed implementation of GUI applications". *Proc. of ACM Symposium on User Interface Software and Technology (UIST'93)*, Atlanta, Georgia, USA, November 1993. pp237-247.

Lampson B. W. "Protection." Fifth Princeton Conf on Info. and Systems Sciences, Pages: 437-443

[Lauwers, 90] Lauwers, J. C. and Lantz, K. A., "Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems." *CHI'90 Proceedings*, ACM Press, 1990, pp: 303-310.

[Patterson, 91] Patterson J., "Comparing the programming demands of single user and multi-user applications". *Proc. of ACM Symposium on User Interface Software and Technology (UIST'91)* November 1991. pp 87-94

[Patterson, 90] Patterson, J. F., Hill, R. D., Rohall, S. L. and Meeks, W. S., 'Rendezvous: An architecture for synchronous multi-user applications', in *Proceedings of CSCW '90*, October 7-10, Los Angeles, Ca., ACM, 1990.

[Rein, 91] Rein, G. L. and Ellis, C. A., 'rIBIS: A real-time group hypertext system', *International Journal of Man Machine Studies*, 34(3), March 1991, pp 349-368.

[Roseman, 93] Roseman M. and Greenberg, S., "Building flexible groupware through open protocols" *Proceedings of COOCS'93 ACM international conference on Organisational Computing Systems*, San Jose, November 1993, ACM press. Pages 279-288

[Seliger, 85] Seliger, R.., *The design and implementation of a distributed program for collaborative editing*. Masters thesis, MIT, Cambridge, MASS., Sept 1985. Also Lab for Comp. Sci. Tech. Rep. TR-350

[Shen, 92] Shen, H. and Dewan, P. (1992) 'Access Control for Collaborative Environments' proceedings of CSCW92, Toronto, Canada, ACM Press, pp: 51-58.

[Smith, 93] Smith. G. and Rodden T., "Using an Access model to configure multi-user interfaces" *Proceedings of COOCS'93 ACM international conference on Organisational Computing Systems*, San Jose, November 1993, ACM press. Pages 289-298

[Stefik, 87] Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L., 'Beyond the chalkboard: computer support for collaboration and problem solving in Meetings', *Communications of the ACM*, 30(1), January 1987, pp 32-47.

Chapter 10

The Use of Adapters to Support Cooperative Sharing

Jonathan Trevor, Tom Rodden and John Mariani

Lancaster University

This chapter examines the importance of providing effective management of sharing in cooperative systems and argues for a specialised service to support the cooperative aspects of information sharing. The relationship between features of the cooperative shared object service and existing services is briefly examined. A number of management services of particular importance to CSCW systems are identified. The chapter presents a technique of realising a shared object service by augmenting existing object facilities to provide management of their cooperative use. These facilities are realised through object adapters that provide additional cooperative facilities and greater control over the supporting infrastructure.

10.1 Introduction

Shared information plays a central role in CSCW systems. It is often the primary means used to develop a shared understanding across a number of users working together. The nature of this sharing and the forms of cooperation it facilitates vary greatly from application to application. For example, systems such as Ensemble [Newman-Wolfe, 92] allow a number of potentially distributed users to work together in real time while annotation systems such as QUILT [Fish, 88] allow a group of users to work in a time-independent manner.

Information sharing is strongly dependent on the facilities provided by the supporting distributed infrastructure. However, a mismatch often exists between the manner in which facilities are provided and the diverse nature of cooperative applications. This is most visible in the balance of control between users, applications and the management of the facilities provided by the supporting platform. Distributed systems have traditionally employed a prescriptive approach to management and control. Successful control has focused on hiding the problems associated with distributed systems from users. For example, both the LOCUS [Thiel, 92] and ANSA [ANSA, 89] projects identify a number of distinct areas of control, and associated with each a *transparency* that allows the feature to be hidden as necessary.

Unfortunately, the combination of prescriptive control based on transparency and the desire to create a single user view of the system can hide the existence of users from each other, actively prohibiting cooperation [Rodden, 92]. This

conflicts with the inherent nature of CSCW systems which requires users to work together often using common tools and information. In contrast to the existing system view of control, CSCW systems and environments need a *user* oriented view of control. This control should seek to promote direct involvement of users in managing the cooperative aspects of the infrastructure.

The aim of this chapter is to examine the management and sharing requirements placed on underlying platforms by CSCW applications, and to report new mechanisms developed to support the sharing required from these infrastructures. The work reported here is the result of the development of a set of services to manage information sharing across cooperative systems. Many of the facilities provided in the service have been informed from a social consideration of the use of documents [COMIC, 93]. Our particular focus within this chapter is on the manner in which we can augment existing distributed systems with cooperative facilities. These facilities are provided in a manner that promotes user involvement in the management of sharing.

10.2 A Shared Object Service

The intent of a shared object service is to provide the necessary platform for sharing in cooperative systems. Thus, rather than consider the problems of CSCW as being essentially a user interface issue, we are examining the more fundamental and generic information storage implications that emerge from a consideration of cooperative work. To do this we need to make a clear separation between the facilities required for shared objects, and those which need to be provided to support ‘real-time’ shared interface facilities. This separation is realised by the provision of a number of closely related services that manage a particular set of facilities. Two services of particular note are the shared interface service and the associated shared object service. The shared interface service makes considerable use of the facilities provided by the shared object service. The relationship between these services and the different user applications that make use of them is shown in Figure 10.1. The diagram is intended only to illustrate the functional relationships within the service rather than to suggest any particular architectural structure or separation for the service.

The explicit aim of the shared object service is to provide a set of services that allow objects to be cooperatively shared by a community of users. Previous services such as those provided in the COSMOS project [Araujo, 88] have focused on the provision of multi-user storage facilities. The distinguishing feature from existing multi-user storage services is the removal of prescriptive control and the provision of mechanisms which support the management of cooperative sharing. The shared object service provides a set of facilities for a group of users that abstracts from the properties of underlying infrastructure to provide a well-defined abstract set of cooperative services.

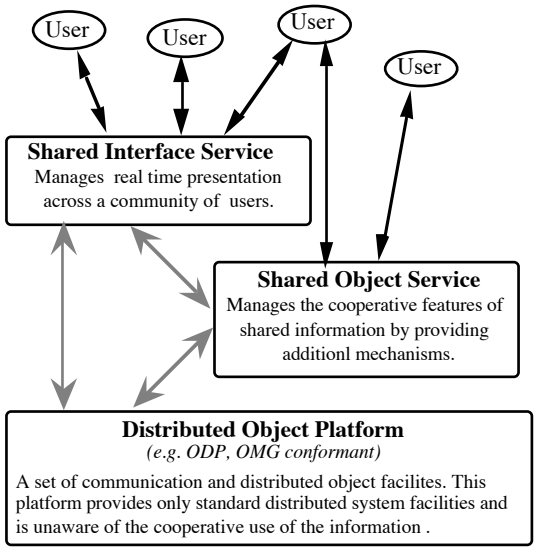


Figure 10.1: Role of the shared object service

Our particular concern in this chapter is the shared object service and the means by which services are realised within the shared object service. The separation between the shared interface service and the shared object service allows us to highlight the close relationship between a shared user interface and the form of support provided for cooperative work. For example, one form of shared interface service could be workstation based visualisation facilities such as those provided by the Rendezvous [Patterson, 90] and MEAD [Bentley, 92] systems. Other alternatives include the shared hypertext facilities provided by the Sepia [Haake, 92] and TheKnowledgeNet [Marmolin, 92], or the spatial representation and interaction provided by a VR presentation [Benford, 93]. The point is not that any of these is particularly appropriate but that all can make use of some form of shared object service. The provision of a service thus provides us with a platform from which to consider supporting cooperation through shared objects and integrating a number of approaches and applications to cooperation in a relatively seamless manner.

10.2.1 Shared Object Service Requirements

Many of the facilities provided by the shared object service are motivated by a number of previous studies of work [COMIC, 93]. These combine with a number of technical limitations to outline a set of general requirement for shared objects in the cooperative object service. The most notable of these include:-

Lightweight semantics through a policy and mechanism separation

The dynamic nature of cooperative work means that the manner in which shared information is accessed and used at any given point can change quite dramatically. Given the variable nature of this context of use it is important that the information content of objects is preserved independently of their use. To

ensure this, an object should not contain any semantics regarding the cooperation setting in which it is used. It is thus essential to allow policy to be imposed separately from basic objects which, while functionally complete, may require additional facilities within a cooperative setting. This separation also allows existing single-user objects to be used in a shared manner by overlaying new cooperative facilities.

Awareness

To promote cooperative sharing in the service, objects will need to make their use public across the service. The key to allowing this form of object use is the provision of awareness facilities in the service. These facilities need to combine an awareness of:

- The users accessing the object.
- The environment in which the object operates.
- Users' interests in the information contained in the object.

The provision of this awareness within the service is intended to balance the existing transparency facilities provided by distributed infrastructures.

Flexible access control and dynamic presentation

CSCW systems and the activities that they support can change rapidly and quite dramatically. Access to objects and the operations that they provide must be flexible and dynamic enough to reflect changes in the state of the group activity. Greif and Sarin [Greif, 87] also outline the need to present shared data differently, depending on the access criteria allowed to the client. Consequently, the supporting infrastructure needs to allow a number of alternative presentations of objects to different clients.

Adjustable and extendible semantics

There is a need for both existing and new objects to cope with the extra demands, both functionally and semantically, placed upon them by multiple cooperating clients. For example, if a normal file object is used by multiple clients, some of its methods need to be *extended* to incorporate the knowledge necessary to preserve the consistency of the data in the file. New methods must be attached to objects to allow clients to make use of these existing, but extended, operations.

These requirements apply to the cooperative sharing of objects in general and need to be addressed at an *appropriately general* level. By developing a cooperative shared object service, support comes from a combination of the *additional cooperative services* and the *underlying distributed system*. A key to the provision of these services is the way in which objects are represented and managed in the shared object service.

The use of objects within the service is directly influenced by existing object models provided by popular distributed systems such as ANSA [ANSA, 89] (based upon ODP [ISO/IEC, 91]) and CORBA [OMG, 91] (based upon the OMG

object model). Objects within the shared object service are abstract units that embody both state and procedure. Objects managed by the service publish an interface that makes the behaviour they provide available to other users of the service. The shared object service provides a set of user oriented facilities to manage the cooperative features of shared objects. Before considering the nature of the facilities provided by the shared object service it is worth briefly reviewing previous approaches to object management in distributed platforms.

10.3 Managing Objects in Distributed Systems

A common abstraction within distributed systems is the notion of an object which encapsulates both state and behaviour. An object model associated with the distributed system defines common object semantics for specifying the externally visible characteristics of objects in an implementation-independent way. The externally visible characteristics of objects are described by an *interface* which consists of operation signatures. These model the external view of both object behaviour and object state. Objects make public these interfaces to the distributed infrastructure through some process of registration with a central Trader or object broker. Clients exploit the services provided by objects by calling an operation associated with the object. This often makes use of specialised distributed system techniques such as remote procedure calls (RPC). The general arrangement is shown in Figure 10.2.

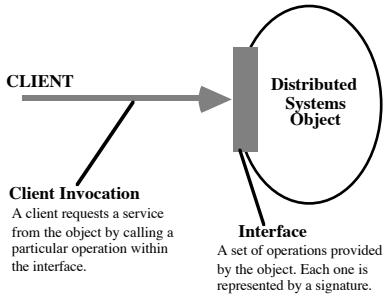


Figure 10.2: Objects in a distributed System

Management of objects within distributed systems should be flexible enough to allow a wide variety of control. However, Davies et al. [Davies, 93] indicate that systems which aim to be flexible often tend to tie the system into supporting only the design environments for which they were configured initially. This makes evolution, or the provision of support for multiple environments difficult. A key reason for this paradoxical situation is that the management mechanisms include semantic information concerning the domain of application. In contrast, we would suggest that flexible management is dependent upon the provision of an appropriate set of lightweight mechanisms. The need for flexibility in any general

distributed system has generated a number of mechanisms which can be used as the basis of flexible management. The most notable of these include:

- *Dynamic run-time binding* allows a client to invoke an operation on an object without requiring the object's interface (typically inserted as a stub into the client) to be known when the client is first compiled. This type of binding allows dynamic presentation of objects and extendible interface semantics, since interfaces no longer need to be fixed permanently at any given time.
- *A separation of mechanism and policy*. Many distributed systems separate the mechanisms which the client employs (provided by one or more servers) from the policies which determine what those mechanisms will do. This separation minimises the amount of semantics which needs to be embedded in an object. Finally, provision of group mechanisms in a distributed system enables objects, and groups of objects, to be controlled and manipulated as a whole or as individuals [Birman, 93].
- *Extending the use of proxy objects*. Dave et al. [Dave, 92] extend the initial definition of proxy objects [Shapiro, 86], as local representations of remote objects, in the CHOICES operating system by introducing a number of techniques which allow applications to access resources in a uniform way (regardless of whether they are local, in the kernel etc.) and introduce a new remote procedure call which can be dynamically reconfigured using a table lookup. Proxy objects therefore provide indirection and late, run-time binding.

These techniques can offer significant support for managing distributed objects. Further, additional techniques may also be employed by the system or application to augment existing management features. These often make use of application domain knowledge in object management. Within the shared object service we focus on the provision of facilities to manage the cooperative aspects of shared information.

10.3.1 Management in the Shared Object Service

There are a number of ways the management functionality provided by a shared object service can be realised and presented. Each approach offers different potential benefits and drawbacks for the service. To illustrate the point consider locking, a fundamental component of any multi-user system. Locking can be provided in one of a number of ways:

- *As a basic mechanism of the service:* Traditionally, within database management systems, this would be implemented as a fundamental component of the DBMS service.
- *As part of each object:* Within the object-oriented database world, this functionality might be implemented as part of the methods and state associated with the most basic object type. Thereafter, the user can safely

assume that every object type possesses the ability to lock access to (portions of) its state.

- *Using multiple inheritance:* A further approach might be the use of an associated object linked using multiple inheritance to the object being locked. For example, we may have a "person" object and a "lock" object. By creating a new shared subtype, we can create a "lockable person" type.
- *Using object adapters:* An object-oriented alternative would be to use interface adapters that dynamically handle objects' invocations to provide the additional functionality required.

The philosophy adopted by the shared object service is to provide all aspects of the service as objects that can be interacted with using a similar set of mechanisms. Thus both objects within the services and the facilities provided by the service can be used by clients in an orthogonal manner. Given this orthogonality it is important that we maintain a separation between the behavioural semantics of the objects within the service and the management facilities provided by the service. For example, the use of multiple inheritance in the case of locking, mixes together management and behaviour semantics and potentially overloads the class hierarchy. We have chosen to use object adapters as a means of maintaining this separation.

10.4 Interface Adapters

Two concepts are important in considering interaction with an object in the shared object service, interfaces and interface adapters. Interfaces describe the means by which a client interacts with the services provided by the object. Interface adapters provide facilities that abstract over these services to provide different services to users based upon context. The service provides mechanisms to create and manipulate both interfaces and interface adapters. An application or user can interact with an object by:

- Access to the basic interface provided by the object.
- Access through an interface adapter which abstracts away from the object.

The relationship between these different forms of interaction is one of abstraction. Thus an interface represents an object and an interface adapter replaces the interface it represents. Object interfaces within the shared object service are analogous to the use of interfaces within distributed platforms such as ODP and ANSA. Interface details are described using an Interface Definition Language similar to that suggested by either ODP or OMG. We would expect this definition language to have strong similarities to those provided by most distributed systems. We do not see the shared object service becoming overly concerned with the exact details of how methods and parameter details are defined and leave this as an issue for supporting platforms. For the purposes of the shared object service a sufficient overview of an interface is to consider it as an access list. Thus each node requiring services from the shared object service contains

information about methods used, invocation protocols, parameters and different methods' location.

Adapters provide a level of indirection between object interfaces and object users. This level of indirection allows interface adapters to provide additional facilities to manage the invocation of methods depending upon the context within which they are defined. Thus object adapters can be used to support access, coordination of users, and task sensitivity. The basic structure of object adapters is as a set of mappings which map method names from those presented to users to those used within the object interface. This overview of interface adaptation within the shared object service is shown in Figure 10.3.

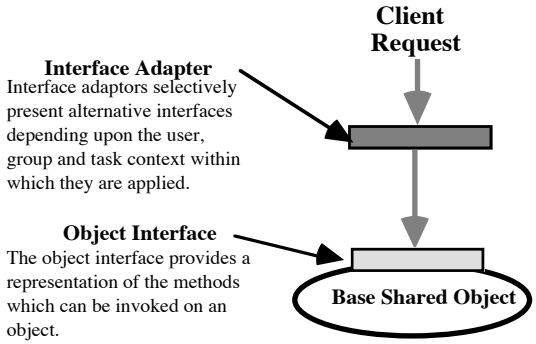


Figure 10.3: Objects, Interfaces and Adapters

The use of adapters builds upon a number of previous themes in distributed and object oriented systems. We are particularly influenced by the use of delegation in object oriented systems [Lieberman, 85], adapters in OMG and active values in Smalltalk [Goldberg, 84]. Delegation allows objects to provide specialised behaviour by forwarding messages to associated objects. Object adapters within OMG clad units of functionality provided by different programs to allow them to appear as objects and in Smalltalk active values are first class objects adding specialised behaviour by intercepting object messages.

The use of adapters maintains a clear separation between behavioural and management semantics. Adapters in the shared object service build upon this separation to allow us to focus on the cooperative sharing of objects. We provide a set of interface adapter objects that provide facilities to manage and support the cooperative features of shared objects. The use of adapters in this way provides a number of benefits:

- A clear separation between the behavioural semantics of objects and the management of objects.
- An explicit identification of the cooperative aspects of shared objects.
- The ability to experiment in order to investigate the correct requirements for object sharing in a cooperative setting.
- A simple mechanism that allows the shared object platform to evolve to support new functionality as it is identified.

At no point is an underlying object interface directly visible to a client, and the only way of invoking the operations is through an object adapter. This allows adapters to manage *dynamic access control* within the service**.** Interface adapters also dynamically enhance (or reduce) the underlying object interface to provide *cooperative object presentation*. This allows objects to present the set of services they offer in a variety of ways depending on who is using the object.

10.4.1 Dynamic Access Control

Multi user systems have used access mechanisms to control the different action of users on the shared systems. A variety of different models exist which offer different facilities to different application communities. Most of these access models are derived from the Access Matrix Model [Graham, 72]. Cooperative applications present a number of distinct problems for the access matrix [Shen, 92; Smith, 93]. In particular, the assumption that access is set up and only occasionally altered by a single administrator has resulted in a static view of access control. However, access models within CSCW systems must provide alternative facilities. In particular, they should provide features which:

- Make access control more dynamic because access is altering frequently,
- Allow access rights to be inferred from the state of some activity or a users role,
- Allow a group of users to manage changes to access,
- Offer a finer granularity of access control than that offered by the access matrix.

The approach adopted by the shared object service is to use interface adapters to provide access management facilities. Access adapters contain a table of interface users and a set of rules that determine the access of users to the object. The access rules within the adapters also make use of specialised state information to determine access.

Clients of the shared object service may be people, services, or applications. Each of these are clients of the service and have an associated context**.** This context information can be used to represent external activities surrounding the shared object store. For example, within a lightweight activity model [Trevor, 93] supported by the service this context is described by a triple of the form:

<name, role, activity_condition>

To perform access control, the client's context must be presented to the adapter when an operation is invoked. Each adapter contains a set of rules that determines which operations are available to a client. When a client invokes an operation on the object, through the adapter, the client's context determines the client's access rights for the operations presented by the adapter. The resulting "rights" are an *access level*, and a *ladder reference number*. Access levels are a less cumbersome way of defining which operations can be performed on an object. Instead of listing all the operations with each potential access rule, the object(s) operations

are listed in one or more "*ladders*" of importance. Each ladder is identified by a ladder reference number. The greater the allowable access, the "higher" up on the ladder a client can reach and the more operations can be used. This arrangement closely ties access to the cooperative use of objects. The access model currently provided by our shared object service is associated with a lightweight activity model [Trevor, 93].

10.4.2 Cooperative Object Presentation

Object adapters can also selectively present and change the interfaces of one or more underlying objects. Operations presented by an adapter are mapped onto an underlying object interface and may filter out operations provided by an object. Unlike interfaces within distributed systems which cannot change once they have been assigned, an object adapter provides a set of rules based on the client's current status within the system. These rules dictate which operations a user can invoke at any time during the activity's life cycle. Two particular presentation techniques are exploited to promote cooperation within the shared object service.

Extending object semantics

Adapters can extend the operational semantics of the objects they adapt to support cooperative behaviour. These additional operations masquerade as standard operations to the client. Unlike object operations, these new methods have side effects which are necessary to change the underlying object into a more cooperative one. For example, an adapter can extend the single user oriented write operation of a file object, by checking that the client invoking the operation can do so and still preserve the consistency of the file. The client can now use the write operation, presented and modified by the adapter, as though is was directly on the file object. However, unlike the original write operation, it can now fail gracefully if any consistency problems occur.

Extending object interfaces

Adapters can amend the presentation of more than one object interface. This means that an adapter can present a combination of more than one interface (and hence more than one object) to a client. For example, consider a document object shared between two users Tom and John. This is merely an extension of a simple file object, which would normally have read and write operations and the basic function of the file should remain the same when the object is shared. However, the semantic information regarding when the methods affecting the objects state, for example write, can be applied must change to accommodate multiple users.

The document object is realised in the service as an adapter which combines two object interfaces, the basic file object and a special schedule object, which determine when it is appropriate for a client to be refused write permission (Figure 10.4). Therefore while the adapter "read", "append" and "write"

operations come from the file object, the “Request write permission” and “Release write permission” methods come from the schedule object.

Additional specialised operations are also provided by the adapter which use and act upon knowledge of how the object is supposed to be shared. These additional operations characterise different classes of adapters which provide different management facilities with the shared object service. A couple of these are examined in the following section.

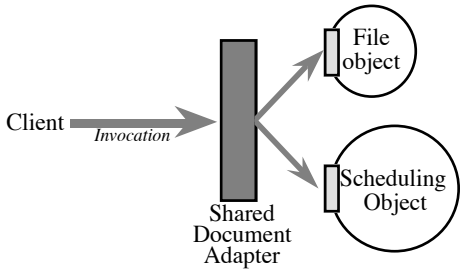


Figure 10.4: Making a shared file through adaptation

10.5 Adapters Within the Service

A variety of different adapters can be supported by the service, each allowing different aspects of the service to be made available to users for management. Rather than consider all of the different aspects of the service in full we wish to focus on two particular uses of adapters within the service:-

Awareness

Current multi-user sharing systems have adopted an approach of strict access transparency. In contrast, cooperative systems need to promote awareness across the user community. The service provides a set of adapters designed to promote just this awareness.

Locking

Effective CSCW systems require a close linking of the locking mechanisms provided by the shared infrastructure and the co-ordination and cooperation needs of user groups.

Each of these different aspects of the shared object services is examined in the following sections.

10.5.1 Awareness in the Object Service

The shared object service is distinguished from other multi-user storage systems by its explicit goal of providing an awareness of the action of others across the

service. Thus, when an object is accessed or altered in the shared object service, the system informs relevant users the action has occurred. In addition, the shared object service provides facilities to allow users of the service to make others aware of actions of interest to them. The principle mechanism used to provide this form of awareness is events. The shared object service provides facilities to allow events to be defined, created and related to others.

In supporting cooperation among a wide set of different users and platforms, possibly with different interfaces to applications and even diverse language bindings, a uniform and clean event mechanism is needed. In the shared object service an *event handler* provides a central facility responsible for event propagation. This portion of the service also maintains a record for the clients of which types of events are of interest to them.

A special form of object adapter called an event adapter is used to inform others of changes to an object in the service. In essence this object converts existing objects which are unaware of the existence of others to cooperation aware objects (Figure 10.5). An event adapter knows about interest relationships through the event handler. If an object declares interest in events to or from another object or agent then an event adapter "filters" events to and from the interested object. If an event is of "public" interest that event is also propagated.

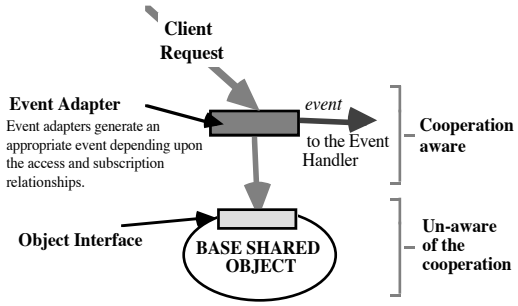


Figure 10.5: Event Generation using an event adapter

10.5.2 Lock Adapters

The locking of information within a multi-user setting has two distinct effects. It ensures the integrity of the information and reduces the possibility of error caused by simultaneous alteration. Locking also provides a means of mediating and controlling the work taking place around the shared information. Potentially, locking provides a means of co-ordinating access to information, sequencing the use of information and identifying the parties using the information. Investigating the use of locking in a cooperative setting requires us to separate the *locking mechanism,* realising the locks upon information, from the *locking policies,* that control how that information is locked. The intent within the shared object service is to maintain this separation and provide a means of making the locking policy explicit.

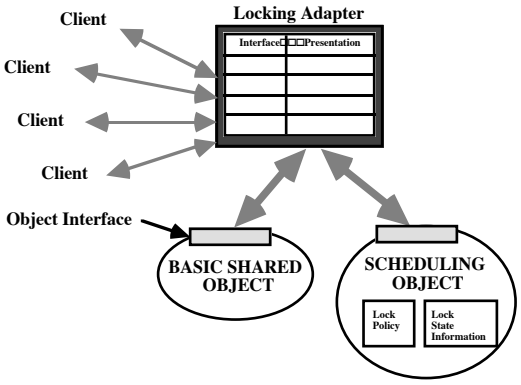


Figure 10.6: Locking Adapters

If an object requires the locking services, an interface adapter is placed in front of the object itself. Requests that were originally targeted to the underlying object are now directed to the adapter. The adapter contains "state" information relative to the locking functionality it provides. Moreover, it contains the "algorithm" or "protocol" used by that particular locking regime. Traditional locking strategies have long been recognised as unsuitable for CSCW applications [Ellis, 89; Greif, 87; Rodden, 92]. Embodying the locking mechanism(s) in a set of adapters allows the flexibility to prototype and experiment with a set of different mechanisms. Thus interface adapters contain an interface description, locking state information and an appropriate locking policy. The arrangement is shown in Figure 10.6.

Lockable interface adapters provide a set of management facilities that allow the locking policy to be modified. Often this policy can be represented externally with reference to a particular policy object. As in the case of access, locking adapters may use representations of the status of activities outside the service to infer locking conditions [Trevor, 93]. For example, a lock adapter may use the stage of a project to determine the locking policy to be applied.

While events promote 'real time' awareness, locks can be used to promote awareness in a time independent manner. The shared object service supports the notion of "identified locks" within adapters. This means when a user (or an application executing on behalf of the user) locks an object, the user's identification is part of the lock. This provides information regarding "who" is using the object. In certain applications, this may be complemented by recording a reason for the object being locked within the adapter. This combination of "who" and "why" provides a useful form of awareness across the service.

We have shown how the use of architectures within the shared object service allows the addition of mechanisms to support cooperation. The following sections consider the architecture needed to support object adapters and the particular adapters provided within a shared object service realised as part of a lightweight activity platform.

10.6 Architecture

A number of components of the service architecture are relevant in supporting object adapters. These include:

- a Trader, which enables clients to locate, and services to register, interfaces.
- an object factory, to create objects.
- an adapter manager, which acts as a central repository of adapter information and creates the objects required by an adapter.
- a Binder, which obtains an object's interface reference for clients, after binding all the objects needed by adapters.

Each of these components are realised directly as high level managers specific to the service. These map down onto the object services provided by the distributed system, such as a low-level object Binder and Trader. For example, the Binder in the shared object service manages the binding process based upon the cooperative environment it is in. However, it still uses a lower level system Binder to perform the actual binding. Likewise, the shared object services Trader makes uses of the mechanisms provided by a system level Trader while overlaying specific knowledge and functionality.

10.6.1 Trader

The trading service imports and exports interface references for objects within the shared object service. Upon creation, objects export their interface descriptions to the Trader which are rendered unusable without being passed to the binding service. Interfaces are described using an Interface Definition Language (IDL) derived from the existing OMG IDL. Clients obtain these interface references by requesting any interface which matches an interface description passed with the call. Any description which matches this interface description has its reference returned to the client. Interface descriptions can be either constructed, or selected from the result of a more general Trader query.

10.6.2 Adapter Manager

The adapter manager acts as the central point of coordination and reference for the adapters and contains information about the object adapters. This includes:

- Basic adapter definitions, exported to the factory when the adapter object is created.
- Invocation path lists. These are lists of object requirements for adapters which describe the necessary invocation path that an adapter requires. Path relationships are expressed as:

  Adapter A, interface I **adapts** Object B, with interface J

  This indicates that the adapter object A is an adapting object over object B, which may be a simple object or an adapter itself.

10.6.3 Binder

The Binder's function is to set-up the connection between a client object and the interface of a service object. A client passes an interface reference to the Binder, which may be to a normal object or an adapter object. The interface reference contains its network address. The binding process consists of assessing whether the required binding is legal and converting the interface reference to an identifier. The actual binding itself takes place by calling a system level Binder. Upon completion, the Binder returns an interface identifier to the client which can then be used to invoke operations on the object.

10.6.4 Object Factory

The object factory holds a list of *object templates* and corresponding *object definitions* for the objects in the platform. The definitions for adapters and masters are exported to the factory by adapter managers upon their creation.

In order to create an object, a client passes an object template to the factory. The template is tested against the factories template list. If a successful match is made, the object, stored along with the matched template, is created from its definition, and the resulting reference to the new object is returned to the client.

10.7 Managing, Initialising, Binding and Invoking Shared Objects

A client may use any adapter or object within the service, but to do so they must first request that the service *bind* them to the required object or adapter. If the object does not yet exist the client must first ask the service to create it. Creation of objects can also occur during the binding process if the object being bound is an adapter. Successful completion of the binding process on an adapter object forms an *invocation path* between the client and the underlying object(s). Adapters are then subsequently managed through an associated *management interface*.

10.7.1 Invocation Paths

An *invocation path* is the route an invocation can take from a client, through zero or more adapters, to a basic object. Since the method for binding objects is recursive, enabling an adapter to use another adapter (as adapters are only a special form of object), paths of any size can be created. Information about which objects belong in a path, and how to construct them, is stored in the master adapter. Figure 10.7 shows a client, John, bound to an extended version of a shared file adapter. To provide a high degree of flexibility in deciding when someone may write to the file object, the scheduler is not considered as a single object, but as an adapter with a number of separate policy objects. The initial

binding has caused the creation, and subsequent binding, of the objects who belong to the invocation path from the client to the scheduling policy objects or file.

From John's point of view, this path is transparent. The only "visible" section of the path is to the shared file adapter. This transparency holds for any adapter (who is effectively a client of the next adapter in the path) with each client being only aware of the subsequent object within the path they are invoking and no more. Figure 10.7 highlights the various different views of the invocation path using an extended version of our shared file example. The client (John) is only aware of the shared file adapter, the shared file adapter can only see the file object and the scheduler, which itself is only aware of the policy objects to which it has access.

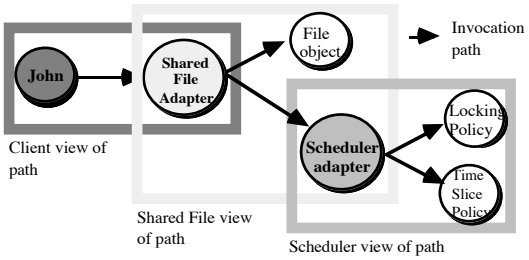


Figure 10.7: Views of an Invocation path

Once created, the entire path may be extended by the addition of further adapter objects on the first original adapter. This extension of the path causes no access conflicts with the existing adapters as a new adapter cannot bypass the old.

In order for a client to bind successfully to an object in the shared object service, the requested object must not be contained *within* the original invocation path. This path integrity condition allows a client to bind either to an existing interface at the start of the original path (the shared file adapter above), or to adapters which have extended the original path.

In addition to the path integrity condition, we place a final restriction on adapters and their relationships with objects. An object may only be presented by one adapter type at any one time. This condition may be relaxed if the different adapters do not invoke conflicting underlying object methods that alter some state. Otherwise, it is necessary in order to preserve the consistency and security of the underlying object state.

Finally, it is the responsibility of the underlying distributed system to make sure that shared service objects (those being used by multiple clients) are isolated and cannot be accessed from outside of the service.

10.7.2 Adapter Management

Each adapter contains a management interface which is used to manage the adapter and the adapted objects. The interface may or may not be used depending

upon the adapters position in the path. The head adapter of the original invocation path and all subsequent adapters added to the path externalise the interface but those contained within the original path do not. This adapter interface provides a number of important functions:

- It manages and co-ordinates the group of adapters under its control. This means, for example, that locks can be freed if an adapter locks an object and then fails.
- It holds associated adaptation information, such as who may access the underlying object, which adapters can be attached to it and which objects it can abstract over.
- It participates in the creation of the adapters to be under its control.
- It enables the state of the adapter to be changed during the adapters life-time, such as the access rules and ladders.

10.7.3 Binding Process

A client asks to bind to an interface using an encrypted interface reference. This reference is decomposed by the binding service into the physical network address of the interface and the object type. If the object is not shared (i.e. it is not adapted) then the Binder proceeds to bind, using the distributed system Binder where available, and returns a usable interface reference to the client. If the object is an adapter, the associated management interface is consulted to check what position the interface occupies within an invocation path.

If the interface is unavailable then the Binder aborts as the client fails to satisfy the path integrity condition. Otherwise, the binding proceeds and a usable interface reference is returned.

10.7.4 Object Creation Process

In order to create a shared object a client initially calls an adapter manager in the service with a template which describes the object they wish to create. The service attempts to match the template with one of an internal list of adapter object templates. If no match can be made then the object is assumed not to be an adapter and the object factory service is used with the template. The factory subsequently creates any matching object definition and returns the new object reference.

If the adapter exists, the adapter manager calls the factory to create the adapter. Creation of all other related and required objects is done by calling the adapter manager itself. This recursive creation procedure enables adapters to “adapt” over other adapters, whose nature will remain hidden from the client adapter.

If a client wished to create an adapter to extend an existing path, the adapter manager would be invoked with the object template that they wish to add to the path, and the interface reference of some top level interface in an existing invocation path. A similar process to the above then takes place.

10.8 Realising Adapters Within COLA

Our realisation of the shared object service has taken place as part of a more general platform to support for cooperative work called COLA [Trevor, 93]. The platform is motivated by the need to promote a more open approach to supporting cooperation. Existing purpose built CSCW support tools and environments allow little access from outside the system itself. In addition, many include a constraining amount of semantic information about the cooperation being supported. This heavyweight and closed approach make them unsuitable in providing CSCW systems within real-world heterogeneous environments.

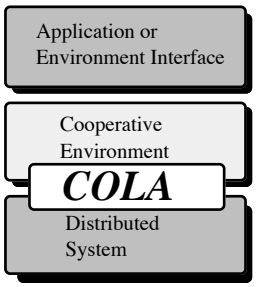


Figure 10.8: COLA as a supporting “veneer”

Unlike existing cooperative environments, the COLA platform provides mechanisms to support sharing but with as few of the semantics as possible. This separation enables COLA to act as the supporting “veneer” between semantically laden cooperative environments and distributed systems (Figure 10.8). The platform is underpinned by a lightweight model of activities [Trevor, 93]. The model exploits the nature of the platform to employ specialised object adapters as the basis for a lightweight activity facility which relies on information sharing rather than exchange.

10.8.1 The COLA Model and Platform

A central part of the platform is a lightweight activity model which represents the cooperation of the users and provides a context in which the objects are shared. The activity model separates different goal oriented (however vague) forms of cooperation into separate activities. Each activity is managed by an associated manager who facilitates group awareness through the provision of an externalised state.

The platform is considered as providing a set of *policy free mechanisms* to allow a wide spectrum of different activity features to be represented. Therefore, activity *policy*, such as the enforcement of deadlines and role allocation, resides exclusively with the application and not COLA. It is the prerogative of the application or cooperative environment to decide how and when the cooperative mechanisms should be employed. This mechanism and policy separation not only

allows a *variety* of existing cooperative models to be built on top of it, but also enables users to circumvent applications and directly interact with the platform.

Within each activity, objects (users and applications) can play many roles at once and a role may be played by any number of objects. No semantic attachment to the notion of a role is made beyond grouping objects with some commonality. If required, the application using the activity can use the role names as hooks to attach any semantic information they wish.

Users, applications and objects are kept aware of each others' actions, inside and outside of their own activities, by event delivery. Events are delivered through subscription to an event facility. All these services are supported by the ISIS distributed toolkit which was chosen because of the provision of mechanisms for reliable message passing, group multicast, synchronisation and fault tolerance [Birman, 93].

COLA Object Adapters. All COLA object location, binding and creation is performed by the a particular instantiation of the shared object service. The shared object service uses specialised COLA adapters to support the lightweight activity model. Because the information within an adapter is necessarily activity dependant, adapters are uniquely defined with respect to a given activity. This separation of the sharing policy from the basic object mechanisms allows all the underlying object definitions to be globally instantiated. The adapter definitions and requirements are stored in an activity manager provided as part of the COLA platform.

To understand the use of COLA adapters it is worth considering a simple scenario of cooperative work based on the use of the platform within an academic department. The scenario centres on the setting and marking of an examination held electronically within the COLA platform. The examination has four basic phases: *creation* (of the paper), *approval* (that the paper is satisfactory), *sitting* (when the student actually take the paper) and *marking* (when the student answers are checked). Jonathan and Nigel are in charge of creating the exam paper, a shared object, and are both assigned the "creator" role within the activity. Tom must approve the paper and is assigned a "reader" role, while Bill and Ben are students who must take the paper and have a "student" role. Dave is a user of the service but has no role within the examination activity. The exam paper is being shared among a number of different people who have different reasons for using it.

COLA adapters exploit a user context realised as triples to manage the presentation of objects. These triples consist of a user, the set of roles which the user is playing, and the activity they are in. The platform ensures the triple is complete by making clients members of a *default* activity and in any given activity provide a *default* role. The defaults, while having no specific function other than for filling in a context, enable all objects to use, and be used by, the activities and services within the platform.

As in the general shared object case when a client invokes an operation through the COLA adapter the client's context is checked against a set of access rules. The

first rule to succeed results in an *access level* and a *ladder reference number* for the user. If no rules are applicable then the invocation fails.

Any client can execute a method on a COLA object by presenting the name of the method they wish to call, the parameters the call requires, and one of their assigned COLA context triples. The platform passes the information to the COLA adapter object which uses the context to decide what access level and ladder the client can reach with the current context. If the desired operation has an access level in excess of the one calculated then the operation fails. The invocation method (provided by the underlying distributed system) is the same for both normal, unshared, objects and adapted objects because object adapters are implemented as only special instances of normal objects.

Figure 10.9 shows a rule set for the examination paper COLA adapter used in the examination scenario. Any students, for example Bill and Ben, will only be able to read the paper during the *Sitting* stage of the activity, while anyone with the *writer* role, Nigel and Jonathan, will always be able to invoke every method on the operation ladder 1. If the activity is in any stage before *Mark* then clients without a specific role cannot invoke any operations.

As in the shared object service, to prevent tampering with a shared object, objects created using the COLA platform can only be used through the COLA platform. This *still* allows applications and environments outside of the activity owning the adapter to share the object. The degree of object sharing across different activities depends on how leniently the access rules for each object have been specified, and how activity dependant the objects are. For example, the access rules of a shared examination paper would make little sense outside of an examination activity because that object is highly activity dependant. Conversely, it may be desirable to allow access to information contained within an object representing a shared whiteboard across more than just one activity.

Shared Exam Paper Rule Set:

User	**Role**	**Activity**	**Level**	**Ladder**
any	Student	stage = Sitting	3	1
any	Reader	stage >= Approve	2	1
any	Writer	*none*	1	1
any	*any*	stage >= Mark	3	1

Operation Ladder: 1

Level	**Operation**
1	Write
1	Release Write Permission
1	Request Write Permission
2	Append
3	Read

Figure 10.9: A shared file ladder and rule set

10.9 Preliminary Issues and Experiences

The managers and objects which make up the COLA platform and its shared object service have now been realised, together with several graphical browsers and a suite of library routines which support user and application interaction with these components. One of the first applications which has attempted to exploit many of the lightweight features provided by this platform, most notably object adaptation, is a variant of the examination example presented above.

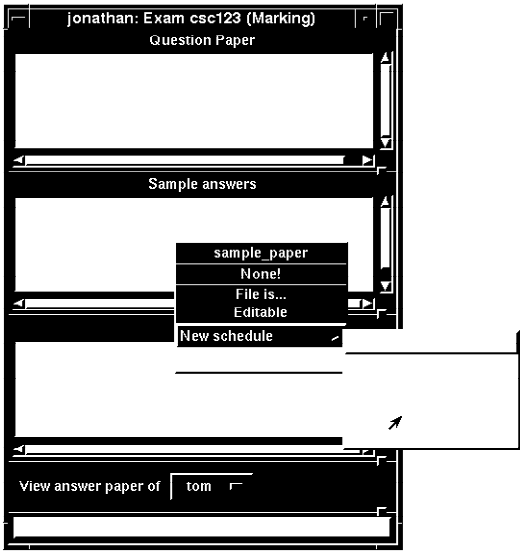


Figure 10.10: A lecturers view of various shared file objects in an examination activity.

Figure 10.10 illustrates a lecturer's view of a particular examination activity. Each of the shared file objects which represent and hold the state of the various papers in the activity (sample, question, answer and marks) are adapted by special locking adapters and presented through simple text widgets. These adapters intercept all the underlying file object operations and actively alter the write/insert/delete operations in order to apply the locking policy. A simple popup menu on each file's widget (Figure 10.10) interacts with the corresponding adapter and allows the application and user to select which particular locking policy they wish the adapter to apply. Read operations on the file, while still propagating through the adapters, remain unaffected by the actual adaptation (except where the read may conflict with the access control rules which COLA enforces on the adapter).

In this application the use of adapters has allowed the successful separation of:

- the underlying file objects mechanisms and semantics from the application itself,
- the problems of multi-user access on a shared, single, object from the application and the file object.

This separation of application semantics from the multiuser issues and from the basic object functionality has allowed us to concentrate on each of these aspects

individually without becoming concerned with the whole, and has made it far easier to change and adjust the application as new requirements become apparent.

One of the biggest potential drawbacks of object adaptation is the performance cost of any invocation on an adapted object. While no detailed performance evaluation has yet been made, it is obvious that in a straight-forward implementation of the shared object service a normal invocation using adapters will be proportional to the length of the adapter's invocation path. Indeed, this is the case in COLA. However, this cost can be reduced in a number of ways. Objects in the ANSA testbench which are related in some way can be grouped into "capsules" [ANSA, 89]. When one object in a capsule invokes another within the same capsule this invocation is mapped onto a more local procedure call and not a remote one. Because it is often the case that the objects being adapted and the adapters themselves are tightly coupled, such a technique can be applied, and would reduce the overhead of the number of remote, slow, calls to an adapted object. In the next worse case, where an adapter is being added to an existing object (the invocation path is being extended), the potential benefits of simply making the adapter local with respect to the adapted object is again worth exploiting as any reduction in network use will be beneficial (for performance). In the worst possible scenario, where the adapters must reside on a different location than the adapted object, the impact of advances in technology in the near-future should not be ignored. More sophisticated shared virtual memory areas and higher speed networks could dramatically increase the performance of a remote object invocation, therefore decreasing the current penalty that object adapters may currently incur.

10.10 Conclusion and Further Work

Successful information sharing is critical if users and applications are to cooperate together. Previously, CSCW applications have relied on distributed systems to provide the necessary management and control of the information which is to be shared between users. Unfortunately the traditional approach adopted by many distributed systems to management does not involve users and actively prohibits the cooperation and sharing they need to support.

By considering the generic implications that emerge from cooperative work, we have proposed a shared object service which explicitly provides the necessary platform for sharing across any cooperative systems. The service supports a number of general requirements for shared objects which include:

- Lightweight semantics through a policy and mechanism separation.
- Awareness.
- Flexible access control and dynamic presentation.
- Adjustable and extendible semantics.

The management and creation of shared objects within the shared object service takes place through a variety of cooperation-oriented services, many of which augment those provided by the underlying distributed system.

This chapter has concentrated on *interface adapters* within the service which help to satisfy the requirements for shared objects. Adapters maintain a clear separation of the behavioural semantics of an object and their management by providing a level of indirection between a shared object interface and its users. This indirection enables adapters to provide dynamic access control and cooperative object presentation. We have shown how two types of adapters within the service, events and locking, provide an essential degree of cooperative awareness between objects and how the cooperative work can be mediated and controlled from the nature of that work itself.

While the focus of this chapter has been on the use of object adapters to support cooperative sharing, object adaptation can usefully be incorporated into other application domains. For example, the transparent presentation and modification of an object through an adapter may provide a very intuitive and workable approach to integrating heterogeneous systems or objects. In addition, the ability to present a combined or amalgamated view of one or more objects can be applied to object oriented-databases, where the results of multiple queries may be "joined" using some form of adapters.

We have highlighted how many of the concepts and services introduced in this chapter have been realised and built as an integral part of a more general platform, COLA, to support CSCW applications. Further work is required to see how successful adapters and other services provided by the shared object service through COLA, are at supporting the needs of cooperative users for sharing objects.

10.11 References

[ANSA, 89] ANSA, *ANSA: An Engineers Introduction to the Architecture*, Projects Management Limited, Poseidon House, Castle Park, Cambridge, CB3 0RD UK, November, 1989

[Araujo, 88] Araujo, R. B., Coulouris, F., Onions, J. P. and Smith, H. T. The Architecture of the Prototype COSMOS Messaging System. In *Proceedings of Euteco'88* (April 20-22, Vienna, Austria) Elsevier Science Publishers B.V., North-Holland, 1988, pp157-1709.

[Benford, 93] Benford, S. and Fahlén, L. A spatial model of cooperation in large virtual environments. In *Proceedings of ECSCW93* (Sept. 13-17, Milan, Italy). Kluwer Academic Publishers, 1993, pp109-124.

[Bentley, 92] Bentley, R., Rodden, T., Sommerville I. and Sawyer, P. An architecture to support tailorable multi-user displays. In *Proceedings of ACM CSCW92 Conference on Computer-Supported Cooperative Work* (Oct.31-Nov.4, Toronto, Canada), ACM Press., N.Y., 1992, pp187-1941.

[Birman, 93] Birman, K.P. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM 36*, 12 (December 1993), pp36-53,103

[COMIC, 93] COMIC, *Informing CSCW System requirements*, Deliverable 2.1, Esprit Project 6225, ISBN 0-901800-29-5, Available via. FTP ftp.comp.lancs.ac.uk, 1993.

[Dave, 92] Dave, A., Sefika, M. and Campbell, R.H. Proxies, Application Interfaces, and Distributed Systems. In *proceedings of the Second International Workshop on Object Orientation in Operating Systems (IWOOOS)*, (Sept. 24-25, Dourdan, France). IEEE Press, 1992, pp.212-220.

[Davies, 93] Davies, N.A., Davy, M.J., Blair, G.S., and Mariani, J.A. Object invocation and management in the Zenith distributed multimedia information system. *Information and Software Technology 35,* 5 (May 1993), 259-266

[Ellis, 89] Ellis, C.A. and Gibbs, S.J. Concurrency Control in Groupware Systems. In *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*. (Portland, Oregon). ACM Press,N.Y., 1989, pp 399-407

[Fish, 88] Fish R.S., Kraut R.E., Leland M.D. and Cohen M. Quilt: A collaborative tool for cooperative writing. In *COIS88 Proceedings of conference on Office Information Systems* (March 23-25, Palo Alto, California) 1988. pp 30-37.

[Goldberg, 84] Goldberg, A. *Smalltalk-80 The interactive programming environment*. Addision-Wesley Publishing, 1984.

[Graham, 72] Graham, G. and Denning, P. Protection: Principles and Practices. In *Proceedings of the AFIPS Spring Joint Computer Conference*. 1972, pp 417-429.

[Greif, 87] Greif, I. and Sarin, S. Data sharing in group work. *ACM Transactions on Office Information Systems 5,* 2 (April 1987), 187-211.

[Haake, 92] Haake, J. M. and Wilson, B. Supporting Collaborative Writing of Hyperdocuments in SEPIA. In *Proceedings of ACM CSCW92 Conference on Computer-Supported Cooperative Work*. (Oct.31-Nov.4, Toronto, Canada). ACM press, N.Y., 1992, pp138-146

[ISO/IEC, 91] ISO/IEC. *Basic Reference Model of Open Distributed Processing*, Working Document RM-ODP - Part 1: Overview. (*December 1991*) Available through national standards bodies.

[Lieberman, 85] Lieberman, H. Delegation and Inheritance: Two Mechanisms for Sharing Knowledge in Object Oriented Systems. *Journées d'Etudes Langages Orientés Objet*, AFCET, Paris, 1985, 79-89.

[Marmolin, 92] Marmolin, H., Sundblad, Y., Tollmar, K., Avatare, A.. and Eriksson H. CoDesk - An interface to TheKnowledgeNet, Design and Implementation. In *Proceedings of the 4th MultiG Workshop*. (May, Stockholm), Kungl Tekniska Hogskolen, Stockholm, Sweden, 1992.

[Newman-Wolfe, 92] Newman-Wolfe, R.E., Webb, M.L. and Montes, M. Implicit Locking in the Ensemble Concurrent Object-Oriented Graphics Editor. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*. (Oct. 31-Nov.4, Toronto, Canada). ACM Press, N.Y., 1992, pp 265-272.

[OMG, 91] OMG. The Common Object Request Broker: Architecture and Specification', OMG Document Number 1991.12.1, Revision 1.1, OMG, Draft 0, December 1991.

[Patterson, 90] Patterson, J. F., Hill, R. D., Rohall, S. L. and Meeks, W. S. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of CSCW'90* (October 7-10, Los Angeles, Ca.). ACM Press, N.Y., 1990, pp.317-328.

[Rodden, 92] Rodden, T., Mariani, J. and Blair, G. Supporting Cooperative Applications, *CSCW: An international Journal 1*, 1 (Oct. 1992) pp41-67

[Shapiro, 86] Shapiro, M. Structure and encapsulation in distributed systems: The proxy principle. In *proceedings of the 6th International Conference on Distributed Computer Systems*. (May) IEEE, 1986, pp198-204.24.

[Shen, 92] Shen, H. and Dewan, P. Access control for collaborative environments, In *Proceeding of CSCW92*. (Oct.31-Nov.4, Toronto, Canada), ACM Press, N.Y., 1992, pp 51-58.

[Smith, 93] Smith, G. and Rodden, T. Using an Access model to configure multi-user interfaces. In *Proceedings of COOCS'93 ACM international conference on Organisational Computing Systems*. (Nov.1-4, San Jose, Ca.). ACM press, N.Y., 1993, pp289-298

[Thiel, 92] Thiel, G. LOCUS operating system, a transparent system. *Computer Communications 14*, 6 (July 1992) 336-346.

[Trevor, 93] Trevor, J., Rodden, T. and Blair, G.S. COLA: A Lightweight Platform for CSCW. In *Proceedings of ECSCW93*. (Sept. 13-17, Milan, Italy), Kluwer Academic Publishers, 1993, pp 15-30

Chapter 11

Supporting User Awareness with Local Event Mechanisms in the GroupDesk System

Ludwin Fuchs and Wolfgang Prinz

GMD

11.1 Introduction

In the CSCW community the problem of supporting shared awareness among the users of systems for the support of cooperative work has become one of the central controversial topics [Dourish, 92; Fuchs, 94; Jarke, 92; Sohlenkamp, 94]. The discussion is motivated by the lack of current CSCW systems providing an overview of the state of work. It addresses two specific issues: on the one hand the problem of making currently ongoing activities of interest visible to the users of the system, and on the other hand providing an overview of changes in the past concerning the objects of work.

Approaches to solve these problems differ greatly in their orientation. They range from systems settled in traditional database technology, such as version and configuration management systems, to multimedia based information systems or three dimensional virtual worlds. All these systems have in common that they focus on just one of the sub problems mentioned above. As an example, the spatial model of interaction, developed in the second part of COMIC Strand 4, has proven to be especially suited to provide an awareness of synchronous events in the environment and to support guidance of synchronous communication in potentially dense populated spaces, whereas achieving the visibility of asynchronous changes seems to be more problematic.

In this chapter we present some ideas to enable an integrated description of the state of cooperation. Instead of conceptually separating the acting user from the objects of work the model integrates the users, work artifacts, tools and resources, into a common organisational context and allows the provision of information concerning synchronous as well as asynchronous situations. The model has been developed in close cooperation with the work on organisational context presented in COMIC Deliverable 1.2 and is based on the representation of the working context as a semantic net. The nodes of the net represent the work artifacts, the actors (users), and organisational entities, such as departments, roles and procedures. The edges of the net are formed by different typed relations. Such a

relation may describe similarities of artifacts in terms of content, or currently ongoing activities in the environment. Relations are also used to embed objects into the organisational context. The net is formed and continuously modified by the interaction of the users with the system.

As the central means to describe the state of affairs, the model uses the notion of cooperative events which are closely related to the notion of events in natural languages [Smith, 91; Vendler, 67]. They may be divided into two basic categories: activities and modifications. Activities describe operational events involving an actor and an object. Modifications describe changes of the internal state of objects, usually caused as a side effect of an activity. Associated with activities is a temporal duration, whereas modifications are considered to be instantaneous. The proposed model is capable of treating both types of events in a uniform manner, thus integrating strategies for solving both of the awareness sub-problems introduced above.

A flexible event distribution strategy is applied, which distributes the events based on a metric that can be derived from the semantic net: events get distributed according to node specific rules, using the edges as the distribution medium. Users may get informed dynamically about events that happen currently or that have happened in the past in the surroundings of their actual position in the work environment. This strategy has the advantage that the visibility of events is restricted to the user's current position in the system, in other words to the user's current occupation. Hence, the model provides a conceptual approach to prevent information overload. It allows support for orientation in a very general sense: information about events is not only present in the directly involved objects, but also at objects that are related to them in some specific way. For modifications this behaviour plays an important role, since the state of artifacts often cannot be determined clearly in isolation from related objects.

In the first part of the chapter we outline which kinds of awareness the event model is capable of supporting. This is followed by a description of the representation of the work setting. We present the core event propagation mechanism and show how it uses this representation. We proceed by examining different work scenarios and show how the model is capable of providing the necessary information to support the respective modes of awareness in these situations. The theoretical aspects of the model however are only briefly presented. A more detailed description of the event model can be found in the COMIC Strand 1 Deliverable 1.2.

In the second part of the chapter, we introduce the GroupDesk system, a first prototype implementation of the event model, and show how the event related facilities of the system make use of these concepts and enable an implicit awareness of the users about the overall dynamics and state of work.

11.2 Modes of Awareness

We start our consideration with a brief introduction of different modes of awareness. It is important that systems support each of these categories and we show subsequently how the event model presented in this chapter can treat each of them in a homogeneous way.

Orientation in cooperative processes is based on events in these processes. We use a notion of events that allows a description of the state of cooperative situations and is suited to provide information to support each of the following modes of awareness:

	synchronous	asynchronous
coupled	What is currently happening in the actual scope of work ?	What has changed in the actual scope of work since last access ?
uncoupled	What happens currently anywhere else of importance ?	Anything of interest happened recently somewhere else ?

Figure 11.1: Modes of awareness

Synchronous awareness is concerned with events that are currently happening, whereas asynchronous awareness considers events that have occurred at some time in the past. Support for the latter mode needs to be derived by a summarising interpretation of a whole sequence of events, that have happened in the meantime. Synchronous awareness should be supported by an immediate reflection of the ongoing affairs at the graphical user interface of the system.

Orthogonal to this classification we distinguish between coupled and uncoupled awareness according to the current interest of the user. Coupled awareness denotes the kind of overview that is closely related to the current occupation of the user. An example of this kind of orientation is the knowledge of a user who wants to edit a certain document, that this document is currently being read by someone else. Asynchronous coupled awareness applies to a user working on a certain object who is informed about changes that happened to this object in the past during a period of absence.

Uncoupled awareness applies in situations where information about events needs to be provided independent of the user's current focus of work. As an example for uncoupled asynchronous awareness consider a situation where a work flow system sends an object, such as a spreadsheet or a folder of documents to be worked over, to somebody who is currently on holiday. If there is a deadline attached to it, then it may be very important to notify the initiator of the work flow about this – even if he is at the moment concerned with something else.

11.3 The GroupDesk Model of a Working Environment

In this section we explain how the different entities in the cooperative setting will be modelled, so that it is possible to derive the information required to support the different modes of awareness presented above.

11.3.1 Objects

The basic units of information in the system are objects. Work artifacts in the environment, such as documents, tools or working resources of any kind, are modelled as respective objects. The same assumption also holds for more abstract entities that compose the organisational context of work: groups, departments, organisational roles and rules are all simply objects in the system. Furthermore, we integrate objects that represent the users of the system. In terms of the model, they are basically treated in the same way as any other entity the system manages. In the following, we will however refer to objects representing users by the term actor, to distinguish them from the other objects in the system. In this sense, we make no assumption about the concrete semantics of the cooperative artifacts, nor do we prescribe any specific classification of objects or formal organisational structuring. The only assumption we impose, is that all entities in the work setting are modelled in compliance to a uniform object model, so that all objects may inherit a basic behaviour. The proposed event model may then be implemented as a part of this basic object behaviour. In this way we can provide a uniform treatment of all objects and are also free to refine the behaviour more specifically, if certain object classes need a more specialised treatment.

11.3.2 Relations

Relations are used to place the actors and artifact-objects into a collaborative context. For reasons of simplicity, we will assume in the following that relations are always 1:1 relations. The model is however capable of handling any kind of *n:m* relations. Hence, a relation has a source and a destination. A relation's source as well as its destination is always formed by an object in the system. Furthermore, relations always define an inverse relation, in which the source object becomes the target object and vice versa. In terms of implementation, we assume that relations are modelled as objects in their own right. We will however abstract from this and only use the term relation to distinguish them from usual objects in the above sense. Note also, that although we may always implement any relation and its inverse as a single relation-object, they are actually treated as two different entities in the model.

Relations are typed and may be grouped into four basic categories:

- structural relations
- operational relations

- semantic relations
- interest relations

Structural relations are used to describe any kind of relationship between objects and an associated organisational context. A simple example for this might be a relation between an organisational role, such as head of department, and an actor, describing the fact that this actor plays this specific role. Further examples are all kinds of membership of entities and actors in specific contexts, such as projects and departments. Operational relations are always relations between an actor and an object. The general semantics of these relations is the fact that the corresponding actor is currently involved in some kind of activity concerning the destination object. In an environment for document production, we would e.g. express the fact that a user is editing a document by a corresponding operational relation. Semantic relations are used to express any semantic similarity between two entities in the systems. They are highly dependent on the concrete nature of work to be performed. Finally, we include a special kind of relation, which is used to formulate a specific interest of users in a given entity of the system. This kind of relation is necessary when it comes to the provision of uncoupled modes of awareness.

Hence, the general form of the overall representation spans a semantic network and we further assume that the relations between the objects are visible and accessible from outside, i.e. by some access functionality encapsulated in the meta-model of the objects. The actual maintenance and evolution of the network is triggered by the interaction of the users with the system: users may create objects, place them into a context or move them around as they like. Also, the establishment of operational relations is derived automatically by the system, according to the actions the users are performing: if a user enters a certain workspace, the system will automatically insert a corresponding operational relation between the actor object representing this user and the workspace object. In many cases it is also possible to derive semantic relations from the system, e.g. a versioning system could introduce specific similarity relations between different versions of design objects.

11.3.3 Events

Events carry the specific information that the model uses to support the different modes of awareness. Events and information about events are used in the model to answer questions like the following:

- Who is currently active in the system?
- What tasks are other users currently concerned with?
- Is someone working on similar issues?
- What has changed in the environment since last week?
- Is my work potentially conflicting with other people’s work?

To achieve this, we distinguish two basic types of events: modifications and activities.

Modification events are generated by the system each time the state of an object changes due to some action of a user, and if the set of relations associated with an object changes. More precisely, a modification of an artifact can be one of the following events:

- *Change of internal state*: e.g. if the content of a document changes, according to a press of the save button in an editor, or if one of the object's attributes, such as access rights, changes.
- *Creation*: A new object is created, either from scratch by some tool, or as a copy of an already existing object.
- *Destruction*: An object is deleted.
- *Movement*: The location of an object, i.e. the position of this object in the semantic network, has been changed, (e.g. an object gets moved into another workspace)
- *Renaming*: The name of an object has changed.

Activities describe synchronous events related to the users in the system. Here we may imagine the following events:

- *Usage of tools*: This type of activity comprises events like editing a document by means of an editor or running a compiler.
- *Presence:* The fact that somebody is accessing a workspace or any kind of object implementing a working context.
- *Communication:* We may describe the usage of synchronous communication facilities by a further type of activity event.

The list presented above is only supposed to give some ideas about possible types of events a system might provide in order to support shared awareness. We can basically imagine any kind of event that has a certain relevance when it comes to coordinating the work in a given setting.

A system that distributes information about events of the types listed above clearly needs to satisfy certain requirements concerned with issues of privacy. We assume that a suitable implementation of the concepts presented here will provide facilities that let the users influence the degree of publicity they are willing to grant to their actions. Such facilities would be straightforward to integrate into the model and we will thus not discuss these questions further.

In the next section, we describe how events, once they are generated by the system, are distributed over the network, in order to enable a shared awareness of the users about the work of others, while at the same time preventing information overload.

11.4 Event Distribution

Events are usually not only relevant for the immediately involved object, but also for other objects that have some relationship to the former. Thus the model needs to distribute events in a sensible way in the surroundings of the object that raised it. On the other hand, it is evident that events in collaborative environments should not be distributed globally in the system, i.e. their presence should be reduced or modified as the distance to the originally affected object increases. As an example consider the modification of an include file in a programming environment. This modification is not only important for people accessing the include file, but also for people working on files that include it.

To achieve this behaviour the model applies the following strategy: operations on objects as well as actions of the users in general create an event of the class that corresponds to the type of operation. The event contains the following information:

- a unique identifier
- a reference to the actor which performed the action
- a reference to the object which is affected by the action
- a time stamp
- an initial event intensity

Subsequently, the event is distributed along the relations that embed the object into the work environment. As the event travels along the relations, the intensity of the event is diminishing with each transition. The decrease of event intensity depends on the particular type of relation it is passing. More specific, it is determined by:

- the type of the event
- the type of relation
- a relation specific intensity factor ≥ 1
- the direction in which the event passes the relation

The intensity factor describes the percentage of decrease of the event's intensity that is performed as the event gets distributed along the relation. If a relation has defined an intensity factor for a given event type, we say that the relation has a rule for events of this type. The direction of distribution is important, because a single relation object actually represents two different relations; the relation itself and its inverse, which may have completely different characteristics of event propagation.

The distribution proceeds by propagating the event recursively in a depth-first-search strategy over the semantic network, starting at the originally affected object and passing along all relations that have this object as their source. Each passing along a relation in turn diminishes the intensity according to the relation's intensity factor. The distribution process terminates if one of the following situations applies:

- the intensity of the event drops below an event type specific threshold.
- the event reaches an object it has visited before and the new intensity level is less than the old one.
- there is no outgoing relation from the current object that has a rule for the event.

In the next chapter we explain how the event model is capable of supporting the different modes of awareness.

11.4.1 Coupled Synchronous Awareness

If a user is active (present) in a certain working context, the system will represent this by establishing an operational relation between the actor and the corresponding workspace object. Activities and modifications that occur synchronously in this context, can be distributed by the event mechanism to this actor, and the system may report this information immediately. The fact that the event travels to this actor also means that the user's focus of work (which is described by his position in the system and his operational relations) also includes the object which is affected by the event. Hence we have the situation of coupled synchronous awareness as introduced earlier.

Consider the following example of a simple working context, containing a few actors and two work artifacts (Figure 11.2):

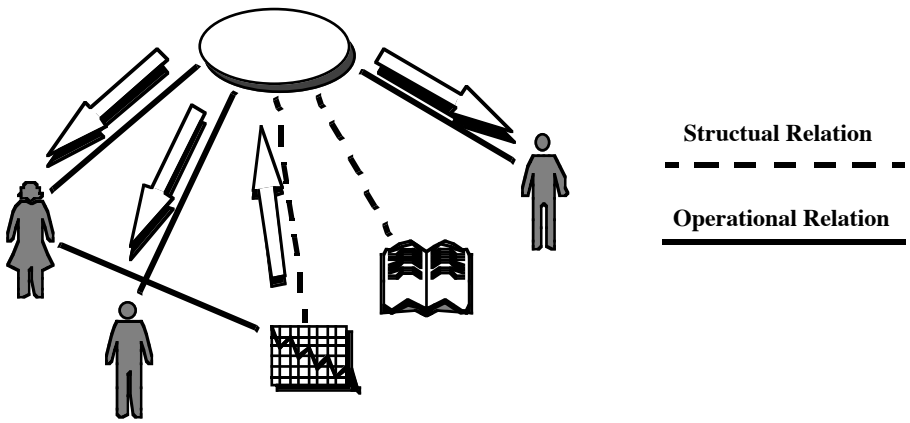


Figure. 11.2: Synchronous awareness

As one of the users works on an artifact, an operational relation between the actor and the object is established and an activity event is generated by the artifact. This event subsequently travels along the structural relation to the context object and gets further forwarded along the operational relations to all other actors who are currently active in the context.

In this example the distribution is defined such that structural relations transport the event only in one direction; from the inferior to the superior object. Operational relations always propagate events to the involved actor. In this way,

all actors in the workspace may be synchronously informed about the actions of other actors.

This strategy may also be used to cross the boundaries of working contexts: in the following example (Figure 11.3), two work artifices are placed in different workspaces and are connected by a semantic relation.

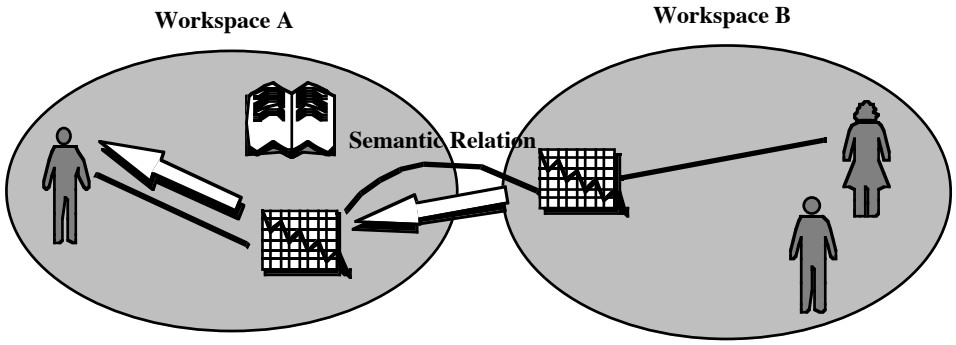


Figure 11.3: Peripheral awareness across working contexts

If we define the propagation rules correctly, modifications and activities that involve one of the objects can be transmitted from one context to the other, and users working on the object can be aware of events that happen to similar objects in another context. Distribution of events along semantic relations is a very powerful concept to support overview, because information about changes of semantically related objects is often of central importance concerning work coordination.

If we additionally define an intensity decrease for these relations, the intensity of events affecting an object diminishes as they travel along the relation. This may then correspond to the decreasing relevance of such events when they arrive at semantically related objects and contributes to prevent information overload. In the example above, we could define the intensity factor of the semantic relation in such a way that the intensity of an event, arriving in the other workspace, drops below the threshold to be propagated even further. This would prevent users working in this context from being informed about events in the other workspace, except if they access this particular object at which the events arrive.

11.4.2 Coupled Asynchronous Awareness

In order to provide an overview of the work that has been performed in the past, the event manager needs to save the events for each object along the propagation path. To achieve this, events are stored in object specific event lists on each object they pass during the distribution process. This may be performed efficiently, by just saving a weighted reference to the unique event object. The weight of the reference determines the intensity of the event at this object. We may also imagine overwriting events with new ones of the same kind and just add a time stamp in

that event. As an example, consider a modification event for a document: instead of creating a new event, each time the user hits the save button of the editor we can simply update a single modification event with the new time stamp. If an actor accesses an object after some period of absence this object can then present a summary of all events that happened in the meantime. This summary may also include information about events that happened with other objects in the surroundings, and that have been distributed to this artifact. This strategy enables coupled forms of awareness, since the considerations in the examples above equally apply in the case of asynchronous event notification.

We assume that the interpretation of the event lists is performed by system services that evaluate the event list each time a user accesses an object. Additionally, a history service may be designed that presents information about all events which affected an object during its lifetime or during some user defined time interval[1].

11.4.3 Uncoupled Awareness

In the case of uncoupled forms of awareness it is necessary that users specify their interest explicitly. At this point the interest relations come into use. If a user specifies interest in a certain object, the system inserts an interest relation between the actor object representing the user and this particular object. Interest relations describe which types of events the user is interested in by means of event propagation rules. For each type of event the user wants to be notified of, the interest relation has a corresponding rule, which defines the intensity factor for the events of this type. Events that occur at such an object are subsequently propagated in exactly the same way as via other relation types and can immediately reach the actor. The event model thus allows a homogeneous treatment for all events and is capable of providing information for the support of the different modes of awareness by applying a clear and simple mechanism.

[1]In order to produce a summary of event lists, we propose a user specific interpretation strategy. This interpretation mechanism is not described in this deliverable. For a detailed description see Chapter 3 of the Strand 1 Deliverable 1.2.

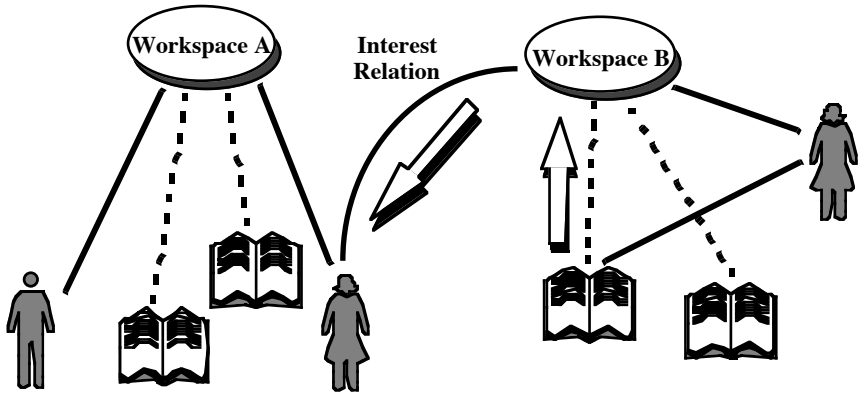


Figure. 11.4: Uncoupled awareness by event distribution along interest relations

In the above example a user has specified their interest in the right working context. The system represents this by inserting an interest relation between the user's actor object and the corresponding context object. All events that happen in the right context and get distributed to the context object are now also forwarded to the actor, who may get an overview about the state of affairs in this workspace. This is a typical example for uncoupled awareness, since the user receives notifications about events that are by no means related to his own working focus.

11.5 GroupDesk

In the remainder of this chapter we describe the GroupDesk system, a prototype CSCW application, that was specifically developed to demonstrate the event model described in the first part of this chapter. To evaluate the event model, the design of the system has dropped any features that would have complicated the investigation of the event related concepts. As a result, GroupDesk has developed as a small platform, supporting distributed work in a simple environment for document production. The second design rationale behind the system has been the evaluation of novel object oriented development paradigms. For the implementation, a distributed development platform, compliant to OMG's CORBA standard, has been chosen [OMG, 91; OMG, 92]. The experience with the object services also had a strong influence on the discussion concerning the COMIC Shared Object Service (SOS). However, the system is currently still under development and does not support all of the conceptual features that have been presented so far.

11.5.1 GroupDesk Functionality

The system implements an environment for collaborative development and sharing of documents. The basic metaphor for coordinating and structuring cooperative work used in the system is the shared workspace, which forms the environment for coordination of document production. A workspace may be

assigned the work artifacts and a set of members, which form the group of users that has access to these objects and may freely modify them. There are currently only documents of different types and editing tools that can be placed in a workspace. Additionally the system implements folders that may contain any number of artifacts, to allow a more detailed structuring of the objects in a workspace. Workspaces may be thought of as rooms in which the objects are visible and accessible and where the group members see each other and meet in order to perform shared tasks. In addition to the group workspaces, the system establishes a private workspace for each user that is registered in the system. Private workspaces may only be accessed by their respective owners. However, the system propagates awareness information related to the these workspaces to a certain degree: other users may, for example, see if a user is currently active in their private workspace.

Workspaces in GroupDesk allow members non sequential, unrestricted access of the objects they contain, thus supporting the accomplishment of tasks that require continuous access of documents and background material by the group members. Workspaces provide users with a shared view on the objects they contain at any time. The actual location of the artifacts in the distributed environment remains hidden from the users. In order to keep the design of the system as easy as possible, GroupDesk imposes no restriction or semantic prescription on the action of users. There are no conflict avoidance mechanisms implemented, to prevent two users from simultaneously modifying objects, for example. The system addresses these problems by continuously providing the users with an implicit overview about all activities that are currently going on in the environment and thus enables an awareness of the users to prevent these situations. Similarly, the system only implements a minimal support for access right management. Workspaces are by default accessible to any user, even if he is not a member of the corresponding working group. Private workspaces however are only accessible by their respective owners. Furthermore any user in the system needs to be registered in order to be able to work with it.

The interface of the system presents workspaces as windows. The objects in the workspaces are shown as icons. The members of the workspace are also shown as labelled icons, showing the picture of the corresponding users. Interaction with the system is implemented by the usual drag and drop mechanism: objects may be moved freely around in the workspace and may be arranged as the users prefer. The arrangement on this level is not shared by all members. Each user may have their own individual view on the contents of a workspace.

Interaction with the system may be performed by double clicking on the respective object icon. If the object is a document, the system will launch the corresponding editing tool to allow the user to read and modify the document. Double clicking on folders and workspace icons opens a window, showing the contents of these objects. The system additionally supports synchronous and asynchronous communication facilities, which are attached to the actor icons.

Double clicking on these symbols launches a video conference to the corresponding user. Artifacts may be moved into another location, i.e. workspace or folder, by simply dragging the object onto the destination's icon or window and dropping it. Each icon additionally has an associated menu attached, which gives users the possibility to delete, copy or rename the object.

11.5.2 Architecture

GroupDesk is designed as a distributed CSCW application, consisting of an object server and an arbitrary number of client applications that may request services from the server. The server manages a repository of objects and is responsible for administering and admission of the users entering the system. The functionality of the system and the distribution of events and object changes is completely controlled by the server. Furthermore, the server serves as an instance that keeps the object repository consistent and enables a common view on the overall state of work.

The implementation is based on a CORBA compliant distribution platform which hides the aspects of localising objects in the domain and granting access to remote objects from the clients. This is performed by an object request broker which is integrated in the distribution platform. The entities in the object repository as well as the clients themselves are modelled as CORBA objects. Clients may request services from any object in the system directly and do not have to be concerned with the interaction with the server. Conversely, client applications also provide a set of methods, which may be invoked by the server or other clients.

Interoperability between different domains is possible, although not yet based on the interaction of different domain servers. Currently users may start a client locally and access a server over the internet. Independent of where the server is running, communication with the server is completely hidden from the user.

The system is structured according to Figure 11.5:

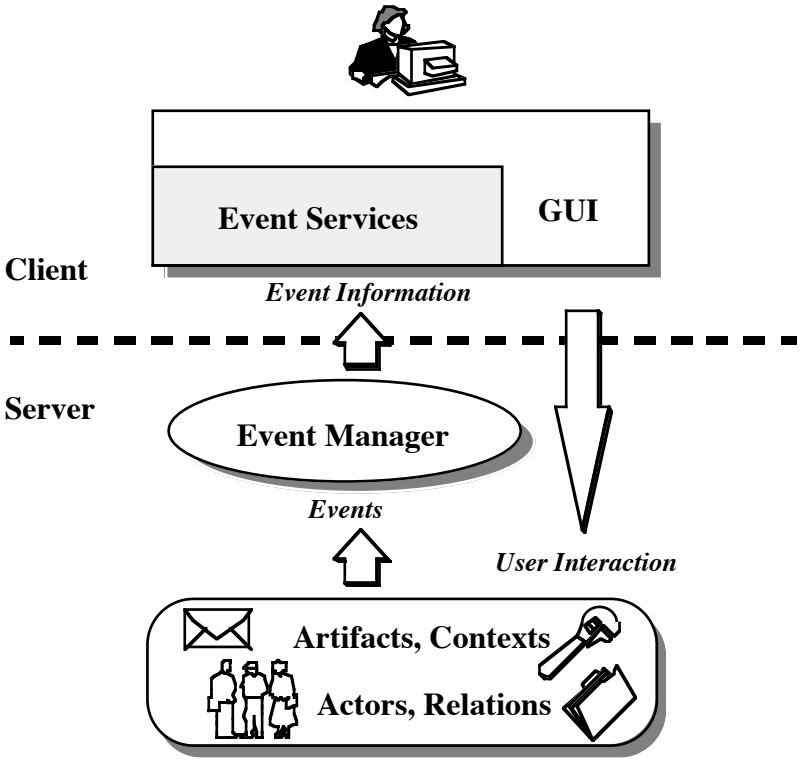


Figure 11.5: GroupDesk architecture

On the client side, GroupDesk offers services that enables users to interact with the system. This basically consists of the graphical user interface, which is responsible for offering the functionality to access and manipulate objects. Additionally, the user interface displays the changes in the state of objects as well as the dynamics in the work setting whenever the server notifies it about new events. Also on the client side, the system provides the management facilities that allow the user to request event related information. Currently the system only implements a history service. It is planned to enrich the facilities with an event notification service that allows users to specify interest in arbitrary events and an event distribution service that enables users to individually tailor the propagation of events in the object repository.

On the server side, GroupDesk implements the common facilities to serve client requests for accessing objects, such as opening documents, deleting objects, or moving entities to another location in the repository. The object repository maintains the representation of the organisational context, i.e. the structuring of artifacts by different typed relations, to form a semantic network. At the current stage, the system only supports structural and operational relation types. Furthermore, these relations are not explicitly implemented. Instead, they are realised as usual object references without a behaviour on their own. The server also implements an event manager, which handles the generation of events each time a user performs some action that results in a change of the object repository and subsequently performs the propagation of the events according to the structure of the semantic network. The event manager is further responsible for

storing events in object dependent event lists and notifies all interested clients about the changes that took place. Additionally, it may receive event retrieval requests from clients and access and return event information.

11.5.3 Awareness Facilities in GroupDesk

The emphasis in developing the GroupDesk system has been the support of user awareness, by applying the strategy of local event distribution. Event related services present to users the dynamics of the work process. Events caused by other actors and external influences are displayed by the system in an unobtrusive manner and include active notifications of changes in the work setting, as well as inactive presentation of event information on user request.

Events in GroupDesk

Currently GroupDesk has implemented two kinds of activities; presence in a workspace and generic working activities.

Whenever a user enters a workspace, the system adds an operational relation between the actor object and the workspace object. Furthermore an activity event is generated which describes this action. The event contains a time stamp and a reference to the actor who has entered the workspace. Subsequently all events that have happened in the workspace since this user has accessed it the last time, are forwarded to the actor object and the user can immediately see what has changed. The system forwards only modification events that have happened in the past and the currently ongoing activities of other users in the workspace to the new user, since activities that happened in the past are not considered important to provide an overview. The user may however request the system to inform him about activities that are already finished, if he desires. This contributes to keep the amount of event information small and concentrate on the current state of work.

The generic work activities include any type of action the user performs on an artifact other than workspaces. Currently this involves editing a document or opening a folder. In both cases the system establishes an according relation between the actor and the corresponding object and presents event information related to the object.

Among the modifications, GroupDesk has implemented the following types:

- object updates, which are generated whenever the content of a document, folder or workspace changes.
- creation of new objects
- deletion of objects
- changes in the locality of objects: this type of modification is generated when an objects is moved into another workspace or folder.

For each modification, the system generates a new event object and stores it in an object specific event list.

Event propagation

Event distribution is currently only statically defined. Users cannot specify individual propagation strategies. This is due to the fact that the system is currently in an experimental stage and still lacks many of the concepts that have been presented before. Similarly, the types of relations currently supported have to be complemented. They currently consist of structural and operational relations. Furthermore, the structural relations only support the basic types of relations to structure the work artifacts in workspaces. A typical GroupDesk scenario is shown in Figure 11.6:

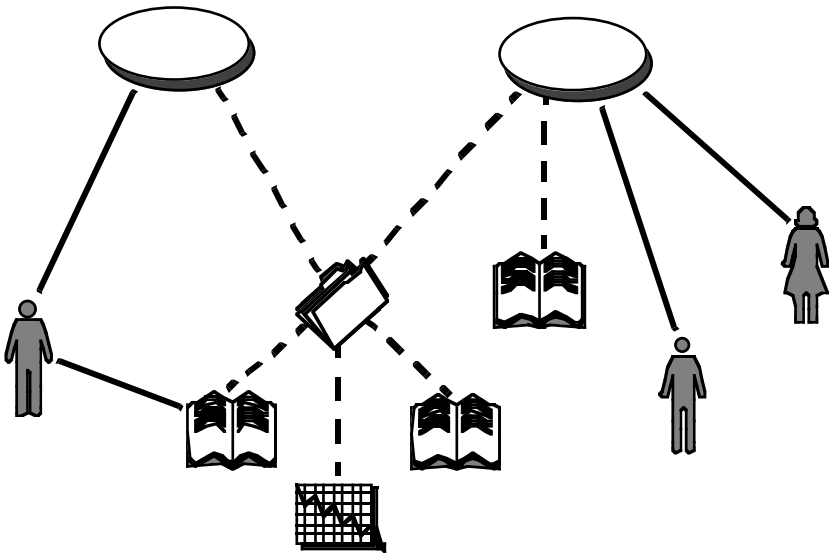


Figure 11.6: Shared workspaces in GroupDesk

In this example, two workspaces are modelled. Structural relations place objects into the respective workspace context and are also used to describe the contents of folders. Operational relations consist of two types, those that describe presence of actors in a workspace and those that represent activities associated with artifacts, e.g. editing a document. Artifacts may be shared among workspaces. In the example above, a folder object is contained in two workspaces simultaneously.

The distribution of events is defined as follows:

- Structural relations always forward events from the inferior object to the superior object, but not vice versa.
- Operational relations always distribute events to the involved actor.

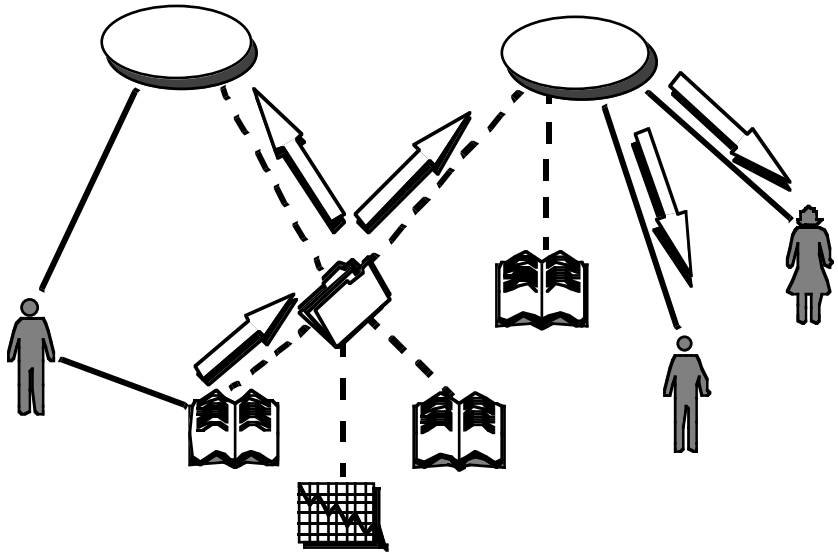


Figure 11.7: Event Propagation in GroupDesk

The activity performed by the actor in the left workspace on the document in the folder is distributed upwards to both workspace objects and subsequently down again to the actor in the workspaces. This is due to the fact that the folder is contained in both workspaces. The users in the right workspace thus get a peripheral awareness about the ongoing work in the left workspace.

Event visualisation

The display of event information is integrated in the standard user interface, which presents the work objects. Modifications on artifacts are indicated by changing the colour of the object's icon. Different colours are provided for the different types of modifications. The system however presents only the most recent modification on an object. This is usually sufficient to give an overview at a glance about the state of affairs in the workspace. If more detailed information is needed, the user may request the complete summary of changes and activities concerning an object via the history service.

Synchronous events, i.e. currently ongoing activities of other users in the same workspace are shown on the graphical user interface by coloured connection lines linking the icon of the actor who is currently performing the activity with the icon of the object that is involved in the activity. The icons of workspace members are always shown in the workspace window, even if they are not currently active. If a member enters a workspace, this activity is shown by changing the member's icon from grayscale to coloured. Non-members entering the workspace are indicated by adding their actor icon in the workspace windows of all other users that have opened this workspace.

In general, the visibility of events is restricted to the visibility of objects in the user's view. This means that events are usually shown at the topmost object in the structural hierarchy, which is visible to the user. If the user wishes to see more details, he can open this object and inspect its contents. As an example, consider

the modification of an object contained in a folder. If the folder is closed, the user may only notice that some modification happened to the folder. If he wants to see more details, he may open it and inspect its contents.

The history service allows users to get a detailed description of the events that happened during an object's lifetime. The service is available for any type of object except actors. In the current version the description is text based and does not allow users to filter event types. Nor does it support event interpretation. The current implementation is, however, useful when it comes to exactly determining what has happened in the past with an object.

This approach of presenting the default event information graphically at the user interface results in an implicit overview for the participants in the work process about the state of affairs, without overloading them with too many details.

Figure 11.8 shows a GroupDesk Interface. The first window displays a list of workspaces. Workspaces with ongoing activities or changes are displayed in another colour. The second window shows an open workspace, with two active users and two editing activities indicated by the connection lines.

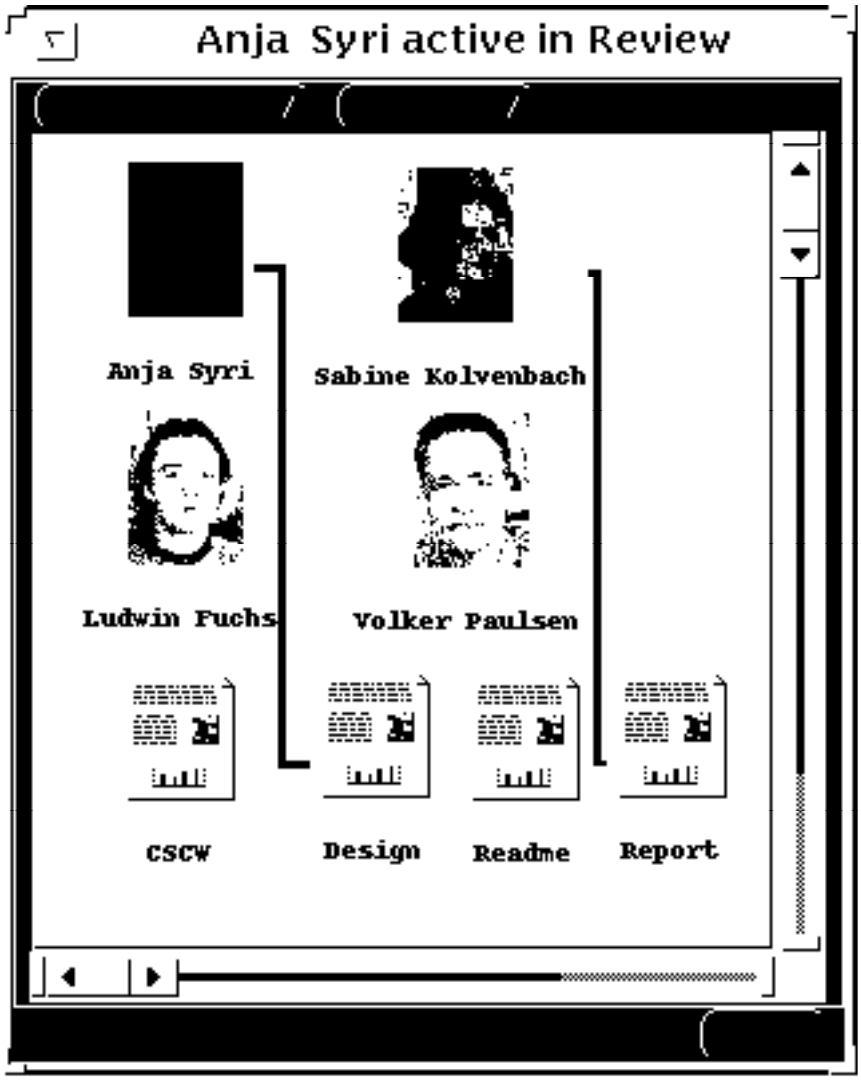


Figure 11.8: The GroupDesk interface

11.6 Future Work

The implementation of the GroupDesk prototype is currently only realising a minimal environment for experimenting with the concepts of event propagation and support for shared awareness presented in this chapter. In order to capture the whole facilities of the model, many things remain to be done. Most notably, the representational issues need to be extended, i.e. the types of relations need to be extended by interest relations and semantic relations. Additionally, the existing relation types have to be further diversified. To capture the dynamics, it is necessary to implement the concept of event intensity and introduce the facilities for individual tailoring of event propagation.

Conversely, the system needs to be enriched with more sophisticated event services on the client side. It is planned to extend the history service with facilities

for graphical display of history information and to include services for event interpretation. To achieve support for uncoupled awareness, an event notification service has to be integrated. Last but not least, it would be necessary to provide full object persistency in order to make the prototype stable enough for real practical work. This is currently not realised to a full extent.

11.7 Conclusion

In this chapter we have presented an event mechanism which is capable of providing information to describe the dynamics and state of work in CSCW applications, and thus may be applied to support shared awareness in systems for the coordination of cooperative work. The model proposes the representation of the environment as a semantic network. Awareness about changes and synchronous activities in the system is supported by the generation and distribution of events in the semantic network. The propagation mechanism provides the flexibility to distribute the information, so that it may be accessed in places where it is relevant, and on the other hand prevents overloading the user with unnecessary details.

GroupDesk, a prototype implementation of this model, has been presented. The system is implemented on the basis of a distributed object service platform. The system implements a simple environment for coordination of distributed work and enables the support of shared awareness for the users by applying the event model and visualising the event information using the desktop metaphor.

11.8 References

[Dourish, 92] Dourish, P. and Bellotti, V., “Awareness and Coordination in Shared Workspaces” in *Proc. of CSCW ´92 - Sharing Perspectives,* Toronto, Canada, ACM Press, 1992, pp. 107-114

[Fuchs, 94] Fuchs, L., Pankoke-Babatz, U. and Prinz, W. Ereignismechanismen zur Unterstützung der Orientierung in Kooperationsprozessen, to appear in *Proc. German Conference on Computer Supported Cooperative Work*, Marburg, September 1994.

[Jarke, 92] Jarke, M., Maltzahn, C. and Rose, T., Sharing Processes: Team Coordination in Design Repositories, *International Journal of Intelligent and Cooperative Information Systems*, Vol. 1, Nr. 1, pp. 145-167, 1992.

[OMG, 91] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG TC Document 91.12.1, 1991, available on request from the Object Management Group.

[OMG, 92] Object Management Group, *Object Management Architecture Guide*, edited by R. M. Soley, OMG TC Document 92.11.1, 1992, available on request from the Object Management Group.

[Smith, 91] Smith, C. S., *The Parameter of Aspect* Reidel-Kluwer, 1991.

[Sohlenkamp, 94] Sohlenkamp, M. and Chwelos, G., Integrating Communication, Cooperation and Awareness: The DIVA Virtual Office Environment, To appear in: *Proc. Conference on Computer Supported Cooperative Work - CSCW 94*, Chapel Hill, NC, October 1994.

[Vendler, 67] Vendler, Z., *Linguistics in Philosophy* Cornell University Press, Ithaca, NY, 1967.

Chapter 12

Interface Builders and Bulletin Boards, Techniques and Requirements on SOS/SIS

B. Eiderbäck and P. Hägglund

KTH

12.1 Introduction

The main objective of this chapter is to emphasise some essential facilities of the SOS and SIS. We illustrate how to use the services for CSCW applications where users:

- have essentially the same view of the document.
- have tailored views of the document.
- interact with different parts of the document.

Here a document can be any set of objects. The first case illustrates how users can use SIS "metatool" facilities to increase their awareness of each other's actions and also for communicating temporal meta-information related to the current cooperative setting. For the second case, we describe tools aimed at easing cooperation among users with different presentations of the same document. For the third case, we propose techniques and tools to support awareness about changes to objects and presence of other users. In this case, we focus on awareness among users who cooperate, but work on different aggregates of overlapping documents.

We describe and demonstrate an Interface Builder (IB) prototype, Collaborative Work Interface Builder (CWIB), which enables developers to construct CSCW applications of the three types above. We describe how CSCW applications can be developed with the help of CWIB, and demonstrate how it can be used as a distributed application in its own right, especially for cooperating developers while building or maintaining CSCW applications.

The differences between the requirements of asynchronous and synchronous communication are stressed by demonstrations of a Bulletin Board (BB) service.

A discussion and demonstration of how applications can communicate independently of lower level communication protocols is also presented. With regard to this, techniques to migrate applications among various types of communication protocols are elaborated. We describe and present the benefits of proxies and adapters to simplify communication and cooperation among objects

in a distributed environment. The distribution package MultiGossip, which is aimed at supporting CSCW applications, is also briefly described.

12.2 Interfaces

In the SOS requirements chapter different types of computer supported collaboration were identified. In this section, we elaborate the different types of collaboration further.

We have identified various sets of interfaces and techniques in CSCW applications. We have made the following major distinctions and classifications, important to our work, of the type of sharing:

- **Shared:** All involved clients (essentially) have the same view of the document. Some minor differences are allowed if they not are significant to the collaboration or context; e.g. colour, cursor shapes, etc. Applications are allowed to change parts (i.e. objects) of the application that, in the current context, do not belong to the domain of essential cooperative objects (e.g. objects that definitely do not belong to or are related to SOS).
- **Partly shared:** Another possibility is that different users have a partly shared view of documents but still with their own style of interaction and presentation even in the shared parts.
- **Different outfits:** Another type of sharing is when the users (developers) compose new interfaces by reusing components defined in a diverse set of other applications. In this type of sharing an application that visualises two (perhaps completely different) documents, which gives some new type of information and awareness, can not rely on other users' "document-viewers" to be conscious of any such combined change.
- **Sharing through objects:** In the final form, sharing occurs at the object level. The sharing is essentially through SOS. Most issues related to interaction must be imposed as a layer on top of these more fundamental objects sharing mechanisms. This form of sharing is the most general but at the same time demands most of the SIS services in order to support the exchange of required meta-information.
- **Adoption and tailoring:** The users, or developers, should be able to adopt the interfaces to their own, or the environments, specific needs and requirements. In SIS, and also in SOS, it is stressed that cooperation among users with slightly different interfaces must also be possible.

12.3 The Distribution Package MultiGossip

The distribution package MultiGossip is intended to make it possible to build and experiment with distributed interfaces that easily and seamlessly distribute an extensive set of objects. One of its objectives is to support experimenting with,

and change of, the distribution style of the involved objects. It also supports migration of applications from non-distributed to distributed settings. MultiGossip supports distribution of:

- ordinary Smalltalk objects
- special purpose designed objects
- different kinds of media objects.

This is achieved through the use of a large set of different protocols. It also contains tools that assist users and objects to establish and maintain cooperation and connections. The primary intention of the package is to release programmers from the linguistic and semantic burden when distributing interactive applications.

12.3.1 The Distribution Model

MultiGossip is based on a high level object oriented model of the distribution where all parts are defined by classes and all services are realised by objects.

In MultiGossip all ordinary Smalltalk objects can be transformed into shared objects by giving them commands to distribute themselves. When doing so, a *shadow* or *proxy* is created as an alias of the existing object. This new object knows how to distribute messages and arguments of the object, but still appears to function as the ordinary one from the "client's" point of view. The use of proxies for distribution makes it easy to adopt and change these mechanisms on a per object basis.

Messages are passed between nodes and invocations of messages at remote nodes are administrated through encapsulation of the message descriptions into special purpose high level message passing objects.

In MultiGossip we can identify two main services:

- *connection service*: establishes, upholds and synchronises connections among objects, users, nodes and images, and is responsible for mappings between higher level actions and lower level protocols.
- *object communication service*: transforms objects into distribution objects of different kinds and further knows how message passing between the objects should be handled.

12.3.2 Distribution of Objects

In MultiGossip we can specify if an object should be distributed by reference or value. It is also possible to choose between *eager* and *lazy* distribution. In the former situation, objects are copied as soon as they are referenced but in the latter case they are only copied when used (with possible caching for further references). The package also supports a mix between these two extreme strategies. A mixing strategy is one where a shallow distribution is made, i.e. an eager copying strategy of the objects until a certain level and then a lazy one of objects deeper than the prerequisite level. The latter is, for instance, suitable for

distribution of objects of collection or graph type, where the elements that constitute them refer to other further objects. We can change these strategies during runtime depending on the application's requirements.

12.3.3 Architecture

MultiGossip contains some prefabricated special purpose distribution classes, general classes and methods that make it possible to easily distribute any existing (non-distributed) object. Distribution of all types of objects with MultiGossip relies on a technique based on *proxies* and *adapters*.

12.4 Building Blocks

In this section we describe some of the fundamental building blocks used in the construction of the prototypes and the distribution package MultiGossip as well.

12.4.1 Proxies

A *proxy* is a representative of an object, or more specifically (as in Figure 12.1 below) a distributed entity. It is responsible for making the distribution invisible and transparent in programming code, while it "hides" communication and distribution details.

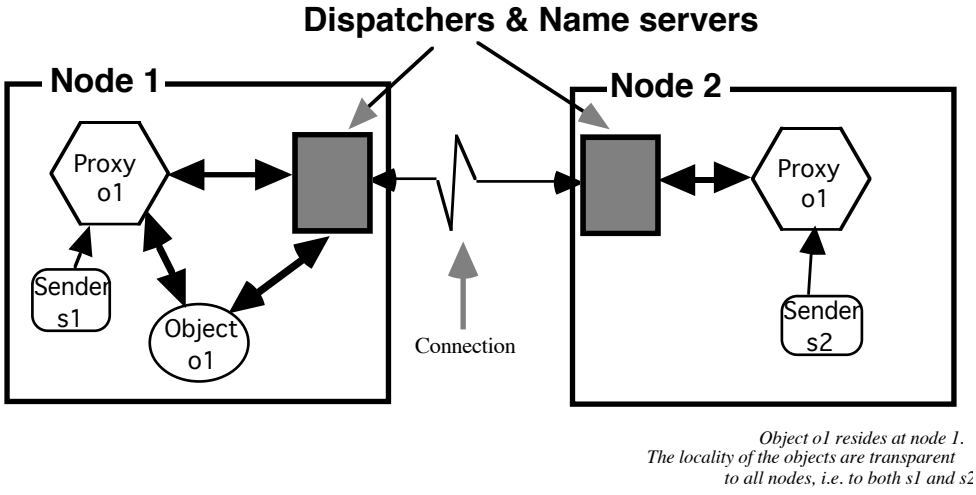


Figure 12.1: Example of a proxy

The figure illustrates how the proxy "Proxy o1 " represents "Object o1" at each of the two involved nodes. The senders, s1 and s2 respectively, cannot in principle tell the difference if the proxy or the original object is currently being referenced. All messages intended for the object are captured by the proxy and sent onto the object. The reply is finally sent back to first the proxy and eventually to the

sender. In the figure the proxies (implicitly) use the name server and dispatcher for their communication.

A proxy can also be considered a more restrictive, special purpose adapter.

12.4.2 Adapters

Adapters are objects designed to make objects talk to each other with as uniform protocols as possible, by providing a standard set of accessing messages. In Smalltalk adapters of this type are used to make views adaptable to different models and vice versa.

An adapter can also provide a specific service; e.g. interfacing a file or catching all events directed to a certain view, hiding all the details from clients. The latter makes it (for instance) easy to replace a file aimed at storing version information for an "ordinary object" or a socket communication without any impact on the clients' objects.

In the SOS/SIS the subscription, locking and objects accessing mechanisms are easily implemented with adapters. We have also used adapters for implementing (and hiding) distribution of objects and for implementing awareness (for further details see the later sections about the demonstrators).

12.4.3 Wrappers and Containers

Many frameworks for building interactive graphical applications use a technique of wrappers and containers to ease the use of, and combination of, views and interaction techniques when defining a GUI. We have used techniques related to this to build and combine CSCW applications.

Wrappers and containers can also be considered specialised adapters. Furthermore they have some visual and spatial knowledge which in some extent can be used to define characteristics of a document at a specific node.

12.4.4 Virtual Screens, Over Layering, Superimposing

One can use adapters/wrappers that make it possible to communicate meta-information through "over layering" documents by other documents and applications.

Consider for instance working with a text editor. By superimposing this editor with a drawing editor and some adapter that glues them together and an "over layered" wrapper that dispatches command from users, one can relatively easily provide a freehand annotation facility without changing either the text or drawing editor. This technique allows us to take advantage of the benefits of the respective editors (progressing) capacity.

12.4.5 Events

To make the distribution and sharing mechanisms as dynamic as possible and as independent as possible on the actual environment we mainly use events to propagate changes and transmit information among the cooperating nodes or objects.

We have layered the event model on an object oriented model where the events are first class citizens. In MultiGossip we transform all messages sent to remote objects into event objects. These event objects know (implicitly by their belonging to a class) how to invoke the “real” object at the remote host, passing it the message and finally returning the result (or an exception object if anything goes wrong) to the sender of the message.

12.4.6 Dependency

The concept of separating the responsibilities among different types of interface and interaction components has proved to be very useful in the description and development of most interactive applications. We have also found this kind of separation very promising when developing distributed interactive applications. Central to most of these techniques is the possibility to make objects dependent on each other. This allows for better ways of structuring the problems and encapsulates implementation details very well.

The dependency technique is used in various sets of different fundamental interaction models as in the famous Model View Controller concept (see section 12.5.1), in Value holders, active values, etc. Our work has also shown dependencies to be suitable for implementing subscriptions, awareness, and other services identified within SOS and SIS.

12.4.7 Value Holders

The technique of using active values or value holders is very powerful when it comes to defining interactive applications. One of its foremost benefits is that it tries to purify the model point of view of the objects instead of considering a component with both view and interaction. Value holders can also be first class citizens, i.e. objects that belong to classes, which enable us to implement different type of awareness directly built on value holders.

A value holder is a specialised adapter with more direct (non transformed), often destructive, access to its encapsulated object/subject.

The value of a value holder is read or changed by sending messages to it.

A value holder also informs all entities that have registered interest in it if the value changes. One can use this to link the user interface and the underlying application. If an application and a user interface share a value holder, for example one holding a string, both the application and the user interface express interests in changes to the string. The interest is administrated by the dependency

mechanisms described in the previous section. If the string is changed by any of the dependent objects, all but the originator of the change are informed about the change. For example, if the application changes the string, then the user interface is informed and can show the new string to the user.

12.4.8 Behaviour Objects

In order to loosen the coupling of the interactions with presentations of objects we use objects at a higher order of abstraction. These objects aim to hide details of both interaction and presentation from the actual model or application. By separating and encapsulating communication between different models in behaviour objects it is easier to hide platforms, language or interface details from "sharing" models.

A behaviour object is a conglomeration of an adapter, view, controller, wrapper and even a model to some degree. The main objectives are to filter, transform and put the communication into a "canonical form", i.e. it filters both incoming and outgoing events.

12.5 Techniques for Building Distributed Applications

In this section we describe techniques aimed at constructing distributed applications of CSCW type. We show how we have exploited and extended techniques for construction of interactive graphical interfaces, as MVC and active values, to better suit requirements while developing distributed applications. We also propose ways to separate interaction components from each other. Finally we compare some different migration strategies used in MultiGossip and our demonstrators.

12.5.1 Model View Controller

The Model-View-Controller (MVC) paradigm invented at the end of the seventies is a long standing model of how to develop interactive graphical interfaces. In this section we describe how we can use this technique to build distributed CSCW applications.

Distributed Model View Controller (DMVC) is an extension of the ordinary MVC paradigm. We need a specific MVC for distributed environments as it provides a way to distinguish between events that are tightly related to the local context of the MVC-triad and events that can be of interest for foreigners.

In the (pure) normal MVC settings are changes to the model transmitted as change-events to dependent objects. In a distributed situation, most of the "local-scope-events" are of interest for foreigners as well. Nevertheless we strive for as smooth techniques as possible to propagate events that can be mainly related to interaction or alternatively to changes in the presentation. Some problems related

to this can be solved by imposing more responsibility on the model, and in some circumstances on the view-controller-pair as well. Some information is more related to the context of the GUIs and the discrepancies between them. We think that problems related to this are better solved by extending the abilities to tune the passing of events between applications than by imposing the new requirements on the model.

We exploit the MVC model as a base for DMVC by:

- extending the ordinary dependency transformers for distribution. As a proxy for the dependent objects in Smalltalk one can use dependency transformers. A dependency transformer knows which events the real dependent objects are interested in and filter them before they are eventually transmitted. By exploiting this special form of adapter and extending it with rules of distribution we provide a simple basis for a more controllable events distribution mechanism.
- preparing the view and controller.

Locally (i.e. within a single node), we use the ordinary dependency mechanisms given by the MVC model and extend it with the ability to define dependencies among nodes in a network, i.e. we both use the “old” dependency mechanisms locally, and distributed ones to establish dependencies among objects in a distributed network.

12.5.2 Active values

As mentioned in previous sections, interfaces built on the idea of active values, where the active values are first class citizens, is very attractive. This model encourages separation of the components for interaction and visualisation from the model.

12.5.3 Behaviour

In this subsection we give an outline of an attempt to separate the MVC paradigm further by including an object intended to control the behaviour of the triad in a more abstract manner. This is perhaps a harder problem than constructing “ordinary” behaviour objects, but is probably even more desirable in SIS and CSCW than in non distributed settings.

12.5.4 Migration

Core to our development is the ability to migrate objects, classes and descriptions between ordinary objects, distributed objects and local “emulated” distribution objects. We have particularly developed techniques to migrate objects from local scopes into global ones. We have found that the ability to construct, test and debug applications in a local setting before distribution significantly shortens the development time of CSCW applications.

12.6 Problems Posed by Cooperation

In this section we seek to identify problems related to cooperation and give some proposals for techniques and tools that solve these problems. A distributed application aimed at supporting and easing cooperative work among users must generally be prepared for both asynchronous and synchronous communication and cooperation. It is obviously important that these are supported in a flexible and seamless manner allowing users to move between synchronous and asynchronous work at low cost. Additional issues include the users' possibilities to orient, the amount of awareness and focus and telepoint in the shared domain.

In many ways the issues of asynchronous support are more relevant to the shared object service than the support to be provided by a shared interface service. The provision of tools to provide asynchronous interfaces for cooperation among the participants are been investigated within this domain. These are affected by the intended number of users, the awareness of change and temporal issues of history, etc. In this section we wish to focus on the needs highlighted by synchronous work.

12.6.1 Supporting Synchronous Interaction

In a synchronous situation it is more obvious that we will benefit from tools and building blocks especially devoted to SIS than in the former asynchronous situation. Three important situations are central to the provision of support for cooperation:

A single user

If no cooperation ever takes place this is trivially the same problem as single user interaction with the application. Although the single user case must certainly be prepared to "transform" into a multi user case or update information in an asynchronous mode. In short the applications must be prepared to change form of cooperation.

Two users

This arrangement is harder than the single user case, but still this biparty form of cooperation can rely more on the involved users ability to sort out possible problems in the "group" by more conventional methods. There is limited need to provide identification mechanisms to make users aware of who else is involved in the interaction.

Many users

This is considerably more involved and complex than in the two users case. Here we must provide tools and mechanisms that make it possible to distinguish the different users' actions from each other. It must also be possible for a user to point

into, or emphasise a certain section of different users visualisation of the shared document. The latter is important to, for instance, increase other users attention of a specific part of a document (perhaps to discuss a intended revision).

Each of these three situations highlight particular demands which need to be met by any toolkit which intends to support cooperative interaction in real-time. It is essential that the mechanisms involved are provided in such a manner that the divisions between these situations are blurred and that users can move between the different arrangements. Consequently the development mechanisms need to be open and extensible.

12.6.2 Awareness

The issue of awareness is central to the provision of cooperation support across a community of users. A number of different types of awareness are supported by the SOS and SIS. In the case of the SOS these focus on the awareness of action over a period of time. It is less clear what forms of awareness are only supported by SIS and how we achieve migration between the different forms of awareness involved.

Within the SIS the principle means of providing awareness of others is through their effect on objects and the use of different kinds of media to reflect a number of key properties. The means by which these properties are reflected at the interface determine the level to which a cooperative interface promotes awareness. Significant properties include:

The status of those involved in interaction

In this case status has three significant components which need to be displayed:

- *Presence*: Displays information to show if a particular object or person is present at this moment.
- *Activity*: Indicates the level of activity associated with the object. In particular display if the object is dormant or active
- *Change*: Highlights the degree to which an object been changed and if it is changing at the moment

Spatial arrangement

The spatial layout of objects and users is significant in determining the level of awareness of the users involved. The spatial arrangement of objects is straightforward in VR but less obvious in 2-D applications. We have solved some of the problems by encapsulating important and interesting interaction elements by distributed value holders.

Logical

How do we make the clients aware that some changes directly affect SOS and some others momentarily do not. We need to reflect that interaction will only affect the current SIS context and may later propagate to SOS

Temporal

In some situations it is important to preserve temporal information in the propagation of events, thereby will it certainly be necessary to make the clients aware of this, e.g. when, in what order, etc.

Audio

In our tools we use audio for communication between cooperating users, as clues to a specific object's actions and finally as a way to annotate texts and other involved objects. This form of awareness is essential to the support of cooperation.

12.6.3 Pointing and Gesturing

Pointing and gesturing is central to shared interaction. In particular some indication of the point of action associated with each user is essential within shared interaction. Action pointers are important to allow users to emphasise certain sections of the cooperative work on each others views. If all users have the same views of the documents then this is a quite easy problem to solve but if all the users have their own distinguish views then this will be much more cumbersome to solve. Currently, we focus on action pointers of the following kind:

- Telepointing: If one user wants to emphasise, stress or simply help another user they can "tele-operate" in the other users view. This indicates on the user's display that action is been remotely controlled.
- Focus: An important action pointer is one which tells other users where the current user's focus is.

In this section we very briefly attempted to outline some of the key requirements emerging from our development activities. In particular, we have focused on the different facilities which need to be provided to support shared interaction where a community of users are involved in controlling a cooperative application. Many of these element are reflected in the prototypes constructed and are currently being assessed for their general applicability.

12.7 Servers and Services

In this section there follows a short description of servers and services that are illustrated in our demonstrations. There are still services that not are described

here, e.g. history and versioning services, as we have not focused on these problems in our demonstrators. We require:

- Name servers: Essential to cooperative computer applications is the use of name servers. A name server can be regarded as a Finder which knows how to find objects residing somewhere in a set of cooperating nodes. The name server can look up objects in accordance with several different criteria. One of the most common ways to identify objects is to assign globally unique identifications to them:
- Connection and association servers: Tools and techniques to establish contacts between users and objects.
- Repository services: Using the model of adapters it is very easy to (for instance) hide a repository service, e.g. by a conventional data base, from the applications.
- Contact services.
- Spontaneous contact services:
  - Logical, interest, etc.
- Incidental contact:
  - Spatial
  - In same location.
- Explicit:
  - Telephone type
  - Group
  - Team
  - Board or Room.

12.8 A Demonstrator of an Cooperative Work Interface Builder

We now demonstrate our Cooperative Work Interface Builder (CWIB) aimed primarily to enable fabrication of Graphical User Interfaces (GUIs) among a group of cooperating developers and also to support construction of distributed GUIs of CSCW-type.

12.8.1 Constructing with CWIB

CWIB is intended to construct applications in either a single user mode where all the developers have their own running copy of CWIB or in multi user mode where the developers work in the same space of applications and widgets. In the former case cooperative applications can still be developed by:

- one developer at a time: a developer defines a component that will belong to a shared name space

- several (simultaneous) developers: several developers cooperate in defining their own (pre-) tailored view of a shared application. The shared components are agreed on via other channels than CWIB. Here the developers typically “negotiate” with help of audio, video, text tools and by mailing code-fragments and widgets. We also experiment with different forms of awareness within this form of cooperation. In the latter case, when CWIB explicitly assists in the cooperation, the developers work in a shared space of widgets and code. In this situation the cooperating developers can also work with further types of sharing:
  - shared: all the developers have the same view and all interactions with the model have an immediate effect at all users’ interfaces, e.g. dragging a component will result in the dragging of the same component in all the developers’ views.
  - partly shared: the developers share the model but do not see all the details of the interaction, e.g. dragging of objects, but they can still be aware of them. The unification takes place at the ending of each “mini-transaction”. Within this model, the users can work on different parts of the application. In this model the shared type above, with (for instance) explicit dragging awareness, can take place in those parts of the interfaces that are mutually equal.
  - object sharing: we also experiment with sharing through objects where users define which parts belong to the shared environment, and which do not. Here we achieve better support for cooperative development with more overlapping (considered from a component based approach) than in the former two situations. This latter situation agrees more with the single user mode but automatically gives more support for cooperation within the CWIB environment.

12.8.2 Using CWIB to Construct Shared Applications

By the use of adapters, value holders and separation of models from interaction and presentation many interfaces are easily distributed. In Figure 12.2 below, two different interfaces are constructed. The darker ovals describe the shared objects domain and the brighter ones some local (at the moment) non-shareable domain. All entities have an identification, agreed on by the cooperating applications’ name servers.

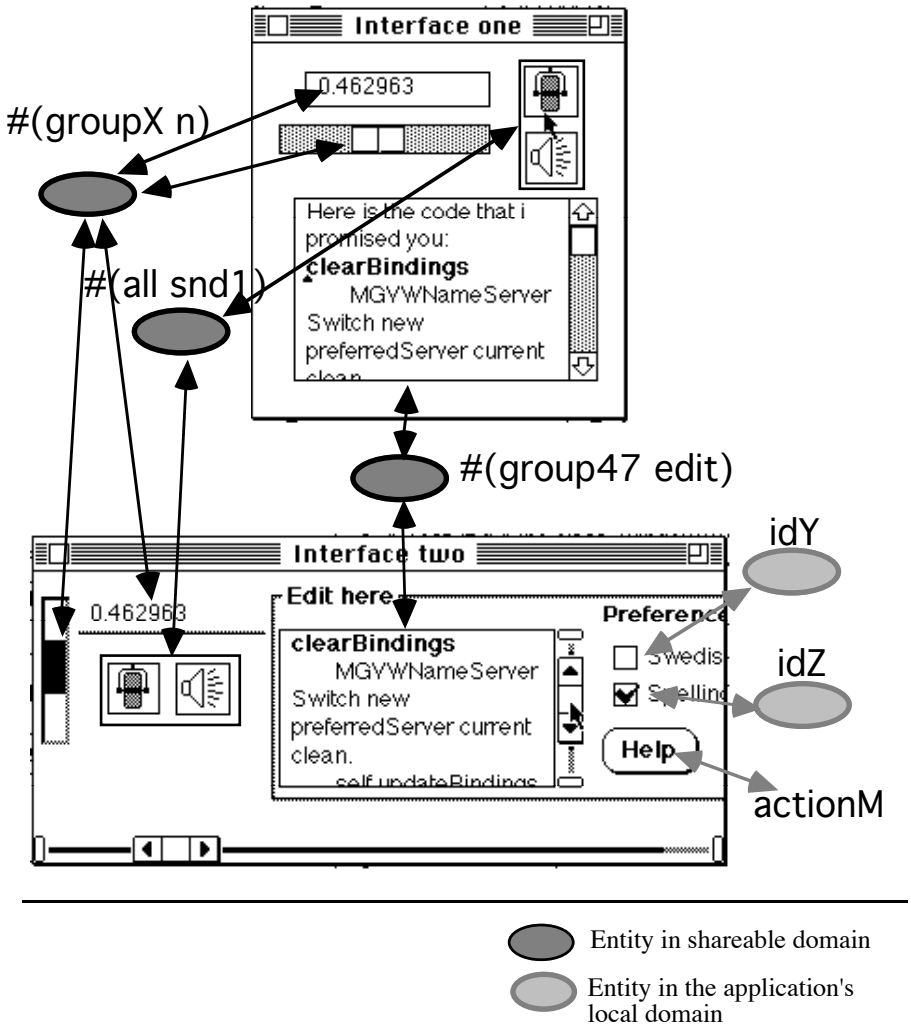


Figure 12.2. Shared objects interfaces

There is no need, from the context of the application's point of view, to really distinguish the names of entities that belong to the local domain from those that belong to a shared domain. However, to simplify the construction of the CWIB (which is based on Smalltalk/VisualWork's IB) and to make it easier for users of CWIB to distinguish among the different types of objects, we have chosen to identify a distributed entity, i.e. by a context (group) name and a local identification within the group. In Figure 12.3 below we present the definition of the object's aspect. We have been able to reuse the already existing "properties tools" and changed appropriate parts of the interface builder to recognise an aspect given as an array as a shared entity and thereby bind it to a proxy that uses the NameServer to find out about the "real" objects type and location.

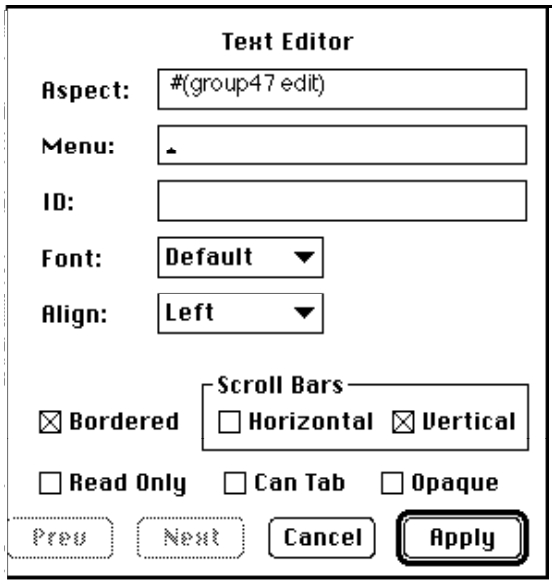


Figure 12.3

Implicitly we get a primitive form of persistence store within our solution. The reason is that if an object with the actual aspect name already exists then, instead of creating a new object of the appropriate type, we create a proxy which refers to this existing entity.

12.9 CoBoard: Description and Demonstrator

CoBoard is a computer based form of a standard physical bulletin board that can be shared between a group of people. It uses the bulletin boards metaphor for its interaction.

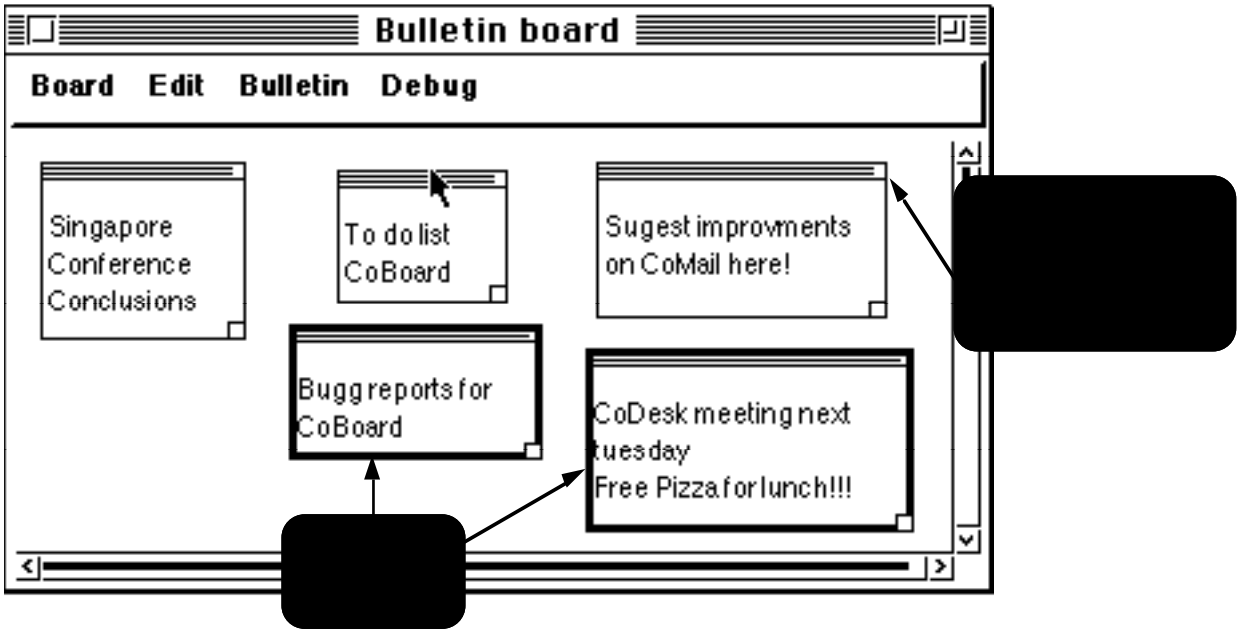


Figure 12.4. A bulletin board[1]

CoBoard is an open system where each member of a bulletin group has the same rights to a board. As with physical bulletin boards everyone that can look at the board add a new bulletin, correct errors in a bulletin, and even remove somebody else's bulletin.

The restrictions on what members are allowed or are able to do are entirely on a social and perhaps legal level.

One reason to chose the bulletin board as the metaphor for a message system is that the 2D board gives a spatial order among all messages and this gives the users a way to organise all messages. One can use the spatial memory to find a bulletin earlier read and see if new bulletins are added. This means that the bulletin board metaphor hopefully gives the user a good overview of all messages.

12.9.1 Awareness in CoBoard

To exploit the computerised environment while creating CoBoard we think it is desirable to extend the metaphor of a physical bulletin board. A major aim is to make the information at the board as easy as possible to assimilate, e.g. it is essential to emphasise bulletins of possible interest for the current user. As an example, we highlight bulletins that contain non-read information. Of course there are many different possibilities; e.g. highlighting the bulletins from the user's

[1]CoBoard is written in Smalltalk□80 and where it is possible to choose among a various set of widgets, e.g. Motif, OpenLook, Mac, MS-windows or Smalltalk's own look-and-feel. The picture in figure 10a was made on Macintosh with Smalltalk's widget set.

superiors, etc. This kind of tailorability may be supported in later versions of CoBoard.

To increase a user's awareness of other people in the system we add information about their presence and actions. Two important aspects to visualise are the reading and editing of a bulletin by a user.

A way to visualise the fact that somebody is editing a bulletin can be to simply remove it from the board so that nobody else can edit it. This appears reasonable following the metaphor of a physical Board where someone may remove a bulletin while editing it.

A more useful way to visualise that somebody is editing a bulletin is to put some sort of mark on it. Then it will be possible to see both the old version, and check who is editing the new one (and maybe contact him/her).

This simple scheme with lock marked bulletins described above gives a way to handle a lot of problems with synchronous communication.

Another important form of awareness in CoBoard is to notify the user about things that have happened in the past. Examples of information that may interest the user of a Board include:

- What has changed since I last looked at the board?
- Who has read a particular bulletin and who has read any versions of it?
- Who is the author of this?
- Who made the last change?

12.9.2 History of a Bulletin

In many situations it is useful to have a history of the development of a bulletin. Versions can be used to point out changes of the bulletin since the last time a person looked at it. This is important if, for example, a bulletin announces a meeting, and the time of the meeting is later changed. On the other hand, people who not are involved do not want this emphasis on a new version.

12.9.3 Cooperation with CoBoard

CoBoard allows a group of people to share bulletins on a board. The semantics of this sharing is that at any time there exists only one instance of each bulletin. The system ensures the integrity of the instance, so simultaneous accesses to it will have predictable results. In the present implementation of CoBoard only one person at a time can change a bulletin; in the future we will support cooperative editing of bulletins.

An example of the usefulness of the single instance approach of CoBoard is when scheduling a meeting for a reasonably large group of people. Then a user's suggested times can be written on a bulletin. While people read it they can make comments on the times; e.g. tell which times that are impossible for them, etc. Since everyone is writing on the same (single) bulletin instance it will always be

guaranteed to be accurate and reliable. Compare this to scheduling a meeting with the help of electronic mail, for example.

12.9.4 Implementation of CoBoard

Events and subscription in CoBoard

Events and subscription are used to update the user interface when some aspect of the board or a bulletin changes. This means that all interface aspects (such as size, position, contents of a bulletin) subscribe on events that reflect changes on those aspects. So if somebody changes the contents of a bulletin by including new information that affects other users, then the bulletin on their screens receives an unread event and their view of it becomes lighter with a thick frame around it.

Adapters and value holders in CoBoard

In this section we describe a way to implement CoBoard and the different aspects described above. The implementation techniques are general and can be used in different kinds of collaborative applications. It is based on Adapters and Value holders described earlier in sections 12.4.2 and 12.4.7.

Value Holder

The basic idea of a value holder is that it stores a value. This value can be read or changed by the value holder by sending message to it.

In addition, the value holder also informs all entities that have registered interest in it if the value changes. One can use this to link between the user interface and the underlying application. Then the application and the user interface share a value holder (for example a value holder with a string). Both the application and user interface expresses interests in changes in the value holder. Then if the user changes the text, the application will be informed and vice versa (i.e. if the application changes the string, then the user interface is informed and can show the new string to the user).

Shared value holders

If we extend the value holder, so it can be shared between different users, then the value holder can be used as a basic component in a shared application. Every time a user changes the value of a shared value holder, all other users who share that value holder will be informed and obtain the new value. With the extension of value holders to shared value holders it is possible to build applications with interfaces to many hosts in an easy way. This is simply done by attaching all parts of the interface on every host to the shared value holders.

An example of a shared application is the first simple form of bulletin board in the upper part of Figure 12.5 below. All hosts will share three value holders for each bulletin, its position and size, its title and the text it contains. The bulletin is then displayed as a paper on a board on each user's screen. It can be manipulated

by all users without any concurrency control and the only awareness mechanisms employed is that changes made by a user will be immediately shown to all other users.

Adapters and collaboration

An adapter is an object with in principle the same interface as a value holder. The difference is that the adapter does not store the value inside itself but transfers it to another object with perhaps another interface. In addition, adapters can add functionality to the objects they adapt. For example there is a standard Smalltalk class BufferedValueHolder that makes it possible to discard or accept changes made to the value, before it is stored in the real object. This is useful if one has a user dialogue where it is possible to accept, cancel or revert changes made in the dialogue.

Awareness adapters

The objective of an awareness adapter is to estimate how aware the user is of its value, and make that information available to other users. It is hard or even impossible to find a general way to estimate how aware a user is of a certain value that is presented to him. Depending on the way the value is used the designer of the user interface can make some estimation. This means that awareness of the value greatly depends on the context where it is used. It also means that in different situations one needs different types of adapters that calculate the users awareness factors in different ways, and depending on different circumstances.

The awareness for a value calculated by the awareness adapter is stored in a value holder itself so it is possible to subscribe on changes in awareness for other users.

Locking adapters

The purpose of a locking adapter is to reject or grant read or write access to an object. This means that an application can set a write-lock on a particular value or set of values so that nobody else is able to write any of those values.

What happens when a read request is sent to a read-locked LockingAdaptor will depend on the application. For example, the process may be stopped until it is able to read the value, an exception may occur or a null value be returned. Because it is usual to have long locking times in this type of interactive applications it is probably not a good idea to stop the process.

By sharing the lock between many locking adapters it is possible to lock all values in an object or group of objects. The lock itself is a special form of SharedValueHolder and has all the behaviour of an ValueHolder, so it is possible to add awareness to the lock itself. This gives an easy and uniform way to obtain information of who currently has the lock, in the same way that one can get information on who has changed other values.

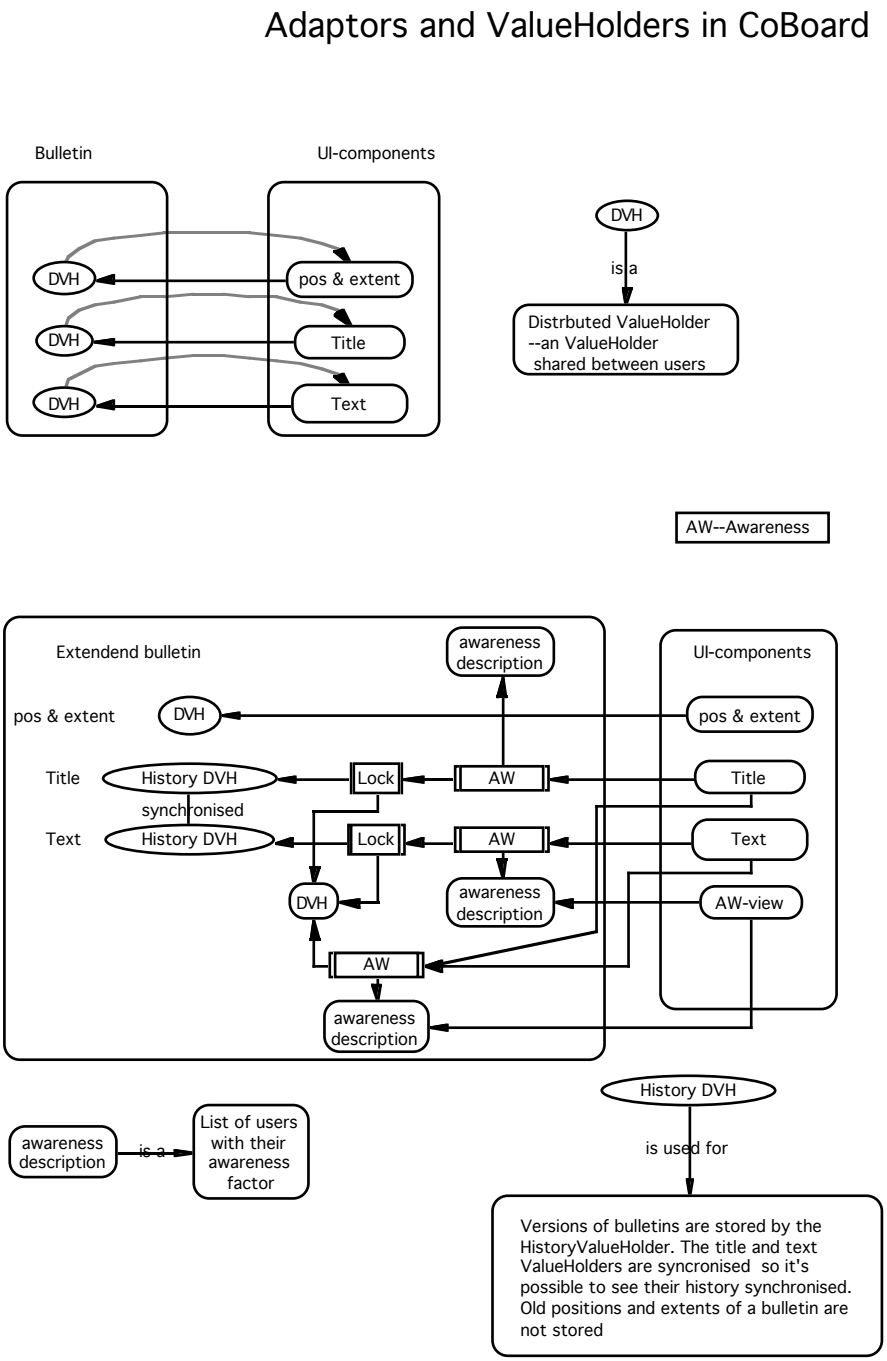


Figure 12.5. Adapters in CoBoard

12.10 Conclusions

In this chapter we have discussed the development of a series of prototypes and their use of the sharing of information. Each approach to sharing has different implications for, and impact on, the underlying services to be provided by the

SOS and SIS. In multi-user systems we must be able to handle distributed shared information. This implies the need for a distributed infrastructure; in our case, we have developed a distribution package MultiGossip. MultiGossip uses the concept of a shadow or proxy object to act as the local representative of the remote object. A number of the features of MultiGossip, adapters, events, value holders, etc., meet some of the requirements for the SOS and SIS.

In our work we have identified several specific problems and questions which we try to solve and respectively answer. Here is a somewhat incomplete list of problems we will try to solve in the near future of our SOS/SIS work:

- Spatial models: How does SIS handle spatial information? Which parts are handled by both? Are there spatial objects that partly belong to both SOS and SIS?
- Domicile of an Object: Is the object residing in SOS, in SIS or in both? How to move objects between the different stores?
- Subscriptions: The SOS idea of subscriptions can be used in SIS as well. The difference is that in SIS one probably also must supply subscriptions on interaction elements and changes of different users' visualisation. How can we do that in a consistent and well defined way?

We have also explored techniques for building distributed applications and presented an extension of the popular MVC paradigm specifically aimed at distribution. We identified a number of problems presented by the need to support cooperative applications, a set of necessary services and a discussion of objects in the SOS and SIS. Finally, we presented a discussion of the CWIB (Cooperative Work Interface Builder) and the Co-Board bulletin board application. This, we believe, illustrates the considerable power of conceptually simple techniques such as shared value holders, adapters, events and subscriptions.

Chapter 13

The Collaborative Desktop - Experience from Designing and Building an Environment for CSCW

Yngve Sundblad and Konrad Tollmar

KTH

Based on our experience from developing tools for, and prototypes of, a CSCW environment -- the Collaborative Desktop -- we examine models for CSCW and elaborate on some building blocks for such environments. Designing and building efficient environments for CSCW includes abstraction into understandable building blocks for future support of new tools and extensions. We examine and separate the support into the fundamental components which emerge from considerations of cooperative work.

We classify the need for support into three categories of services: the user interface toolkit, the collaborative services for tool management, messages and real-time communication, and a shared object service that handles the shared object space. Real life cooperative work experience has shown that CSCW systems based on "a shared information space" are attractive as both time and location independent but need to be augmented by direct user communication.

This work is our first attempt to realise aspects of the COMIC Shared Object Service into a CSCW environment.

13.1 Introduction

The purpose was originally to develop design ideas for a CSCW environment based on broadband communication networks. We soon found that support for cooperation mediated through shared objects is crucial, not instead of direct communication but as an equally important complement.

Our work is based on studies of knowledge work, design and software engineering. We are aware that this spans a limited range of work but as studies have shown this form of "free work" is becoming more and more common into several work settings [Kling & Iacono 1985]. Another argument for using these forms of free work as a basis is to get rid of structured models of cooperation that obviously lack in reality by assuming a "perfect world" that seldom occurs in more serious situations [Bannon & Schmidt 1992].

In reports, videos and prototypes we have analysed collaboration in design and specified some general functional requirements and interface design principles for CSCW support in a distributed design environment and distributed software engineering [Marmolin, Sundblad & Pehrson 1991] [Ahlström, Marmolin & Marmolin 1993] [Ropa & Ahlström 1992].

We could summarise our earlier results in terms of a set of general requirements that a CSCW environment should support: informal collaboration, sharing and record keeping of information, presentation of ideas, sharing of background knowledge, strategies reducing the need for co-working and tools that support co-working.

This chapter introduces our basic CSCW environment based on our research and experience in CSCW in section 13.2. As will be argued, building such a system is hard since many of the needed building blocks are not designed for support CSCW systems.

Based on our experience from developing tools, for and prototypes of a CSCW environment, the Collaborative Desktop, we examine models for CSCW and elaborate on some building blocks for such environments. Designing and building efficient environments for CSCW includes abstraction into understandable building blocks for future support of new tools and extensions. We examine and separate the support into the fundamental components which emerge from considerations of cooperative work.

We classify the need for support into three categories of services: The user interface toolkit, the collaborative services for tool management, messages and real-time communication, and a shared object service that handles the shared object space. This is based on real life cooperative work experience that CSCW systems based on "a shared information space" are attractive as both time and location independent but need to be augmented by direct user communication.

Within COMIC several of the underlying mechanisms and techniques for realising CSCW systems have been further developed. This joint work has given us an model for an environment for CSCW as well as building blocks to build a working prototype of the Collaborative Desktop, CoDesk. Sections 13.3, 13.4 and 13.5 describe the CoDesk design and implementation of the COMIC Shared Object Server (SOS).

This work is a first attempt to realise parts of the COMIC Shared Object Service into a CSCW environment.

13.1.1 But...

Building CSCW applications and systems is difficult and special development tool support is needed. A number of CSCW toolkits and environments have emerged for development of multi-user interfaces, e.g. Rendezvous [Patterson et al. 1990], shared editing systems, e.g. GroupKit [Greenberg & Roseman 1992] and conference systems, e.g. MMConf [Forsdick 1985]. Criticism by e.g. [Trevor, Rodden and Blair 1993] argues that unfortunately many of the earlier CSCW environments:

> " ... provide little or no support for representing the cooperation taking place."

They and we argue that the most obvious drawback is that many earlier environments are closed applications rather than open platforms. Clearly the

underlying technology is not mature enough and the architecture in distributed systems and database management systems (DBMS) maps badly onto the concepts needed in CSCW. Note that we do not mean that CSCW support should not be built from existing technology but that special attention must be put on supporting cooperation concepts.

Thus there is a need for basic CSCW environments that could be used for basic cooperative work and that support future specialisation and incorporation of new tools.

13.2 Environments for CSCW

In the still novel art of designing CSCW systems most of us look for general overall solutions. We will first examine earlier work that has tried to capture the "nature of work" and argue why we think that many of those models lacks in realism and are more or less useless in real work situations and hence give as an alternative our approach which is a non-model environment.

13.2.1 Models of and for CSCW

Many models for CSCW tend to be goal oriented. Most include some conception of an activity that has some goal. The more specific the support, the more specialised and narrow the goals. [Trevor, Rodden and Blair 1993] have classified the different models for cooperation into three classes: procedural models, activity models and frameworks.

With procedural models one tries to model and capture procedures that are intended to happen while performing a certain task in, e.g., an office environment. The models concern control and the user carries out a given task. Examples of these kind of systems are the Coordinator [Winograd 1992] and DOMINO [Kreifelts 1992].

To give more flexibility activity models have been introduced, e.g. the Amigo Activity Model [Pankoke-Babatz, et al. 1986], that focus on what and how the work is done to better and non-statically describe cooperative work.

Frameworks are, in the definition by Trevor & al, "the most general form of cooperative environment" and intended to go one step further than activity models by focusing on the coordination of activities in groups or teams without a specific application or domain in mind.

In reality, work is not well structured or defined, people more often do the unexpected than the planned [Suchman 1983] to perform a task. Several systems focus on coordination while cooperation in work is often mediated through the material, the documents or the notes. A Model of cooperation based on messages seems too limp, cooperation based on sharing seems equally important.

We therefore wish to add another class of CSCW models to the previous list – CSCW environments. Environments do not implicitly model cooperative work.

Rather they model the environment for work, as well as cooperative work. In those environment users could find support for CSCW with different mechanisms and tools. But also, as a natural part of the environment, building blocks are needed to be able to "live" in the environment, to extend it and rebuild it.

A number of tailorable systems have been built with the intention to give the users "a model of control and access", e.g., OVAL [Malone, Lai & Fry 1992]. The general idea in OVAL is to find a small but substantial set of building blocks that even a casual computer user could use in her own way to create and modify her own or someone else's application in the way she likes.

A second class of tailorable systems is now emerging, i.e. growable systems that have the potential of scaling. Small prototypes could be developed and later when both the organisation and the technology become more mature be integrated.

13.2.2 The Collaborative Desktop as an environment for CSCW

The Collaborative Desktop (CoDesk) is an attempt to make collaboration a natural part of the daily use of a computer. Our way of achieving this is to put the user in the centre of the computer in a similar way that applications and documents are defined and visualised in the desktop metaphor.

Our philosophy is neither to introduce a "new model" nor a generic tailorable toolkit. We have developed CoDesk from something that we know works: the desktop metaphor that, despite some pros and cons, has made daily computing a lot easier and error tolerant.

CoDesk is a basic environment for CSCW. We have extended the traditional desktop metaphor with a few new objects that enable cooperative work. Without limitation to a specific model of cooperation each user could tailor or form her desktop to their individual need for cooperation and communication. In CoDesk it should be as easy to look for your colleagues as for shared or individual working material. Central in CoDesk is support for groups or teams to form cooperative settings.

Primarily CoDesk provides mechanisms that extend the network from a computer network to also be a user network by integrating the essence in communication and collaboration via different tools and media.

Before going into more detail about the CoDesk environment we want to briefly describe two of the fundamental artifacts that are being used, the KnowledgeNet and the tool approach.

13.2.3 The KnowledgeNet vision

One important need in modern working life is efficient handling of information in a society that produces so much information that traditional text-based and TV-based media are insufficient. The need to handle the "information overflow" has been characterised as a change in the social paradigm of society [Kumon S., 1992]

and different visionary computer based solutions have been suggested [Bullen and Bennett, 1990] and [Engelbart 1990]. These solutions are all focused on the management of published information in global and open but personalised libraries. The KnowledgeNet is a complementary vision, based on sharing and integration of unpublished private knowledge.

The KnowledgeNet [Marmolin 1991b] is designed to support a vision of the social work situation in which collaboration among peers can take place by sharing and integration of knowledge. The information overload in the society of today is often handled by using other people as references rather than excessive reading of documents [Kedziersky 1988]. With the KnowledgeNet we aim to support this process by shared knowledge bases of experts accessible by CSCW tools and make undocumented knowledge public in the same way as libraries make documented knowledge public.

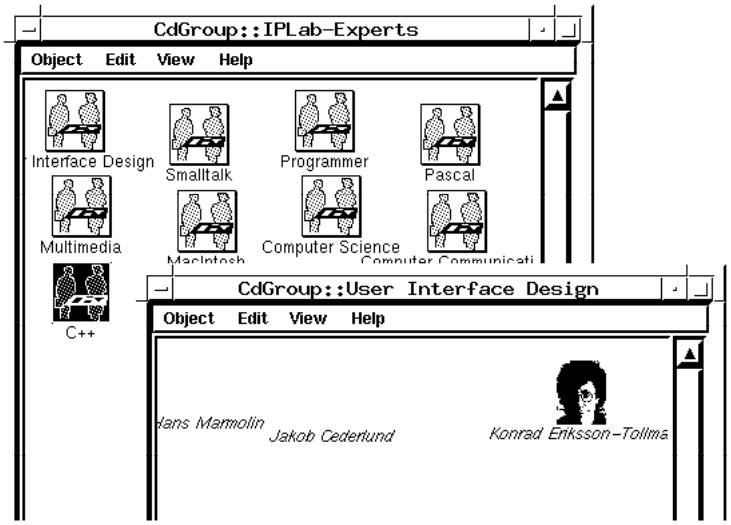


Figure 13.1 - Users informal knowledge forms competence groups.

There are many ways in which undocumented knowledge can be shared in a distributed environment. Firstly, knowledge can be broadcast in the form of lectures, announcements etc. Secondly, knowledge can be obtained by explicitly asking others for information and advice. Thirdly, knowledge can be implicitly transferred during meetings.

Fourthly, a database about “who-knows-what” can be made accessible and maintainable by the participants, Figure 13.1. This information space is “peopled” with the originator of the information visible and accessible. Note that one of The KnowledgeNet main characteristic is to be user controlled, i.e. the members themselves decide if , e.g., a document should be made public or, e.g., define and maintain their own expertise areas.

13.2.4 A tool approach

CoDesk strives to support the KnowledgeNet vision by providing all users with tools for a seamless integration of synchronous and asynchronous modes of interaction, for example to enable social ad-hoc communication and let the user toggle between activities as in real life. We think that one can use CoDesk both to talk and work as a complement of more formal and planned work processes.

The tool-oriented approach, as in [Sundblad et al. 1987], aims at designing a user controlled environment that allows the users to do what they want, without limitations and assumptions imposed by the system.

Usually the tool perspective focuses on individual use that one might find contradictory to cooperative work. But as other researchers suggest [Greenberg 1991] user control is a key factor for usability, certainly for CSCW systems. With a tool-oriented approach the users can apply and develop individual original skills that will form the core as the basic resource in cooperative work teams.

So instead of designing groupware based on some analysis of the design task or collaboration task to be fulfilled, we propose, like other researchers, the design of generic collaborative tools [Moran & Andersson 1990] [Bannon & Robinson 1991].

13.2.5 Members, groups and rooms

The most central type of object in CoDesk is the individual person or member, represented both as icons and as forms (e.g. “cards”) with attributes, including name, communication lists and KnowledgeNet who-knows-what information.

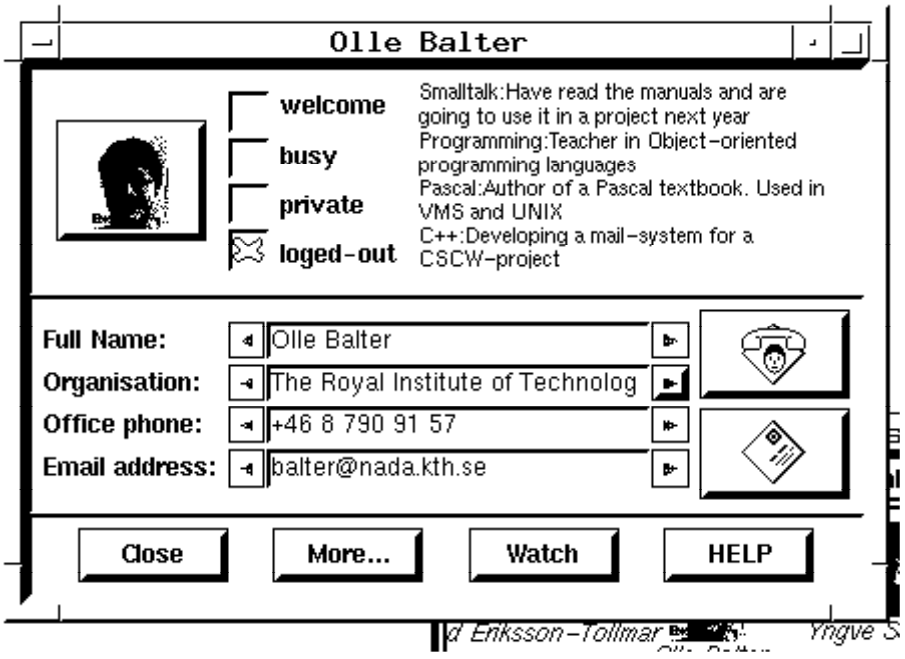


Figure 13.2 - Extended User Info with KnowledgeNet Data and access to direct communication.

Groups are simple collections of members. Each member is connected to a key group that can be viewed as the default group with which that member is interconnected.

We, as other researchers [Borning & Travers 1991], have chosen a room metaphor, where rooms are used to represent a collaborating group or a specific action. The rooms are additions to the groups and should been seen as dynamic cooperative settings. Rooms are familiar environments for cooperation and work. Rooms are where you meet people, do your work, read a paper etc. etc. For movement and navigation in the rooms the desktop metaphor is used through pictorial representation, the graphical user interface, and search-and-retrieve tools. Note that rooms are not only for sharing but also for individual use like a private mail list.

We do also explore the role of rooms in supporting "social browsing", as in Cruiser [Root 1988], by "group awareness" mechanisms. The user can set allowed “disturbance level” of group members, in the same room or making a "random walk" visiting a couple of rooms. The most common way to communicate with some members will be to install a common room with some tools and working material, e.g. documents, specific for that group. To support temporary connections with other group members a temporary room could (automatically) be installed by for example a direct phone call to an user.

13.2.6 Documents, Tools and Folders

As found by [Reder and Schwab 1990] work behaviour is characterised by multitasking, and many activities and interactions are structured into communication chains that criss-cross each other. This means that tools for collaboration should allow and support many nagging collaborative activities at the same time. A user can jump from one activity to another, have “sleeping” activities that will be continued later on, etc. The ability to adopt different kinds of tools has been proven [Grudin 1988] to be a main key in successful CSCW systems and has therefore also been one of our major goals. We believe that our architecture makes it possible to integrate and use a large amount of ordinary single user tools into the Collaborative Desktop.

Common tasks for which collaboration through the computer is very suitable include writing text, designing graphics, sound or video together. Here the collaboration is mediated through the “material”, documents, we work with. As stated in [Bannon, L. & Schmidt, K. 1992] to design CSCW system from the viewpoint of a common information space could be very valuable and useful.

The folder object gives a simple and convenient container for sorting and organising documents. Folders are collections of documents and/or folders (recursively), which gives a hierarchy similar to groups

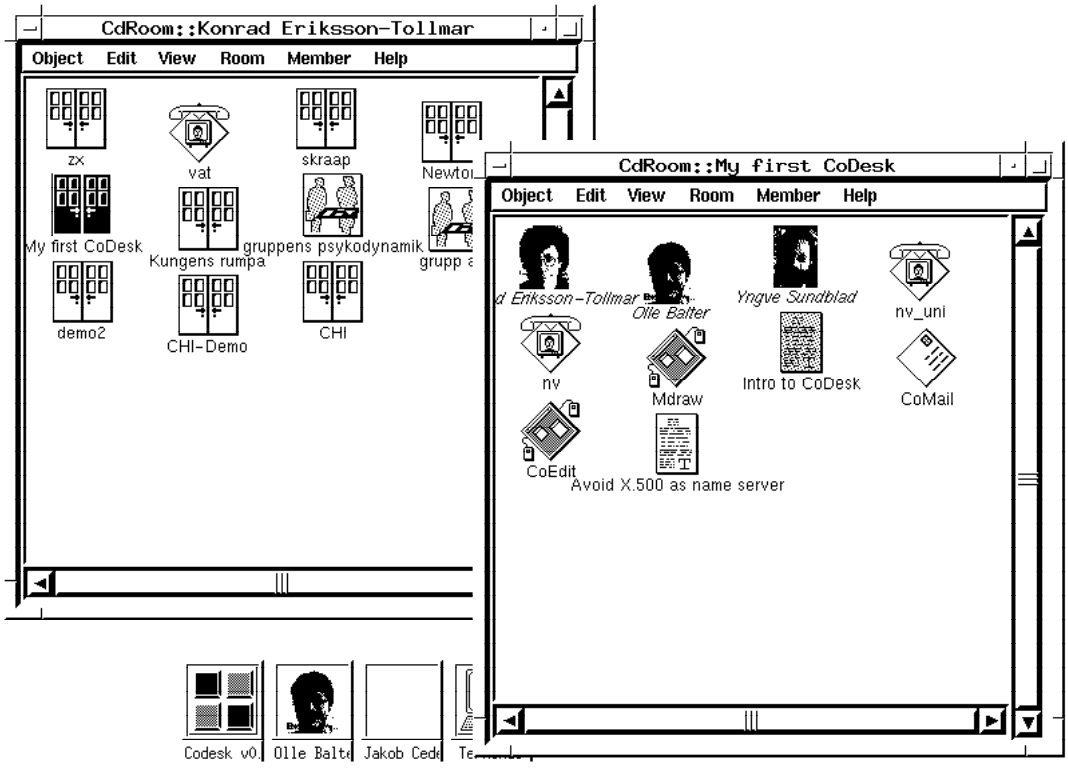


Figure 13.3 - CoDesk - An environment for CSCW

Scenarios of use of the Collaborative Desktop.

To exemplify the use of the CoDesk environment take a look at Figure 13.3. It's a screen shoot of a simple CoDesk environment. In the figure you will see a room with my name, that's my "home". Here I have a couple of different rooms that I share with some colleagues. One of the rooms is: "My first CoDesk". In that room you will find those who share that room, the tools they have chosen to use and two different documents that are in progress. One of them, "Intro to CoDesk", is slightly greyed out indicating that it's currently used.

There are basically two different ways of forming such a room, either directly or indirectly. Both normally start with one of the members creating a room on her desk, with the members and some tools inside the room (more could be added later on).

The direct way to share this room is then to use one of the tools in a conference session in that room. The room will then be available to those that participate in that conference session and a participant could, when then session ends, decide to keep a copy (or reference rather) of that room on her desk.

The indirect way is started up in the same manner, create and place objects inside a room, but instead of starting a conference session there is a option of distribute a key to that room. The key will be wrapped into an ordinary email and could be used by participants to get a copy of that room into their desk.

13.2.7 Building and Extending the Collaborative Desktop

Providing efficient support for CSCW system includes abstractions into understandable building blocks rather than considering the problem of CSCW as being essentially a user interface or database or distributed system issue. From here we will examine and separate the support into the more fundamental components that emerge from considerations of cooperative work.

CSCW systems based on "a shared information space" have gained a growing attention. Such CSCW systems are attractive as both time and location independent but from our earlier studies we know that they need to be augmented by direct user communication.

We classify the support into three different categories of services:

- The user interface toolkit
- The collaborative services for tool management, messages and real-time communication
- A shared object service that handle the shared object (information) space.

The primary method to extend the Collaborative Desktop is to build tools. Tools in CoDesk could utilise the support in the building blocks by essentially three means. Primarily are all tools are started using scripts. Those scripts can easily be localised, extended or replaced to fit specific needs. Secondly CoDesk has a well defined RPC (Remote Procedure Call) interface accessible for external tools. In a mailer one can search for addresses and in a phone-tool one can look up a telephone number given a name.

Additionally the Collaborative Desktop is distributed with most of its source so more advanced changes could be done by derivations from documented and well defined classes.

We have developed some tools to validate these ideas and test if different development systems are to be recommend. Examples are a Smalltalk based mailer, a text editor using FrameMaker's developer kit, a "collaboration aware" draw editor based on Isis (see below) and a generic "shared window" tool to encapsulate standard single-user tools to become "co-editors". We have also integrated available videoconference tools and some single-user applications.

13.3 The Collaborative Desktop User Interface

In general there are two classes of interfaces for CSCW, collaboration aware applications and collaboration transparent shared applications, a categorisation from [Lauwers & Lantz 1990]. Usually collaboration aware applications share some underlying objects presented by different user views and interaction of the application objects. Collaboration transparent systems, on the other hand, often use some standard single-user application and at some level tap the users' event streams and branch that through some protocol back to the application. Both of

these sorts of CSCW applications are useful and needed, since converting all applications of interest in cooperative use to be collaboration aware is unfeasible.

Recently a number of similar applications and systems have been developed in which one more specifically tries to solve the problems in handling sharing of objects with mechanisms that support the interaction between the participants [Ishii & Kobayashi 1992].

13.3.1 Basic principles

One of the key principles in CoDesk is to provide an easy-to-use environment. In general the CoDesk user interface could be described as a direct manipulative graphical environment where we have defined a set of graphical objects, represented as icons, for CoDesk principal objects: *Member, Group, Room, Document, Folder and Tool*. These graphical objects can be manipulated with drag-and-drop actions by suitable tools. Examples are dropping an user (icon) onto the telephone (icon) to connect a phonecall to him (see Figure 13.4) or dropping a tool into a room to install that tool to be used and shared by the team in the room.

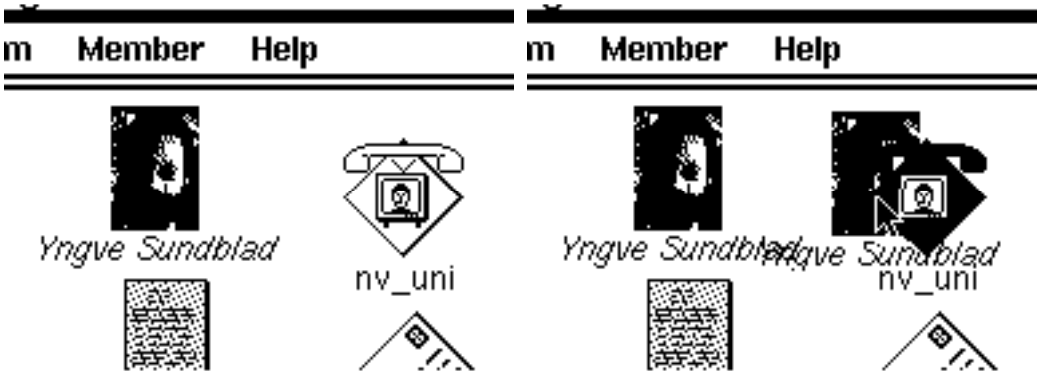


Figure 13.4 - Drag and drop operations in the GUI

13.3.2 Views

The window system is the first thing an user sees when entering the most commonly used GUI systems, e.g., Mac, Windows, Open Look and Motif. Most, not all, of these systems also define a Desktop layer, with some desktop tools. A system like CoDesk extends this with an additional layer, the network layer. That layer gives access to remote and shared data, communication and access control. This puts a cognitive load on the user that needs to be reduced by a simple and easy to learn Desktop layer.

One typical problem encountered is that multi-user interfaces often provide unexpected access to material that should not be accessible. We try to avoid compromised security by make it clear what is shared and available.

Different views of the same data can personalise and increase effectiveness of multi-user interfaces but can lead to poor awareness of the state in the data and radically decrease gained benefits and reduce the understanding of the system. Most views in CoDesk have at least three different representations, as an Icon in a standard browser and as an information dialogue, both with a colour key indicating the current access state.

These problems highlight the tricky balance between the demand of reducing the complexity in the system and the need for efficient representations. Visual clues within the views take care of some of the problems. It is very important to make clear how accessible the data is, from free and unlocked to unavailable for any kind of change.

It is also important that changes propagate immediately to all views to give a strong feedback of cooperation. In general the minimum requirement on CoDesk is to function properly, without deadlocks and race conditions, in concurrent tasks for distributed operations where more than one task performs user interface operations at the same time. Different input and output streams are handled in separate threads by default. As a consequence new kinds of objects need to handle locking mechanisms to ensure safe access to data. We assume that concurrency is a part of interface definitions; so CoDesk objects do not by themselves define that control. We provide mechanisms leaving an application programmer to define the form and granularity of access control.

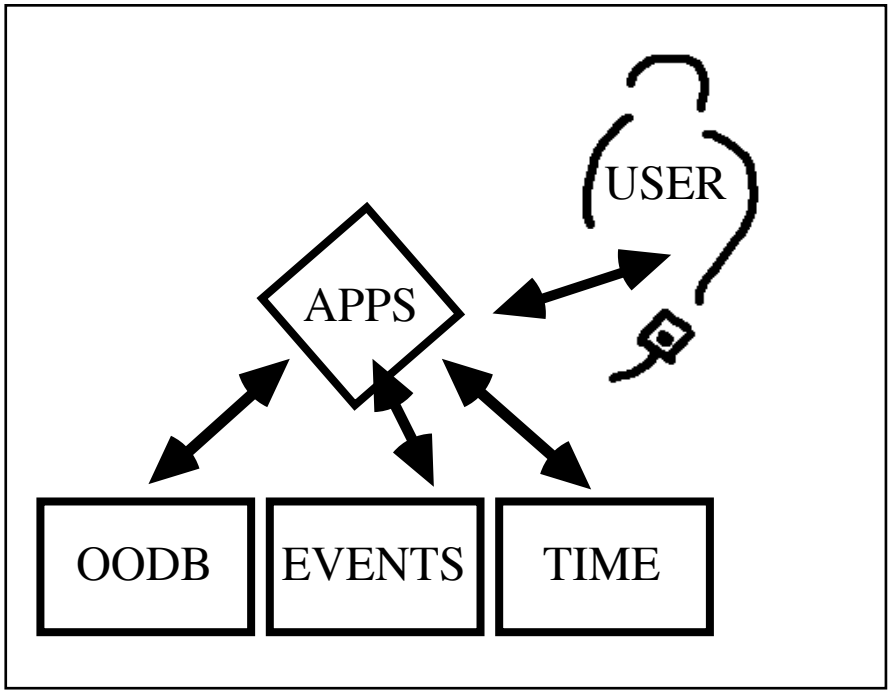


Figure 13.5 - Multiple input streams

13.3.3 Awareness

Studies [Marmolin et al. 1993] [Kraut, Egido & Galegher 1990] of work have shown that casual interactions are of utmost importance when considering how work is done. Regular interaction through non-planned meetings etc., gives the rich possibilities for exchange of ideas and conversation that a good workplace could offer. Therefore one of the key characteristics of CoDesk is to support "group awareness".

This cooperation awareness and "social browsing" is provided in the graphical user interface by different forms of highlighting objects. So far we have defined five generic forms of awareness: open, locked, available, notified and, as default, passive.

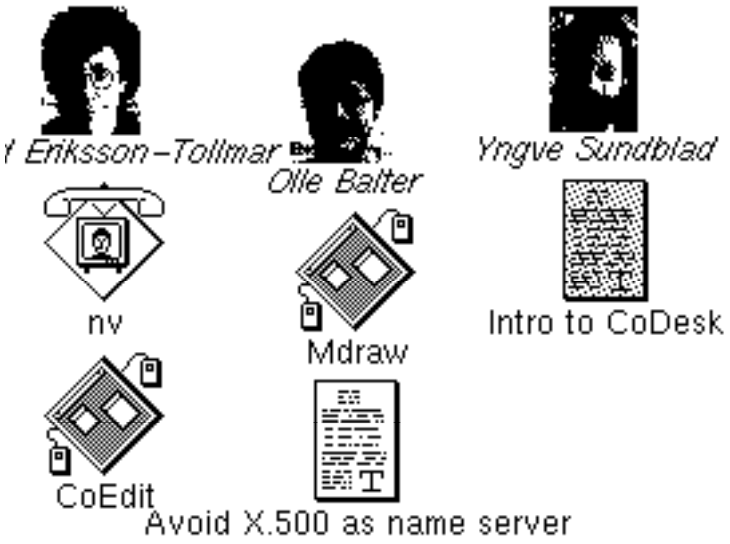


Figure 13.6 - Awareness in the GUI, welcome resp. logout member, open resp normal document.

An active open object indicates that it is used. The resources that are under use and locked are actively locked, e.g. an active user who is busy or occupied. By making an object notified we can trigger attention to it (see PATM attribute below). A notified object could be either public for all or directed to a selected colleague. A notified object usually expires after a certain time and becomes a normal passive object.

13.3.4 Principal classes

The basic structure is based on an object oriented view where we develop the graphical user-interface through interactive objects. We have been using InterViews [Linton et al. 1992] for this. Like other modern user-interfaces we make a distinction between data objects and view objects.

The basic graphical components define the user interface to the subjects. The Subject corresponds to the abstract information comprising the application

without any information of how it will be represented on the display. The information about how to display and interact with an instance of Subject is contained in the interactive object.

Our interaction model strives to support three basic direct manipulation operations:

- Select operation, e.g. mouse click.
- Action operation, e.g. double click.
- Select and drag operation, e.g. drag-and-drop or move operation.

These are the basic operations that every interface object understands. By combining the basic operations we can have composite operations like e.g. multiple selection drag-and-drop.

Four different classes contain the graphical components, `Browser`, `Editor`, `Info` and `Icon`, each representing different views of the abstract `Subject` class. A `Browser` is an abstract class for all kinds of graphical containers while an `Editor` could be used to derive a specific editor for an object type. `Info` is an abstract class for displaying properties and attributes of a particular object. `Icon` is an abstract class for icons containing the drag-and-drop functionality.

To make the interface consistent we will need some restrictions on the basic operations. These restrictions are mainly properties of the `Subject` such as, e.g. a drop operation indicates different visual feedback of the drop target dependent on the target state. Different kinds of states could simply be visualised by a generic update mechanism embedded into all graphical components.

13.4 The Collaborative Desktop Collaboration Services - COS

A major goal with CoDesk, as stated before, is to support a smooth mixed mode of asynchronous and synchronous cooperation. While some claim that: “Communication is just another way for performing operations on objects” we have marked a separation between our shared object services and the collaborative services where the latter focuses mainly on communicational aspects of sharing and cooperation.

Even if asynchronous computer based conference systems have been around for a while [Palme 1984], innovations for message based cooperative applications still arise. In Oval [Malone 1992] messages are extended from a passive to dynamic medium to also include behaviour, e.g., a message contains instructions that could be executed by the receiver. We believe that forthcoming techniques in distributed object-oriented system will further push for object based "semi-asynchronous" CSCW systems.

Originally real-time conference systems were divided into co-located and remote systems. Examples are Xerox’ Colab system and earlier versions of MMConf [Forsdick 1985]. More recent research has blurred this distinction. Later

work with real-time conference systems like Rendezvous [Patterson et al. 1990] has pointed out a similar architecture based on a shared conference session service, where e.g. a request to form a new conference session and add a new member is posted through some kind of Remote Procedure Calls (RPC) to a centralised conference server managing the current sessions.

[Roseman and Greenberg 1993] have with their work on GroupKit, a toolkit for "real-time work", listed a couple of human-centred as well as programmer-centred requirements for a real-time conference toolkit. Important observations are that such a toolkit should:

- Support alternative methods for joining a meeting, including informal meeting as well as "closed" meetings,
- Persistent sessions
- Integrate other forms of communication.

The most well known effort in distributed object-oriented environments is probably the Object Management Group (OMG) – Object Management Architecture (OMA) – Object Request Broker (ORB), a mechanism that provides transparency of object location, activation and communication. OMG strives for a concrete standardisation of ORB into the Common Object Request Broker Architecture (CORBA) [OMG 91]. A number of significant services in CORBA are of direct relevance to CoDesk collaborative and shared object services.

We are using a "predecessor" of CORBA, ISIS/C++, developed partly in-house in association with the ISIS project [Birman & Cooper 1991]. ISIS/C++ gives us, as a main feature, alternatives between a purely replicated and a centralised architecture with the possibility to design a decentralised distributed computing environment.

13.4.1 COS Design

Given a toolkit as rich as ISIS/C++ it will become much more simpler to implement CoDesk collaborative services based on good design. Here we describe some design features and considerations.

Events

The CoDesk collaborative services should enable all users to be aware of the actions of other users . The mechanism used to provide this form of awareness within the services is events. When any action takes place on an object, that is of possible interest for other objects, events are generated. The collaborative services provide facilities to allow events to be defined, created, related to other users and handled in appropriate ways. Users and objects can declare interest in certain events or types of events by registering a subscription.

Within the CoDesk system events belong to a special class of system objects. Based on the information held in the event objects, the service client can calculate awareness factors between objects and thence between users (of those objects).

The Subscription Handler is central to the event services that support users' (and objects') awareness of each others'' actions.

Mechanisms are needed to handle a subscriber's

- Immediate information about changes in an object
- Information about changes in an object when accessing it
- Immediate information about anyone taking interest in (access to, or even just pointing at) an object
- Information, when accessing an object, that someone else has accessed it
- Information about interests of other users

These services are in fact "shared" between the collaboration services and the shared object services to provide "live" awareness as well as supporting awareness on shared objects.

Here we might argue that some of the services overlap, and yes they do. System complexity and performance issues will at the end make the final decision on which ways to go.

Messages

Messages are mainly directed towards a group of members. CoDesk messages could carry instructions that are performed on delivery (if wanted). The delivery form is store-and-forward, allowing messages to be stored for delivery whenever the recipient is ready to receive them. Since a mailer in CoDesk is regarded as an external tool we could not rely on such support.

Tools

Since *tools* are key objects in CoDesk, support for starting tools and monitoring performance and error messages is a central part of CoDesk. With ISIS/C++ we can supervise other processes on the local machine as well as on other machines. We can trace load balance and monitor crashed tools.

Conference sessions

The basic conference session protocol is simple. Let us imagine a situation where a CoDesk member tries to establish a conference session between some CoDesk members using some tools.

The origin is either a shared or a temporarily created *room* containing some tools, documents and the members. The caller sends a `call` to all the members of that group. If the number of positive replies is at least one an instance of the group for this conference session is created. Future exchange of unique resources and arguments for the tools are exchanged with broadcasts within the group. When the group is established the tools are executed at each site

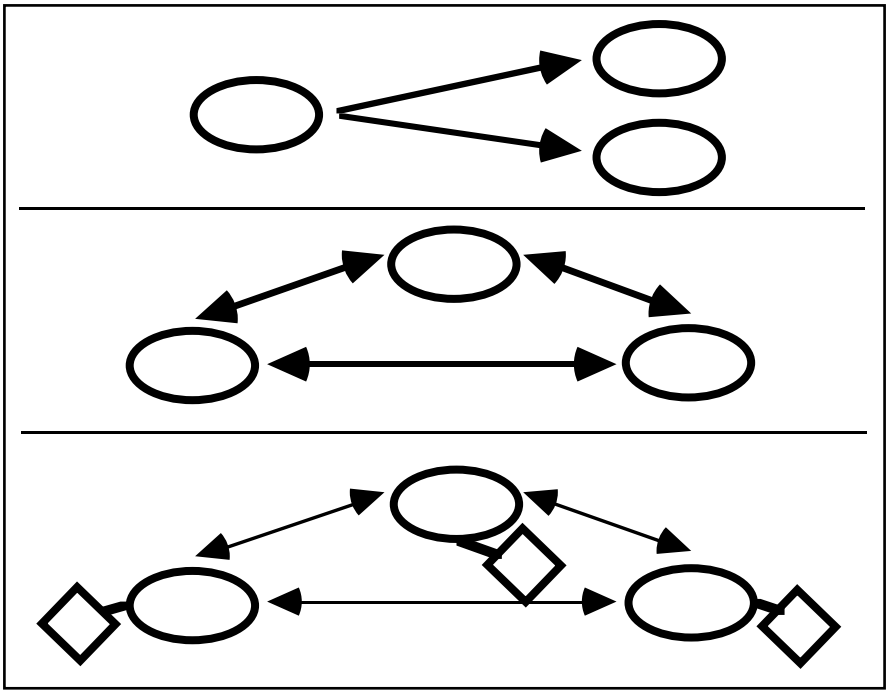


Figure 13.7 - Conference session diagram

13.4.2 Implementation of COS

In this subsection we will describe how we use ISIS++ to implement CoDesk - COS.

ISIS/C++

ISIS/C++ [Hagsand, Herzog, Birman & Cooper 1991] is an object oriented interface to ISIS written in C++. The key idea is that objects can be grouped into groups onto which requests could be sent for a particular service. Of specific interest for CSCW system is that the object groups are very dynamic, adding members and recovery from crashes are basic functionality in ISIS/C++. ISIS has as main goal to make distributed computing reliable.

ISIS/C++ is built on the client-server model. There is a set of server objects implementing services and a set of client objects accessing these services. A client is an object which is requesting a service of another, possibly remote, object. A server is an object giving some service, defined by an interface description. Note that sometimes a distributed application can not be constructed as a pure client-server model. For example, a simple distributed drawing editor is both a server and a client because an editor needs to give services and request services. If a user draws a line, this line is distributed to the other editors. The request is “add-line”, so the client at every site is requesting for the service “add-line” and then the

server at every site is drawing this line. This is not a problem for ISIS/C++ since both clients and servers could share the same address space.

An interface description defines a server interface which consists of a set of entries. The server interface is used by a stub-compiler to generate communication stubs. Each entry corresponds to a method supported by the service. When a client is requesting an entry the corresponding method is invoked. Service interfaces are specified by an Interface Description Language.

Support for reliable distributed systems with ISIS/C++

Our concepts of a distributed system are based on object groups. The `Baseobject` implements the singleton RPC-like communication methods:

- Call - Synchronous request/reply
- Bcast - Non-blocking announcement
- Request / Collect - Asynchronous request / reply

The classes `Objectset` and `Group` refine the communication methods to multicast and reliable multicast.

The asynchronous event notifying mechanism is implemented through objects. These objects implement event handling methods and are registered with notification handlers. `Basehandler` implements a functionality orthogonal to the behaviour of the `Baseobject` class. Instead of using compositional techniques, which would force us to introduce multiple inheritance in the core design, we added the event handling mechanism as a general property of objects in the class tree.

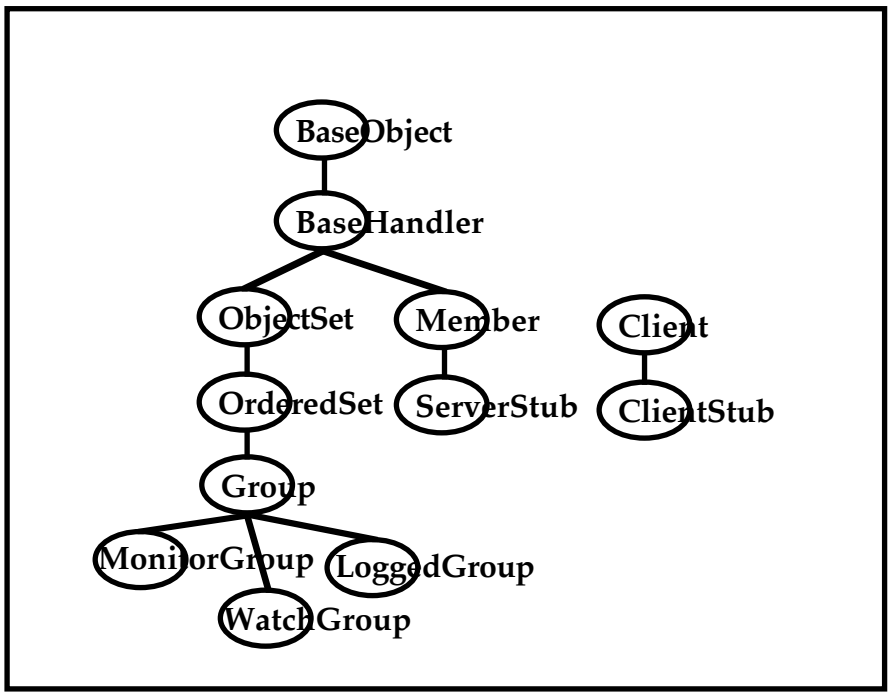


Figure 13.8 - ISIS/C++ classes

The `Group` class thus implements our notion of object groups. Figure 13.9 shows the `Group` interface. The `Group` class is further specialised for services like monitoring, group changes and state transfer. A local nameserver is also available for resolving names and addresses.

```
Interface Group : public OrderedSet {

        Group(char*);
        Group(address*);
        ~Group();

    int     join(Member*);
    int     leave(Member*);

virtual  int    call( int, int, message*);
virtual  int    bcast(int, int, message*);
virtual  int    request(int, int, message*, void*);
virtual  int    collect(int);

};
```

Figure 13.9 - ISIS/C++ Group Interface

`Member` and `Client` introduce behaviour related to server and client functionality of distributed objects.

COS - Events

Several monitoring and watching capabilities are offered by the ISIS++ system. This is done by supplying a handler object which receives and acts upon a specific event. The Basehandler class supplies virtual templates for event handling, see Figure 13.11. Monitoring or watching is invoked by calling watch/monitor methods of an indicated ISIS++ object, other kinds of object need first to be encapsulated by an ISIS++ event adapter.

Using a watch, e.g., are most often done by

i) instance a Watchgroup

ii) if it its an external, non ISIS++, object create an event adapter

iii) might join the object to a group, and last

iv) invoke the watch mechanism, see Figure 13.10.

```
Watchgroup *group = Watchgroup("/aname/agroup");
Eventobject* eo = new EventObject(AN_EVENT);
eo->append(anObj);
```

```
// might.. group->join(eo);
group->watch(eo);
```

Figure 13.10 - Example of using the Event adapter

```
interface Basehandler {
  Basehandler();
  Basehandler(address *addr);
  ~Basehandler();
  virtual void object_watch(address *);
  virtual void group_watch(address *gaddr,address *paddr,int event);
  virtual void group_monitor();     // = 0
  virtual void site_watch(site_id, int);  // = 0
  virtual void siteview_monitor(Siteview *);     // = 0
}

interface Watchgroup : virtual public Group {
public:
        Watchgroup(char *gname);
        Watchgroup(address *gaddr);
        ~Watchgroup();

        int  watch(Basehandler *obj, Baseobject*pobj, int event);
        int  watch_any(Basehandler *);
        int  watch_cancel(int);
};
```

Figure 13.11 - Basehandler och Watchgroup interface.

COS - Conference session services

As mentioned earlier an advantage of using ISIS is the ability to construct reliable distributed systems. To address that in CoDesk COS all of the services are distributed into a object-group, e.g. of conference managers. That is a distributed service for the management of conference sessions that could plausibly run on in several processes on the net. Here follow a couple of services:

- List available sessions, this will list current available conferences within a given scope, e.g., public conference at a members default group
- Find conference session given a name search through the list of conference sessions
- Start conference session, send a invitation to a number of people and upon approval create a new session

- Invite to conference session, will invite a new member to a conference
- Join conference session, this will send a request to join a conference session
- Leave conference session, this will disconnect a member from a conference session split conference session this will split a conference into two equivalent parts
- Register a conference resource, this will register a unique resource that this conference will need and use from the network.
- Unregister a conference resource, this will unregister a unique resource that a conference session have been using from the network.

The to implement this two different things need to be defined. First, and what is most natural, what data are needed. The data needed here are only a couple of list of Strings,

- For authorisation.
- Current active groups by name (the nameserver will help to look-up those groups).
- A table for network resources (e.g. IP-multicast addresses).

Secondly we need to define this service interface, this done in Figure 13.12. The functions LIST and FIND are defined to be asynchronous request (which could in another thread later pick up the answers). CREATE, JOIN and SPLIT are defined as blocking calls that need to be answered by all servers (in order) to continue. And finally LEAVE, REGISTER and UNREGISTER are handled by non-blocking announcement CAST's.

```
interface CONFSERVER =
    LIST : REQUEST ALL: list[int] RETURNS [int, message];
    FIND : REQUEST ALL: find[int;string] RETURNS [int, message];
    CREATE : CALL ALL: join[int;string] RETURNS [int, message];
    JOIN : CALL ALL: join[int;string] RETURNS [int, message];
    SPLIT : CALL ALL: split[int;string] RETURNS [int, message];
    LEAVE  : CAST: leave[int;string];
    REGISTER : CAST: register[int; string];
    UNREGISTER : CAST: unregister[int; string];
END;
```

Figure 13.12 - The COS Conference session services - Interface description

13.5 The Collaborative Desktop Shared Object Services - SOS

As stated above, cooperation mediated through a group's working material needs to be supported in a CSCW environment. A considerable amount of work in

CSCW systems has been devoted towards coordination and communication, while the need for CSCW support in time and place independent shared object needs more attention.

Several recent commercial systems extend DBMS techniques for document management and work-flow system, e.g. Lotus Notes and Documentum, with support for versioning, clustering, replication and group modelled access control mechanisms. Some OODBMS now also claim to provide supporting techniques for building CSCW system, e.g. ObjectStore.

In the CSCW research community work on shared information spaces has mainly been made in multi-user hypertext system, such as SEPIA [Haake & Wilson 1992]. These system focus on how to break up and link fine grained information objects and intradocument structures. Interdocument systems where the bodies are stored as single units have been used e.g. for cooperative authoring system, either by supporting different roles such as in Quilt [Fish et al. 1988] or by supporting annotation and reviewing of documents as in Prep [Chandhok et al. 1992].

Today's expanding "public" information system, e.g. Gopher and WWW [Berners-Lee et al. 1992], can also be seen as cooperative shared information spaces. The WWW approach is very beneficial in the production of an organisational knowledge browser. More advanced forms of organisational memory have been developed with directory service toolkits such as X.500 [Robbins & Kille 1991]. The growing interest for support of organisational context and memory will hopefully drive the understanding of large scale heterogeneous systems forward.

New standard efforts in distributed system, such as OMG, described earlier, have also defined persistent storage as an part of the distribution services. This can have a large impact on design and implementation of forthcoming CSCW system.

13.5.1 Mechanism for Sharing in SOS

The feature which most distinguishes the CoDesk shared object service from other multiuser storage systems is the focus on sharing and the provision of mechanisms which support the management of this sharing. The explicit goal is to provide an awareness of the action of others on objects within the shared object service. When an object is accessed or altered the CoDesk shared object service should inform the relevant users that the action has occurred in order to promote awareness across the service.

Naming Policies

Names allow us to identify objects, talk about them and to share them. Usually naming models for distributed systems try to overcome ambiguous and multiple names by use a hierarchical design, such as the UNIX filesystem and BIND nameserver. This is contradictory to human use of names.

Our work here is heavily influenced by research by [Benford 1992] and earlier efforts in the Internet community. We claim that names are not unique, people are aware of the ambiguous use of names and that names changes continuously to indicate a process or to be used for trading and federation of objects. Benford uses the term "Group oriented naming context".

Clearly we have to separate an object's name(s) and its identifier. While the identifier maps addresses, paths and routes to objects, names are a distinct property of shared objects, used by humans. Supporting a flexible naming manager in CoDesk SOS is very important for group identity and cooperation. With multiple names for a single object a user will become free to divide her object space into different contexts, e.g. for dividing a single work activity into several different projects. The CoDesk shared object services needs to provide mechanisms for:

- Setting, adding and changing an object's name(s)
- Resolving a name into a set of objects.

The use of common names is a key factor in group cohesion, therefore we have, as Benford suggests, developed two different mechanisms for name proposals:

- Name proposals given a new object.
- Name proposals given a new user.

One of the name proposals' most obvious use is when a user tries to merge an object to a set, finding an appropriate name in that context. Another use is for new user for whom most objects lack names. Name proposals are here given to the user for each object in that set.

Awareness and Subscriptions

Events give a mechanism for "live" awareness but awareness of the shared objects is equally important. Some objects may possess a greater "pay attention to me" attribute than others. As in the Comic SOS we suggest that every object possesses a PATM attribute and accordingly methods to get and set the value of that attribute.

An object announces an interest in other objects or events by declaring an interest to the Subscription Handler or directly to the objects of interest. The same holds for subscription to PATM attributes. A client, application or user, could declare interest in a certain PATM attribute, internally or externally, by declaring the interest to the Subscription Handler. If the subscription is an external entity then a "shadow", or proxy, object that represents the real object is used.

Queries

In a CSCW environment it is also necessary to handle queries by applications and users. There is a need to specify them as transitory (the resultant collection is discarded as soon as the view operation that brought it into being is out of scope of the surrounding transaction) or persistent (the resultant collection is saved and

updated if new objects are added which match the condition). To support the dynamic addition (and indeed, deletion if some object changes the value of their first name attribute) of objects to a collection, we need some sense of a "future" query, which can be handled by the event mechanism.

History

An important kind of query that has to be supported is on history of objects. The objects and their attributes must contain time stamps on creation (and deletion when applicable). In order to keep the granularity of time specification on a per object basis we assume that such an important change of an object that the previous state should be part of the history creates a new object with reference to its ancestor.

Given these assumptions, the SOS service should be able to provide history if it is constructed to cope with temporal clauses delimiting the range of a query to a set time period.

External references

It is also important that real physical objects can be subject to similar kinds of operations as the objects stored within the shared object service. For example, just as we can issue queries regarding stored objects, it would be nice if we could issue queries regarding real physical objects such as "Who currently has a copy of a specific book?".

In some senses, an object which contains a reference to an external object can be considered as being incomplete. Unlike a fully internal object, we do not possess all the information "at hand" to display, manipulate, indeed generally consider the object. This impacts on issues of how we display or present an incomplete object, how we query and retrieve an incomplete object, etc.

The purpose of the external reference is merely to give us a hook into the physical objects, where presently none exists. If we search for a copy of the specific book, and we can query the (incomplete) objects which reference the book, we can hope that we may find some clues as to whose office currently contains it.

13.5.2 Implementation of CoDesk SOS

We started off our first implementation of the CoDesk SOS using the X500 directory service. We have been using the X.500 directories as a distributed database. The directory is intended to support human user querying, allowing the user to find telephone and address information of organisations and other users. We will in the next chapter describe how we used X.500 and highlight some off the most valuable points in X.500.

The second approach that we are currently working on is to use an Object Oriented Database Management System (OODBMS), like e.g. ObjectStore or

ODE. There are several features needed in such an environment to realise the CoDesk SOS and as will be argued some could be done in extended services while other are much fundamental and really needed to be in the OODBMS core.

Using X.500 to realise the CoDesk SOS

The X.500 directory is:

- a database.
- intended to be very large and highly distributed.
- hierarchically structured, the entries are arranged in the form of a tree called the Directory Information Tree (DIT).

The directory holds information about various entities such as a person or an organisation. The information is held in an information object typically referred to as an entry. An entry consists of a set of one of more attributes. An attribute is represented by a type and a set of values. Some attributes are called distinguished attributes. The collection of these attributes forms the relative distinguished name (RDN). A distinguished name (DN) is the sequence of RDN's that uniquely define a node in the DIT.

We use X.500 as a general distributed database. It is used for saving data and information shared by members on an Internet domain. We have found the naming facilities very useful and rather close to our requirements. Accessing an object is done by either its DN or iterate locally over by its RDN. An object could very well have multiple RDN's that could be used in different contexts either by being used to pick a set or by setting up multiple filters, like "name=Smith & location=Stockholm" that provides a simple search mechanism.

The X.500 directory is not intentionally designed for highly dynamic data which rapidly changes. However, each object has an attribute called: lastModified. lastModified shows when a specific object was changed and by who. We have been using lastModified to poll the database for changes.

CoDesk SOS in an OODB environment

To provide the services in CoDesk Shared Object Server we need to add some functionality to the familiar functionality found in today's Object Oriented DataBase Management System.

Awareness & Subscription

To be able to provide Awareness in CoDesk SOS we need to monitor the database for some conditions. We do that by issuing a subscription via a trigger. Triggers are usually associated with objects and are of two types, once-only or perpetual. Note that this distinction is needed since a perpetual trigger can't be simulated with once-only triggers by re-activating the trigger in the trigger action. The problem with doing so is that the trigger action is executed as a transaction at some time later.

As a result the trigger will be inactive for the period between the firing of the trigger and until the activation instruction in the trigger action is executed. Triggers are specified within the class definition, e.g.

```
class A {
    some-list
    void do_something();
Public:
    Name aname;
    Status mode;
    void update(Status xmode);
Trigger:
    Perpetual subscribe() : changed(mode) ==> do_something();
}
```

Fig 13.13 - A Perpetual trigger definition for a subscription.

Queries

A major criticism of the current OODB's is their inability to pose arbitrary "join" queries and that query processing "smells" of pointer chasing. A rich set of iterators that also allows the expression of recursive queries is needed. Recursive queries are especially useful for deductive databases like the kind of search mechanism needed for the KnowledgeNet search-and-retrieve functions.

Aho & Ullman have shown that the least fixed point operator (recursion) is an essential addition. What needs to be supported is while iterating over a set or a cluster do the same over the elements that are added during the iteration.

The much more complicated issue, and so far not solved in any OODBMS, distributed nested transactions can't be, as we see it, put on top of a OODB (compare with Perpetual triggers). That will certainly limit our scope that we can reach in building efficient support for CSCW which is by it's nature distributed. So the basic architecture will remain non-distributed while it probably is feasible to set up a system with multiple servers that could utilise replication and caching techniques.

History

The means of implementing history in a OODB are most commonly done by versioning. Normally there are no pre-defined limits on the number of versions of an object and if the OODBMS supports transaction logging a simple trace can, given a specific time, bring a historical object back to the scene. Note though that versioning is an object property and not a class property. Objects belonging to the same class have different numbers of versioning.

13.6 Summary and Conclusions

From the Collaborative Desktop effort we have learnt that

- Producing a working prototype of a generic CSCW environment is, even with the best generally available tools, a very considerable effort
- Tools and components for distributed systems, interface design and implementation and distributed databases are not well designed for integration into one system
- Awareness of other users, crucial in CSCW applications, puts specific demands on distribution, interface and database tools, including making access visible rather than "transparent"
- Sharing objects and information as a way of mediating cooperation is a feature as important as the communication channels in real world applications
- Specific collaboration services and shared object services are useful modularisations for building CSCW environments

We conclude that CSCW system development tools where the cooperation taking place can be represented through collaboration and shared object services are very useful and should be elaborated further according to the principles and features described here. For gaining better experience and knowledge of the human and social factors of CSCW in real world applications it is crucial to make production of working prototypes for field testing much easier and faster. The ideas, experience and services presented here contribute towards the goal of a development environment for fast building of functional CSCW prototypes and efficient construction of full scale CSCW applications.

13.7 Acknowledgements

We are indebted to the IPLab communities for numerous discussions on the issues presented here, especially Hans Marmolin, our joint designer of CoDesk, Björn Eiderbäck and Per Hägglund, our joint developers of the Shared Object Service ideas.

13.8 References

[Ahlström, 93] Ahlström B, Marmolin H. and Marmolin T., Ongoing Evaluation Studies of Collaborative Work Within the Swedish MultiG Research Program, *Proceedings of INTERCHI'93*, Amsterdam, The Netherlands, 1993.

[Bannon, 91] Bannon, L. and Robinson, M., Questioning Representation. *ECSCW 1991. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, 1991, Kluwer, Amsterdam.

[Bannon, 92] Bannon, L. and Schmidt, K., Taking CSCW seriously, *CSCW: An international Journal,* vol. 1, no. 1, Oct. 1992, Kluwer.

[Benford, 92a] Benford S., *Group oriented naming context*, Computer Communication Laboratory, Nottingham University, 1992.

[Benford, 92b] Benford, S., *From rooms to cyberspace: models of interaction in large virtual computer spaces* (Working Paper No. NOTT 4.3). Nottingham University.

[Benford, 93] Benford S. & Mariani J. editors, The COMIC Shared Object Service, ch.10 in: *Shared space and objects, COMIC Deliverable D4.1*, Nottingham & Lancaster University.

[Berners-Lee, 92] Berners-Lee et al., World-Wide Web: The Information Universe, *ENRAP* vol. 1 No. 2, 1992.

[Birman, 91] Birman K. & Cooper R., The ISIS Project: Real Experience with a Fault Tolerant Programming System, in *Proc. Workshop on Fault Tolerant Distributed Systems*, New York, ACM Press, 1991.

[Borning, 91] Borning A. and Travers M., Two Approaches to Casual Interaction over Computer and Video Networks, In *Proceedings of CHI'91*, ACM Press, 1991.

[Bullen, 90] Bullen, C. V. and Bennett, J. L., Learning from User Experience with Groupware. In *Proceedings of ACM CSCW'90 Conference on Computer-Supported Cooperative Work*, ACM Press, 1990, 291-302.

[Bush, 45] Bush V., As we may think. *Atlantic Monthly,* 1945:176:101.108.

[Bødker, 87] Bødker S., Ehn P., Kammersgaard J., Kyng M. and Sundblad Y., A UTOPIA Experience; On Design of Powerful Computer-Based Tools for Skilled Graphic Workers, In *Computer and Democracy*, Edit: Bjerknes, Ehn, Kyng, Aldershot: Avebury

[Cool, 92] Cool, C., Fish, R. S., Kraut, R. E., & Lowery, C. M., Iterative design of video communication systems. In J. Turner & R. Kraut (Ed.), *CSCW '92*, (pp. 25-32). Toronto: ACM Press.

[Danielson, 86] Danielson, T., Pankoke-Babatz, W., Patel, A., Pays, P., Smaaland, K. and Speth, R., The Amigo Project - Advanced Group Communication Model for Computer-Based Communications Environment, in *Proceedings of the ACM CSCW'86 Conference,* Austin, Texas, ACM Press, 1986.

[Engelbart, 90] Engelbart, D. C., Knowledge-Domain Interoperability and an Open Hyperdocument System. In *Proceedings of ACM CSCW'90 Conference on Computer-Supported Cooperative Work*, Los Angeles, ACM Press, 1990, 143-156.

[Fish, 88] Fish R. S., Kraut R. E., Leland M. D. and Cohen M., Quilt: A collaborative tool for cooperative writing, in *Proceedings of conference on Office Information Systems,* Palo Alto, California, 1988.

[Forsdick, 85] Forsdick H. C., "Explorations in Real-Time Multimedia Conferencing", In *Proc. 2nd International Symposium on Computer Message System*, IFIP, September 1985.

[Greenberg, 92] Greenberg, S. and Roseman, M., GroupKit: A Groupware Toolkit for building Real-Time Conferencing Application. *Proceedings of CSCW92,* ACM Press, Toronto, 1992.

[Greenberg, 91] Greenberg, S., Personalisable Groupware: Accommodating Individual Roles and Group Differences. *ECSCW 1991. Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, Kluwer, Amsterdam, 1991.

[Grudin, 88] Grudin J., Why CSCW Application fail: Problems in the design and evaluation of Organisational Interface, In *Proceedings of CSCW'88*, Portland, ACM Press, 1988.

[Haake, 92] Haake, J. M. and B. Wilson., Supporting Collaborative Writing of Hyperdocuments in SEPIA. *Proceedings of ACM CSCW92 Conference on Computer-Supported Cooperative Work*, Toronto, 1992.

[Hagsand, 91] Hagsand, O., Herzog, H., Birman, K. and Cooper, R., Object Groups: An Approach to Reliable Distributed Programming, *Proceedings of the 3rd MultiG Workshop*, KTH, Stockholm, 1991.

[Ishii, 92] Ishii, H., and Kobayashi, M., ClearBoard: A Seamless Medium for Shared Drawing and Conversation with Eye Contact, In *Proceedings of CHI'92*, Monterey, ACM Press, 1992.

[Kedziersky, 88] Kedziersky, B., Communication and management support in system development environments. In I Greif (ed.) *Computer Supported Cooperative Work*, M Kaufman, San Mateo, 1988.

[Kling, 85] Kling R. and Iacono S., *Desktop Computerisation as the Product of Social Movement*, University of California, Irvine, 1985.

[Kraut, 90] Kraut R., Egido C. and Galegher J., Patterns of Contact and Communication in Scientific Research Collaborations, in *Intellectual Teamwork*, J.Galegher, R.Kraut and C.Egido (eds), Erlbaum, Hillsdale New Jersey, 1990.

[Kreifelts, 91] Kreifelts T., Hinrichs E., Klein K-H, Seuffert P. and Woetzel G., Experiences with the DOMINO Office Procedure System. in *Proceedings of ECSCW 1991*. Kluwer, Amsterdam, 1991.

[Kumon, 92] Kumon S., *From wealth to wisdom: A change in the social paradigm*. Proceedings of CSCW'92, Toronto, ACM Press, 1992.

[Lauwers, 90] Lauwers J. C. and Lantz K. A., Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems, In *Proceedings of CHI'90*, ACM press, 1990.

[Linton, 92] Linton M. et al., InterViews Reference Manual, Stanford University (USA), 1992.

[Malone, 92] Malone T. W., Lai K. and Fry C., Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. *Proceedings of CSCW92: ACM 1992 conference on Computer Supported Cooperative Work,* Toronto, ACM press, 1992.

[Marmolin, 91b] Marmolin H. (1991b), The KnowledgeNet, *Proceedings of the 3rd MultiG Workshop*, KTH, Stockholm.

[Marmolin, 92] Marmolin H., Sundblad Y., Tollmar K., Avatare A. and Eriksson H., The KnowledgeNet with the collaborative desktop interface. Formal demonstration *Proceedings of CSCW92,* Toronto, ACM press, 1992.

[Marmolin, 91a] Marmolin H., Sundblad Y. and Pehrson B., An Analysis of Design and Collaboration in a Distributed Environment. *Proceedings of ECSCW91,* Amsterdam, September 1991, pp 147-161.

[Medina-Mora, 92] Medina-Mora, R., Winograd, T., Flores, R. and Flores, F., The Action Workflow Approach to Workflow Management Technology. in *Proceedings of the conference on CSCW92,* Toronto, ACM press, 1992.

[Moran, 90] Moran T. and Anderson R., The Workaday World as a Paradigm for CSCW Design. In *proceedings of CSCW '90,* Los Angeles California, ACM press, 1990.

[Neuwirth, 90] Neuwirth, C. M., Kaufer, D. S., Chandhok, R. and Morris, J. H., Issues in the Design of Computer Support for Co-Authoring and Commenting. In *Proceedings of ACM CSCW'90 Conference on Computer-Supported Cooperative Work*, Los Angeles, ACM Press, 1990, 183-195.

[OMG, 91] The Common Object Request Broker: Architecture and Specification, *OMG Document Number 1991.12.1, Revision 1.1, OMG, Draft 0*, December 1991.

[Palme, 84] Palme J., 'You Have 134 Unread Mail Do You Want to Read Them Now?' in *Proceedings of the IFIP Conference on Computer Based Message Services*, Smith, H (ed) Nottingham University, 1984, North-Holland:Amsterdam.

[Patterson, 90] Patterson, J. F., Hill, R. D., Rohall, S. L. and Meeks, W. S., Rendezvous: An architecture for synchronous multi-user applications, in *Proceedings of CSCW 1990*, October 7-10, Los Angeles, Ca., ACM, 1990, pp. 317-328.

[Reder, 90] Reder S. and Schwab G., The Temporal Structure of Cooperative Activity, In *Proceedings of CSCW'90,* Los Angeles California, ACM press, 1990.

[Robbins, 91] Robbins C. J and Kille S. E *The ISO Development Environment: User's Manual, Volume 5: QUIPU*, July 1991, X-Tel Corp.

[Root, 88] Root, W. R., Design of a multi-media vehicle for social browsing. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)*, Portland, Oregon: ACM Press, 1988.

[Ropa, 92] Ropa A., and Ahlström B., The Video Viewer, Interface design example for video communication in a CSCW environment, Video presentation at CHI92, May 1992.

[Suchman, 83] Suchman, L.A., Office Procedures as Practical Action: Models of Work and System Design, *ACM Transactions on Office Information Systems*, vol. 1, no. 4, October 1983, pp. 320-328.

[Sundblad, 92] Sundblad Y., Marmolin H. & Tollmar K., *The Collaborative Desktop: Video* 12 min, IPLab, NADA, October 1992.

[Trevor, 93] Trevor, J. J., Rodden, T. and Blair, G. S., COLA: A Lightweight Platform for CSCW. in *Proceedings of ECSCW93,* Milan, 1993.

Chapter 14

The design of the Resource manager and the Trader

G. Rodriguez

UPC

This chapter shows a possible solution to integrating the Resource manager and the Trader into the SOS environment. This is done by using the computational model and the engineering model. To carry out the engineering model, we use the guidelines of the OMT method.

An initial attempt at understanding the necessary functionality to scale up these elements inside a cooperative environment is presented in this chapter.

14.1 Why do we Need the Resource Manager and the Trader?

In an environment where there is competition to use resources we need to add instruments that assure orderly access to them. These controls are provided by the Resource Manager.

In order to enforce these controls we need to apply *policies*. We define policies as the set of rules that govern access to resources or objects. But policies are not only the restrictions, they are also the way in which the authority applies these restrictions and how the authority interprets all the policies.

Anytime we want to access a resource we must observe the policies that constrain its use. The purpose of applying policies is to guarantee correct use, load sharing, no monopolisation, etc. Examples of policies that can be applied to a user request are the following: checking security access (who has access rights to the object and what kind of access is available), accounting control, load balancing, and more. In an environment where resources are shared by many users it is very important to establish restrictions on who may and who may not use some resource. These limits are established in order to protect resources from incorrect use by unauthorised users. We have named this task as checking security access.

Often, it is not enough to supervise access rights. This does not prevent an authorised user from monopolising a resource prejudicing other users as a result of their action. This situation can be avoided with accounting control which measures the usage that every user makes of resources, either to limit the utilisation or to charge for it. For instance, a system could have a shared disk which is accessed by many users. This disk has a fixed capacity which is not

infinite. We need a control mechanism to ensure that someone does not prejudice other users when they operate the disk, for example storing too much information. The accounting control could consist of charging money according to the space used, or forbidding use of the disk anyone exceeding the accounting limit.

In an environment where more than one object provides the same kind of service there is also concern for not overloading a resource. If more than one server provides the same type of service (it has exported the same interface) the management policies should provide for load balancing, and these apply in the selection of lightly loaded servers amongst those chosen by the trading service (see below) as possible candidates. The load balancing could be checked at connection stage or dynamically. The former is carried out when the connection is established between user and object (task and service). The latter could be done during a session if the service goes wrong.

These three controls are not enough to guarantee correct access to objects and resources inside a heterogeneous and distributed environment. Controlling a set of users that use an object at same time must be done without prejudice. This and others controls are beyond the scope of the Trader and the Resource Manager, and are the responsibility of other elements inside the SOS [Rodríguez, 94].

In a large distributed environment a user cannot know all the available services or the access point of each service (its interface reference). In this kind of environment the services could change their access point and service offered dynamically. Therefore, to be able to profit from the multitude of services available in a distributed environment, a means is required for finding the resource that best fits our needs. Trading is the mechanism that allows to find these resources and isolates the user from changes of name and location and even from destruction or creation of new resources. Therefore, the trading service permits users to move inside an environment that could change.

On one side, users need to find the required services, and on the other side there are resources that want to make public the service that they provide. The interaction between the two sides is ruled by the trading functionality.

Each offer published has a type of service (or interface) associated with it, and a list of attributes which defines the quality of service. All these allow the trading functionality to find the most adequate server for every request. Of course, more than one server can offer the same interface. The trading service performs a type match and tries to find the offer that best matches the quality demanded, and returns a pointer to the server that has been chosen along with the list of attributes that define its quality [Deschrevel, 92].

The characteristics of these environments constrain us to add mechanisms in order to solve these problems. These mechanisms are the Resource Manager and the Trader (inside the SOS). However, these mechanisms cannot be used when the connection is established. These mechanisms can act only when a connection is established with an object. We need to add new elements to hide all the environment changes during a session between user and object. These elements are called adapters.

14.2 Computational view

14.2.1 Requirements of the Computational Model

While existing distributed platforms offer support for some of the functionality of the Trader and Resource Manager (T/RM), they have a number of drawbacks:

- They do not have a correct representation of the organisational context. All platforms are designed as a static model but the external world which they represent is not static. The organisational context or the external world is extremely dynamic.
- The current platforms are very restrictive models. Users in these platforms are excessively controlled.
- Not all platforms have an awareness service. This is the biggest drawback of current platforms.
- Many platforms offer limited support for shared object and shared information among environments and users. This makes it difficult to work in a cooperative environment with this kind of platform. We believe that current platforms are an adaptation of non-distributed systems, and this in itself is the source of many drawbacks.
- Many distributed platforms are designed with a hierarchical schema. Cooperation between several environments can be done with a hierarchical schema, for example with a tree structure, but this structure is too static. Each entity inside the hierarchical schema is too controlled by its owner. Changes on owner entities can affect owned entities. Also, this kind of structure does not permit two entities to begin cooperative work. This is because several entities have their own rules and an entity does not want to lose its rules and independence in order to establish the communication.

Bearing these problems in mind, we can determine the following list of desired characteristics for the design of our Trader and Resource Manager (and for the design of our SOS view):

- An understandable and flexible representation of shared information and cooperative work (not static).
- A design which disjoins the object internal representation and its semantic (offered functionality, interface representation). That allows several interface representations for an object, to add new functionality, to hide implementation changes to users, and to increase the dynamism of the system.
- A complete set of awareness tools.
- An easy scalability. The system should have an easy, quick, highly adaptable structure in order to represent a new situation of organisational context e.g., interface adapters, non-hierarchical structure, etc.

- The ability to cross boundaries between entities. It must permit communication among heterogeneous environments.
- The ability to guarantee the independence of entities that collaborate in cooperative work.
- The facility to offer various levels of transparency depending on the level of transparency that we want to offer to users.

14.2.2 Components of the Computational Model

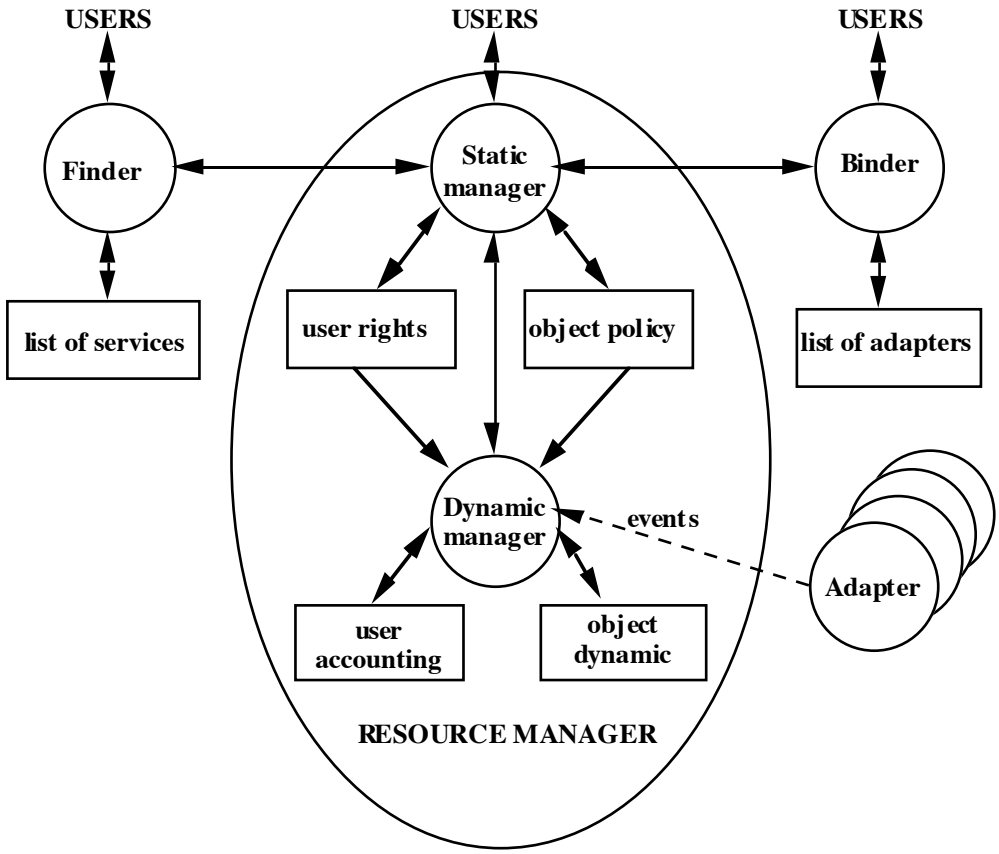


Figure 14.1 The Role of the Resource Manager

The *resource manager* is the element that controls the access rights of each user, controls the users' action on objects with *accounting adapters*, and attaches the objects to users with the load balancing policies. The resource manager cooperates with the *Trader* in order to offer a yellow page service. When a user wants to establish communication with an object he interacts with this service and it establishes the communication giving an access point. The Trader gives a list of candidates and the resource manager selects the best one depending on some policies. In order to do this correctly, the resource manager must maintain information about environment state and a set of policies.

The resource manager has the following data; characteristics and access rights of users and objects, a set of policies, and environment information that is given

by events. Events such as the following can be raised; notification of the current load factor of an object, unauthorised user's access to an object, overloading of accounting limit, etc.

In order to control users' access over the objects, the resource manager has to maintain a list of characteristics of users and objects.

We can divide this list according to users' data and objects' data. Users' data could contain things as:

- Owners: current users of each object. A user is an object owner if he has a connection with it.
- Moderator: the moderator is a user who has the capability to update dynamically the policy applied over the object. This modification must be done under administrator control. In order to control this dynamic modification, the object attributes or policy data applied on the object can be invisible, visible or editable, depending on the necessary security level. The connection between users and the object exists while the moderator exists. When he leaves his connection then all the owners must leave their connection. Who the moderator is and who inherits his rights are two important issues which depend on each object. As examples of inherit policy, the first user who establishes communication with the object is its moderator and when this user leaves their connection then the following moderator is the next user that establishes communication with the object.
- The object creator. This has the capability to destroy the object.
- Maximum load factor for each object, used to control the load balancing among objects.
- Maximum accounting factor for each object.

The system maintains information about the rights of each user on the objects. These access rights could be the following: write, read, modify, etc. Also it maintains the accounting factor by each user, if the object needs to control it.

In order to extend control over the environment the resource manager needs more than data. It also needs tools to apply this data on the objects and users. We name these tools *policies*. The policies control when the system must generate an event, under which special conditions, how it must extend the control over users and objects, how it must interpret the information about the environment, etc. The resource manager should have a set of policies and mechanism that it could apply on objects. Depending on the object, the user's characteristics and on the received information, it selects one policy for each object.

The environment information consists of information about the environment structure (model) and the performance information [Franken, 94]. The performance information consists of data about the current quality of the services. The resource manager obtains the performance information from events. These events can come from its adapters (accounting adapters) or from other managers (like the monitor). The performance information is used to detect system failures or to detect an overload of an object in order to carry out the dynamic load

balancing control, or it is used to assign the lightly loaded services to one user request (static load balancing control).

The model contains several components and graphs. In our view, the environment has tasks, services, and system parts as main components. Tasks represent a certain amount of work to be realised by the system, and are started by a user. Services represent the elements that are capable of realising a task. System parts are independent components that are capable of executing a service. They are divided into machines (CPU, processors) and network links that connect machines.

Each task uses a set of services to carry out its goal. Each task has a relation with the services (offered by the object-servers). The services also have a relation with both the task level and the system part level. Each service is used by a set of tasks and each service uses a system part to execute itself. A service could also need to cooperate with other services in order to perform its task correctly. So, a service could have relations with other services. And, finally, the system level (machines and network links) has relations with services, to execute them, and with other system parts like machines and networks. All this information is represented in an environment graph. Nodes are tasks at the top level, services in the middle, and system parts at the bottom level of this graph. Its arcs are the relation between the nodes. Arcs could be relations between task on services, relations between services on system parts, cooperation between services, and network links between machines or environments.

To summarise, we use a set of graphs to represent the structure information of each environment, its relations with other environments, and the relations between the environment elements. This information is used to apply the policies correctly.

The automaton is a function that establishes the session among users and objects. It searches for the candidates from the environment, calculates their fitness to satisfy the user's request, and attaches the most suitable. This function operates with the information put into the request queue. This queue is inside the resource manager and keeps all the necessary information about users' incoming requests. The automaton has several states; each state represents one control that a candidate has to excel in order to comply with the characteristics of the user's request. The candidates that do not excel these controls should be erased from the candidates queue.

The automaton's states could be:

- Outstanding requests: requests in this state are still to be attended to. A petition contains a candidate list, their required properties, and the identifier of users who makes the requests.
- Selection of candidates: the search of the feasible candidates is conducted. This search is carried out by the Finder. It could return zero, one or more candidates.
- Security control: the candidates on which the user has no rights are discarded.

- Accounting control: for each candidate, we check if the user has exceeded its accounting quota.
- Estimation of each candidate's fit: it is done with the information gathered during the second state.
- Establishing a connection with the federation if the request so requires: this connection could be done in two ways. The first is at the second state (selection of candidates). In this case, the Finder could return all the candidates searching in all the entities of the federation. And the second way is at this current state. The resource manager is then responsible for when this communication is made. The resource manager establishes communication with other entities when it believes that the remainder candidates are not suitable. This second solution could be better than first solution because in this case we only establish the connection when it is strictly needed.
- Sorting of the remaining candidates: the candidates are sorted according to their fitness and the candidates that do not reach a minimum are discarded. This threshold is determined by the administrator and is applied to every request.
- Assign resources to the request: at this state the connection is done among user and object. The Binder and the resource manager are the elements that apply the necessary adapters in order to establish correctly the communication.

The Trader is composed by two elements: *Finder and Binder*. The Finder is the element that searches for the candidates that could satisfy users' requests. The Binder, and the resource manager, are the elements that establish the communication between user and object. Both apply all the necessary adapters to check this connection according to the environment rules. When these processes finish, they return the object access point.

In order to perform these tasks the Trader keeps the following information: a list of services and who offers them (list of interface references), list of characteristics of objects (if they need security control, accounting control, locking control, etc.).

The service list isolates users from changes of object interfaces and offers location transparency. The users do not need to know the access point of each object, they only have to know their names on the services list. When a change is made to the representation of an object, the system only needs to modify the information contained in this list.

The Binder keeps a list of an object's adapter. This list maintains information about the access point of the object attached to the adapter, adapter's identifier (ID), and whether the object needs any special control such as locking, history, etc. In order to do its task correctly the Binder also needs to know which sessions are established, and for the established sessions, how many adapters it has connected.

The Binder and the resource manager establish the minimum set of adapters for each communication between users and objects. How they attach these adapters and how many for each session are important questions solved by the policies.

In order to carry out the tasks seen below the resource manager adapter keeps the following data: the object interface, the policy that it applies, accounting data, load policy and load data, who the moderator is, who the creator is, and users' access rights. All these attributes can be invisible, visible or editable in order to increase system dynamism. All this information plays the same role as the information maintained in the resource manager, but here they are specialised for each object.

The resource manager and the Trader offer three interfaces to users. These are:

- A yellow pages service: users can establish sessions with objects. This service, when a user's request come in, should select candidates (Finder) and should establish a session (Binder and resource manager) according to policies (automaton, resource manager).
- A management interface to control the policies and to adapt them to new changes on the environment.
- An interface to manage the list of services seen below. Through this interface users can create a new object reference in the environment. Also users can modify and delete them.

We have used the OMT method to design our prototype [Rumbaugh, 91]. The results of this method are shown in theappendix, and consists of our engineering view.

14.3 User Interface

This chapter gives a first impression on the class of information a user would see and how it could be presented.

We take as our example the display of an offer of user interface. This user interface has been designed to ease the user's work.

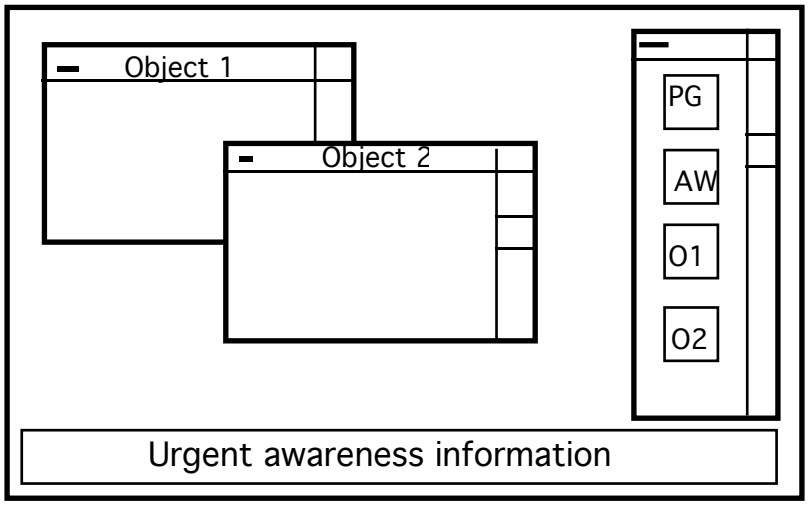


Figure 14.2 A Potential User Display

On the last line the user can see short messages that give him information about current awareness news. If he wants to know more information about this message or any other then he must select the awareness icon inside the object-menu.

All the objects that the user has established for the session are represented inside the object-menu. Each of them has its own icon, and when the user wishes to use an object, he double-clicks on its icon.

This object-menu always contains two icons:

PG Yellow Pages service. This service allows users to establish a session with objects without knowing their access point.

AW Information interface of the awareness manager. Through this interface users can obtain more awareness information about objects.

14.3.1 Yellow Pages Interface

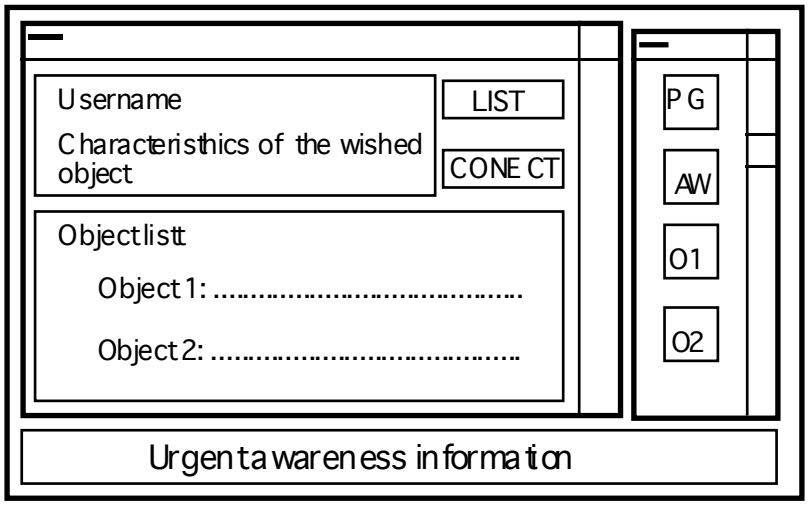


Figure 14.3 A Directory Interface

LIST lists objects according to some desirable characteristics. These characteristics could be: object's operations, parameters, service type, service quality, etc. This function returns a list of objects which match with the desired characteristics. This list contains the object name, object access point and its methods. If the user wishes more information about the object (variants, versions, history information, etc.), he can select the object and produce a new window.

If the selected object does not satisfy the user requirements then the user hits the cancel option and then he can select another object from the previous list. If the selected object satisfies him then he hits 'connect' from the yellow page window. Then the connection process with the selected object will start (Binder, resource manager, adapters). When these tasks finish the system shows OK or WRONG depending on the result of carrying out the connection.

14.3.2 Resource Manager and Binder Control Interface

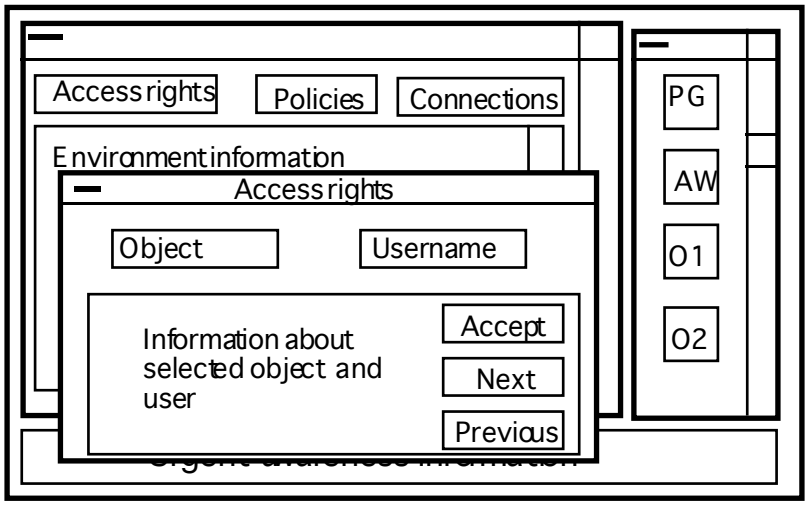


Figure 14.4 Control Interface for the Resource Manager

This interface allows the user-manager to modify the control policy values of the Resource Manager and the Binder.

With this interface, the user-manager could modify the access rights of each user, the object policies like inheritance of moderators or event rules, the user's policies as accounting control. This interface allows the adjusting of the adapter policy (how many adapters does an object need?) according to the environment stages. Finally, this interface gives information about the current established session between user and objects. This information contains data on which adapters are linked to the object, which users have a session linked to the object, who the moderator is, etc.

14.4 From Small To Large (Federation)

We have determined how our elements work inside a heterogeneous distributed environment, but there is one last question. How will we establish a cooperation among distinct managers in order to hide the heterogeneous environment to users [Marquès, 93; Deschrevel, 92].

In order to establish a cooperation between heterogeneous entities we have taken as schema the federation structure. In this structure, distinct entities resolve to cooperate without decreasing their independence. A central authority does not exist. Communications take form between anyone that is inside the federation without the cooperation of the central entity. Thus, the entities are completely independent and the central entity does not exist. This means it is possible that two different entities could have distinctly working rules, and it can provoke several problems. The *Interceptor* is the element that permits us to cross these

boundaries. It is on the boundaries among entities and it has to hide the differences between entities.

There are two kinds of interceptors depending on how we position them on the boundaries. The first one is the *off-line interceptor*. This kind of interceptor is on the boundaries but it only acts when entities establish their communication. The interceptor is not needed while the communication is carried out. Before establishing the communication between entities, each of them establishes a session with one interceptor and, with its collaboration, they make a conjunction of communication rules. When these rules are set, the collaboration of the interceptor is not necessary, the communication is carried out by the entities according to the established communication rules.

The second kind of interceptor is named as the *in-line interceptor*. This interceptor is also positioned on the boundaries, like all interceptors, but now accords between entities do not exist. All messages between both entities will need the collaboration of interceptors. The interceptor will make all the necessary translations in order to allow communication between both entities.

The entities choose the correct kind of interceptor depending on the boundary characteristics that the entities need to cross. If the entities only need to establish an accord then the first solution is the best one. However, if they only need to establish a short session between them then the first solution is much too expensive, and the second one is preferable.

Therefore, we have one or more *federation managers* inside several environments. These elements are connected with the interceptors and these managers control them. These managers have to be able to manage the communication with interceptors and any manager that wants to establish a session with another manager placed in another entity must communicate with these managers. These managers have a set of communication policies. These policies help them to efficiently carry out the federation communications.

14.5 Using Interface Adapters

We have determined the necessary functions and the components of the Resource Manager and the Trader. But we must address one last important issue: how the managers act over the environment and from where these managers obtain their necessary information.

In order to address these issues the managers need help from the interface adapters. These elements intercept users' requests and make transformations according to the policies of several managers. Therefore, managers can act on the objects sending control orders to adapters. Adapters can perform these controls because they lie between users and objects. This position also permits them to extract the necessary information about objects for managers.

Interface adapters have an interface access point, as normal objects, and they are associated with exactly one object, but the object could be an object group, a

relationship object or a single object. Adapters are situated between the user and the object and when the user sends a request to the object, he really sends it to the adapter, but he does not know it. The interface of an adapter must have all the methods of the associated object. In order to make the manager controls and to adapt the user's request according to policies the adapter can modify this copy of the object interface. The adapter can delete, add and modify some interface methods.

Each adapter is managed by a manager and it only makes one check control on the associated object. We would need more than a single adapter in front of an object in order to make all the necessary controls. As an example, you could imagine that you have an object that represents a database. This database can have many users working at the same time. So, you must put several adapters to guarantee the correct access. This set of adapters could be: locking adapter, accounting adapter that it provides accounting control over the users, event adapter that it generates events, and one versioning adapter.

In order to adapt the objects to the environment we need to add special elements that give distinct views from the same object. These new elements are interface adapters, and their position between user and object allows them to hide the internal representation of objects and permits them to offer more than one view. The adapters offer several views depending on the environment and user information.

We have identified three ways to use the adapters:

- One adapter for each object which manages all the controls (too difficult).
- One adapter for each control on every object.
- one adapter for each session established for every user on every object.

The second way is the best solution that solves locking and versioning performance (see the SOS functions). If we should put one adapter for each session then we must allow communication among all the adapters in order to apply correctly the locking and versioning policies, and this may be very expensive. But this solution does not solve the problems of the RM adapters. The accounting policies need to have one adapter per user's session. This third solution allows the Resource manager to increase the dynamism of the environment, changing the data of each adapter according to the environment state. Also, it could allow the dynamic reconfiguration of the environment to be carried out.

14.6 Future Issues

We are going to study the new processes and objects that we need to carry out the dynamic reconfiguration. This would attach new servers to a user when his session goes down (the object failed). This reconfiguration must be hidden from users whenever possible. We want to study the real performance when two or more managers are connected via the interceptors (federation). Finally, we think

that studying the relation between other SOS elements and the Resource manager is an interesting issue and will offer a coherent external view to the users.

14.7 References

[Deschrevel, 92] Deschrevel, J.P. *Trading across Boundaries*. ISA Project, 1992.

[Marquès, 93] Marquès, J.M., Navarro and Sarmiento, L.M. *From small to Large Scale*. Comic Project Deliverable 1.1, 1993.

[Rodríguez, 94] Rodríguez, G. and Navarro, L. *A view of the SOS: refinements*. Comic internal review, UPC-4-2, 1994.

[Franken, 94] Franken, L. J. N. and Haverkort, B. R., *The Performability Manager*. IEEE Network 1994

[Rumbaugh, 91] Rumbaugh, J., Blaha, M. and Premerlani, W. *Object -oriented modelling and design,* 1991.

Chapter 15

Realisation of the COMIC Shared Object Server on Basis of CORBA

Anja Syri

GMD

This chapter introduces the concepts of the Common Object Request Broker Architecture (CORBA) and the Object Services provided by the OMG. A possible realisation of the COMIC Shared Object Server on basis of CORBA is presented. In particular, the eventing service within CORBA is useful for the eventing, locking and versioning concepts of the SOS. The chapter concludes with a brief overview of the related work within the GroupDesk system.

15.1 The Common Object Request Broker Architecture

The work of the OMG aims to create a platform that provides software developers and users with the possibility of developing and using integrated software systems. The object oriented approach supports modular production of software and encourages the reuse of code. The next step is to use the concepts for distribution of objects and to provide facilities that make it possible to call remote objects across different operating systems and hardware platforms.

15.2 The Object Model

This section describes the common object semantics for specifying the externally visible characteristics of objects in an implementation-independent way.

The OMG Object Model defines a core set of requirements that have to be fulfilled by each CORBA-compliant object model. Further on it defines the concept of components which are extensions to the Core Object Model and not to be supported by all systems (like exceptions). Profiles define groups of components to get domain-specific sets of extensions (such as an Object Database Profile).

15.2.1 The Core Object Model

The OMG Object Model is based on the following basic concepts: objects, operations, types and subtyping:

- **Types** characterise the behaviour of their instances by defining the operations that are supported. These externally visible characteristics are specified in an interface (defined in an IDL, an interface definition language) by a set of all signatures of operations that are supported. There is no notion of multiple named interfaces defined on a type that export a smaller type specification.

  Types can be related to each other through supertype or subtype relations. The root of the type hierarchy is formed by the type Object. One type (say S) is a subtype of another (say T) if the first is a refinement of the second: S inherits from T and for each operation of T there exists an operation of S with the same name and the same number and types of parameters and results. The Core Object Model supports multiple inheritance but does not allow the redefinition of inherited operation signatures.
- **Objects** are created as instances of types and cannot change their type. Each object has a unique identifier that is based upon the characteristics of the object.
- **Operations** are used to model the external interface to the state of an object. Operations may return results, cause side effects or raise exceptions.
- **Non-Objects Types** are defined in a (Non-Object-) component and included in profiles. CORBA defines the following non-object types: short, long, ushort, ulong, float, double, char, string, boolean, octet, enum, struct, sequence, union and array.

The realisation of a type is put into the implementation. Implementation details (like inheritance of implementations) are not specified within the object model. The combination of a class specification and implementation is called a class. For one type specification different implementations are allowed, but it is not specified whether an object can change its implementation during its lifetime or not.

15.3 The Reference Model

The Reference Model identifies and characterises the main components of the Common Object Request Broker Architecture which are:

- Object Request Broker
- Object Services
- Common Facilities
- Application Objects

as well as the interfaces and protocols which determine the interaction among these components.

Whereas the broker provides the means to transport requests and results, the other components are groups of services that differ with regard to their universality.

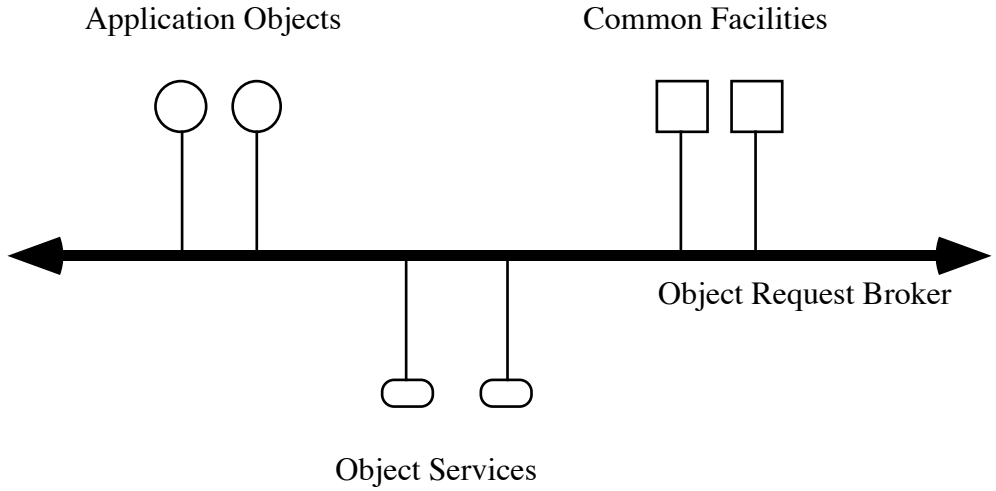


Figure 15.1: The Common Object Request Broker Architecture

15.3.1 Object Request Broker

The Object Request Broker (ORB) provides mechanisms to support the interaction of objects. Objects may invoke requests as well as receive requests and results in a transparent way; the underlying platform (hardware, operating systems, etc.) is hidden.

The ORB receives the created request and forwards it to the corresponding object, possibly using Object Services to do this work. Strictly speaking the broker receives the request from the client, encodes the parameters of the client's representation and determines the method which is to be invoked (request dispatch). The broker decodes the parameters and delivers the request (using TCP/UDP/IP, ISO/TPn) to the corresponding object.

Additional services of the broker, some of them performed with the help of an Object Adapter explained later in this chapter, are:

- Mapping from the naming space of the invoking object to the naming space of the invoked object.
- Synchronisation of client and server.
- Activation and deactivation of target objects and implementations.
- Exception handling.
- Security mechanisms (authentication and protection mechanisms).
- Generation and interpretation of object references.
- Mapping object references to implementation.
- Registration of implementations.

15.3.2 Object Services

Object Services serve as building blocks for compound objects. They are the target of the OMG's standardisation effort. Up to now only naming, events, life cycle and persistent storage are standardised. During 1994 associations, transactions, security (access control at an appropriate level of granularity on objects and their components) and licensing will follow. Other plans concern the management of classes and instances, queries and versioning.

Object Services will be presented in more detail in section 15.4.

15.3.3 Common Facilities

Common Facilities are objects that provide general purpose capabilities useful for many applications.

Document linking, help facilities and printing are to be standardised in 1995. Other candidates for Common Facilities are:

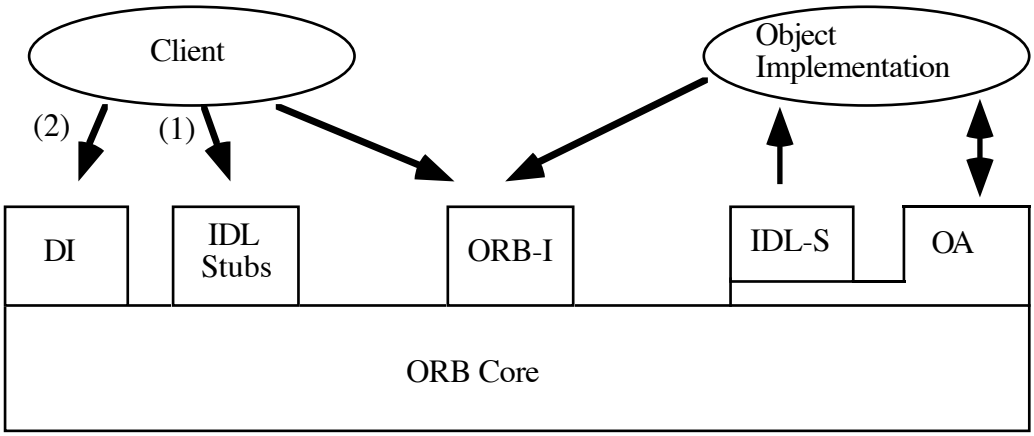
- Cataloguing and browsing of classes and objects.
- Reusable user interfaces.
- Error reporting.
- Electronic mail.
- Tutorials / computer-based training.
- Common access to remote information repositories.
- Agent facilities.
- Interfaces to external systems.
- User preferences.
- Profiles.

15.3.4 Application Objects

Application Objects are specific to end-user applications and not standardised by the OMG. Here traditional, non-object-oriented applications may be integrated: office applications, CADD applications, CASE tools, network management applications, information access or knowledge-based systems.

15.3.5 Invocation of an operation

The Object Management Architecture (OMA) provides the means to invoke operations upon remote objects across different operating systems and hardware platforms. The general invocation mechanism is shown in Figure 15.2.



DI□□□□□□□□□□□Dynamic Invocation Interface
ORB-I□□□□Object Request Broker Interface
IDL-S□□□□□IDL Skeleton
OA□□□□□□□□□Object Adapter

Figure 15.2: Invocation of an operation

The client object sends out a request. It may use the interface-specific stubs (1) to specify the request or dynamically create the request (2). The ORB is responsible for finding the corresponding object implementation for the request, it prepares the object implementation to receive the request and to communicate the data. The request is transmitted to the object implementation via an IDL skeleton. When the request is complete, the results and control are returned to the client.

The *Object Adapter* (OA) compensates the functionality of the ORB and generates and interprets object references. New server objects register themselves at the OA. The OA should be tailored to the object implementation. For example, as database applications want to register all their objects at once it has to provide this functionality. During a method invocation the Object Adapter takes over the duty to activate and deactivate the target object or implementation.

The *Interface Repository* provides facilities for the management of interface definitions and provides persistent objects that represent the IDL information. This information may be used by the ORB to perform requests and by a client program to encounter objects that were unknown during compile time of the program (dynamic invocation).

The *Implementation Repository* contains information that allows the ORB to retrieve and activate object implementations.

15.4 The Object Services

Object Services serve as building blocks for compound objects and provide general services for the realisation and management of objects. They will be presented here to give an idea of the services already available. In the next section

they will be examined with regard to their relevance for the Shared Object Service (SOS) of COMIC.

15.4.1 The Naming Service

The Naming Service provides the ability to bind a name to an object relative to a naming context. Therefore it provides the principal mechanism through which most clients (resp. the ORB) locate their target objects in an ORB-based system. A naming service can be implemented using an underlying enterprise-wide naming server. Names can be presented to programs through a names library.

15.4.2 The Eventing Service

A standard CORBA request results in a synchronous execution of an operation. To get a more decoupled communication model an Eventing Service is provided. This service provides the means to have asynchronous events and reliable event delivery; event fan-in, notification fan-out. It can be used to provide notifications about changes.

The Eventing Service differentiates two roles for objects, those which produce events (suppliers) and those which process event data (consumers). According to this approach two event delivery models are introduced:

- *push event delivery model:* here the initiation is made by the supplier of events. The supplier and the consumer exchange object references (here called PushConsumer and PushSupplier object references). Then the supplier pushes event data to the consumer by invoking push operations on the PushConsumer interface.
- *pull event delivery model:* here the initial steps are made by the consumer who pulls event data from the supplier.

Event channels provide the means for a decoupled communication between suppliers and consumers and enable communication between multiple suppliers and multiple consumers. They are standard CORBA objects and serve as a kind of buffer in between the partners. With the push-style communication, the supplier pushes event data into the event channel which forwards the events to the consumer. In addition to the pull-style communication a mixed style communication is also possible.

Event channels may be chained. The definition of a generalised filtering event channel is an area for future standardisation. They provide the means to get event channels that filter events supplied by another channel.

Event data exchange can be generic or typed. In the latter case the event data exchanged within the push or pull event delivery process is defined by an IDL-interface.

15.4.3 The Life Cycle Service

The Life Cycle service provides means for creating, deleting, copying and moving objects in different locations.

If a client object wants to create an object, it searches for a corresponding factory object with the help of a finding mechanism (as a naming context). Factory objects support operations for the creation and initialisation of new instances, they are object implementation dependent. If there is no standard interface for a factory, a generic factory interface is defined by the life cycle service. Generic factories search for a factory.

In order to delete an object the client sends a delete request to an object that supports the LifeCycleObject interface.

To get a copy of an object or to move an object, the client sends a copy or move request to an object that supports the LifeCycleObject interface. The implementations of move and copy can use the factory Finder to find appropriate factories in the remote place.

Further questions considered here are:

- Can the client control the location of a new / a copied object?
- Can the location be determined according to some administered policy?
- What entity does the client communicate with in order to create a new object / to copy or migrate an object?
- How does the client find that new / copied entity?
- How much control does the client have over deciding the implementation of the created object?
- Can the client influence the initial values of the newly created object?
- Can the client create an object in an implementation specific fashion?
- What happens to the implementation code of a copied or migrated object?
- What are the boundaries of a graph of distributed objects?
- If multiple objects are affected, how is the life cycle operation actually applied to those objects? Are cycles in the graph preserved?

Externalisation describes the transformation of an object into a suitable form for storage on an external media, internalisation describes the other direction of transformation.

15.4.4 The Persistent Storage Manager

The Persistent Storage Manager (PSM) simplifies the task of defining and managing a CORBA object's persistent state. It provides the basic functionality of an object database and efficient local data caching for the object implementation. The objects are stored in a simple file, in an OODB or in a relational database.

The PSM is used through an API and not through the ORB which is more efficient. It consists of a DDL (Data Definition Language) compiler and the PSM library.

The DDL compiler takes a schema as input and produces declarations and executable code (called schema binding) to create and manage persistent objects. The PSM library provides runtime support required by these schema bindings to access the non-schema-specific facilities of the PSM.

To store an object's persistent state the designer of the CORBA object implementation first uses the PSM-DDL to describe the object's persistent state as one or more data object interfaces. These data object interfaces are grouped into units called schemas (files of DDL statements that define the content of persistent storage).

The DDL compiler takes a PSM schema which is invariant over computing architecture and language as input and produces a language and architecture specific schema binding.

CORBA objects may then be implemented by code organised as servants composed of skeleton code generated by the IDL compiler, user written code and the schema binding (data object bindings). The PSM provides the ability to create instances of data object interfaces. A data store holds these data objects - typically for a single CORBA server process. Changes to data objects are encapsulated into transactions, the unit of atomic update is defined by the clusters within the data store.

15.4.5 The Association Service

The Association Service provides facilities to establish connections between two or more objects. Bi-directional associations are maintained between associates, n-ary relationships can be introduced through intermediate objects.

The Association Service supports referential integrity for the bi-directional connections (for updates and deletions by the Life Cycle Service). Until now, no query language is defined on associations.

The following operations are possible:

- Creating, deleting and managing relationships between objects.
- Iteration through connections by navigating component-by-component.
- Creation and maintenance of connections in graphs of objects.
- Enforcing constraints on types of objects in a relationship by associate objects.
- Register consumers for different types of associate events in the push model of the Eventing Service (Reporting Associate Interface).
- Support of the cooperation between associates: an operation is invoked upon the associate (Associate Cooperation Interface).

15.4.6 Transactions

Transactions support the atomic execution of one or more operations which may be issued to one or more objects. All modifications of an object's state become

visible when the modification has succeeded completely or do not become visible at all. Short and long nested transactions shall be supported.

15.4.7 Object Versioning and Configuration Management

This service supports the identification of alternate versions of existing interfaces and object instances and the management of mutually consistent collections of these interfaces or instances. It provides means for incremental modification and management of interface and object instances. Such a service might support hierarchical trees of versions.

The version service should insure that consistency is maintained between interfaces and any existing instances (old instances may use old versions of the interface or all object instances can use the newer interfaces).

The configuration management server provides means of creating, deleting and accessing collections of version references that are mutually consistent.

15.4.8 Object Security

The Object Security Service provides operations which perform discretionary access control on objects and interfaces. The control is set by the object's owner and cannot be circumvented in normal cases. Access control may become mandatory. It determines which clients can make what requests on an object (specification of authorisation and rights).

Provided operations support setting, revoking or changing of access controls. Assigning / delegating privileges to groups of users may be supported, user authentication schemes may be provided.

15.4.9 Object Interchange Service

The Object Interchange Service supports the negotiation of protocols used for the interchange of data between objects. Different qualities of data interchange and the provision for declaring different intended usage paradigms are possible.

15.4.10 Object Concurrency Control

The Object Concurrency Control mediates concurrent access to one or more objects by one or more clients such that both, source and target object, remain consistent and coherent.

15.5 Realisation of the SOS with CORBA

The requirements for the COMIC Shared Object Service are described in detail in the COMIC Deliverable 4.1 "Requirements and metaphors of shared interaction".

This section shows how the objectives explained in the chapter “Requirements for the COMIC Shared Object Service” can be reached by taking the Common Object Request Broker Architecture as basis.

15.5.1 How to realise the Shared Object Service

The COMIC Shared Object Service may be realised in different ways:

- As basic mechanism of the service.
- As part of the most basic object type from which all other objects are derived.
- As different objects from which the “real” objects are derived.
- Using interface adapters.

The COMIC SOS should provide a strict separation between behavioural semantics of the objects and the management facilities of the SOS. It is therefore proposed to use interface adapters.

Each adapter may be seen as a further broker that delegates an incoming call to other objects - depending upon the user, a group or a task context: it contains a set of mappings which map methods names (that are presented to the user) to those used within the object interface.

There are two ways to organise an object adapter; as a CORBA Object Adapter or by other objects that play the role of an object adapter. For the client the existence of an adapter remains transparent; he only sees the object adapter interface not knowing the “real” object interface.

The CORBA Object Adapter may provide the functionality of the CSCW object services. Then every instance of the adapter must provide the same interface and services for all ORBs it is implemented upon. According the concepts of the OMG there should be as few Object Adapters as practical so this approach seems not to be suitable here.

Using the second approach, we can realise the object adapter as an object having an IDL interface which is exported. The clients only know this object - thinking they invoke methods upon the “real” object. So every call first goes to the object adapter which identifies the caller (To which group does the caller belong? What rights does the caller have?). It undertakes the initial steps and forwards the invocation to other object adapters, each of them responsible for one duty, and finally to the server object.

15.5.2 Information Modelling Facilities

In the SOS paper three core object types are identified: single objects, relationship objects and collection of objects.

Simple objects may be represented by the type “object” provided by the Core Object Model of OMG. As described earlier, an object type is defined by an

interface specified in IDL. The behaviour of an instance is defined by operations, attributes are defined by set and get-operations.

Relationship objects are not defined within the Core Object Model, they are part of the Object Services. The end of the standardisation of the Association Service is planned for 1994.

Collection objects are not standardised at all. They may be realised as a component or taken from a CORBA compliant database. In addition there should be a possibility to invoke construction and inspection operations on every instance held within the collection instance in a transparent way. A collection may be specified with the help of the any-datatype:

```
interface collection {
    insert (in any elem);
    result getFirst (out any elem);
    result getNext (out any elem);
    result invokeConstructionOnAll (in request specifiedFunction);
    any invokeInspectationOnAll (in request specifiedFunction);
}
```

The any-datatype offers the possibility to ask the incoming parameters for their type and treat them according to their type. Construction and inspection operations may be invoked by using a request interface (as in the dynamic invocation method): a request is constructed, passed to the collection and the corresponding operation of the interface collection iterates over all components and invokes the request on each of them.

15.5.3 Locking

Locking is done for the following reasons:

- To keep the integrity of information.
- To coordinate access to information and reduce the occurrence of errors of simultaneous manipulation of the information.
- To give better possibilities of orientation to the user of the information by providing information such as who else is locking the information.

Locking mechanism (locking types may be NoLock, ReadLock, WriteIntentLock) and locking policies (as for example: NoRWConflict, RWConflict) are separated.

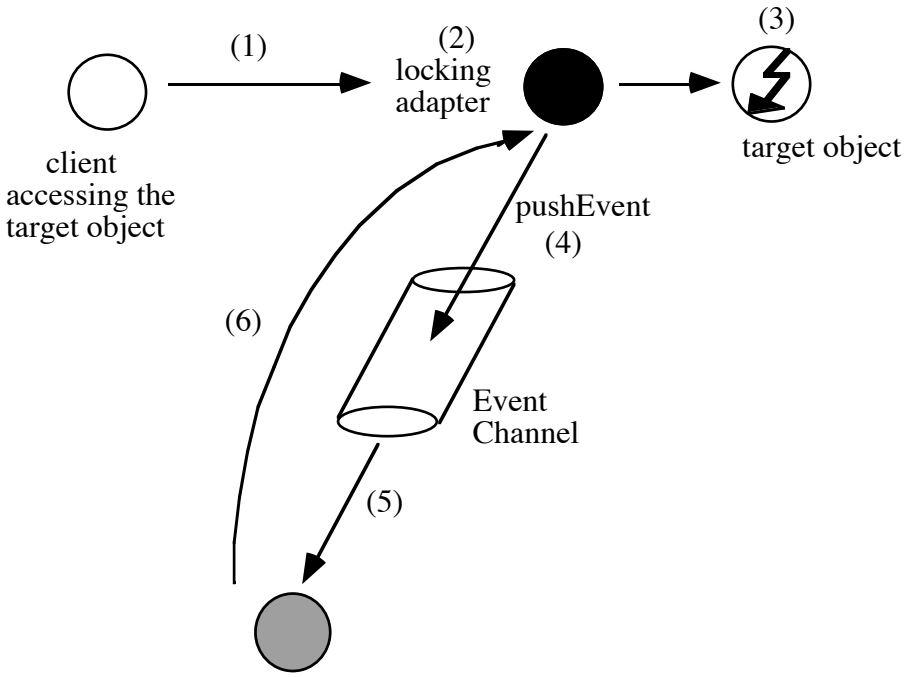


Figure 15.3: Locking

To realise this we need a locking adapter. This object provides the basic functionality for the locking and forwards the request to the target object.

```
interface lockingAdapter {
    readonly attribute accessingUserPairList accessingPersons;
                    // (accessingPerson, reasonForAccess)
    readonly attribute UserIdentifierList waitingPersons;

    res registerInterest (in pushConsumer p);
    res deregisterInterest (in pushConsumer p);
}
```

The IDL-interface provides methods for client objects to ask for the locking state of the object (6). It also provides the facility to register interest in the locking state, so that one is automatically informed. Here we could use the push model of the CORBA Eventing Service: a person object is associated with a pushConsumer reference.

If one person object sends a locking-request (1) to the target object via the object adapter (2) then the request to the object is either directly forwarded to the target object (if no locking conflict exists) (3) or is put into the request queue held within the locking adapter. The locking information is recorded in the locking adapter and forwarded to the event channel which pushes the information to the interested objects (4, 5).

The locking remains transparent when the client does not register his interest in the locking state. He gets information about all accesses when he registers interest.

The object implementation of the locking adapter may have the following appearance:

```
class lockingAdapterImpl : targetObjectImpl{
    // attributes and methods of target object
    // methods realising the interface
    requestQueue          q;
    accessingUserTripleList accPers;
            // (accessingPerson, lockingType,lockingPolicy)
    pushConsumerList   intPers;                    // interested persons
}
```

15.5.4 Versioning

The CORBA Object Versioning and Configuration Management supports the management of mutually consistent collections of interfaces or instances and maintains the consistency between interfaces and existing instances.

The Shared Object Server considers a variation from the existing view of versioning and version control systems and has therefore to be realised by a separate mechanism. It supports the management of versions and in addition it differentiates between versions of an object and variants.

Versions are agreed by a group, and may be merged. New versions are objects for normal usage. Variants signify a difference in opinion and cannot act as objects in use. Existing variants should just get notified about new versions.

For every version a version manager (person or group) is to be identified. The default value could be the creator of the new instance. The versioningAdapter and the versionManager exchange pushSupplier and PushConsumer references. Whenever a change to a version is made, the versioningAdapter may forward a notification to the versionManager who can decide what to do and if a new version is to be created.

```
interface versioningAdapter {
    readonly attribute versionIdentifier versionId;
    readonly attribute objectType              oType;
                    // original, version or variant
    readonly attribute date                    dateOfCreation;
    readonly attribute person          creator;
    readonly attribute predecessorRelList allPredecessors;
    readonly attribute versionIdentifier previousVersion;
    readonly attribute variant         variantOf;
    readonly attribute predecessorList         allPredecessors;
        attribute userIdentifierList versionManagers;
        attribute userIdentifierList variantController;
}
```

In this context we have to think about aggregation of modifications and history.

15.5.5 History

The main goal here is to integrate time into the object model and to record the history of an object. The history and the visualisation of the history supports the asynchronous awareness of the users.

In the chapter “Requirements for the COMIC Shared Object Service” four major time variables have been identified: start and end of the valid (real) time and start and end of the transaction (system) time.

The history adapter may be used to store the life cycle of an object so that it becomes easy to recreate an old state of the object. For this reason, it provides a long list of time - value pairs (four pair-lists for every attribute of the target object). Every action that manipulates the state of the object is forwarded by the history adapter, that adds an entry to its lists.

```
interface historyAdapter {
  // all methods of the target class
  readonly   attribute  timeValuePairList   attr1ValidStarts;
  readonly   attribute  timeValuePairList   attr1ValidEnds;
  readonly   attribute  timeValuePairList   attr1TransStarts;
  readonly   attribute  timeValuePairList   attr1TransEnds;
// same attributes for every target object attribute
}
```

Here again we have to think about the aggregation of events representing the actions, otherwise we have a strong information overload.

15.5.6 Access

Access to an object may be managed by an access adapter. Although the OMG Object Service “Object Security” is not yet standardised, the proposals for its functionality propose the following:

- Specification of authority and rights.
- Operations for the management of access control.
- Assigning and delegating privileges to groups of users.
- User authentication schemes.

This functionality may be sufficient for us. The functionality proposed in the COMIC SOS part of the Deliverable 4.1 differentiates owner, responsible, possessor, moderator and creator of an object. For each object these entities are specified. A table realises the mapping from attribute and person to the condition variable (invisible, visible but not editable, visible and editable). The following interface shows the corresponding interface:

```
interface accessAdapter {
  readonly   attribute  groupIdentifier     owner;
  readonly   attribute  groupIdentifier     responsible;
  readonly   attribute  groupIdentifier     possessor;
```

```
    readonly    attribute   groupIdentifier     moderator;
    readonly    attribute   groupIdentifier     creator;
    readonly    attribute   table               accessRight;
                // attribute & roleOfPerson
                //    -> condition var
}
```

15.5.7 Queries and Views

Queries are one objective of the CORBA object services standardisation effort, but not yet standardised.

The SOS should provide a query mechanism for end-users (through an interface) and for agents to locate sets of objects.

SOS queries should support the following functionality:

- interpretation of query expression and return a list of object identifiers,
- relational operators, boolean operators in a query,
- transient and future queries.

Object adapters can be used to present a special view on the objects. If we have one object adapter per user, the user may send his identification with each request, the “main” object adapter may forward the request to the right object adapter and return the right view upon the object.

15.5.8 Events

Events may be used to inform objects about changes and to gain awareness about what is going on. Whenever an object is accessed or altered, the system should inform relevant users about the action occurred. The SOS provides a set of specific services to handle event objects.

Two different kinds of events have to be differentiated: events that may be created and propagated automatically and events a user has to create explicitly.

The event object service has to support the definition, creation of events, the specification of relations between events and users. In addition the SOS has to provide the means for event handling and subscription of events.

The event adapter forwards all incoming requests to an object and generates the corresponding events. Incoming events may be aggregated to put more semantic into an event object. This means an integration of an event transformation component into the event adapter.

The event adapter (2) distributes the events to interested participants. To fulfil its responsibilities the event handler uses the Event Services supported by the OMA.

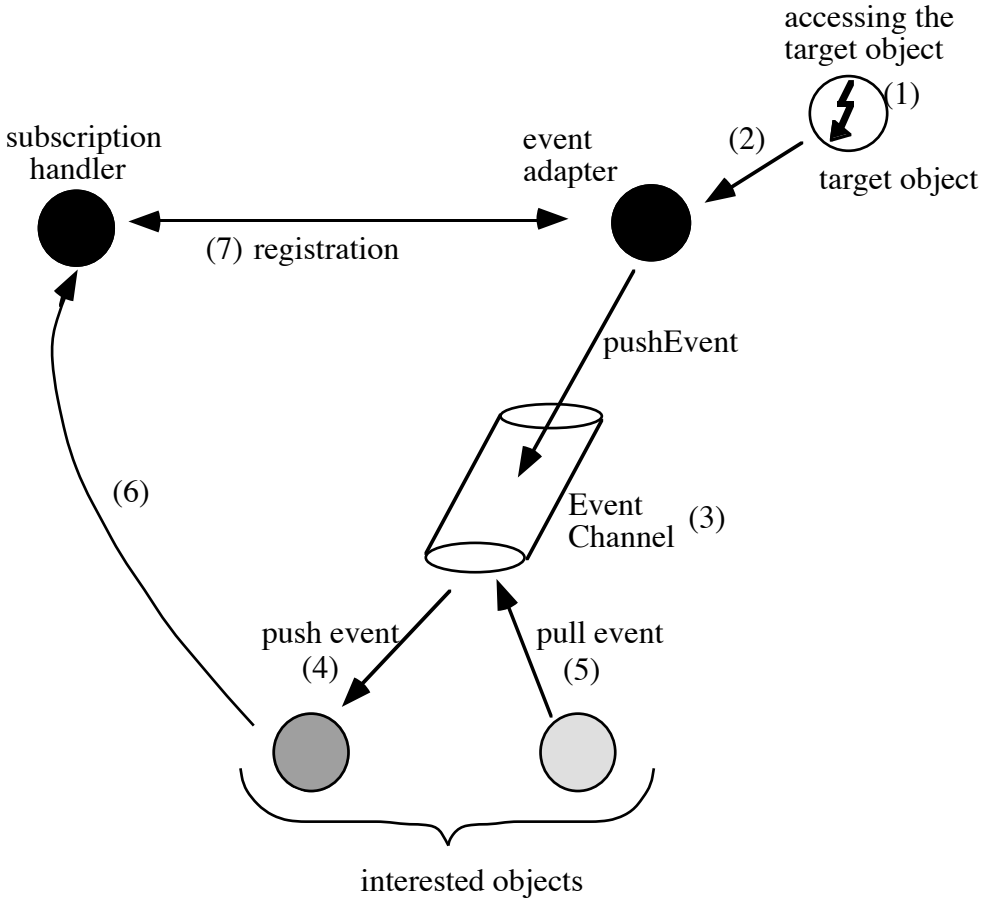


Figure 15.4: Event propagation

The subscription handler maintains the information about which events each client (application or user) is interested in. A client declares interest in a certain event type by registering itself at the subscription handler (6). There are different possibilities for subscribing:

- Lazy or eager subscription that determine when a client is informed about a change (immediately or when next accessing the object).
- Anonymous or public subscription that determine if the subscriber wants to remain anonymous or if others may be informed of his subscription.
- Subscription to specific categories of events: the client may specify its interest in certain events or event types or events with certain properties.

The functionality to be provided is:

- Announcements that no more events will be sent out by an object.
- Declaring and retracting subscriptions.
- Subscription on system events.
- Suppression of announcements (if the event would announce an event that is secret).
- Subscription to other subscriptions.

The event adapter creates an event channel (3) to which event consumers are connected. If the event adapter differentiates eager and lazy subscription objects, it may use different event propagation models: the push model for eager subscription (4) and the pull model for lazy subscription (5). Then it has to cooperate with the access adapter which polls the event data when an access occurs.

```
interface event {
  readonly   attribute  identifier      id;
  readonly   attribute  objectIds       src;
  readonly   attribute  objectIds       target;
                  // public subscribers
  readonly   attribute  eventType       t;      // announcing event?
  readonly   attribute  time            timeOfGeneration;
  res generateEvent (identifier id, object src, eventType t);
}

interface eventAdapter {
  readonly   attribute  eventAdapterID      id;
  readonly   attribute  sHandlerId          subscriptionHandler;
       attribute   pushConsumerList  interestedObjects;
            // here subscription handler can inform                                    //
eventAdapter about new subscribers
  res propagateEvent (eventId id);
}

interface subscription {
  readonly   attribute  userIdentifier      user;
  readonly   attribute  fashion         f; // lazy or eager
  readonly   attribute  publication     p; // public or anonymous
  readonly   attribute  eventCategory       c;
  readonly   attribute  groupIdentifier     creator;
}

interface subscriptionHandler {
  readonly  attribute    eventAdapterList eHList;
  subscriptionId declareSubscription
    (in any eventType,
    in any eventPropertyList,
    in PushConsumer p),
  subscriptionId declareSubscription (in subscriptionId i);
  res retractSubscription (in subscriptionId i);
  res registerEventHandler (in eventhandlerId eId);
}
```

Explicitly generated events may be propagated by the propagateEvent method of the eventAdapter.

15.5.9 Awareness

Every object possesses a PATM attribute (pay attention to me attribute) which may be provided by an awareness adapter.

```
interface awarenessAdapter {
        attribute    long PATM;
}
```

This attribute gives a hint about the importance and relevance of the object. The awareness adapter may also provide means to aggregate events.

15.5.10 Interaction and Object Presentation in cooperative applications

Four different aspects are mentioned concerning the object interaction and presentation: different presentations of objects in SOS, support of awareness of similar objects, support of awareness between participants and support of browsing.

Different presentations of objects are provided by different object adapters and can be compared to the issue of views in the chapter queries. The basic concepts for awareness are provided by the object themselves, the eventing concept and the awareness adapters.

15.5.11 External References

The goal here is to integrate objects that cannot be stored in a SOS, like documents that are not electronically available. Here external adapters may be used as described in the deliverable. A corresponding interface may look like:

```
interface external adapter {
  readonly  attribute    any     holderOfTheOriginal;
  readonly  attribute    string  accessDescription;
}
```

15.6 Related Work Within GroupDesk

The GroupDesk system supports the cooperation of people working in offices that may be located at different sites and possibly at different times. It uses the metaphor of a desk which may be shared with other persons. The desks, representing different workspaces, allow users to structure their workspaces and to share them with others. The major goal is to provide the users with

synchronous and asynchronous awareness of the actions going on within their work environment.

According to the objectives of the project the main focus was on the Eventing Service, the Awareness Service and the History Service. Object Interaction is done by the Object Request Broker.

15.7 Conclusion

We have introduced the concepts of the Common Object Request Broker Architecture and the Object Services that are provided by the OMG.

We have also mapped the COMIC SOS services upon the facilities provided by the OMG. In particular, event is supported by the concepts of the OMG and could be used for eventing, locking and versioning concepts of the SOS.

15.8 References

[1] Object Management Group (1992): *Object Management Architecture Guide*, edited by R. M. Soley, OMG TC Document 92.11.1, available on request from the Object Management Group.

[2] Object Management Group (1991): *The Common Object Request Broker: Architecture and Specification*, OMG TC Document 91.12.1, available on request from the Object Management Group.

[3] Rodden, T.; Mariani, J.; Eiderbäck, B.; Hägglund, P. (1993): *Requirements for the COMIC Shared Object Service*, in “Requirements and Metaphors of Shared Interaction” - COMIC Deliverable 4.1, pp. 203-242.

Chapter 16

A Reference Framework for the COMIC Shared Object Service

Ludwin Fuchs
GMD

Leandro Navarro
Gerard Rodriguez
UPC

Jonathan Trevor
Lancaster

Konrad Tollmar
KTH

16.1 Introduction

The tendency of modern companies to organise almost every aspect of operation by a continuous flow and exchange of data covers all economic areas and involves many different kinds of tasks. Applications range from remote information retrieval systems, electronic access to bank accounts, process scheduling systems for power plants or chemical factories, to coordination, cooperation and communication software in big multi-national enterprises. Additionally, drastically falling hardware prices mean that access to computing resources is no longer restricted to organisations. Private use of electronic services over wide-area networks constitutes a fast growing market. As a consequence, the general ability to handle data is drifting towards a serious crisis. A ten-thousand fold increase in machine and network performance during the recent decades is not the reason for this development. Rather it is caused by the fact that common paradigms for development and maintenance of software are no longer sufficient to satisfy the increasing demand for data management.

The COMIC project has approached the above problems in the domain of computer support for cooperative work. The COMIC Shared Object Server (SOS) is intended to address the specific problems that arise in cooperative situations [Rodden, 93; Rodden, 92]. A range of concepts has been proposed that we believe to be suited to enable effective support for distributed work arrangements, and which are demonstrated by prototype implementations. Some specific facilities provided by the design of the SOS are:

- Provision of integrated facilities that enable the support for synchronous as well as asynchronous communication, collaboration and coordination tasks.
- Enabling shared awareness in distributed work arrangements.
- Support for inter organisational cooperation.

- Management of access and distribution of resources in competitive environments.
- GUI development support for shared applications and visual feedback according to the cooperative state.

The experience in the implementation of the different prototypes has led to a proposal for a common integrated architectural model for the SOS. This chapter presents a reference framework for services and facilities that are integrated in this architecture.

The architecture comprises of three distinguished layers. Each layer provides services in a different granularity for the whole range of issues that deal with support for distributed cooperation and shared awareness that have been considered important in developing the prototypes. Thus, the layers are motivated by the need for different levels of abstraction rather than functionality. Applications using the SOS may access the services of whichever layer is appropriate and suited to serve their specific needs.

Among the prototypes, the COLA system has shown the need for object adapters as basic components that enable lightweight, i.e. open cooperation support in the SOS. Instead of including a constraining amount of semantic information about the work supported in the system, COLA provides mechanisms to support sharing with as few of the semantics as possible. The role of object adapters may be described as a mediating instance between the objects and the client or user. They provide additional cooperative facilities to method invocations depending on the context within which they are defined. This has been applied in COLA to implement dynamic locking of objects and access control. The SOS provides these services accordingly based on interface adapters.

A major goal of the CoDesk system [Sundblad, 94] is the support for a smooth mixed mode of asynchronous and synchronous cooperation. One of the key functionalities in the CoDesk system is the provision of persistency to maintain work processes that span over longer periods, e.g. weeks or months. One of the shortcomings in the current architecture of CSCW environments is the lack of interoperability between persistency services provided by OODBMS and distribution services. CoDesk has tried to bridge this gap by an orthogonal exchangeable architecture. The SOS will reflect this by integrating a persistent storage manager, a properties manager and a naming service.

In the GroupDesk system the use of local event distribution as the basis of increasing the awareness of users in cooperative settings is demonstrated. The SOS architecture takes these issues into account on different levels. Events and links between objects that provide the required functionalities are included in the lowest level of the architecture layer. An event distribution service that is tailorable to specific needs follows on the next level and as an example of high level support for awareness, a history service and an event information and notification service are included in the architectural model.

Scaling up and inter organisational cooperation are demonstrated by the Aleph system. As a consequence the architecture model includes facilities for object naming and a set of services that enable cooperative applications to apply the concepts of federation and trading. Additionally a resource management service has been identified as being a further central part of the SOS architecture.

The Aleph system, a prototype system being developed at UPC, demonstrates the use of the resource management and trading functions. The design of these functions is described in [Rodriguez, 94] and [Navarro, 94]. These functions reduce the complexity of articulation work by stipulating how entities in the SOS are located and used. Cooperative applications do not have to deal with the complexity of being aware of other cooperative applications that are competing to use the same resources, or having the knowledge of how the organisational environment works. This reduces the apparent complexity of the SOS and provides application support for organisational independence, and support for inter-organisational work.

Finally, the SOL system has addressed the need for an interface management service that supports the shared usage of applications. For the SOS architecture this idea has led to the provision of access points as part of the system that allows applications and interface toolkits to subscribe to object related events which in turn enables applications to delegate the display of the cooperative state of objects out of the application into the responsibility of the interface service.

The proposed reference framework is intended to be open and extensible for additional services that may be added as an integral part of the system. In the following section the architecture of the SOS is presented. The functional components in the different layers are described and it is shown how the CSCW specific issues result from the experience of the prototypes.

16.2 SOS Architecture

The SOS is a distributed service platform that provides access to objects that encapsulate the functionality which has been addressed in various input papers of the COMIC project and which is intended to effectively enable the development and management of systems for the support of cooperative work. The object oriented approach has proven to be appropriate in order to achieve systems that are easily extensible. Object oriented methodology allows for structuring of applications, so that their components are easily identifiable and modifiable and thus enables a better maintenance and support of large systems, while at the same time reducing the cost of development. The object server concept has the additional advantage of being easily portable across platforms, since object interoperation takes place via platform independent object interfaces. Implementation details as well as the location of objects are hidden; objects and services may be used on a local machine in the same way as distributed over a wide area network.

16.2.1 Objects and Services

Objects form the basic components that constitute the SOS. They consist of an interface that offers a set of methods (services) and an implementation which is hidden from clients using them. An object may be persistent, i.e. it continues to exist in the SOS after a client has used it, or it may be transient, in which case it is only temporary created for some specific needs of the client and automatically deleted afterwards.

The services an object provides may be used in a given programming language by applying a language specific translation of the object's interface. An object invocation mechanism[1] of the SOS is then responsible for locating the actual implementation of the server object, which performs the required computation. The results of this computation, if any, are afterwards transported back to the client object by the invocation mechanism. The physical location of the server object is transparent, i.e. the client does not need any knowledge of where the server object can be found. It may reside on the same machine or exist remotely, accessible over a network.

The notion of objects in the SOS is very general. There is no formal distinction between transient objects that live only to perform some task, similar to the notion of data structures in programming languages, and persistent objects that keep their internal state until the next invocation; the same holds for whole applications and programs. All these entities are simply objects in the user's view[2].

16.2.2 Service Layers

The architecture of the SOS consists of three layers. The lowest layer provides the basic access and management facilities that are implemented by the service. A second layer of core services allows applications to use the mechanisms that enable specific support for cooperative applications. An extensible set of common services forms the upper layer and implements the general-purpose utilities for administration and user tailoring in the SOS. This is shown in the following figure:

[1]It is not the goal of the SOS to develop such a mechanism (See section 16.2.3).
[2]We will however distinguish applications developed in the framework of the SOS from regular service objects provided by the SOS although they are objects in their own right.

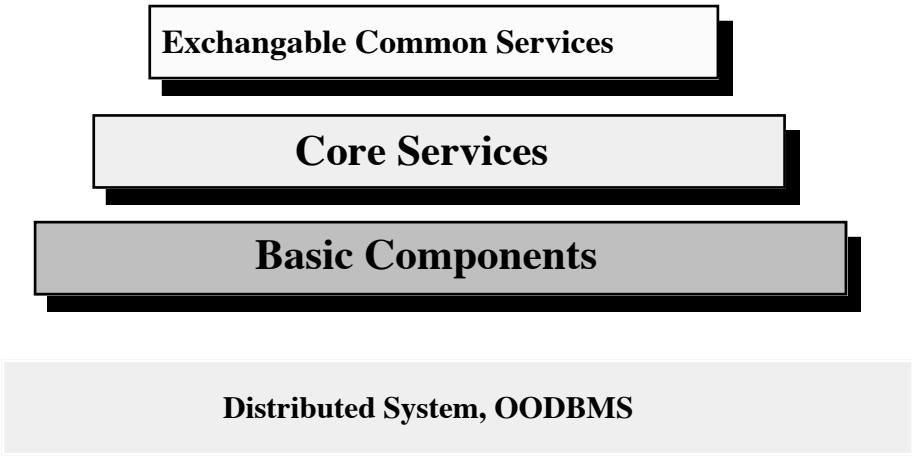


Figure 16.1: The layer view of the SOS

The layered approach gives application developers a way to structure the functionality of systems. However the distinction of the three layers should not be confused with a hierarchical (client-server) model: there is no limitation restricting access of objects in a given layer to the layers below. Additionally, they may interoperate directly with the underlying distributed platform. As an example, consider an event information service specific to a certain work domain being built on top of the SOS as an additional common service. The implementation of this service would make use of the basic components, e.g. define its own event types as subclasses of existing events or use object adapters and associations to implement additional object relationships and event distribution policies. The service may then use the event manager from the core services layer to handle the event distribution and notification.

Objects generally may be clients and servers simultaneously. The classification of services into three groups is solely intended to make the presentation more understandable by categorising the objects according to their level of granularity. New objects may be created based upon already existing objects. In this way the basis of services becomes broader as new objects are introduced; systems extensions do not have to be designed from scratch. Furthermore the object layers provide a means for delegation: an enhancement of object functionality may be implemented by means of a new object that contains an interface for its own new functionality as well as for the old object. Although there are now actually two objects, there is only one in the client's view, and services that rely on the old object now inherit the functionality of the new object.

16.2.3 Relationship to the CORBA Standard

The layers of the SOS are based upon another layer which supplies the functionality for distribution and object persistency. There is currently a range of distribution platforms as well as object oriented database management systems available that are suited for this purpose. However, the need for new approaches

in the distributed management and sharing of information is not specific to the domain of cooperation support. The CORBA object standardisation effort [OMG, 91; OMG, 92] of the *Object Management Group* (OMG) is currently developing a general model for standardised access, distribution, and interoperability for objects, which has already led to the first commercially available distributed object systems. We believe such an implementation would be a natural environment for the realisation of the concepts that are presented here, and in fact one of the prototypes, the GroupDesk system, has been implemented by using the CORBA compliant object platform ORBIX from IONA Technologies Ltd. This strategy also allows the development of the SOS to concentrate on the implementation of the CSCW services that are described here because issues of distribution and platform independence are already handled by the underlying CORBA implementation.

CORBA, as a general-purpose object system, covers a wide range of object related services that are of common importance for applications (e.g. object life cycle services). We will thus not address these kinds of services here and instead concentrate on the description of the novel concepts, which we believe to be necessary in order to make a collaborative object service out of a common object service. Furthermore, the CORBA model addresses many issues that are closely related to several services presented here (e.g. events, persistent storage manager). In these cases we will show how both models interrelate and how the additional functionality of the SOS service is provided.

16.3 SOS Services

In the following, it is shown in some detail how the different layers appear and how applications built on top of the SOS benefit from the services they provide. The description concentrates on the novel CSCW specific issues that have been investigated in the prototypes. The reference framework is intended as a design perspective which takes into account the experience resulting from these prototypes, sometimes by extending the respective concepts.

16.3.1 Basic Components

The basic components of the SOS give applications access to a set of methods that provide the atomic functionality to share and maintain objects. They implement the specific concepts needed to support awareness and object sharing that have been developed in the framework of the COMIC project:

- object adapters
- events
- object associations
- persistent storage manager.

Most of these components appear to be included in the OMG standardisation effort as well. However, instead of having these concepts semantic free, as is the case in the CORBA specification (which is necessary for providing a general-purpose software development paradigm), we believe that for the purpose of the SOS they need to be enriched with additional cooperative semantics in order to meet the goals of distributed support of cooperation and awareness. Subsequently we highlight these services in the framework of the SOS.

Object adapters

Object adapters allow a clear separation of the management semantics of the SOS from the behavioural semantics of the objects. They generally provide a level of indirection between object interfaces and object users as a way to dynamically impose specific cooperative functionality to otherwise cooperation unaware object behaviour. The COLA system [Trevor, 93; Trevor, 94] has shown that using object adapters gives us the following advantages:

- *Object reusability and interoperability*: Instead of including too much cooperative semantics into the objects themselves, this functionality can be imposed from outside of the object by the use of adapters. This leads to more lightweight objects in the SOS and eases tasks like changing and exporting objects.
- *Flexibility*: Because of the simplicity of the mechanism that allows objects to be adapted to different contexts of use, object adapters can be used in the SOS as central components to define the functionality of the object service.
- *Evolutionary design*: Applications using the SOS, as well as the SOS itself, may be developed experimentally. Appropriate requirements for object sharing in specific cooperative settings may be evaluated and new functionality may be supported as it is identified.

The different services offered by the SOS will be implemented by appropriate adapters. This has been demonstrated in COLA with locking adapters and access control mechanisms. Similarly the event handling and awareness supporting facilities of the SOS can be modelled as adapters as well.

Note also, that the concept of object adapters in the SOS is fundamentally different from the concept of object adapters in CORBA. The CORBA basic object adapter (BOA) forms the management interface between object implementation and the object invocation mechanism (request broker). For example, it may be used to physically locate objects, or to perform tasks like process forking on object invocation, if required. Conversely, object adapters in the SOS form the interface between objects and the user. In some sense, they are the objects in the SOS. They determine what the user's view on the object is and have influence on the object's behaviour.

Events

Events are provided as basic components that carry information about the state of collaboration. They are generated by the SOS and get distributed according to the structure of the object repository and a set of distribution policies.

Events in the SOS play a central role concerning the issues of object sharing and awareness. In the GroupDesk prototype, events enable users' awareness of the following:

- Who is currently active in the system?
- What tasks are other users currently concerned with?
- What has changed in the environment since last access?

The SOS provides a number of predefined event types that are suitable to describe these CSCW specific situations. They generally may be divided into the following two basic categories: *modifications* and *activities*. Modifications describe changes in the state of objects. Activities describe operational relations between users and objects. Modifications do not have a temporal duration whereas activities do. In GroupDesk the support of awareness is demonstrated by the use of these two basic event types in an environment for document production and sharing.

Applications built on top of the SOS may derive their own event types with a more specific semantics that may be smoothly integrated in the default event distribution and notification services that are offered by the SOS. Similarly, the SOS may identify more basic event types additionally to the activities and modifications. A useful extension consists of the integration of exception events. They describe neither activities nor modifications but instead different kinds of abnormal situations ranging from standard systems exceptions like access right violations or object inconsistencies to cooperation specific situations such as distribution of work flow to absent users. These kinds of events would clearly form an important contribution to the support of awareness, allowing users a more dynamic and fast reaction on breakdowns.

The CORBA model also defines a notion of events, which would be suitable to use as the implementation basis for events in the SOS. See [Syri, 94] for a description how to implement SOS events in the framework of the OMG concepts.

Object associations

Object associations may be used to structure the objects that are maintained by the SOS. They form the means by which objects are embedded into an organisational, semantic, or operational context. Additionally, associations form the infrastructure for the event distribution facilities of the object server and may thus be regarded as the basic components that enable the provision of user awareness of the cooperative situation.

The GroupDesk system uses associations between objects in the above context. The system currently identifies two basic types of relationships described by

associations: *structural* and *operational* relationships. Structural relationships are used to structure the objects according to the organisational context; operational relationships are associated with users and represent relationships like presence (in a working context) and activity. In addition to these types of associations the SOS provides associations to describe semantic relationships between objects, i.e. semantic similarities between objects that may be different according the task to be done. A special type of associations is used to express interest relationships between users and objects. All these kinds of associations form the medium by which event distribution takes place in the SOS.

While associations in the OMG object standard are intended to provide an efficient means to aggregate objects (e.g. graphs and trees), associations in the SOS are enriched with more semantics. They determine which events to propagate and may perform certain modifications on them. Hence, they can be regarded as objects in their own right, since they encapsulate a behaviour. Relationships modelled as objects are also implemented in the organisational information system TOSCA [Prinz, 93].

Persistent storage manager

While many applications today implement object persistency by simply writing their state on disk, this is far from sufficient for CSCW systems. The experience with the KnowledgeNet functions in CoDesk have given evidence for integrating a persistent storage manager (PSM), that provides additional persistency services for fine grained objects like those created by programming languages like C++ and Smalltalk. The PSM is similar to persistent storage facilities that may be found in CORBA compliant systems, like DOE [SunSoft, 93]. However, it provides some additional features which will be described subsequently.

Unlike most other services that are accessed through an object adapter (OA) on the Shared Object Server, the PSM is designed to be used over an API for efficiency reasons. That means that type checking is not done over the OA interface rather than internally in the PSM. The PSM enables a smooth transition between persistent and transient object spaces, which has proven to be very useful in the CoDesk prototype.

Basically, the PSM defines two different means of using the services:

- Through an Object Oriented Database API.
- Through Object Oriented Database Adapters for persistent storage (OODA).

Another service provided by the PSM is the registration of triggers on objects. This mechanism can be used to define some action to be performed whenever the state of a particular object changes (e.g. do automatic consistency checks). This is also an additional means of providing awareness to the users of the SOS, since the trigger may issue notifications about the state change. Triggers are specified within the class definition, as follows:

```
class A {
  some-list
```

```
  void do_something();
Public:
  Name aname;
  Status mode;
  void update(Status xmode);
Trigger:
  Perpetual subscribe() : changed(mode) ==> do_something();
}
```

Figure 16.2: A Perpetual trigger definition for a subscription.

Another major criticism of current database management systems is their lack of capability to handle arbitrary search-and-retrieve functions. A rich set of iterators that also allows the expression of recursive search-and-retrieve functions is needed.

16.3.2 Core Services

The Core Services consist of different managers that enable a more convenient access for applications to handle the CSCW specific tasks that enable user awareness and sharing. They consist of the concepts that have been addressed in detail in the various chapters of the SOS part of the deliverable and which will be demonstrated in the prototype videos associated with this deliverable:

- the resource manager
- the Trader
- the locking service
- the event manager
- the naming manager.

The resource manager

Resource management is the task of scheduling access to resources in a cooperative environment in a fair and organised way. This implies the application of knowledge about the organisation and the relationships among entities where work is situated. The resource manager is an entity which mediates and supports articulation work situated in a specific organisational context. It applies and observes organisational policies to establish bindings among entities in a continuously changing and potentially very large environment where entities are scarce, and thus where there is competition to use them.

The resource manager is the referee of conflicting requests coming from different agents that must be resolved in accordance with the organisational policies. It has to know about any entities in their environment and how those entities are structured. It is also responsible for keeping track of the dynamics of the environment and adapting to it. Therefore, it reduces the complexity of articulation work by stipulating access to resources, i.e. Reducing local control

that actors have over articulation, or supporting articulation work with incomplete information.

Two functions are especially important in resource management: binding and trading. Binding is a function required to establish relationships among entities in a proper way. Binding two different entities frequently requires the insertion of cushion objects (object adapters or interceptors) to adapt, expand, transform, monitor, supervise or co-ordinate the interactions between entities. The binder is responsible for establishing contacts. It decides upon how entity relationships have to be configured in order to comply with the requirements of the parties involved and the organisational requirements for supervision.

The design of the resource manager component of the aleph prototype is described in [Rodriguez, 94]. It provides a mechanism that takes into account inter-organisational cooperative work. This is a design issue currently being addressed and support for establishing federations (a form of non-hierarchical organisation) is currently under development [Navarro, 94].

The Trader

Trading is the task of determining the most adequate server for every request at a given time. In a large scale organisation, entities change the way they work, the service they provide and possibly their location. Additionally, new entities may appear dynamically.

The trading service in the SOS is responsible for finding resources that best fit to specific needs (which can be specified weakly and incomplete). It has to resolve changes of name, location, and even destruction and creation of resources. On one side, users provide a descriptive name of the entity they need, on the other side, there are entities that may be contacted. The interaction between both sides is mediated by the Finder. The Aleph prototype includes a descriptive naming mechanism to specify to the Finder the requirements of potential entities to contact, isolating applications from a changing environment.

The locking service

Locking of information is of central importance when it comes to coordinating access of information in a multi-user setting. It generally ensures the integrity of the shared objects by serialising simultaneous accesses. The traditional locking approaches incorporated by the reserve-deposit paradigm however are not powerful enough if applied in a cooperative environment: work inherent parallelism often cannot be exploited by following the primacy of conflict avoidance. Shared awareness in the collaborative access of information needs additional semantics incorporated in the locking facilities of the system.

In the SOS, locking facilities are implemented simply as object adapters, as is demonstrated by the COLA prototype. Different locking adapters are provided by the object service that enable the application of different locking strategies.

Access requests for objects are mediated through the locking adapter. The adapter in turn encapsulates state information as well as the respective locking strategy.

Embodying the locking mechanisms in a set of adapters allows us to experiment with different locking policies. Furthermore, we may attach any additional semantics to the locks: e.g. “identified” locks [Mariani, 93] may be achieved by adding the user’s identification to the lock’s state. We may also imagine situations, where it makes sense to add information about the reason why this object is locked. In this way, the locking service of the SOS significantly contributes to the support of shared awareness.

The event manager

The event manager handles the generation and administration of events that occur in the cooperative access of the SOS. The administration consists of a distribution of events in the object repository and the persistent storage of events. Event administration is basically dependent on two factors: the event’s context and its validity.

The context of an event determines the set of objects that are associated with that particular event which may be different, depending on the type of event (see e.g. [Fuchs, 94b; Fuchs, 94c] for a more detailed description). In general the event manager determines the context by distributing the event according to the set of associations originating at the object in question and the type of event that occurred in a depth first search manner.

The validity of events also differs depending on the type of events: activities are only valid during the time in which they really happen whereas modifications are valid as long as the user did not receive a notification about them. According to the validity the event manager is responsible for storing the events in object dependent event repositories and accesses the information in these lists on client request.

Current experience with the groupdesk prototype has shown the need for clients to be able to tailor the distribution of events according to individual preferences. The SOS thus allows for a dynamic configuration of the event manager by means of a set of distribution policies that a client may define. The policies basically consist of rules that define the association types that may be used to transport a given type of event. The rules may also define an intensity decrease which allows the distribution to be locally restricted as well as a user identification to enable individual user tailorable event distribution.

The naming manager

Names allow us to identify objects, talk about them and to share them. Usually naming models for distributed systems try to overcome ambiguous and multiple names by using a hierarchical design, as e.g. In the UNIX file system. However this is not compliant to human use of names; names are usually not unique, people are aware of the ambiguous use of names and that names change continuously as

the environment changes. With multiple names for a single object the user has the freedom to divide the object space into different contexts, e.g. Dividing a single work activity into several different projects.

Naming in the above sense already forms a component in currently developed prototypes of the OMG object standardisation. For example, the naming manager of SunSoft's doe system provides the functionality to bind a name to an object relative to a name context. A name context is an object that contains a set of name bindings in which each name is unique. To resolve a name is to determine the object associated with the name in a given context. Through the use of the naming manager, name manipulation is simplified and can be made representation-independent, thus allowing their representation to evolve without requiring client changes. For more details, see [SunSoft, 93].

The use of common names is a key factor in group cohesion. Therefore it is envisaged in the framework of the CoDesk shared object services to implement mechanisms for name proposals:

- Name proposals given a new object.
- Name proposals given a new user.

The first alternative is useful in situations when a user wants to merge an object into a new context, in order to find an appropriate name in that context. The second alternative is relevant for novice users; as objects may have multiple names, it makes sense to offer name proposals to the user for such objects.

The COMIC shared object server will be based on the naming service offered in CORBA compliant platforms and additionally integrate the concept of name proposals as described above.

16.3.3 Common Services

This layer is intended to be extensible and the services it provides should be exchangeable. Thus the services in this layer in general have the nature of CSCW utility programs rather than methods that may be imported by clients. Depending on the concrete tasks to be done in the environment, work specific services may be added. Here, we will only briefly introduce the history service and the event notification service, as their design is strongly influenced by the experience of the prototypes. Additional services that this layer might contain are a versioning service, synchronous and asynchronous communication facilities (email, conferencing software, etc.), general administration utilities or an organisational information service. An example of the latter is described in [Prinz, 93].

History service

The history service of the SOS enables users to get an overview of the state of affairs in asynchronous cooperation, i.e. It provides summaries for activities and modifications on objects that happened in the past.

Groupdesk currently implements a history service that has proven to be useful in cooperative situations. The interesting aspects differ depending on the type of object in question, e.g. The history of a document gives information about recent changes, who changed it, how much time a particular user spent on working on it etc., Whereas the history of a container object gives information on new objects created, overall activities or workload and which users have visited it.

For the SOS the information provided by the history service consists of a compression of the events that happened for the object in question in order to enable a true overview rather than only a list of all fine grained events. The service is user tailorable along multiple dimensions: clients may specify the timescale of information, the level of detail and granularity as well as the presentation on the GUI.

Furthermore, the types of events that the user is interested in are dynamically tailorable. For example, the service enables a view on the history of an object concerning all activities the object was involved in, leaving the modifications of the object aside. A more elaborate description of the architecture of the SOS history service can be found in [Fuchs, 94a].

Event information service

This service is provided in order to enable users to tailor the event related services of the SOS in a convenient and intuitive way. It provides features that let the user specify individual preferences concerning the distribution and aggregation of events in the core event manager, as well as preferences for the visualisation of event notifications at the graphical user interface.

Another functionality offered by the event information service in the SOS is the possibility for users to define a set of objects they have a special interest in. For such an object, users can define the event types they wish to be notified of.

For such a specification, the event manager establishes an interest relation between the object and the user. The interest relation takes account of all the event types the user is interested in. The standard event distribution mechanism will subsequently distribute such events along this relation such that only relevant events will lead to a notification of the user. This is an important feature of the event related services in the SOS, which on the one hand contributes to avoiding information overload and on the other hand allows for an integrated treatment of synchronous as well as asynchronous event notifications.

16.3.4 Interoperability with a Shared Interface Service (SIS)

Despite the availability of sophisticated user interface builders the design of user interfaces for cooperative applications still is a tedious task. This is due to the fact that interface toolkits are tailored to the development of single user applications. They restrict interfaces to react solely to fine grained events (mouse clicks, keyboard input, etc.), that come from a single workstation, usually the one that is running the application. Formally, we could describe their behaviour in terms of a

deterministic finite automaton, that gets events from a workstation and changes state according to the input received. Each state transition may potentially cause some visual change on the corresponding screen, such as opening a window, or highlighting a menu entry. The fact that there may exist multiple windows on a single screen does not form an essential improvement to this mechanism, it simply means that there may exist multiple automata simultaneously.

For two reasons, this approach is insufficient for the support of collaborative applications. The first reason may be seen in the necessity for CSCW applications to be shared among users. Although this is in principle state of the art in most window systems, it still is still not enough for collaborative environments as since the behaviour is strictly WYSIWIS. An effective support for shared applications needs to provide means that let the interface of the participants react in different ways depending on certain conditions in the cooperative setting. An example for such a condition is the role of the participants, granting a different level of access to the functionality of the application, which in turn needs to be reflected in the visual appearance of the interface. The second reason lies in the restricted notion of events that interface toolkits are based on. Instead of forcing application developers to design interfaces in terms of mouse clicks and pointer movements, it should additionally be possible to specify events with a higher semantics, describing changes in the cooperative setting. Such events are typically raised inside of the object space rather than coming from outside of the system. With such an enhancement, many tasks in the design of interfaces, e.g. presentation of documents, workspaces etc. can be essentially eased, while at the same time integrating facilities that support shared awareness by reflecting their cooperative state (e.g. shared, in use, modified etc.).

To address the above problems, the SOS implements a close interoperation with a Shared Interface Service (SIS) which enables a delegation of managing the cooperative issues in the user interface out of the responsibility of the applications. These issues consist of two general kinds:

- Support for joint usage of applications.
- Support for awareness concerning object sharing.

The SIS facilities for enabling collaborative aware joint usage of applications have been described in detail in [Smith, 94] and are demonstrated in the SOL prototype. In addition to that, a coupling of the SOS to the SIS allows us to design collaborative aware interface toolkits which let the user interface react not only on the standard interface events but also on collaborative events.

In order to achieve this behaviour, the interface objects can be attached to SOS objects, so that the SIS updates interface objects according to a set of (interface) resources every time a cooperative event occurs in connection to that particular SOS object. This gives applications a powerful way to present information contained in the SOS (e.g. a printer chooser could represent the different possible choices in different colours, depending on the size of the printer queue without the application having to care for the updates in the interface).

Furthermore, the SOS offers an access point where the SIS may register its interest and specify the events it wants to be informed about. The access point in turn provides an event queue with an entry for each event that matches the specification. These events may be treated by the SIS in exactly the same manner as usual interface events. User tailorability may be performed also in accordance to the usual description of interface preferences. E.g. in the X-Window system via resources that specify colours, sizes of objects etc. for presenting objects in a specific cooperative state.

The following figure thus gives a global view on the SOS in conjunction with the SIS:

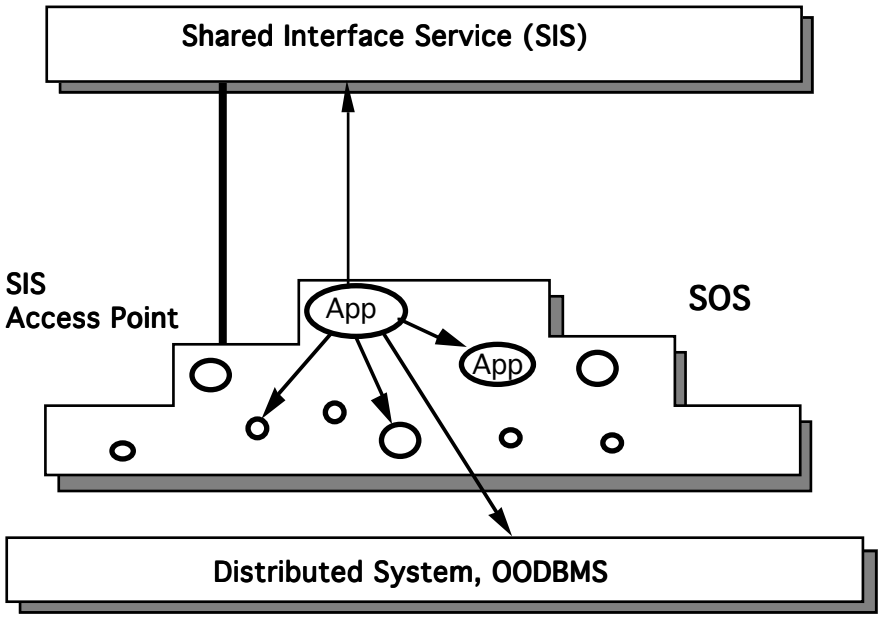


Figure 16.2: SOS-SIS Interoperation

A new application is implemented as an additional object in the SOS. The application may use any services from existing objects, and offers its own functionality to other objects as well. It may access high level services in the SOS in the same way as it can use the primitives in the basic components layer or in the underlying distributed system. If the application implements a user interface, it can use the interface objects offered by the SIS. By attaching interface objects to SOS objects, the SIS can handle interface updates according to the cooperative events managed in the SOS event services which are registered and distributed via an SOS access point to the SIS.

16.4 Conclusion

The SOS is an object oriented service platform for the specific support of CSCW environments and application development. The architecture consists of three service layers, that provide access to objects in different levels of abstraction and

granularity. The lowest layer consists of basic components such as events, object adapters and the functionality for object persistency. The core services layer implements the management facilities which have been identified as central components for the support for object sharing and user awareness in various input documents in the framework of the COMIC project. The upper layer of the SOS consists of a set of high-level services that may be exchanged and extended as needed in a specific work environment. Special attention has been paid to the problem of support in the development of graphical user interfaces for CSCW applications. A close interaction between the SOS and a shared interface service eases the development and augmentation of applications that are to be synchronously shared between users. The whole system is intended to be based on a CORBA compliant distribution platform and the services we have presented are extensions of the CORBA model. They encapsulate the CSCW specific functionality which we found necessary as a result of the experience in the development of the prototypes which are demonstrated in this deliverable.

16.5 References

[Fuchs, 94a] Fuchs, L. and Prinz, W., *Supporting User Awareness with Local Event Mechanisms*, COMIC Document GMD-4-6, 1994.

[Fuchs, 94b] Fuchs, L., Pankoke-Babatz, U. and Prinz, W., Ereignismechanismen zur Unterstützung der Orientierung in Kooperationsprozessen, to appear in *Proc. German Conference on Computer Supported Cooperative Work*, Marburg, September 1994.

[Fuchs, 94c] Fuchs, L., Pankoke-Babatz, U. and Prinz, W., *Organisational Context, Events and Awareness,* COMIC Document GMD-1-10, 1994.

[Mariani, 93] Mariani, J. and Prinz, W., Awareness in Collaborative Object Systems, in *Requirements and Metaphors of Shared Interaction*, Chapter 11, COMIC Deliverable 4.1, 1993, 243-255.

[Navarro, 94] Navarro, L., *Ecology in Global Distributed Systems*, COMIC Document UPC-4-4, 1994.

[OMG, 91] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG TC Document 91.12.1, 1991, available on request from the Object Management Group.

[OMG, 92] Object Management Group, *Object Management Architecture Guide*, edited by R. M. Soley, OMG TC Document 92.11.1, 1992, available on request from the Object Management Group.

[Prinz, 93] Prinz, W., TOSCA: Providing Organisational Information to CSCW applications, *Proceedings of the Third European Conference on Computer Supported Cooperative Work - ECSCW '93,* G. de Michelis, K. Schmidt, and C. Simone, eds., Kluwer, Dordrecht, 1993, 139-154.

[Rodden, 92] Rodden, T., *COMICal Issues on a Shared Object Service*, COMIC Document LANCS-4-2, 1992.

[Rodden, 93] Rodden, T., Mariani, J., Eiderbäck, B. and Hägglund, P., Requirements for the COMIC Shared Object Service, in *Requirements and Metaphors of Shared Interaction*, COMIC Deliverable 4.1, 1993, 203-242.

[Rodriguez, 94a] Rodriguez, G., *The design of the Resource manager and the Trader*, COMIC Document UPC-4-3, 1994.

[Smith, 94] Smith, G. and Rodden, T., *SOL: A Shared Object Toolkit for Cooperative Interfaces*, Chapter 9 in this Deliverable.

[Sundblad, 94] Sundblad, Y. and Tollmar, K., *The Collaborative Desktop – Experience from Designing and Building an Environment for CSCW*, COMIC Document KTH-4-16, 1994.

[SunSoft, 93] SunSoft Corp., *Project DOE: Future Solaris Technologies,* Technical White Paper, 1993

[Syri, 94] Syri, A., *Realisation of the COMIC Shared Object Server on Basis of CORBA*, COMIC Document GMD-4-7, 1994.

[Trevor, 93] Trevor, J., Rodden, T. and Blair, G. S., COLA: A Lightweight Platform for CSCW, *Proc. 1993 Conference on Computer Supported Cooperative Work,* Milan, Kluwer, 1993.

[Trevor, 94] Trevor, J., Rodden, T. and Mariani, J., The Use of Adapters to support Cooperative Sharing, to appear in *Proc. 1994 Conference on Computer Supported Cooperative Work,* Chapel Hill, NC, ACM Press, October 1994.

Chapter 17

Objects in Space, the Spatial Model and Shared Graphs

Tom Rodden

Lancaster University

This chapter addresses the relationship between shared objects and the work of the spatial model. In particular, we consider the way in which shared objects can be used to develop a general model of space and outline particular specialisations from this general model. For example, we demonstrate how the spatial model can be mapped onto general shared graphs. To do so we introduce a mathematical consideration of graph structures. We use this to model the sharing of these general graph structures using concepts drawn from the spatial model. Finally, we consider some potential interpretations of the general shared model when applied to cooperative applications.

The net result of our consideration is the formulation of a generally applicable mathematical model of space and awareness that can be used to reason about the cooperative use of a range of cooperative applications.

17.1 Introduction

In this chapter we wish to consider how existing notions of the spatial model may be mapped more generally to CSCW applications. The intent here is not to gain an exact correspondence between concepts in the spatial model and elements of existing cooperative applications. Rather we wish to examine the ways in which we can use the core elements of the spatial model to support views of CSCW systems. The principle focus in this endeavour is to develop richer means of reasoning about the awareness between users sharing a cooperative application which can be represented as a collection of shared objects. By allowing awareness across a system to be based on a simple set of spatially motivated concepts we can hopefully provide the basis for a supporting theory for future cooperative application developers.

Rather than consider different mappings to particular applications we wish to develop a view of the spatial model that is generally interpretable across a number of cooperative applications. The starting point for this exercise is a consideration of a how we represent a space shared by a community of users. Our notion of a shared space focuses on modelling the presence of users and objects within the space and the proximity of users to other objects occupying the space. This representation of a shared space provides the most general formulation of the spatial model. This formulation can then be specialised to exploit the nature of

particular spaces. In this chapter we consider two particular specialisations of the general model. The first considers how the model can exploit the geometric properties of a shared Euclidean space. The second considers the application of the general model to the shared graph structures commonly found in a range of CSCW applications.

Our consideration of the spatial model concludes by discussing how we may interpret the model across a range of cooperative applications to derive some theoretical consideration of how a cooperative system is being shared by a number of interacting users. In practice, the use of the spatial model in this way allows us to consider the awareness of users cooperating across a number of shared objects. These objects may be configured in such a way that they make up a shared cooperative applications or exist within a shared information repository. We consider the collection of objects as representing a shared space onto which users project their presence.

17.2 Users of a Shared Space

Our starting point for a general spatial model is the representation of space as a collection of objects shared by a number of users. These objects may either exist within an externally defined geometric space or actually constitute the space. We wish to reason about the relationships which form between objects in the space and consequently focus on the shared objects within the space represented by the simple pair

$$SS(U, O)$$

The shared space SS(U,O) associates a set of objects O (determining the space) with a set of users U. The set of users U is a heterogeneous collection of objects that may represent users, groups of users or any object capable of exerting an active presence. This association between users and objects in the space takes the form of a number of mappings which are central to reasoning about the relationships between users sharing the space.

17.2.1 Presence Positions

The presence of user in the space is represented in the general model by the location of the user and a surrounding set of adjacent objects. This *presence position* is represented by the simple pair

$$P(pos, Adj)$$
$$\text{where } pos \in O \text{ and } Adj \in \wp\, O$$
$$\text{note: } \wp \text{ – powerset.}$$

The mapping of user to an object in the space allows us to model the points of presence of users in the space. Position objects represent one of the heterogeneous forms of object within the space. These position objects may correspond to other

objects in the space and may or may not be reflected by some form of embodiment object within the space. This representation of the presence of users in the space forms the basis for representing both focus and nimbus. In the most general case of the spatial model we do not consider restrictions on membership of the adjacency set associated with a particular position point. However, more specialised cases of the model rely on specifying more precisely these features of the model. One particular case we consider in more detail later in this chapter is using the properties of networks to determine the membership of the adjacency sets.

We shall call the set of all presence positions for a shared space the presence set, PP(O) or *presence space*. Where

$$PP(O) = \{P(o,A) | \ o \in O, A \in \wp O\}$$

We refer to the particular parts of the pair using a simple bracket notation. Where P is a presence position

pos(P) is the position object
Adj(P) is the adjacency set

17.2.2 Nimbus

A function nimbus exists which maps the set of users to the shared space. This function effectively places a user in the space. The function is intended to be used to map a user to many potential presence positions in the space. This reflects the most general case of a user having a number of nimbus positions in shared space. Restricting the function to allow only one nimbus position would yield only one of the more specialised examples of the spatial model (discussed below). The general nimbus function is of the form

$$nimbus: U \times Id \rightarrow PP(O)$$

The linking of users to the space using a nimbus function exploits a label that identifies the different nimbi associated with a user Id is a set of arbitrary labels that can be associated with different instances of the nimbus function.

17.2.3 Focus

A function focus exists which maps users to a position point in the space. As in the case of nimbus the focus function uses an arbitrary label to allow a user to have a number of associated foci. The general focus function is similar in structure to the nimbus function and is of the form

$$focus: U \times Id \rightarrow PP(O)$$

The general definition of space, focus and nimbus takes an object centred view of the world. It sets out both the focus and nimbus in terms of the objects of interest in the world. The determination of what these objects are for any given

adjacency set may exploit more complex techniques, for example, Cartesian geometry, field theories and network topologies.

17.2.4 Location, Aggregate Focus and Nimbus

The focus and nimbus functions map a user to a number of focus and nimbus points across the space. It is often essential to reason about the collective or aggregate nimbus and focus of users formed by this set of position points. Three functions can be used to reason about a user location, aggregate nimbus and aggregate focus.

location: $U \rightarrow \wp\ O$
location(u) ={ g| g ∈ O and (pos(nimbus(u,id)) = g or pos(focus(u,id)) = g) }

Agg_nimbus: U -> $\wp$ O
agg_nimbus(u) ={g| g ∈ O and (g∈ adj(nimbus(u,id)) }

Agg_focus: U -> $\wp$ O
agg_nimbus(u) ={g| g ∈ O and (g∈ adj(focus(u,id)) }

17.2.5 Restrictions on the Spatial Model

So far we have outlined the most general form of the spatial model. We can restrict the properties to reflect more specific instances of the spatial model without needing to consider the specific nature of the shared space. In particular it is worth considering the case where the general model is restricted by:

- Co-locating focus and nimbus.
- Making focus and nimbus unique for a given user.

Each of these restrictions are briefly considered below. Where possible we highlight the implications of these restrictions for the resulting models.

Single user location

Our first restriction to the general model is to limit the focus and nimbus points for a given user to the location. This restriction is represented as :-

$\forall_u \in U\ (\forall_{I1} \in Id, \forall_{I2} \in Id$ pos(focus(u,i1))= pos(focus(u,i2)) and
pos(nimbus(u,i1)) = pos(nimbus(u,i2)) and pos(nimbus(u,i1))= pos(focus(u,i1)))

This single location for a user's nimbus and focus may be closely associated with the embodiments exploited in a number of versions of the model. In fact, this restriction allows us to represent each user as a single location in the space with a number of independent focus and nimbus related to different media. This restriction of the general model is most closely associated with the form of the spatial model outlined in Deliverable 4.1.

Unique focus and nimbus

A further restriction of the general model is to restrict users to having a unique focus and nimbus associated with them. This view of focus and nimbus simplifies the calculation of users' awareness levels. This is analogous to considering a user's focus and nimbus for a single media. This is represented as:

$$\forall_{u} \in U\ \forall_{I} \in Id\ \text{nimbus}(u,i).\ \text{not}\ (\exists_{i2} \in Id\ \text{nimbus}(u,i2)\ \text{and}\ i \neq i2)$$

$$\forall_{u} \in U\ \forall_{I} \in Id\ \text{focus}(u,i).\ \text{not}\ (\exists_{i2} \in Id\ \text{focus}(u,i2)\ \text{and}\ i \neq i2)$$

17.2.6 Awareness

The principle aim of the spatial model is to allow us to reason about and represent the awareness of users who share a common space. In doing so we wish to represent some continuous quantitative notion of awareness which indicates whether users are strongly or weakly aware of each other. On some occasions we also wish to represent some qualitative aspects of awareness in a more discrete form. The general model allows this by providing two basic forms of awareness and using separate functions to represent both the continuous and the discrete forms of awareness.

A continuous view of awareness

One view of awareness between users is in terms of the strength of awareness. An awareness function can be defined which maps from two users of the shared space to the set of natural numbers. The value of the awareness function reflects how strongly aware users are of each other. This is expressed as:-

$$\text{awareness\_strength: } U \times U \rightarrow N$$

A number of potential rules can be constructed for an awareness function with this signature. Each of these different awareness functions may be fairly complex and may in turn exploit different particular features of space over which awareness is being calculated. In our general case of the spatial model we focus on the objects in space as a means of determining awareness between users.

In the case the awareness function exploits the overlap between focus and nimbus to gain some continuous measure of awareness. Using focus and nimbus we can derive an awareness set which represents the overlap between the focus and nimbus sets associated with a user. The size of this set and its common membership is useful in reasoning about the strength of awareness.

The awareness overlap for two users u1 and u2 is deemed to be the set:-

$$\text{aware-over}(u1,u2) = (\text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2)) \cup (\text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2)) \cup (\text{agg\_focus}(u1) \cap \text{agg\_focus}(u2)) \cup (\text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2))$$

We can use the size of the awareness overlap, aggregate focus and aggregate nimbus to determine some notion of the strength of awareness between two users in terms of an integer value. A number of alternative formulations of a rule for an

awareness strength function can be derived. For example, we can produce a rule for the awareness_strength function which normalises the strength of the awareness function to have values between 0 and 1. The interpretation of these values is specific to the particular demands of an application.

$$\text{awareness\_strengh}(u1,u2) = \frac{|\,\text{aware\_overlap}(u1,u2)\,|}{|\,\text{agg\_focus}(u1) \cup \text{agg\_nimbus}(u1)\,|}$$

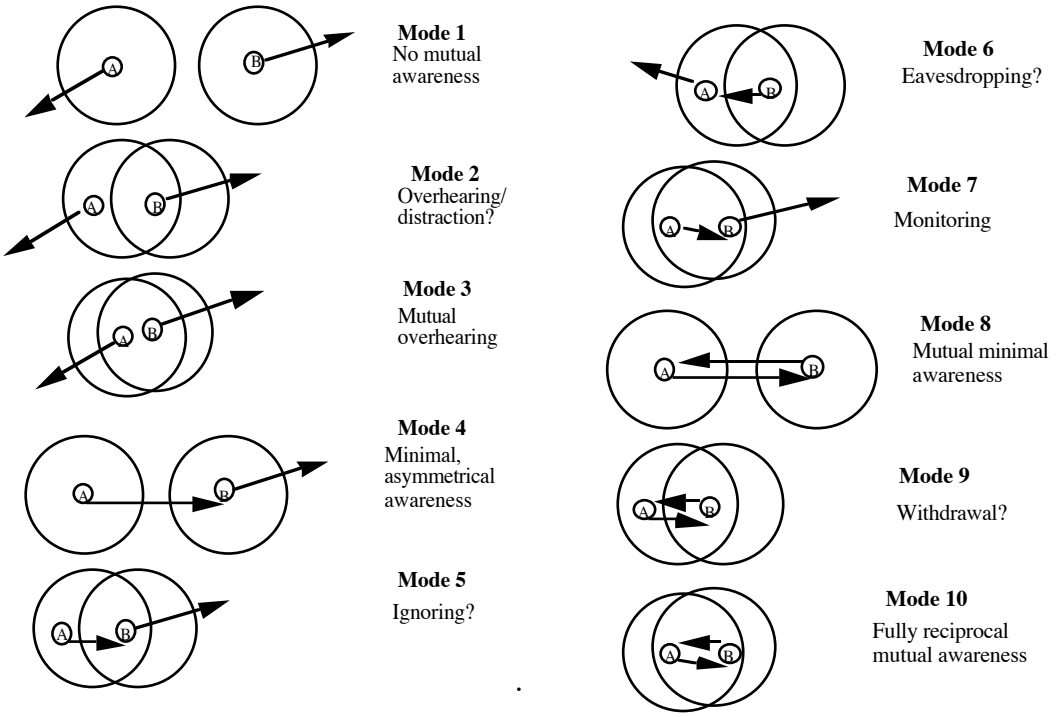
It is worth noticing that this form of the awareness function is asymmetrical in the strength of awareness user one has of user two is not necessarily the same as the strength from user two to user one. It is also worth stressing that this is only one potential form of function for awareness strength. It is almost certain that others exist which exploit the properties of a particular space. The discovery of particular functions for awareness_strength are in fact a matter of empirical investigation.

A discrete representation of awareness

We have defined the notion of a shared space populated by a set of users. For each user we have described associated definitions of focus and nimbus which allow us to consider a definition of an awareness function which can link users. In addition to the continuous case outlined above, the awareness function could be defined to map to a set of values defined from an awareness mode set. Consider the set of modes defined in terms of the placement of users in other focus and the overlap of focus and nimbus. We can define

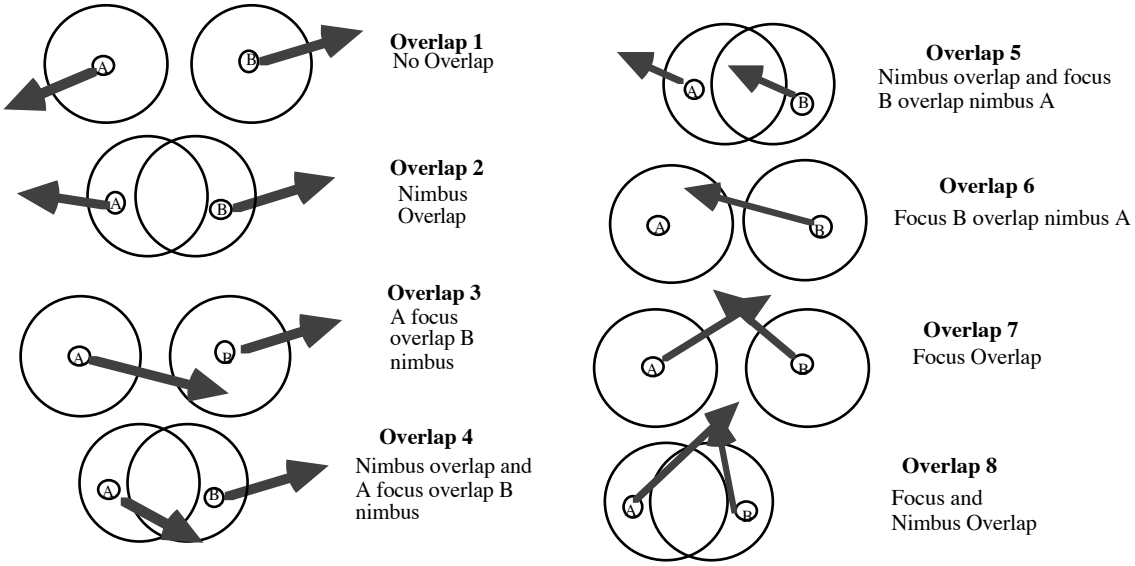
A = { mode1, mode2, mode3, mode4, mode5, mode6, mode7, mode8, mode9, mode 10, overlap1, overlap2 overlap3, overlap4, overlap5, overlap6, overlap7, overlap8}

Each of the elements of the set of awareness A represent the 10 different modes of awareness outlined previously in the spatial model and a further three which represent the overlap of focus and nimbus associated with two bodies. These are briefly reviewed in the following diagrams. In each figure the nimbus is indicated using a circle while the focus is shown as a directed arrow. Each of the modes shows a potential interpretation of each of the arrangements.



.

In addition to the ability to model focusing and nimbusing on bodies within the network we also wish to consider as part of our general spatial model the overlapping of focus and nimbus in the network. The following 8 modes can be recognised for the overlap of focus and nimbus.



These general cases of the spatial model maps the presence of users in the space to 18 different forms of awareness. This is achieved using a modal awareness function. A particular awareness function is defined which maps from two users of the shared graph to the awareness set A. This is expressed

$$\text{awareness\_mode: } U \times U \rightarrow A$$

The rule for the awareness function is defined

$$\text{awareness\_mode}(u1,u2) = \begin{cases} \text{mode1 if } \text{location}(u1) \not\subset \text{agg\_nimbus}(u2), \text{location}(u1) \not\subset \text{agg\_focus}(u2), \\ \text{location}(u2) \not\subset \text{agg\_nimbus}(u1), \text{location}(u2) \not\subset \text{agg\_focus}(u1), \\ \text{mode2 if } \text{location}(u1) \not\subset \text{agg\_nimbus}(u2), \text{location}(u1) \not\subset \text{agg\_focus}(u2), \\ \text{location}(u2) \subset \text{agg\_nimbus}(u1), \text{location}(u2) \not\subset \text{agg\_focus}(u1) \\ \text{mode3 if } \text{location}(u1) \subset \text{agg\_nimbus}(u2), \text{location}(u1) \not\subset \text{agg\_focus}(u2), \\ \text{location}(u2) \subset \text{agg\_nimbus}(u1), \text{location}(u2) \not\subset \text{agg\_focus}(u1), \\ \text{mode4 if } \text{location}(u1) \not\subset \text{agg\_nimbus}(u2), \text{location}(u1) \not\subset \text{agg\_focus}(u2), \\ \text{location}(u2) \not\subset \text{agg\_nimbus}(u1), \text{location}(u2) \subset \text{agg\_focus}(u1), \\ \text{mode5 if } \text{location}(u1) \not\subset \text{agg\_nimbus}(u2), \text{location}(u1) \not\subset \text{agg\_focus}(u2), \\ \text{location}(u2) \subset \text{agg\_nimbus}(u1), \text{location}(u2) \subset \text{agg\_focus}(u1), \\ \text{mode6 if } \text{location}(u1) \not\subset \text{agg\_nimbus}(u2), \text{location}(u1) \subset \text{agg\_focus}(u2), \\ \text{location}(u2) \subset \text{agg\_nimbus}(u1), \text{location}(u2) \not\subset \text{agg\_focus}(u1), \\ \text{mode7 if } \text{location}(u1) \subset \text{agg\_nimbus}(u2), \text{location}(u1) \not\subset \text{agg\_focus}(u2), \\ \text{location}(u2) \subset \text{agg\_nimbus}(u1), \text{location}(u2) \subset \text{agg\_focus}(u1), \\ \text{mode8 if } \text{location}(u1) \not\subset \text{agg\_nimbus}(u2), \text{location}(u1) \subset \text{agg\_focus}(u2), \\ \text{location}(u2) \not\subset \text{agg\_nimbus}(u1), \text{location}(u2) \subset \text{agg\_focus}(u1), \\ \text{mode9 if } \text{location}(u1) \not\subset \text{agg\_nimbus}(u2), \text{location}(u1) \subset \text{agg\_focus}(u2), \\ \text{location}(u2) \subset \text{agg\_nimbus}(u1), \text{location}(u2) \subset \text{agg\_focus}(u1), \\ \text{mode10 if } \text{location}(u1) \subset \text{agg\_nimbus}(u2), \text{location}(u1) \subset \text{agg\_focus}(u2), \\ \text{location}(u2) \subset \text{agg\_nimbus}(u1), \text{location}(u2) \subset \text{agg\_focus}(u1), \\ \text{overlap1 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \\ \text{overlap2 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) \neq \emptyset, \\ \text{overlap3 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) \neq \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \\ \text{overlap4 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) \neq \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) \neq \emptyset, \\ \text{overlap5 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) \neq \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) \neq \emptyset, \\ \text{overlap6 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) \neq \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \\ \text{overlap7 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) = \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) \neq \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) = \emptyset, \\ \text{overlap8 if } \text{agg\_focus}(u1) \cap \text{agg\_nimbus}(u2) \neq \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_focus}(u2) \neq \emptyset, \\ \text{agg\_focus}(u1) \cap \text{agg\_focus}(u2) \neq \emptyset, \text{agg\_nimbus}(u1) \cap \text{agg\_nimbus}(u2) \neq \emptyset, \end{cases}$$

17.2.7 Aura

Continual recalculation of awareness functions across all combinations of objects is costly and often not scalable for large densely populated spaces. As a result some mechanism is needed for limiting the scope of such calculations. In the spatial model this is the role of aura.

An aura effectively allows the system to identify in advance which combinations of objects will have negligible mutual awareness and then to exclude these combinations from awareness calculations. One way of doing this in spatial systems is to have a simple (it needs to be simple to minimise computation) bounding space which is bigger than both focus and nimbus. Alternatively, aura may be related to some other system concept such as a

management domain which marks the borders between different areas. Thus, awareness computations would only be done for those objects within a common domain (many distributed systems embody this idea of a management domain).

In our most general formulation of the spatial model, the notion of aura is reflected as bounds on the functions defining the adjacency sets for the presence positions. For example, in a Cartesian based space we may express this boundary in terms of the objects which lie within a sphere of a given size. Alternatively, in a network space we may express the bounds in terms of the length of a particular path.

The means by which adjacency sets are defined are closely tied to the particular properties of the space within which objects reside. The means of deriving these sets are also the point where the particular nature of focus and nimbus are defined. In the following sections we shall consider how adjacency can be defined based on some of the current formulations of the spatial model.

17.3 Different Kinds of Space and Adjacency

In this section we wish to consider some of the different forms of space considered in the spatial model and how these can be used to construct the definition of adjacency sets associated with presence position. By using the particular properties of a space we can look at alternative ways to specify this. We begin our discussion by considering a definition of aura and focus which makes use of simple geometry and represents focus and nimbus as geometric spaces.

17.3.1 Focus and Nimbus in a Geometric Space

A popular way of expressing focus and nimbus in the spatial model has been to use simple geometry to define shapes associated with bodies in the space. These shapes represent the extent of focus and nimbus associated with the body. The adjacency of objects within the space are dependant on the particular geometric relation between objects and the shape of the corresponding focus and nimbus. Thus anybody within the space can be considered within or outside a focus depending on a set of geometric rules. For example, consider the formulation of the spatial model shown in Figure 17.1. The object has a single focus and nimbus associated with it and is in a space occupied by a number of other objects (each of which may have their own focus and nimbus). Objects are within or outside the focus or nimbus depending on their co-ordinate position in relation to the geometric shape defining the focus or nimbus.

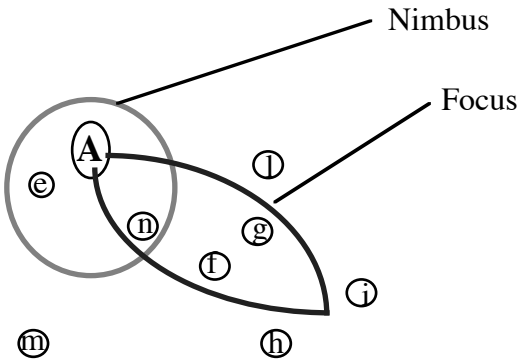


Figure 17.1: A body in shared space

In this case of the spatial model our representation of the objects needs to hold some record of their absolute position in space. Each of the objects in space need to be structured to allow their position to be represented. For any object in a geometrically defined space we can refer to its position relative to an origin using the bracketing terminology we introduced earlier

x(O) is the x position of the object
y(O) is the y position of the object
z(O) is the z position of the object

The ability to represent and access the position information associated with the objects allow us to test if objects are within an object's focus or nimbus using geometric formula. Consider for example a user A represented by an object O in a shared space. The associated object has a focus and nimbus defined by the predicate *in_foc(x,y,z)* and *in_nim(x,y,z)* which return true when a point lies within the bounds of associated geometric functions describing focus and nimbus areas. In this case our focus function in the general model would map the user focus to a presence position of the form

$$focus(u,f\_label) \mapsto P(o,A)$$

Where

o is the object representing the user in the space
and A is the set of adjacent objects within the focus

In this case A would be defined as

$$A = \{o \in O \mid in\_foc(x(o),y(o),z(o)) \}$$

Obviously, a similar arrangement would hold for the nimbus associated with the object. This case of the spatial model allows arbitrarily complex geometric definitions of focus and nimbus to be used. As in the general model, users may have more than one focus and nimbus associated with them. The functions defining membership of the adjacency set can range from simple Cartesian functions to more complex field equations depending on the nature of the application and the type of space being modelled.

17.3.2 A Field Based Notion of Awareness

The properties of a particular form of space can also feature strongly in the calculating the awareness a user has of others in the space. The general model uses objects to represent and reason about the space. Thus the strength of awareness is based on a ratio derived from the overlap in objects lying within the body's focus and nimbus. While representing a useful measure for awareness this ratio takes no account of Cartesian space. Consequently no consideration is given for spaces where the spacing of objects in not particularly uniform.

An alternative measure of awareness which has been suggested is to consider the focus and nimbus associated with a particular object as geometric fields and to use the overlap of these fields as an indicator of the level of awareness between the two bodies. This formulation of awareness relies on a number of geometric properties associated with the focus and nimbus of users rather than the number or type of objects found within users' focus and nimbus.

One way of reflecting this reliance on the geometry of the space is to associate with each user a geometric field defining the focus and nimbus. These functions are represented

$$focus_u()$$

$$nimbus_u()$$

These two functions can be used to define a specialised form of the awareness function field which takes as its parameters two users and returns a number representing the strength of awareness

$$field\_awareness: U \times U \rightarrow N$$

This awareness strength would draw upon the associated field definitions and return the area of overlap between the different fields. If we assume the existence of an additional function

overlap(f(),g()) - returns the area of the overlap of the fields defined by functions f and g.

the rule for the function field_awareness could be given by merely considering the overlap between the focus field of one user and the nimbus of another.

$$field\_awareness(u1,u2) = overlap(focus_{u1}(), nimbus_{u2}())$$

This form of awareness strength is only one example of a number of potentially complex geometric formulations. It is worth stressing that this form of awareness is merely a specific (rather than definitive) form of awareness function which exploits the geometry of the space.

17.4 Mapping to Non-spatial Domains

In this section we wish to consider the general application of the spatial model across a number of domains. In particular we wish to focus on the use of the spatial model to reason about the sharing of a collection of objects such as those represented in the SOS and SIS. Rather than consider different specific mappings to particular non spatial applications, we wish to develop a view of the spatial model that is generally interpretable across a number of cooperative applications. The starting point for this exercise is a consideration of general graph structures and how the spatial model can be mapped to these graphs. A consideration of general graph structures is useful because the general structure can be applied to a range of different CSCW applications including:

- Elements within an object store such as the SOS and the relations between them.
- The structure of 2D application interfaces.
- The recording of dependencies between activities in workflow systems.
- The development of arguments in design rationale/organisation memory systems.
- The structure of conferences in traditional conferencing systems.
- The connectivity of different machines/rooms.
- The exchanges of email messages in threaded mail packages.
- General multi-user hypertext systems.
- The structure of documents (objects) in shared editors.

Our obvious intent is to find a mapping to general network structures which we can then develop into appropriate interpretations to different classes of cooperative systems.

This section introduces a mathematical consideration of graph structures. We then consider how we can model the sharing of these general graph structures using concepts drawn from our more general spatial model. Finally, we consider some potential interpretations of this graph based instantiation of the model.

17.4.1 Graphs and Directed Graphs

A graph is a discrete structure consisting of sets of vertices (nodes) which may be joined by edges (arcs). A simple graph involving 5 vertices and 5 edges could be of the form shown in Figure 17.2.

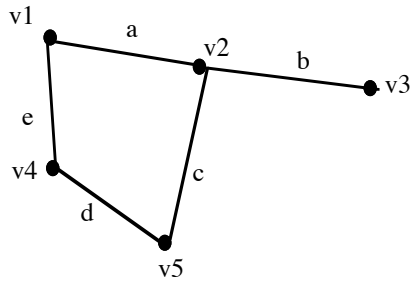


Figure 17.2: a simple Graph

Some simple definitions:

- An edge is said to be **incident** to the vertices it joins. For example, edge a is incident to vertices v1 and v2.
- Two vertices are said to be **adjacent** if they are joined by an edge. For example v2 and v5 are adjacent.
- If edges are incident with the same pair of vertices they are termed **multiple edges**.
- If an edge is incident with only one vertex it is said to be a **loop**.
- A **simple graph** contains no multiple edges or loops.
- A **multigraph** contains multiple edges but not loops.
- A **general graph** or **pseudo graph** contains both multiple edges and loops.

A consideration of general graphs gives us our mapping to the most basic of graph structures and will motivate many of our future mappings.

A graph G has a number of significant constituent parts:

V(G) is a set of objects called vertices.

E(G) is a multiset[1] of edges. Each element of E can be mapped to a multiset of two vertices from V.

A graph can exploit an incident function i which maps edges in E to vertex pairs drawn from V. The incidence function is of the form

$$i: E \rightarrow \{\{u,v\} | \, u,v \in V\}$$

A graph combines each of these elements in a tuple as G(V,E,i). The graph drawn in Figure 17.2 would be of the form

V(G) = {v1,v2,v3,v4,v5},

E(G) = {a,b,c,d,e},
i(a) |→ {v1,v2}, i(b) |→ {v2,v3}, i(c) |→ {v2,v5}, i(d) |→ {v5,v4},
i(e) |→ {v1,v4}

[1] A multiset is a set which allows repetition of elements.

17.4.2 Subgraph

A subgraph of a graph G is a graph whose vertices are a subset S of V(G) and whose edges contain only vertices from S and are a subset of E(G). We use the notation

A < B to denote A is a subgraph of B.
For graphs A and B, $A < B$ iff $V(A) \subset V(B)$ and $E(A) \subset E(B)$.

17.5 Adjacency in Network Structures

The general spatial model for the shared network does not consider the means by which adjacency sets are constructed. This generality allows sets to be constructed for arbitrary collections of the vertices in the network. This freedom allows us to consider non continuous spaces and nodes selected from contextual sensitive criteria such as database queries. However, we may wish to consider assigning continuous spatial properties to adjacency sets. To allow us to do this for each user we use a range of property functions which test the spatial properties of adjacency sets.

17.5.1 Linked Function

We can define a direct linked function which returns true if the nodes in a presence positions adjacent set are directly adjacent to location. This function can be used to specify the properties of particular focus and nimbus. The property function is of the form

$$\text{linked: } PP(G) \mapsto \text{Boolean}$$
$$\text{linked}(pp) = \forall_{V \in} \text{adj}(pp) \text{ and joined}(\text{loc}(pp), v)$$

$$\text{Where joined}(v,g) = \exists x\ (x \in E(G) \text{ and } (i(x) \mapsto \{v,g\} \text{ or } i(x) \mapsto \{g,v\})$$

The linked function allows us to specify that a particular user's focus or nimbus is formed from directly linked nodes. For example if we wished to state that all of the foci associated with a user Steve are formed from directly connected vertices we would state

$$\forall_{l \in} \text{Id linked}(\text{focus}(\text{Steve}, l))$$

17.5.2 Sphere Function

This extension of the linked function reasons about vertices that lie within a specified distance from the location vertex of a presence position. This function is of the form

sphere: PP(G) X N ⊢→ Boolean
sphere(pp, l) = ∀v∈ adj(pp) and in_distance(loc(pp), v, l)

Where in_distance(v,g) = len(path(v,g)) < l or len(path(g,v)) <= l

The sphere function allows us to state that the adjacency set for a focus or nimbus is formed from the vertices within a given range of the location point. For example if we wished to state that the nimbus associated with a user John is formed from vertices within a range of 4 we would

∀l∈ ld sphere(focus(john,l), 4)

The sphere and linked function can be used to specify adjacency properties for particular foci and nimbi identified by the user, label pair. Moving through the network is reflected by altering the location of presence position. Altering focus and nimbus is associated with changing the specified properties of the adjacency sets associated with the focus and nimbus.

17.5.3 Shared Directed Graphs

Our final consideration of a specialised spatial model is to consider the situation when the edges in the shared graph are directed. In this case the general model can be restricted to exploit the directed nature of the graph. We do this by reconsidering the nature of the focus and nimbus.

We consider the adjacency sets associated with nimbus points to consist of vertices connected to the position point by outgoing vertices. This reflects the nature of nimbus as delimiting the range of effects across the space.

Focus on the directed graph is constructed from sets of vertices connected to the location point by incoming vertices. This reflects that focus represents the portion of a space whose effects you are interested in.

These derivative forms of focus and nimbus are constructed using alternative forms of the functions for defining the adjacency set. These become

inlinked: PP(G) ⊢→ Boolean
inlinked(pp) = ∀v∈ adj(pp) and inward(loc(pp), v)

Where inward(v,g) = ∃x (x ∈ E(G) and i(x)⊢→{g,v})

outlinked: PP(G) ⊢→ Boolean
outlinked(pp) = ∀v∈ adj(pp) and outward(loc(pp), v)

Where outward(v,g) = ∃x (x ∈ E(G) and (i(x)⊢→{v,g})

Similarly, equivalent forms of the sphere function exist for delimiting the adjacent sets for focus and nimbus for vertices that are not directly connected

insphere: PP(G) X N ⊢→ Boolean
insphere(pp, l) = ∀v∈ adj(pp) and input_distance(loc(pp), v, l)

Where input_distance(v,g) = len(path(g,v)) <= l

outsphere: PP(G) X N l→ Boolean
outsphere(pp, l) = $\forall_{V\in}$ adj(pp) and output_distance(loc(pp), v, l)

Where output_distance(v,g) = len(path(v,g)) <= l

17.5.4 Hybrid Notions of Awareness

As our final consideration of non spatial mappings of the model we wish to examine a specialisation of awareness which exploits the particular properties of graph to define a measure of awareness. Thus we seek to define a specialised graph_awareness function of the form

graph_awareness: UxU → N

This specialised form of the awareness function draws upon the properties of graph structures. One simple formulation of the model would be simply to exploit the number of paths linking two users of a given length. If we assume the existence of a function of the form

number_of_path(a,b,len) - returns the number of paths between a and b of length len

This could be used to formulate an awareness rule of the form

graph_awareness(u1,u2)=number_of_paths(loc(u1),loc(u2),len(path(loc(u1),loc(u2))

More complex rules which exploit the properties of graph structures to represent the strength of awareness between two users can be formulated. For example, we could develop an awareness function based on a combination of field theory and graph structures. For example each of a surrounding set of nodes can be combined with a value or level for whatever property is being represented. For example, an awareness function might identify the centre of a sphere with an initial field value, a maximum radius for adjacent nodes and a linear decrease in field value. Figure 17.3 shows an example of how such a field may appear on the network.

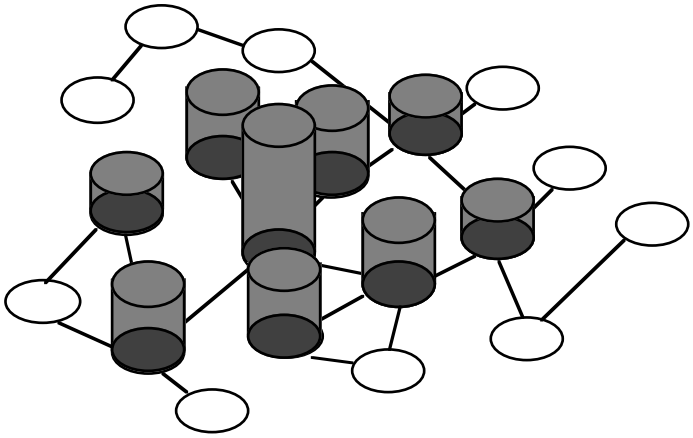


Figure 17.3: Simple example of a spherical presence position

A graph based field function of this form can then be used to return the strength of awareness using the interest levels at each of these nodes. This would be expressed by a rule of the form

graph_awareness(u1,u2)=awareness_level(loc(u1),loc(u2))

We would not like to give the impression that this is the only formulation of awareness possible. Rather, as in the case of geometric spaces we seek to stress that a wide range of equally valid alternate formulations of awareness are possible which exploit the specific properties of particular spaces.

17.6 Interpretations of the Model

The general models of the shared network can be interpreted in a range of different ways to reflect a number of different potential mappings from the shared graph to cooperative applications. The following sections briefly outline a number of possible interpretations of the network based spatial models.

17.6.1 Shared Hypertext

We begin our interpretation of the different spatial models by considering a number of users sharing a multi-user hypertext work. Rather than consider the facilities provided by a specific multi-user hypertext application we shall focus on the general features of multi-user hypertext. We assume that a multi-user hypertext application allows:

- A network of linked information nodes to be represented.
- Users to view different nodes on the network in different windows.
- Users to edit different network nodes.
- Users to navigate the network by following links and opening different windows.

These general facilities allow us to interpret the model by mapping between our general model for shared graphs and the facilities offered in shared hypertext systems. Consider the situation when a network is being shared between two users John and Steve.

Our interpretation of the spatial model is based on the following mappings:

- The presence points associated with focus maps to open network nodes.
- The presence points associated with nimbus maps to network nodes being edited.

Both John and Steve will be associated with presence points using the focus and nimbus functions. For example, Steve has a window open at location ID#245 on the network which has adjacent nodes ID#21, ID#456, ID#98. The associated focus function is

$$\text{focus(steve, win2)} \mapsto \{\text{ID\#245},\{\text{ID\#21, ID\#456, ID\#98}\}\}$$

We have also mapped the arbitrary labels associated with the focus and nimbus to the window identifiers associated with the hypertext node. In this case we have also assumed linked adjacency for this window. This is expressed as

$$\text{linked(focus(steve, win2))}$$

Each user may have a different adjacency function associated with different windows open on the network.

17.6.2 Workflow Systems

Our second interpretation of the graph model considers the specialisation of the model which considers directed graphs. In this case we wish to consider how directed graphs map onto the dependency between activities in work flow systems. The mappings central to our consideration are:

- Activities map to nodes on the network.
- Dependencies between activities (e.g. document exchanges) map onto links.
- Users' involvement in particular activities map them to locations on the network.
- A user has a focus and a nimbus at each location on the network.

In this interpretation of the model we assume that focus consists of the nodes which feed into the particular location while nimbus consists of the nodes which the location feeds. Consider a user Steve involved in an activity *send_info*. The activity relies on the output from the activities *collect_data* and *collect_addresses*. The activity *send_info* delivers output to the activities *check_replies* and *customer_followup*. Steve's focus and nimbus would be

$$\text{focus(steve, role3)} \mapsto \{\textit{send\_info},\{\textit{collect\_data}, \textit{collect\_addresses}\}\}$$

$$\text{nimbus(steve, role3)} \mapsto \{\textit{send\_info},\{\textit{check\_replies}, \textit{customer\_followup}\}\}$$

In this case we have decided to exploit the previously arbitrary labels associated with nimbus and focus by assigning them to some notion of role. The use of the spatial model in this way allows us to reconsider the means by which workflow systems work to directly increase flexibility.

Current considerations of dependencies between activities within workflow systems associated with each dependence a strict deadline. When this deadline is exceeded warnings are sent to those involved in the associated activities. The approach suggested by the interpretation of the spatial model is to consider awareness of dependency rather than some abstract consideration of time as the driving force in the workflow. Thus as activities complete they increase their nimbus on activities they feed, these then become more aware of the need to complete work. Similarly activities awaiting information increase their focus and the supplying activities become more aware of the need to provide information.

17.6.3 Sharing a Version Set

A similar interpretation of the shared graph model is to exploit the version associated with directed graphs to model awareness across a set of versions of information. In this case we rely on the mappings:

- Versions of information entities map to nodes on the network.
- Version dependencies and derivations map onto links.
- The use of particular versions of information map them to locations on the network.
- User of a particular version have a focus and a nimbus at the location on the network associated with that version of the information.

In this interpretation of the model we assume a mapping of focus to incoming nodes and nimbus to outgoing nodes. Consider a user Steve using a version of a software module *dvs_driverV1.4*. This version is derived from merging a previous version *dvs_driverV1.3* with a second module *dive_driverV2.1*. This version has subsequent version *dvs_driverV1.5* and *dive_driverV4.0* In this case Steve's focus and nimbus would be

focus(steve, prog4) $\mapsto$ {*dvs_driverV1.4.*,{*dvs_driverV1.3*, *dive_driverV2.1* }}

nimbus(steve, prog4) $\mapsto$ {*dvs_driverV1.4.*,{*dvs_driverV1.5*, *dive_driverV4.0*}}

In this case we have decided to exploit the previously arbitrary labels associated with nimbus and focus to assigning them to local mappings to private versions.

17.6.4 A Simple Shared Desktop

As our final interpretation of the graph model consider the use of the shared graph to represent a number of users sharing some form of collaborative desktops. We exploit the following mappings:

- Nodes on the network map to icons on the desktop (e.g. folders, applications, disks etc..).
- Links on the network map onto the containment of windows on the network.
- The presence points associated with open icons on the desktop map to focus.
- The icon currently being selected by a user maps to nimbus.

Consider a user Steve who has a folder *COMIC* open. The icon for the *COMIC* folder is in a window called projects that includes folders for *VIRTUOSI* and *DIVE* and the application *proposal_maker*. The folder DIVE contains the folders *Strand1, Strand2, Strand3, Strand 4*. Strand 4 has folders titled *spatial* and *object*.

The focus associated with this open folder is

$$\text{focus(steve, win1)} \mapsto \{COMIC.,\{VIRTUOSI, DIVE, proposal\_maker, Strand1, Strand2, Strand3, Strand\ 4, spatial, object\}\}$$

We are using the arbitrary label in this case to map the nimbus to particular window identifiers associated with the user. In this case we assume that Steve has set his focus for this window to be set to items which are up to two levels away. This is expressed in the model as:-

$$\text{sphere(focus(steve, win1), 2)}$$

17.7 Applying the Model in COMIC

In this chapter we have outlined a general spatial model that uses simple set theory concepts to represent the properties of a shared space independently of the properties of a particular space. More specific instances of the general shared model can be realised for geometric based spaces and for spaces constructed from collections of related objects. A number of possible interpretations of the model are possible and some of these have been outlined in the previous section. In this final part of the chapter we wish to explore how these different interpretations of the spatial model can be applied to a number of the prototype systems developed as part of COMIC. In particular we wish to focus on the prototype systems that have emerged to explore shared object systems and have not been used as part of the investigation of how best to formulate the spatial model.

17.7.1 TheKnowledgeNet

We shall take as our first example system the TheKnowledgeNet a system developed at KTH and augmented as part of the COMIC project. TheKnowledgeNet could be viewed as a distributed “library” of documented and undocumented knowledge that is made accessible by CSCW technology. Areas of undocumented knowledge are made public by expertise declarations describing the kind of knowledge the member possesses. The library consists of these expertise declarations and the documents produced by the members with links to the originator of the information in a way that supports communication and

collaboration with appropriate CSCW tools. The information space is then peopled and the originator of the information is made visible and accessible.

TheKnowledgeNet is in essence a network of linked information shared by a community of cooperating users. We can directly represent the linked information as a graph with the nodes representing items of knowledge and arcs representing the links between information. This allows us to exploit the hypertext based interpretation of the spatial model which reasons about users sharing a common graph structure.

The CoDesk

The developers of the KnowledgeNet stress that it should be regarded as an integrated work environment. The interface to this work environment, CoDesk, is designed to support the management of collaboration in a manner similar to document and application handling in the popular Macintosh desktop interface, for example. The collaborative desktop consists of interface objects representing members, catalogues, groups, documents, expertise areas, CSCW tools and private tools.

The CoDesk model is based on a room metaphor, which models the computer network as a set of virtual rooms or spaces within which people interact. The CoDesk room metaphor models TheKnowledgeNet as a building consisting of a collection of hierarchically structured rooms representing both public, shared and private spaces containing members, tools and documents.

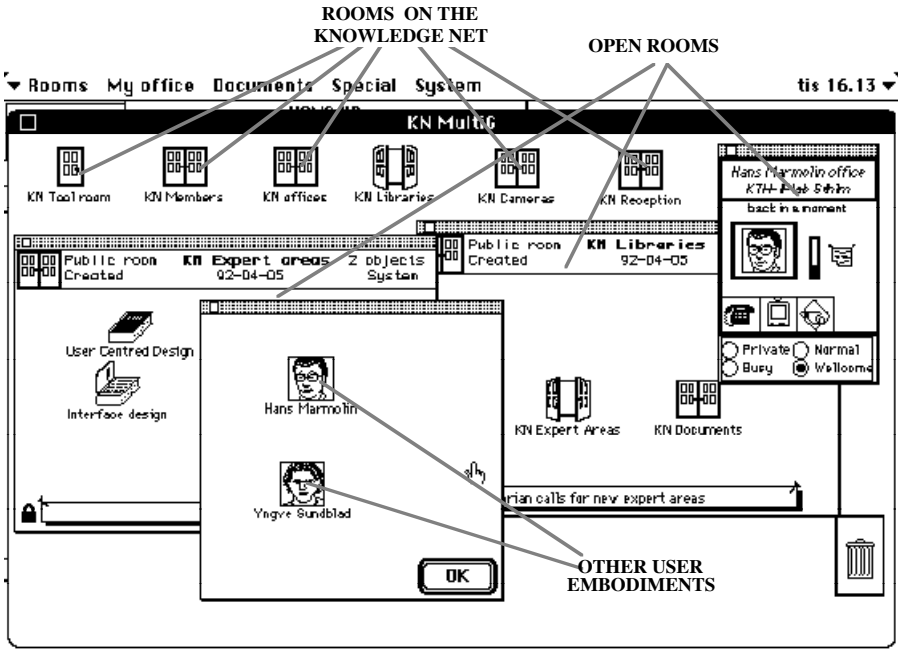


Figure 17.4: The CoDesk interface to TheKnowledgeNet

The screen shot in Figure 17.4 shows the key features of the CoDesk models. Nodes on the knowledge net are manifest on the screen as doors to shared rooms.

Each user has an iconic representation which acts as a user embodiment and is shown in shared rooms. Finally open rooms are shown as separate windows onto which users can run telepointing facilities.

In addition to the mapping of the structure of the knowledge net to a shared graph we can exploit the graph based spatial model to reason about the shared interface aspects of the CoDesk using an interpretation akin to the one for shared desktop interfaces. This can be achieved by mapping users to presence positions corresponding to the open rooms. The calculated awareness functions can be presented to users by amending the user icons. In this case the awareness mode function can be used to calculate the particular awareness states for icon visualisations.

17.7.2 The COLA platform

The COLA platform developed at Lancaster as part of the COMIC project seeks to provide lightweight and flexible support for cooperative activities. The core of the COLA model is based on a number of simple components:

- *Activities*: Activities in COLA are lightweight collections of resource objects and people. They may provide a state which can be externally examined to allow applications to make public particular activity changes.
- *Events*: COLA events link activities together as different activities register particular interests in events.

In general, COLA models a set of events as a network collection of activities to which a user may belong. In addition the core of the model is to promote the cooperative sharing of related objects by providing alternative context sensitive presentations.

COLA allows two alternative specialisations of the model to be exploited. The model's reliance on shared objects which may be linked by a series of relationships offers the possibility of exploiting an interpretation based on shared object stores and shared hypertext systems. Here nimbus and focus would be defined from presence positions given by the user's links with shared objects. An alternative interpretation which could be used would be that of workflow systems. In this case we would focus on the definition of activities within the model and formulate focus and nimbus based on the dependencies recorded between activities.

17.7.3 The SOL Toolkit

As part of the COMIC project Lancaster has developed a shared user interface development toolkit called SOL. SOL relies on using shared user interface objects to define multi-user user interfaces for cooperative application. To define cooperative user interface SOL relies on the use of an access model to define the relationship between user and the functionality provided by the shared interface.

The access model also specifies the level of coupling by defining the extent of propagation of actions.

SOL relies on a common shared representation of the user display which is used to define specific instances presented to different users. This representation of the shared interface exploits a hierarchy of objects to define the structure of the interfaces. The hierarchy of user interface object maps directly onto the presentation of the user interface (Figure 17.5).

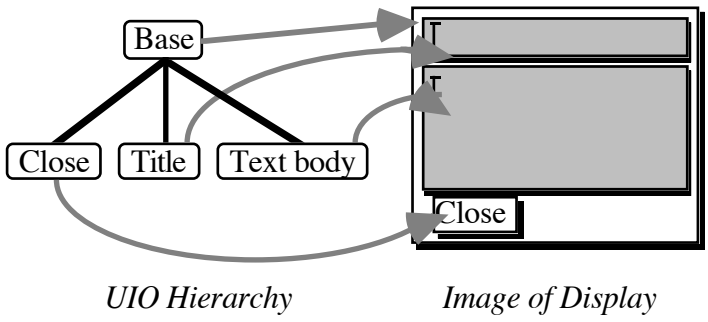


Figure 17.5: Relationship between UIO hierarchy and its representation

The object hierarchy is akin to the general model of shared desktops, and the SOL toolkit can exploit a shared desktop interpretation of the awareness model outlined in the previous sections. In addition the awareness measure provided by the spatial model can be complemented by the properties of the SOL access model to provide an augmented model of awareness specific to SOL.

17.7.4 Extending awareness

As part of the COMIC project Mariani and Prinz have examined the way in which awareness can be supported within shared object stores. This work is particularly interesting in that it augments the existing interpretations of the general and network based model with notions of awareness which are based on understanding the semantics of object interaction.

In suggesting a model for awareness Mariani and Prinz adopt a view of an object store as a collection of interconnected data objects. This allows the network or hypertext interpretation of the network based spatial model to be used to reason about awareness across the object space. In addition, they propose a particular model of awareness which provides an indication of awareness based on the operations users are undertaking on the shared objects. They propose a number of significant metrics which provide a measure of the awareness of shared interaction.

The proposed metrics are based on identifying and classifying a number of significant factors in interaction with the shared object store. Each of these different factors is given a weighting which reflects its impact on the awareness of other users. These factors combine a number of issues in determining a metric, the associated metrics focus on calculating an awareness factor resulting from an

operation on an object. This metric combines a weighting associated with the operation on an object, the number of objects involved in an operation and a classification of the type of object to produce an operation awareness factor **OAF**.

In formulating their final metric of awareness Mariani and Prinz exploit the network based structure implicit within object systems by using the relations between objects as the means of determining an indication of spatial proximity. In particular, they focus on determining a semantic indication of awareness based on direct connectivity. The basic formulation of their awareness measure relies on the following factors:

OAF The original operation awareness factor associated with an operation.

N The number of other objects also connected by a relation to the given object. This exploits a decay function which returns an appropriate decay index. (The one suggested by Mariani and Prinz is a m root curve).

RW A relationship weighting factor which indicates some notion of distance.

I An importance factor for that particular relationship.

These different factors can be readily combined to give an operational view of awareness to complement the other awareness measures available from the network based interpretation of the spatial model. It is important to note here that none of the particular measures of awareness are more significant than any other. Rather each indicates a different form of awareness which exploit different properties of the shared space and may prove more useful within particular circumstances. The formulation of operation_awareness for a set of operations **Op** can be represented by a function of the form

$$\text{operation\_awareness: } U \times U \times Op \rightarrow N$$

We can exploit the general spatial model to map the locations for user 1 and user 2 presence positions to connected objects. We may also define the adjacency sets for these presence positions to be the set of objects directly connected to the location object by a given relation. This allows us to express for the operation_awareness function an associated rule of the form:-

$$\text{operation\_awareness}(u1,u2,op) = \frac{oaf(loc(u1),op) \times RW(loc(u1),loc(u2)) \times Imp(loc(u1),loc(u2))}{\text{Operation\_decline}(\,|adj(u1)|\,)}$$

In this chapter we have attempted to develop a mapping between the concepts central to the spatial model and a wide range of cooperative systems. In doing so we have also sought to provide a theoretical link between the work on sharing within object based systems and shared space systems. The starting point for this mapping has been the development of a general model of cooperative shared spaces.

The general model of shared spaces relies on the construction of a notion of awareness that exploits the general concepts of sharing independently of the specific properties of a particular shared space. The basis of the model is the use of presence positions that relate the presence of an object at a particular place within the shared space with the surrounding adjacency objects. This association allows us to define abstract formulations of focus and nimbus that can be used to generate a number of different measures of awareness.

A number of specialisations of the general model exist. In this chapter we explored two particular specialisations of the general model. The first specialisation considered mapping the general model to geometric spaces and compared this work to the existing VR prototypes developed as part of investigating the spatial model. This was contrasted with a specialisation of the spatial model to network structures to provide a mapping of spatial models of awareness to a wide variety of cooperative systems. The specialisation of the spatial model in this way allows a range of different interpretations across a range of applications. These spatial views of awareness can be complemented by properties of particular applications.

We believe that the mappings outlined in this chapter demonstrate the general utility of the spatial model beyond cooperative VR systems. We also feel that simple mathematical basis of the mapping allows us to develop a theoretical basis for awareness in future cooperative applications that exploit the properties and principles of shared spaces.

Appendices

Appendix

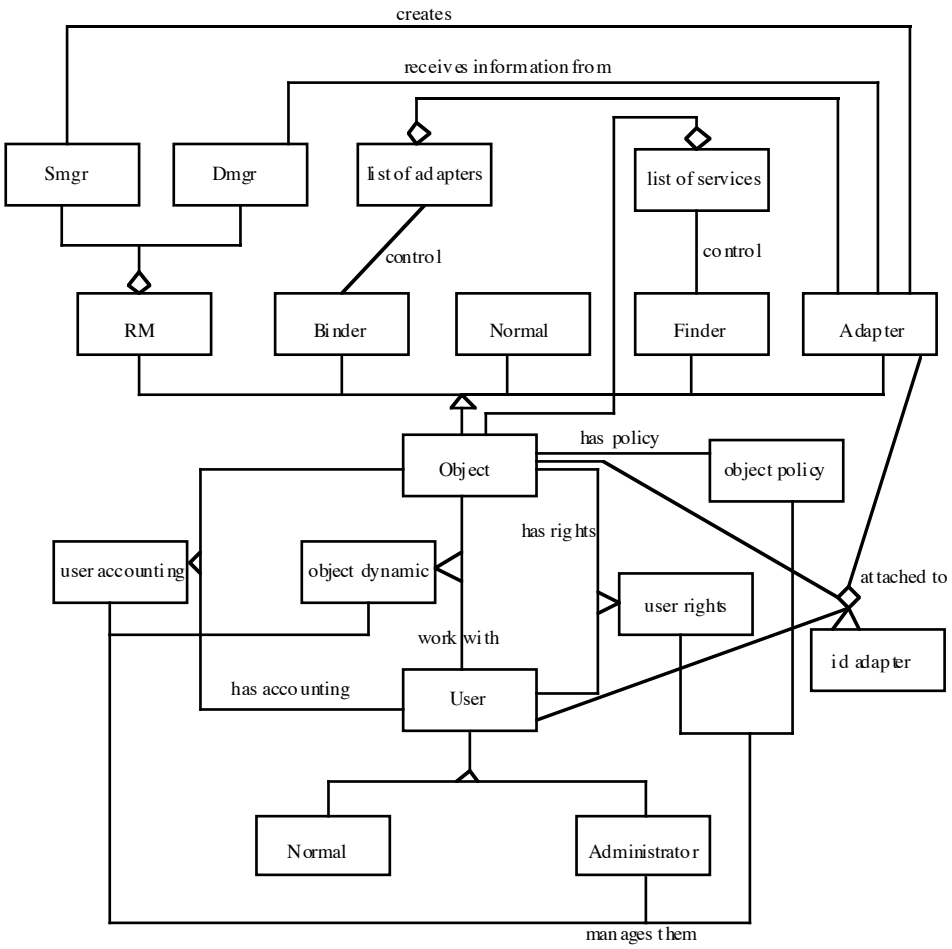
An engineering view of the Trader and Resource Manager

G. Rodriguez

UPC

This is an appendix to chapter 8. In order to design our prototype of the Trader and Resource Manager, we employed the OMT method. In this appendix, we present the result of this method.(OMT-91)

1. OMT object model



1.1. Attributes:

User:	user's identifier, name, plan , etc.
Object:	object's identifier, interface reference (access point), name, characteristics (methods and parameters).
User rights:	access rights, maximum accounting, moderator rights (F.F.S.)
Object dynamic:	list of owners, moderator (F.F.S.)
User accounting:	accounting data.
Object policy:	maximum load, events' rules (F.F.S.), moderator policy (F.F.S.), creator (F.F.S.).
Adapter:	events' rules (F.F.S.), accounting data, moderator's rights (F.F.S.), access rights.

1.2. Methods and properties:

User:
error_handler (error type)
delivery_candidates (list of candidates)
delivery_result (result)
delivery_access_point (access point)
disconnect_ok()
request_list (characteristics)
request_method (object's identifier, method, parameters)
request_connect (desired characteristics of the service)
request_disconnect (object's identifier)
show_candidates (list of candidates)
show_result (result)
establish_a_session (access point)
close_a_session

Object:
request_method (method, parameters) -> result or error

Finder:
request_candidates (characteristics) -> list of candidates or error
request_new_object (object's data) -> OK or error
request_delete_object (object's identifier) -> OK or error
request_modify_object (object's identifier, data) -> OK or error

Binder:
request_adapter's_data (objects identifier) -> data
establish_a_session (adapter's identifier) -> new access point

Static manager (Smgr):
request_connect (desired characteristics of the service) ->
-> access point or error
request_list (desired characteristics of the service) ->
-> list of candidates or error
request_reconfigure (data) -> new access point or error
request_control_policies (data) -> OK or error
request_maximum_accounting (user's identifier, object's identifier) ->
-> accounting data or error
error_handler (error type) -> error
delivery_candidates (list of candidates)
delivery_dynamic_info (data)
delivery_adapters's_data (data)
delivery_access_point (access_point)
dynamical_reconfiguration (F.F.S.)
automaton (list of candidates) -> the best one
establish_a_session (adapter's identifier) -> new access point

Dynamic manager (Dmgr):
error_handler (error type) -> error
request_dynamic_info (object's identifier) -> data
new_owner (owner, object's identifier)
update_accounting (data)
delivery_maximum_accounting (maximum accounting)
delivery_disconnect (result)
object_failed
request_disconnect (data) -> OK or error
delete_owner (owner, object's identifier)
dynamical_reconfiguration (F.F.S.)

User rights:
add_user (user's data) -> OK or error
delete_user (user's identifier) -> OK or error
select_user (user's identifier) -> user rights
add_rights (user's identifier, object's identifier, access rights,
maximum accounting) -> OK or error
access_rights (user's identifier, object's identifier) ->
-> access_rights or error

maximum_accounting (user's identifier, object's identifier) ->
-> maximum accounting or error
delete_rights (user's identifier, object's identifier) -> OK or error
modify_access_rights (user's identifier, object's identifier) ->
-> OK or error
modify_maximum_accounting (user's identifier, object's identifier) ->
-> OK or error

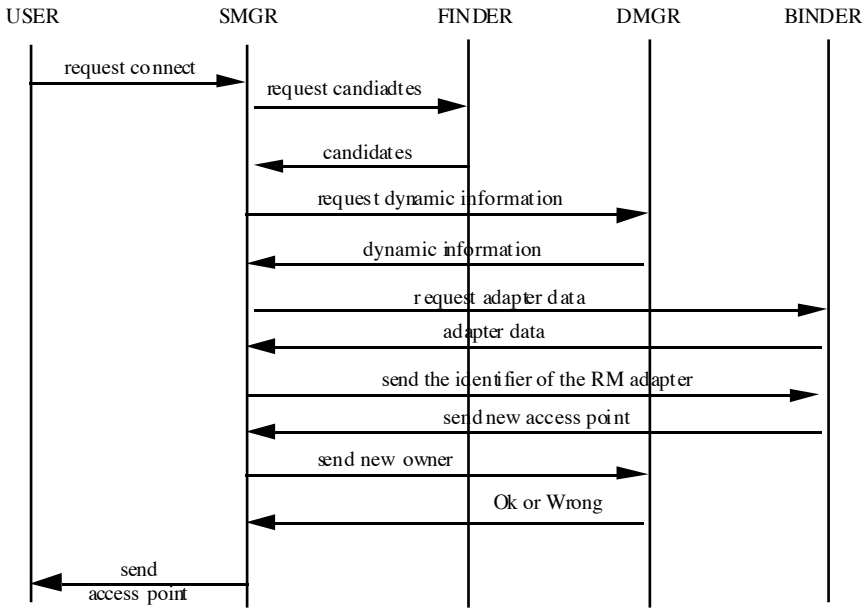
Object dynamic:
add_object (object's data) -> OK or error
delete_object (object's identifier) -> OK or error
select_object (object's identifier) -> object dynamic
add_owner (object's identifier, owner) -> OK or error
delete_owner (object's identifier, owner) -> OK or error
show_owners (object's identifier) -> list of owners
how_many (object's identifier) -> number or error

User accounting:
add_user (user's data) -> OK or error
delete_user (user's identifier) -> OK or error
select_user (user's identifier) -> user accounting
add_accounting (user's identifier, object's identifier, accounting factor) ->
-> OK or error
delete_accounting (user's identifier, object's identifier) -> OK or error
accounting (user's identifier, object's identifier) -> accounting or error
modify_accounting (user's identifier, object's identifier, accounting) ->
-> OK or error

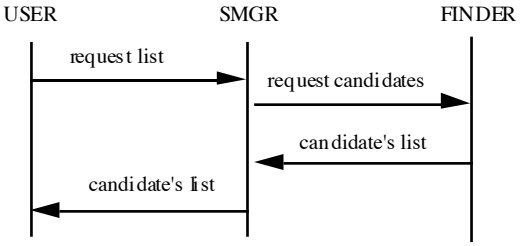
Object policy:
add_object (object's identifier, load) -> OK or error
delete_object (object's identifier) -> OK or error
maximum_load (object's identifier) -> load or error
modify_maximum_load (object's identifier) -> OK or error

Adapter:
Adapter_created (information about the object) -> OK or error
request_disconnect -> OK or error
request_method (method id, parameters) -> result or error
error_handler (error type) -> error
object_failed.
send_result (result) -> result
disconnect_dmgr
dynamical_reconfiguration (F.F.S.)

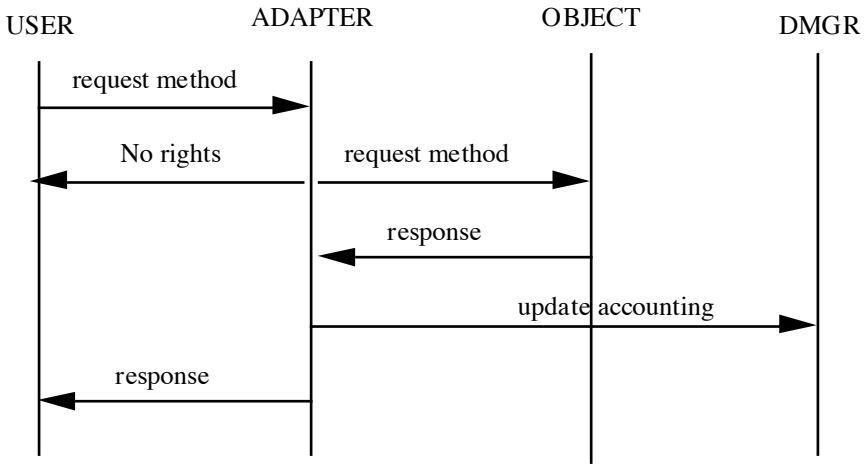
2. OMT dynamic model



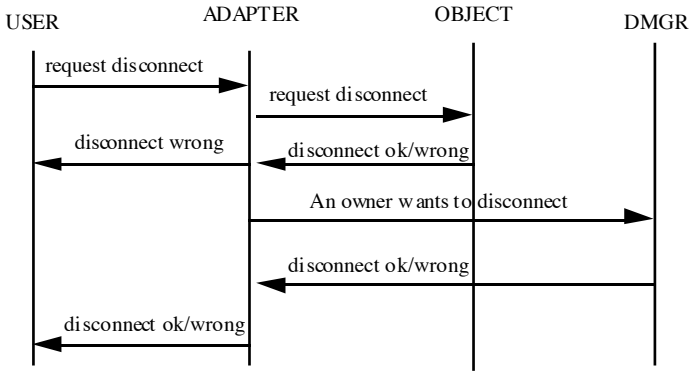
A user wishes to establish a session with an object



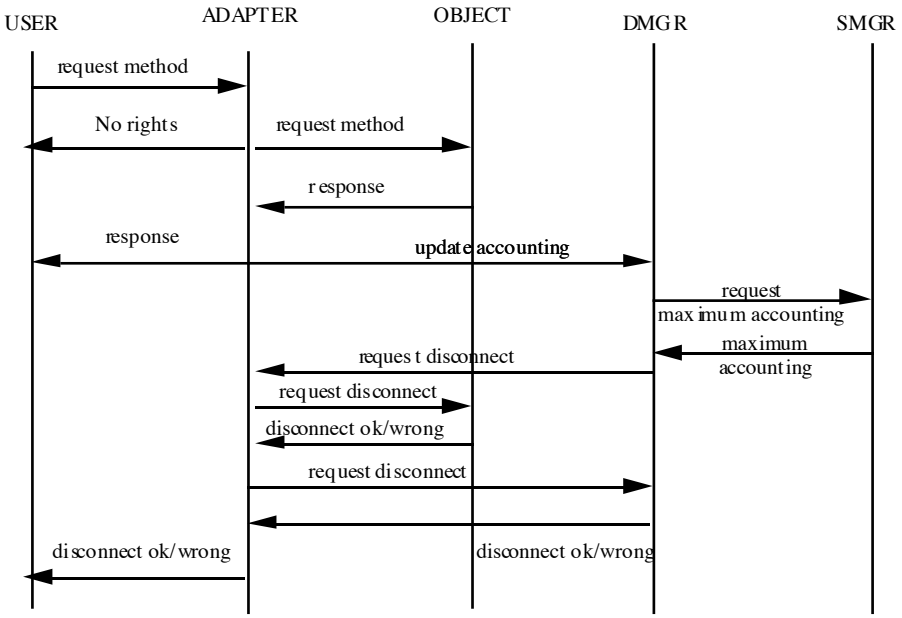
A user calls a list request



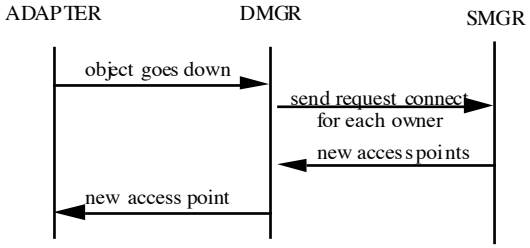
A user uses an object (method)



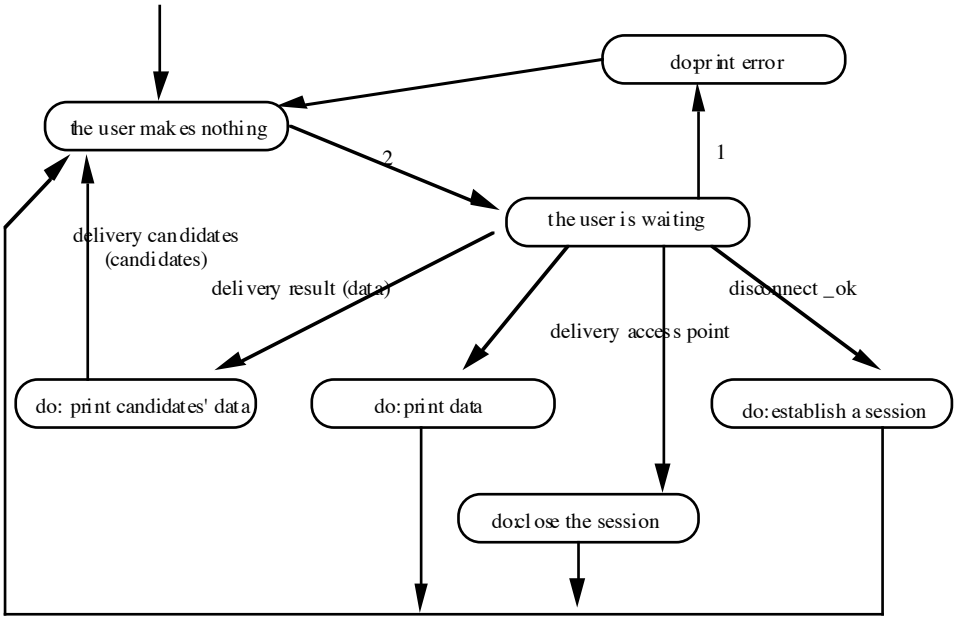
A user is "over-accounting"



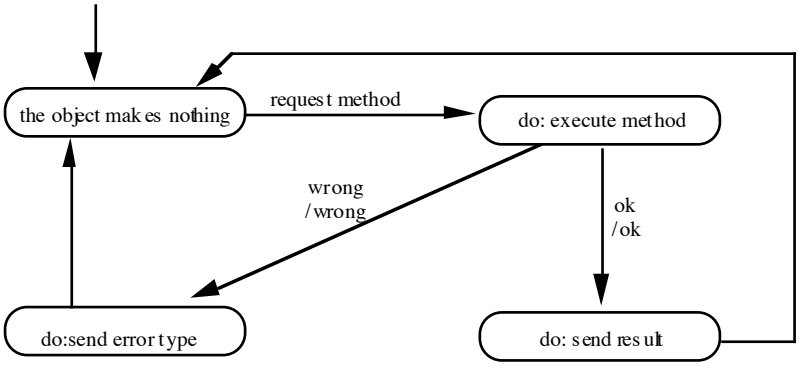
A user calls disconnect request



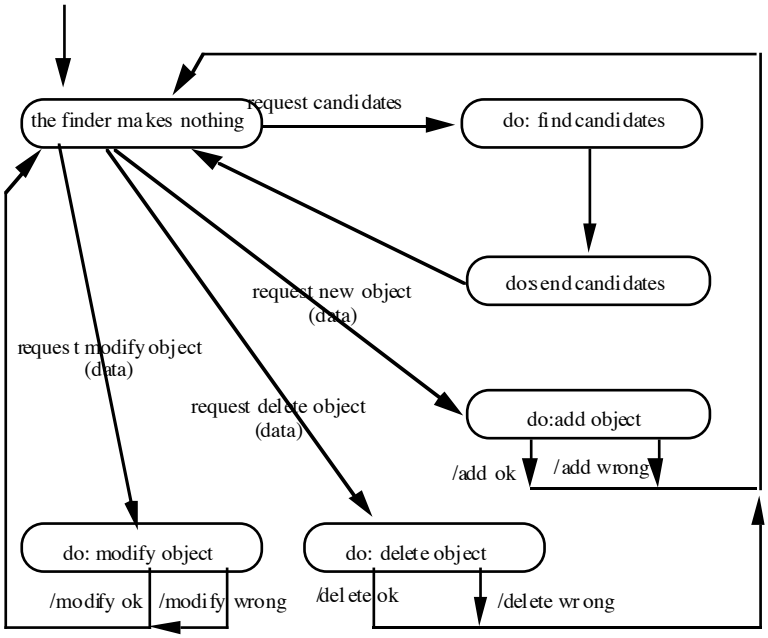
An object goes down (reconfigure) F.F.S.



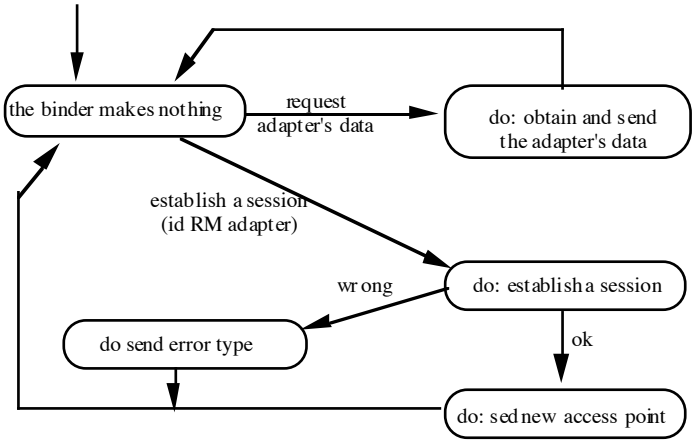
State diagram for class USER
1. no_response, no_connect, no_candidates, disconnect wrong, no access rights.
2. request list, request method, request connect, request disconnect.



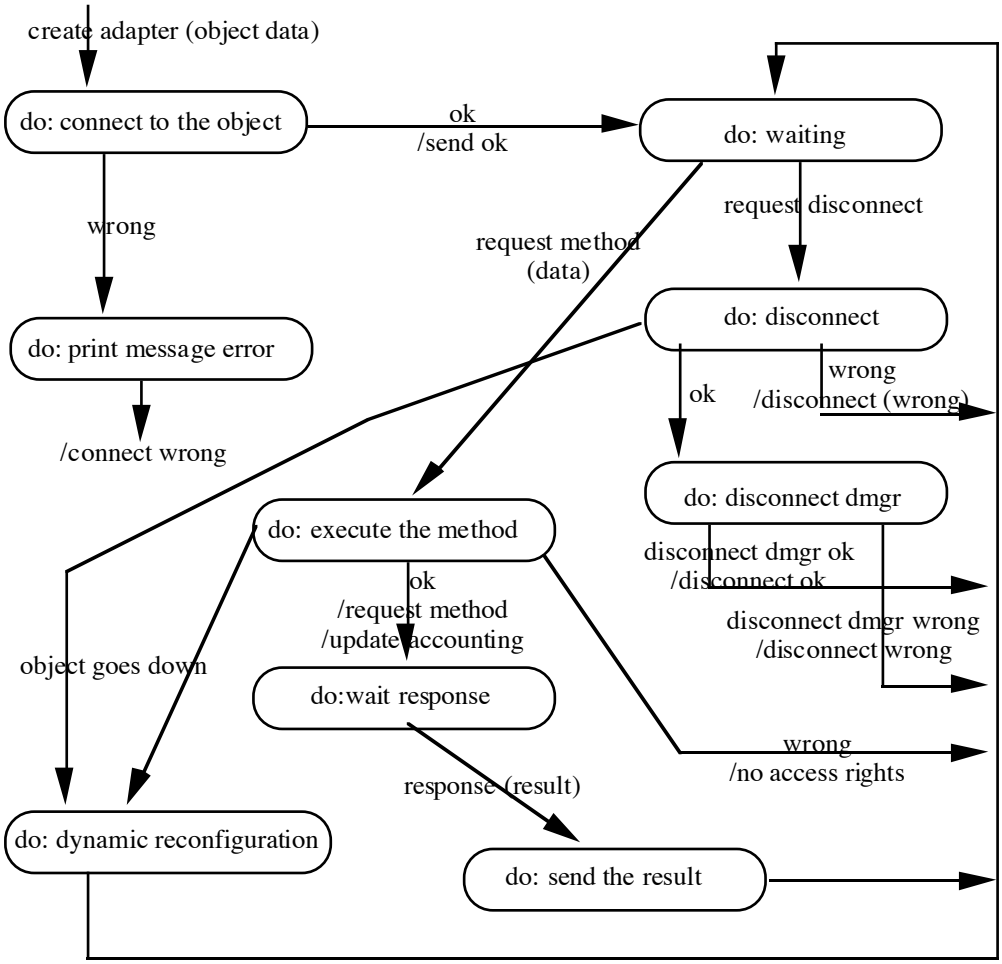
State diagram for class OBJECT



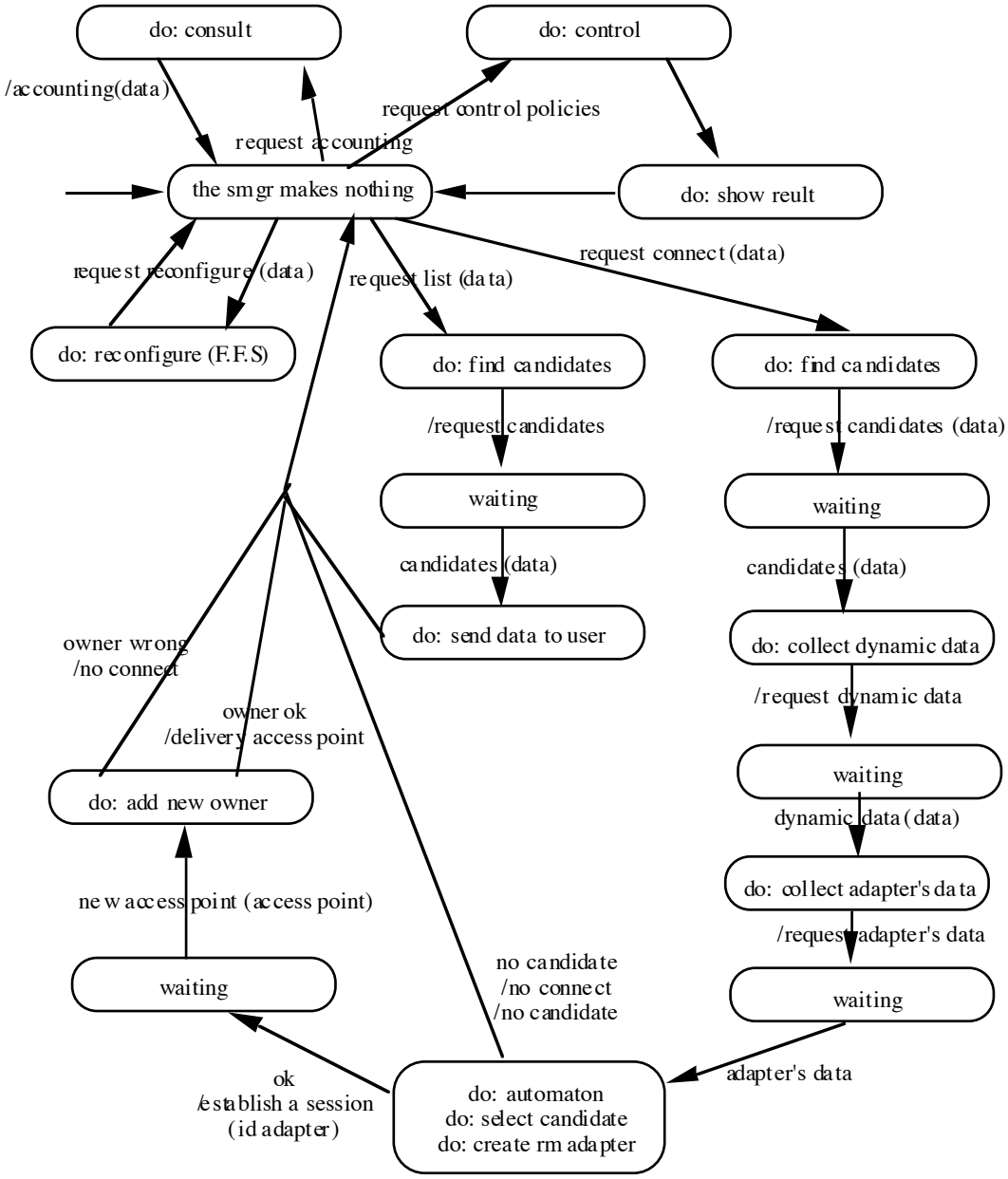
State diagram for class FINDER



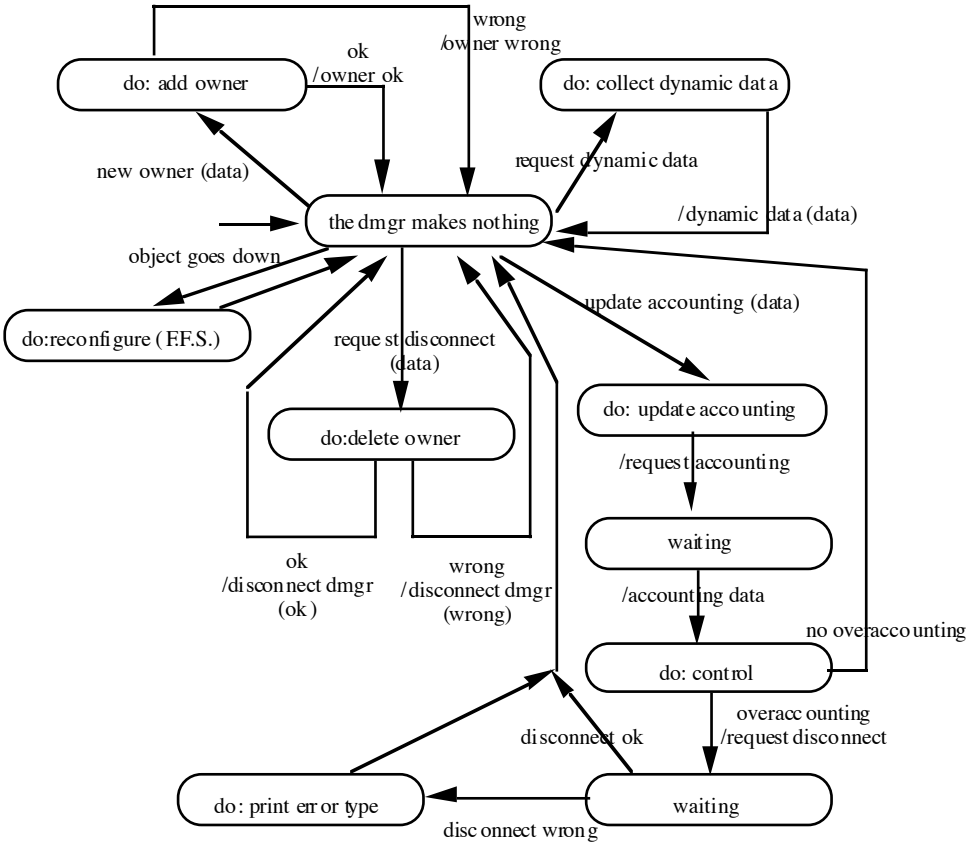
State diagram for class BINDER



State diagram for class ADAPTER

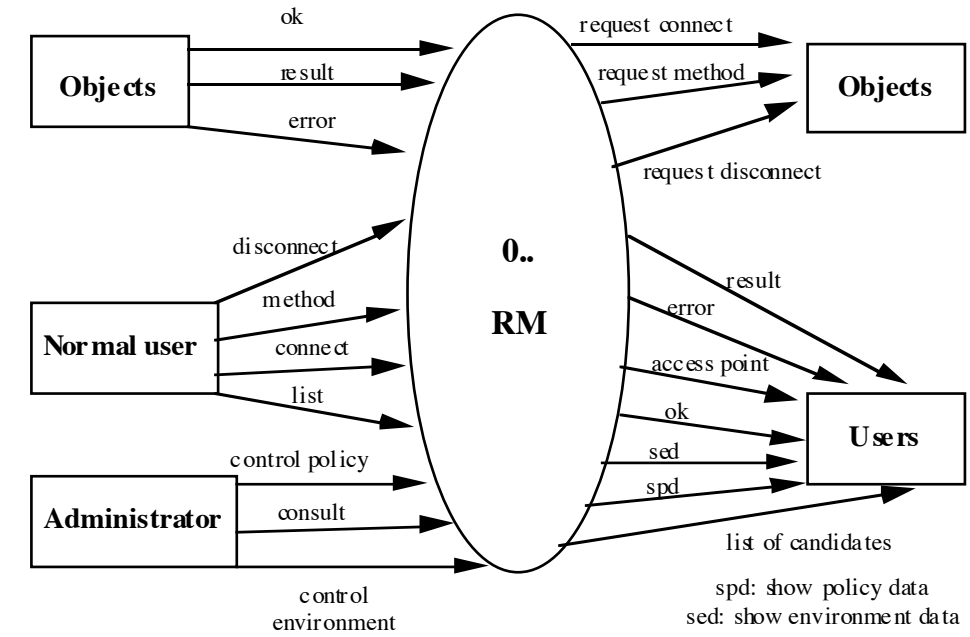


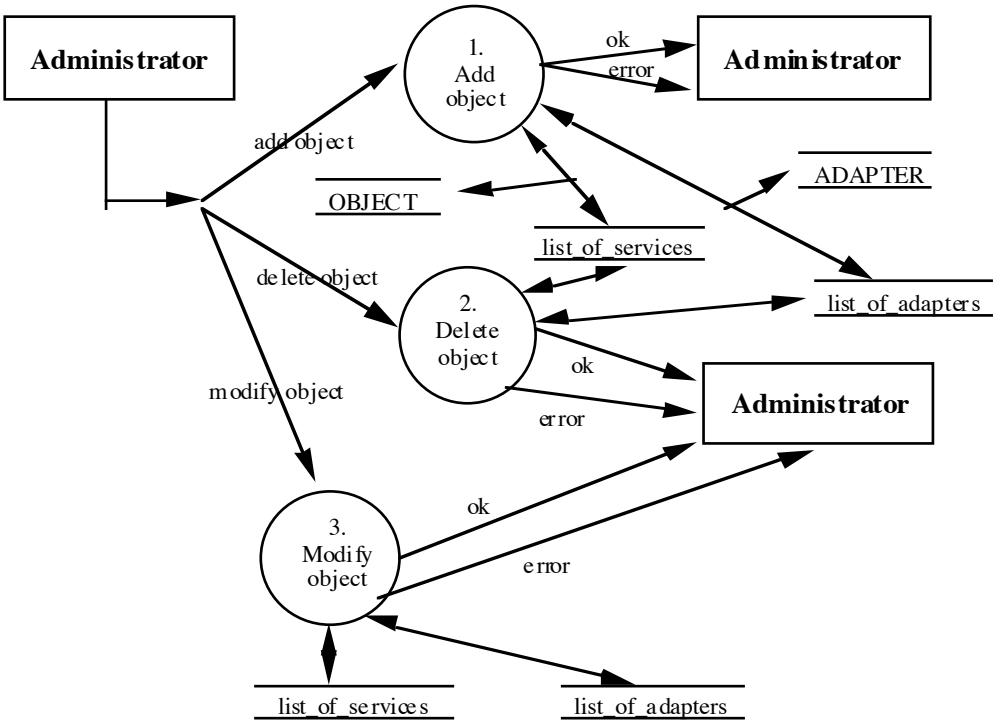
State diagram for class SMGR



State diagram for class DMGR

3. OMT functional model

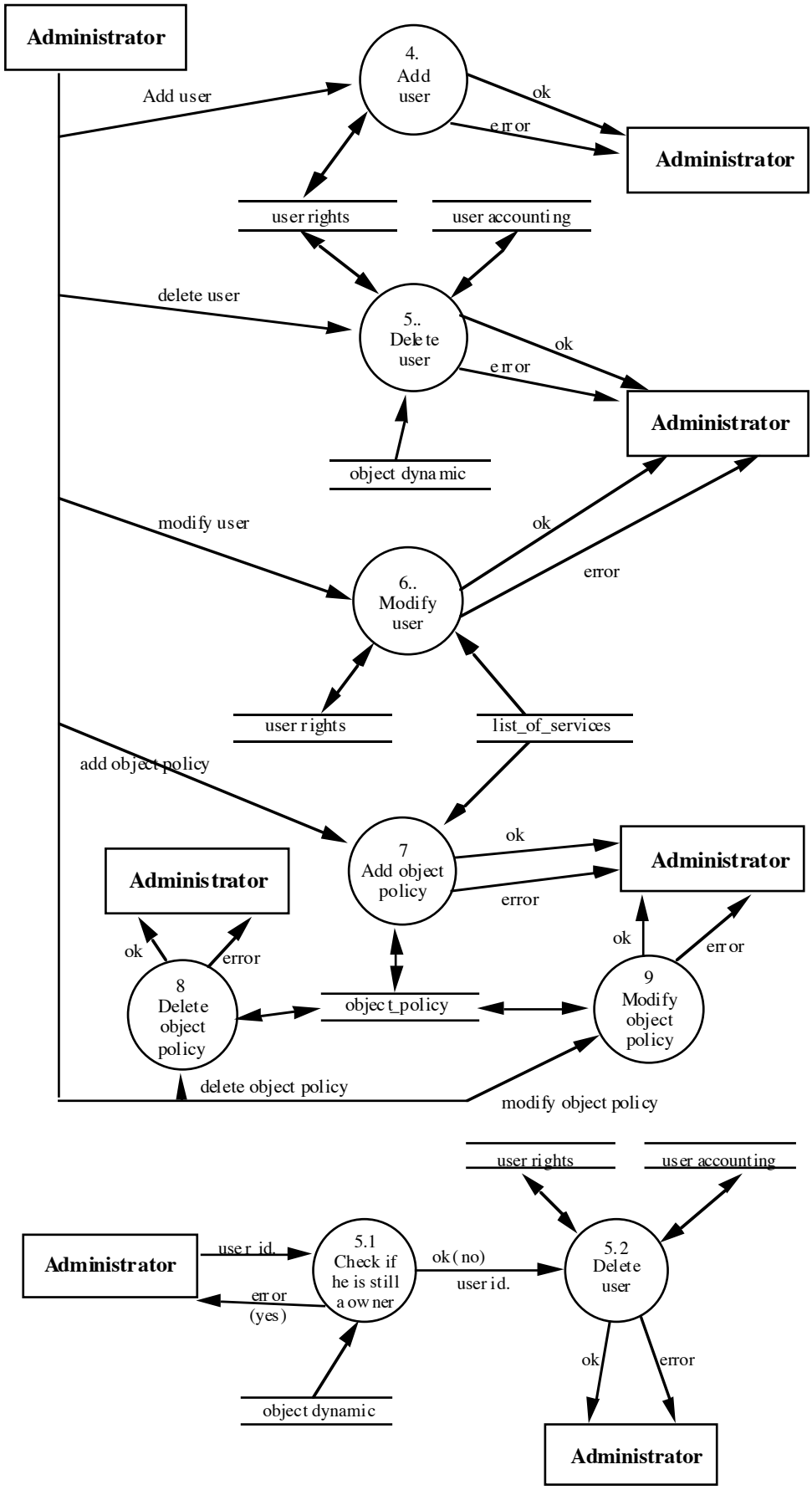


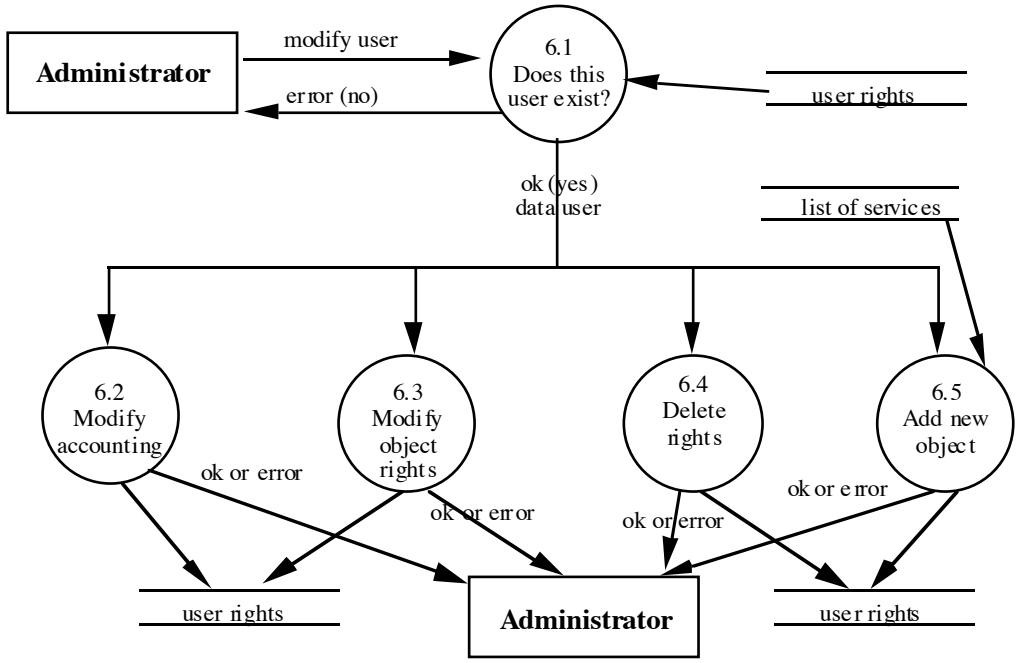


1. Add object (object_id, access_point,
adapters_info, characteristics and values)-> OK or ERROR
If (parameters have errors)
return error (the parameters are not right).
else
If (object_id already exists in list_of_services)
return error(the object already exists).
else
add object to list of services(object_id, access_point,
characteristics and values).
add adapters info to list of adapters(object_id, access_point,
adapters_info,
characteristics and values).
return OK.
endif.
endif.

2. Delete object (object_id) -> OK or ERROR
If (parameter has errors)
return error (the parameter is not right).
else
if (object_id does not exist in list_of_services)
return error (the object does not exist).
else
delete object from list of services (object_id).
delete object from list of adapters (object_id).
return OK.
endif.
endif.

3. Modify object (object_id, data) -> OK or ERROR
If (parameters have error)
return error (the parameters are not right).
else
if (object_id does not exist in list_of_services)
return error (the object does not exist).
else
modify object from list of services (object_id, data).
modify object from list of adapters (object_id, data).
return OK.
endif.


Administrator
4.
Add
user
Add user
ok
error
Administrator
user rights
user accounting
delete user
5..
Delete
user
ok
error
Administrator
object dynamic
modify user
ok
error
6..
Modify
user
user rights
list_of_services
add object policy
7
Add object
policy
ok
error
Administrator
Administrator
ok
error
8
Delete
object
policy
object_policy
ok
error
9
Modify
object
policy
delete object policy
modify object policy
user rights
user accounting
Administrator
user id.
error
(yes)
5.1
Check if
he is still
a owner
ok (no)
user id.
5.2
Delete
user
object dynamic
ok
error
Administrator



4. Add user (user_id) -> OK or ERROR
If (parameter has error)
return error (the parameter is not right).
else
if (user already exists in user_rights)
return error (the user already exists).
else
add user to user rights with no rights (user_id).
return OK.
endif
endif.

5.1. Check if he is still a owner (user_id) -> YES or NO
For each object in object_dynamic do
if (user_id is an owner) return YES.
else
do nothing.
endforeach.
If (he is a owner)
return to user an error (You cannot delete him because he is still a owner).
else
return NO.
endif.

5.2. Delete user (user_id) -> OK or ERROR
If (the user does not exist into user_rights)
return error (the user does not exists).
else
delete from user_rights (user_id).
delete from user_acco (user_id).
return OK.
endif.

6.1. Does this user exist? (user_id) -> YES or NO
If (the user_id exists in user_rights)
return YES.
else
return NO.
endif.

6.2. Modify accounting (user_id, new maximum accounting factor) ->OK or ERROR
If (the accounting factor is less than zero)
    return error (The accounting factor cannot be less than zero).
else
    modify user's maximum accounting factor (user_id,
                    new maximum accounting factor).
    return OK.
endif.

6.3. Modify object's rights (user_id, object_id, new rights) -> OK or ERROR
If (the new rights are not null, read, modify or delete)
    return error (the access rights are not OK).
else
    If (the object_id does not exist in the access right list of the user)
        return error (the object is not in the access right of the user).
    else
        modify access right (user_id, object_id, new access right).
        return OK.
    endif.
endif.

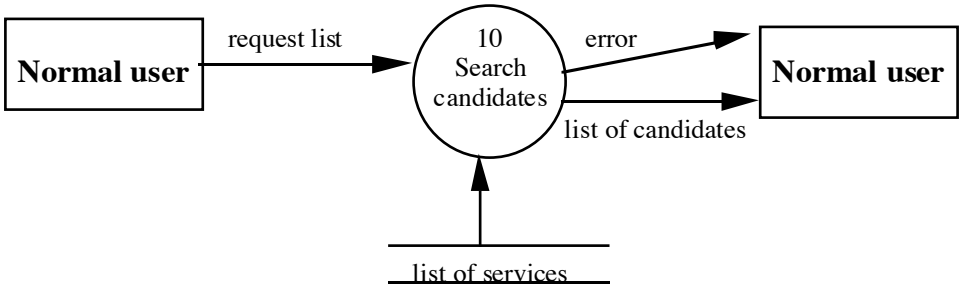
6.4. Delete rights (user_id, object_id) -> OK or ERROR
If (the object_id does not exist in the access right list of the user)
    return error (the object is not in the access right of the user).
else
    delete the object from the access right list of the user (user_id, object_id).
    return OK.
endif.

6.5. Add new object to the access right of the user (user_id, object_id,
                    access rights) -> OK or ERROR
If (the object_id already exists in the access right of the user)
    return error (the object already exist in the access right of the user).
else
    If (the access rights are not null, read, modify or delete)
        return error (the access rights are not OK).
    else
        If (the object_id is not in the list_of_services)
            return error (the object does not exist).
        else
            add new object to access right list (user_id, object_id,
                                access rights).
            return OK.
        endif.
    endif.
endif.

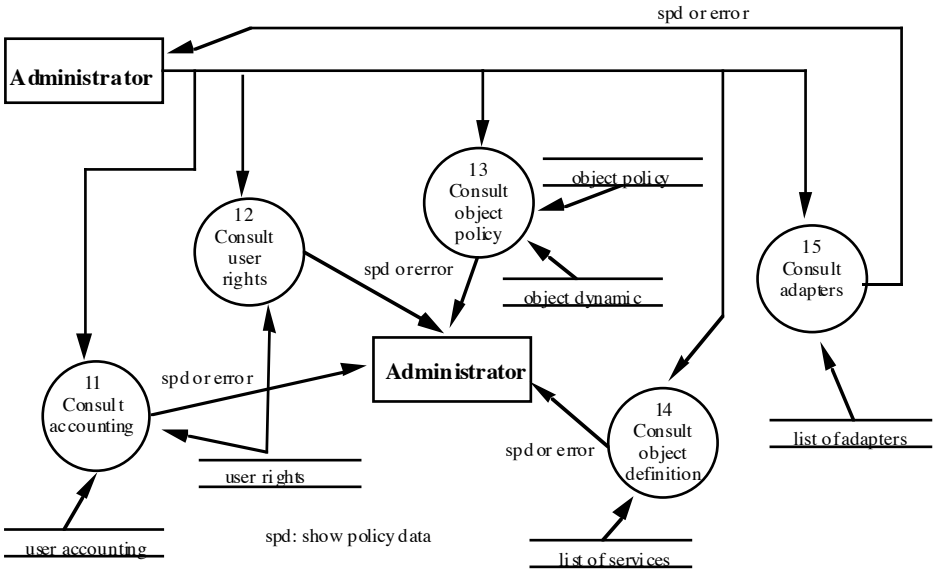
7. Add the policy of an object (object_id,
        policy: in the prototype, this means maximum load) -> OK or ERROR
If (maximum load is less than zero)
    return error (the maximum load cannot be less than zero).
else
    If (the object_id already exists in the list_of_services)
        return error (the object already exists).
    else
        add the object with its policy into the object_policy list (object_id,
                                maximum load).
        return OK.
    endif.
endif.

8. Delete an object from the object_policy list (object_id) -> OK or ERROR
If (the object_id does not exist in the object_policy list)
    return error (the object does not exist).
else
    delete the object from the object_policy list (object_id).
    return OK.
endif.

9. Modify the policy of an object (object_id, new policy) -> OK or ERROR
If (the new policy is less than zero)
    return error (the maximum load cannot be less than zero).
else
    If (the object_id does not exist in the list_of_services)
        return error (the object does not exist).
    else
        modify the policy of the object (object_id, new policy).
        return OK.
    endif.
endif.



10. Search candidates (characteristics) ->list of candidates or ERROR
The Finder selects the candidates from the list_of_services according to the characteristics. If any error occurs then the process returns error.



11. Consult the user's accounting (user_id, object_id) -> accounting data or ERROR

```
If (the user_id does not exist in user_rights)
    return error (the user does not exist).
else
    user = select user (user_id).
    If (the user has accounting data on object_id)
        accounting = user.consult accounting data( object_id).
    else
        accounting = 0.
    return accounting.
endif.
```

12. Consult the user's access rights (user_id, object_id) ->
access rights or ERROR.

```
If (the user_id does not exist in user_rights)
    return error (the user does not exist).
else
    user = select user (user_id).
    If (the user has access rights on object_id)
        access rights = user.consult access rights( object_id).
    else
        access rights = 0.
    return access rights.
endif.
```

13. Consult the policy of an object (object_id) -> object policy or ERROR

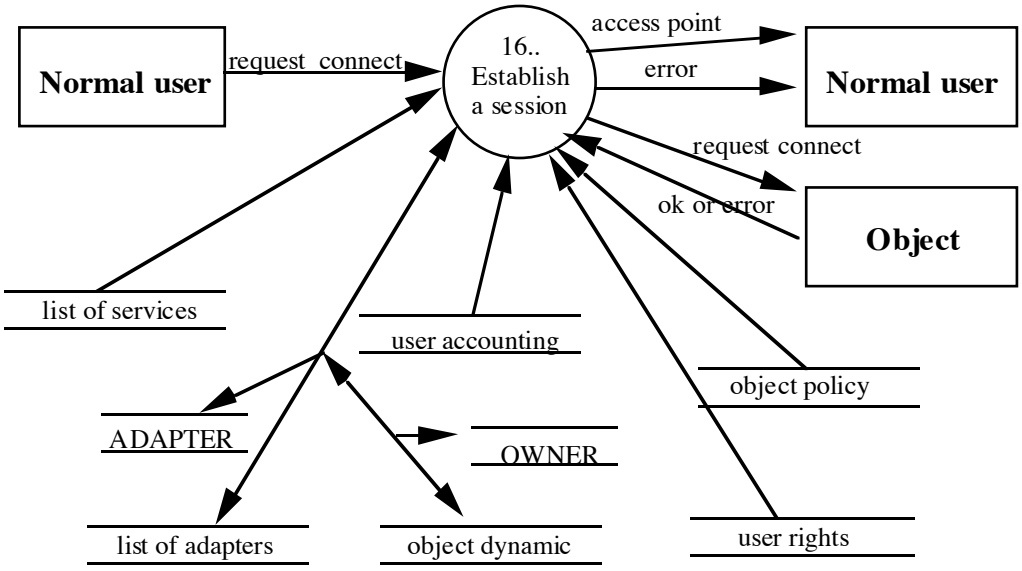
```
If (the object_id does not exist in object_policy)
    return error (the object does not exist).
else
    object = select object (object_id).
    object policy = object.consult object policy.
    return object policy.
endif.
```

14. Consult the object definition (object_id) -> object definition or ERROR

```
If (the object_id does not exist in list_of_services)
    return error (the object does not exist).
else
    object = select object (object_id).
    object definition = object.consult object definition.
    return object definition.
endif.
```

15. Consult the adapter policy of an object (object_id) ->
adapter policy or ERROR

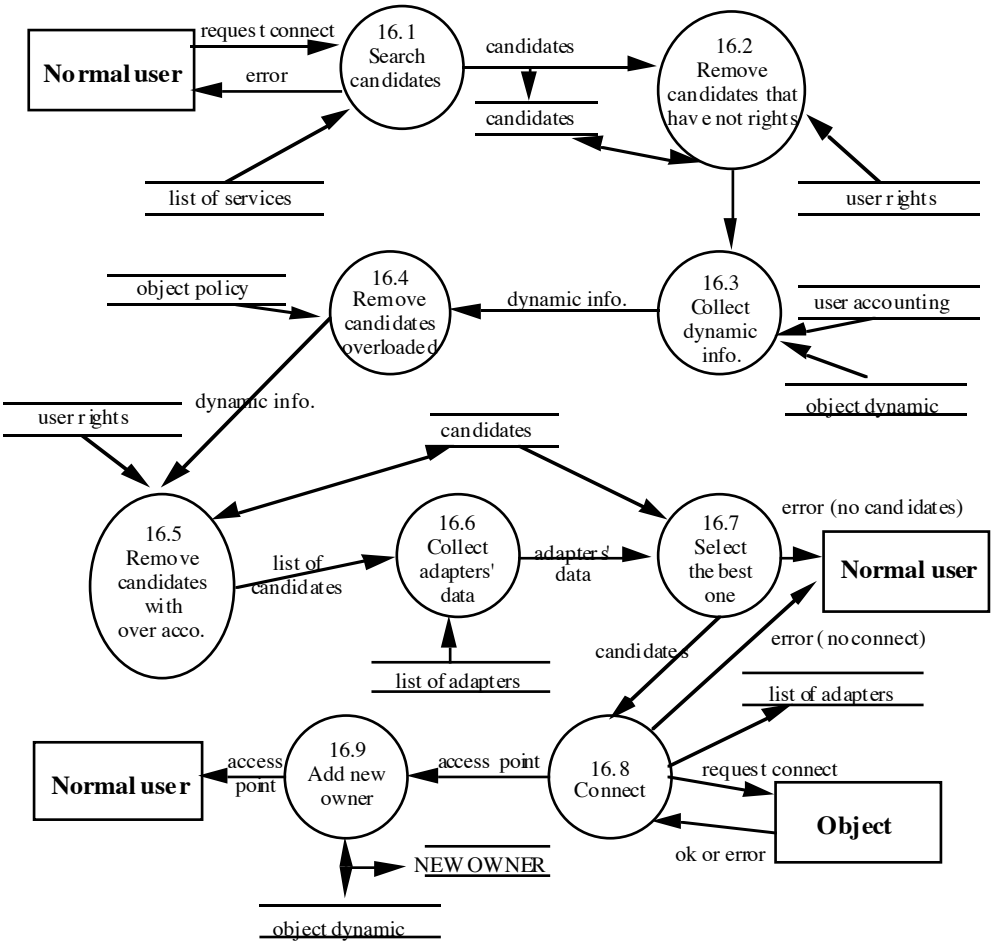
```
If (the object_id does not exist in list_of_adapters)
    return error (the object does not exist).
else
    object = select object (object_id).
    If (the object has adapter data)
        adapter policy = object.consult adapter data( object_id).
    else
        adapter policy = 0.
    return accounting.
endif.
```



16.1. Search candidates (characteristics) ->list of candidates or ERROR
The Finder selects the candidates from the list_of_services according to the characteristics. If any error occurs then the process returns error.

16.2. Remove candidates that the user has not access rights
(list of candidates, user_id) -> list of candidates.

```
If (the user_id does not exist in the user_right list)
    return error (the user does not exist).
else
    For each object in the list of candidates do
        access rights = consult user's access rights.
        If (access right is null)
            remove the candidate.
    endforeach.
    return list of candidates.
endif.
```



16.3. Collect dynamic information (user_id, list of candidates) ->
dynamic information.

```
dynamic information = empty.
For each object in the list of candidates do
    accounting data = consult accounting data (user_id).
    list of owners = consult owner of the object (object_id).
    dynamic information = dynamic information + (accounting data and
                          list of owners).
endforeach.
return dynamic information.
```

16.4. Remove the overloaded candidates (list of candidates,
dynamic information) -> list of candidates.

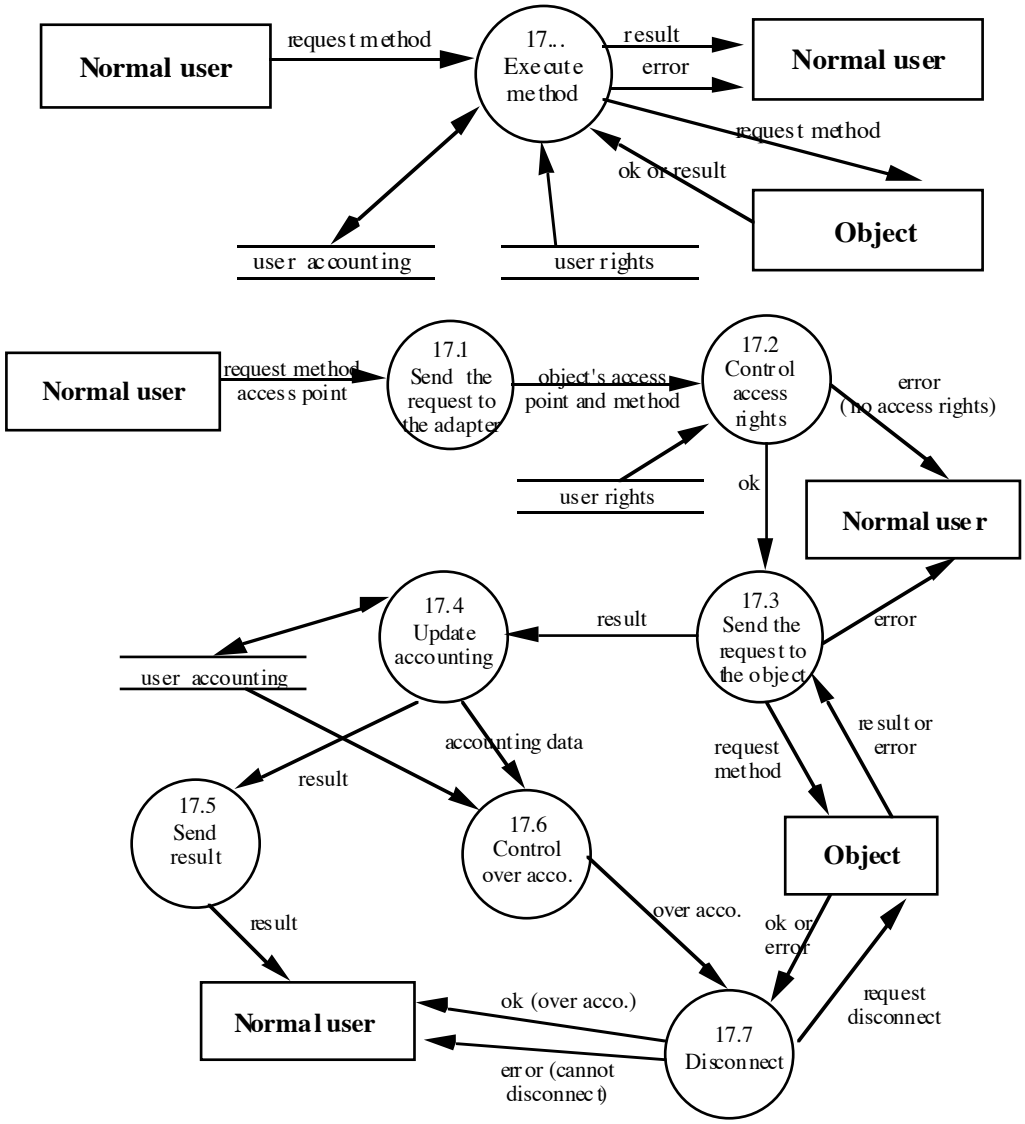
```
For each object in the list of candidates do
    maximum load = consult object policy.
    list of owners = dynamic information.list of owners.
    If (maximum load is less than the number of owners in the list of owners)
        remove the candidate.
endforeach.
return list of candidates.
```

16.5. Remove the candidates with over accounting (list of candidates,
dynamic information, user_id) -> list of candidates.
For each object in the list of candidates do
maximum accounting = consult maximum accounting (user_id, candidate).
accounting data = dynamic information.accounting data.
If (maximum accounting is less than the accounting data)
remove the candidate.
endforeach.
return list of candidates.

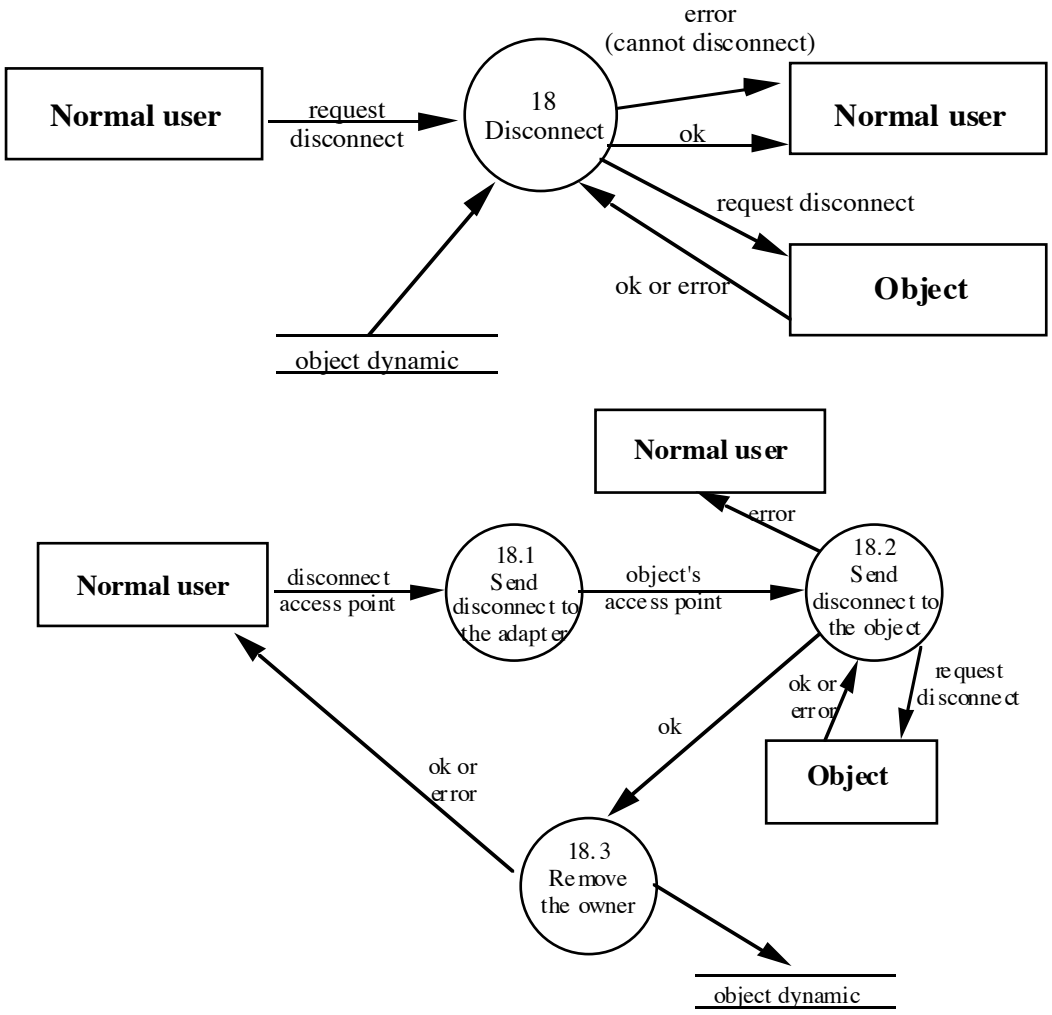
16.6. Collect the information about adapters (list of candidates) ->
information about the adapters.
adapter_info = empty.
For each object in the list of candidates do
data = consult information about adapters of each candidate.
adapter_info = adapter_info + data.
endforeach.
return adapter_info.

16.7. Select the best candidate (list of candidates,
information about adapters) -> the best candidate.
From the list of candidates, this function chooses the best one according to the adapter information.
If (the best one does not exist)
return error (no candidate).
else
return the best one.
endif.

16.8. Connect the candidate with the attached adapters (candidate,
information about adapters) -> access point.
Each object needs one or more adapters to adapt its functionality according to the environment policy. This is done by the adapters. So, this process connect the necessary adapters to the object and returns the access point. It is the access point of the last adapter in the chain of adapters.



17... The user calls an object method.
Send it to the adapter.
If (the access right of the user does not permit to execute this method)
    return error (no rights).
else
    Send it to object.
    response = wait for response.
    Update accounting data, adding one factor to accounting data.
    Send the response to the user.
    If (the maximum accounting > accounting data)
        Disconnect the user.
endif.



18... Disconnect.
Send disconnect request to the adapter.
The adapter sends it to the object.
response = wait for response.
If (response == error)
    return error (cannot disconnect).
else
    remove the user from the list of owners of this object.
endif.
return OK.

4. Data dictionary

In this section we present the main structures that we use in our prototypes. These structures are clustered according to the functionality of the Resource Manager, the Trader (Binder and Finder) or the adapters.

The data held by the Resource Manager represent the user's and object's rights, and the policies that can be applied on the environment elements. The necessary structures are the following: object_policy, user_rights, object_dynamic and user_accounting.

```
Type object_policy is a record with
        object is ident_object;
        moderator_policy is choose (only_first, sorting);
        maximum_load is integer;
        rules_of_events is list of rules;
        creator is ident_user;
end_type.
```

In this structure, we represent the object policies that the system needs to control the environment objects. These policies are the following:

- the inheritance policy of the moderator.
- maximum user load : this could be represented by number of users or number of sessions.
- rules to generate events.

```
Type user_rights is a record with
        user is ident_user;
        object is ident_object;
        rights is choose (read, modify, delete);
        maximum accounting is integer;
        moderator_rights is array(1... number_of_object_attributes) of
                choose (invisible, visible, editable);
end_type.
```

In this structure, we represent the user's access rights on several objects. These rights include :

- read : allows consulting operations,
- modify : allows updates and consulting operations
- delete : allows any operation.

The methods of each object are clustered according to these rights seen below and according to the security level that any operation needs. Inside this structure, the moderator policy and maximum accounting factor are also represented.

```
Type user_accounting is a record with
        user is ident_user;
        object is ident_object;
        accounting_factor is integer;
end_type.
```

In this structure, we represent the policies to control the users. In our prototype these policies are the accounting controls. Therefore, we store the user accounting factor that they have accumulated with their requests.

```
Type object_dynamic is a record with
        object is ident_object;
        owners is a list of ident_user;
end_type.
```

In this structure, we represent who has a session established with the objects. This information is used to calculate the dynamic load of an object.

The data held by the Trader are clustered according to the Finder functionality (list of services) or Binder functionality (list of adapters).

```
Type list_of_services is a record with
        methods is array (1...maximum_methods) of method;
        access_point is ident_object;
end_type.
```

This structure represents the interpretation of an object inside the environment. This interpretation is constituted by the access point (interface reference) and a set of object methods.

```
Type list_of_adapters is a record with
        object is ident_object;
        user_adapters is set of adapters;
        system_adapters is set of adapters;
        adapters_on is set of adapters;
        new_access_point is ident_object;
end_type.
```

This structure holds all the necessary information to establish a connection between both user and object. User_adapters is a set of adapters that the creator of an object can define in order to control their object; this set could be empty. System_adapters is a set of adapters that defines the minimum group required to control the access on each object. Adapters_on indicates how many adapters are connected with the object and which these are. Lastly, new_access_point indicates what the new access point is; it is the last connected adapter. This datum is an internal datum. The user always uses the same object name (interface reference). The adapters are invisible to users.

The data held by the adapters are the data of Trader and Resource Manager, but these data could be modified according to the state of the environment. These data could include the following:

```
list_of_methods is array (1 .. maximum_methods) of method;
access_point_to_object_or_next_adapter is ident_object;
list_of_owners is a list of records with
        username is ident_user;
        rights is choose ( read, modify, delete);
        moderator_rights is array (1 .. number_of_object_attributes) of
                choose (invisible, visible, editable);
        maximum_accounting is integer;
        accounting_factor is integer;
moderator, creator are ident_user;
moderator_policy is choose (only_first, sorting);
rules_of_events is list of rules;
```