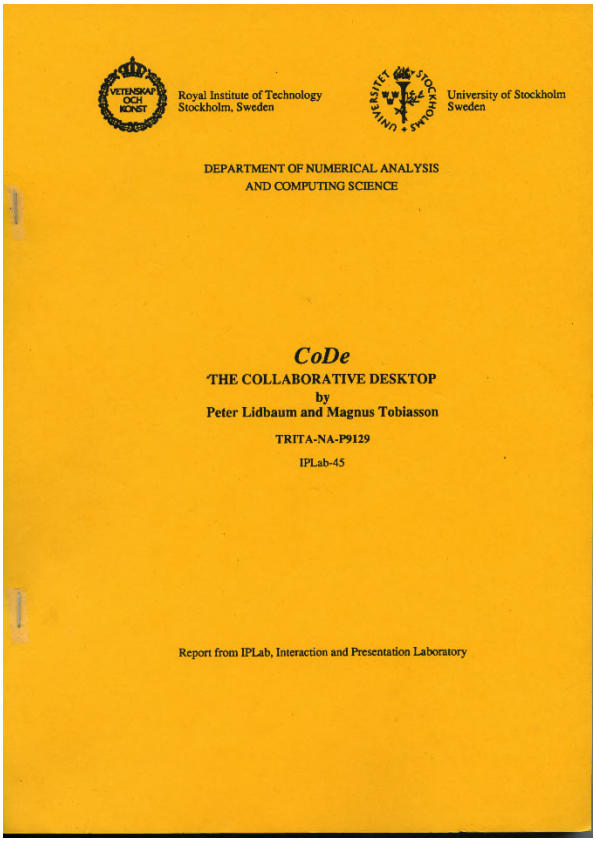
PDF-version

Peter Lidbaum & Magnus Tobiasson, Lidbaum,
CoDe – The Collaborative Desktop,
TRITA-NA-P9129, IPLab-45.
IPLab, Institutionen för numerisk analys och datalogi, KTH,
Stockholm 1991, 100 p.

I denna rapport, daterad okt. 1991, redogör Peter Lidbaum och Magnus Tobiasson för sitt gemensamma examensarbete i datalogi vid Kungl. Tekniska Högskolan, Stockholm, vilket resulterade i CoDe (The Collaborative Desktop), en samling samarbetsverktyg avsedda att användas i Unix-miljö. Mot slutet av sin civilingenjörsutbildning, ungefärligen då examensarbetet genomfördes knöts Peter och Magnus till olika projekt inom programmet Digital Litteratur och blev därmed kvar vid IPLab (Interaction and Presentation Laboratory), Institutionen för numerisk analys och datalogi vid KTH

Denna pdf-version återger rapportens utseende men tillåter inte OCR-tolkning. Innehållet är således inte omedelbart sökbart.



KUNGL
TEKNISKA
HÖGSKOLAN

CoDe

The Collaborative Desktop

PETER LIDBAUM
d87-pli@nada.kth.se
Datateknik, KTH

MAGNUS TOBIASSON
d87-mto@nada.kth.se
Datateknik, KTH

Referat

Denna rapport behandlar vårt arbete med CoDe, The Collaborative Desktop. Detta har gått ut på att skapa en mängd verktyg för datorstött samarbete och integrera dessa i den vanliga datormiljön. Först utvecklades en enkel prototyp i HyperCard för att prova och åskådliggöra våra idéer i ämnet. I det andra steget har de flesta av dessa idéer implementerats i UNIX-miljön. Verktygen inkluderar bland annat

- *Call Manager, en elektronisk telefonväxel med ljudöverföring över Ethernet och möjlighet att använda en distribuerad riteditor.*
- *Answering Machine, en telefonsvarare med möjlighet att lämna och avlyssna röst- och dokumentmeddelanden.*
- *Team Map, en översikt över deltagare i en arbetsgrupp.*

Verktygen utgör en påbyggnad till den existerande miljön (SUN:s Open Windows) och är tänkta att fungera smidigt i en verklig situation.

Contents

Förord

Så var då äntligen allt färdigt — efter snart fem månaders arbete. När vi nu blickar tillbaka kan vi konstatera att våra ansträngningar burit frukt, att alla sena kvällar och kriser inte varit förgäves.

Systemet vi skapat, **CaDe**, har naturligtvis både brister och fel. Trots detta är det inte *helt* utan stolthet vi i denna rapport presenterar det vi gjort. Vi tar tacksamt emot kommentarer och synpunkter via `e-mail`, och kan kanske till och med svara på en och annan fråga.

Ett arbete av denna storleksordning går inte att utföra i ett vaacum. Vi står i djup tacksamhetsskuld till bland annat följande personer.

- Våra handledare, (i bokstavsordning) Hans Marmolin, Yngve Sundblad och Konrad Tollmar. De har gett oss en perfekt avvägd kombination av stöd och frihet, och kommit med uppmuntran då vi behövt det mest.
- JoAnn Gerdin för att hon ordnat fram allt vi behövt, fläktar i sommarvärmen, pengar, *allt!* Stort tack även till andra IPLabanställda för att de stått ut med vår närvaro hela sommaren.
- NADA:s systemgrupp för att de alltid ställt upp (på de mest konstiga tider), hjälpt oss med krånglande datorer och guidat oss genom UNIX inre.
- Alla vi kommit i kontakt med på SICS.
- Våra "nära och kära".

Därmed ska vi inte slösa mer tid på förordet, utan överlämnar rapporten i era händer, kära läsare. Hoppas att den motsvarar alla förväntningar och inte gör någon besviken.

Stockholm, oktober 1991

Peter Lidbaum

Magnus Tobiasson

1 English Summary

Abstract

CoDe, The Collaborative Desktop, is a set of tools designed to support cooperative work, integrated in the desktop. The tools include:

- The `Call Manager`, a computerized telephone exchange with sound digitally transfered over Ethernet. `Call Manager` can control up to 4 calls simultaneously, each of which can share synchroneous tools, such as the `WhiteBoard`, a distributed drawing Editor.
- The `Answering Machine`, a process to control the flow of messages within **CoDe**. This is accomplished much in the same way as a real answering machine, thus using familiar metaphores.
- The `Team Map`, a visual overview of the current activities of the participants of a work group.

The tools have been added to an existing environment, SUN:s Open Windows.

1.1 Introduction

This is a brief summary of our MSc. thesis **CoDe**, The Collaborative Desktop. The full version (available only in Swedish) describes our work in detail, *what* we have done and *how* we did it. This summary only deals with the first of these issues.

CoDe is part of a Swedish research program, MultiG. The purpose is to develop and study new methods for Computer Supported Cooperative Work, as presented in Hans Marmolin's report TheKnowledgeNet[1]. This document provides a *vision* of how to add cooperative support to an existing environment. **CoDe** is an attempt to implement *some* of these ideas.

The tools are designed to support informal rather than formal collaboration. When the need for cooperation arises, **CoDe** should offer the necessary support in a way allowing users to adopt their own creativity. The tools include sound communication, voice mail and co-editing. It is important to stress that our implementation is not necessarily the *final* implementation, but rather a first attempt.

CoDe consists of a *set* of processes where some are collaborative tools, and others act as *tools for the tools*. The tools of the first category are the `Call Manager`, the `WhiteBoard`, the `Team Catalogue`, the `Catalogue Card` and the `Answering Machine`, while the second category includes `CoDe, Sound` and the `Displayer`.

[1] Marmolin, Hans, "TheKnowledgeNet — An Environment for Distributed Design", in *Proceedings from the 2nd MultiG Workshop*, Stockholm, June 1991

1.2 Platform

The environment used when creating ***CoDe*** was 2 SUN SparcStation IPC:s running UNIX and SUN:s Open Windows. The programming language is C. Programming tools include

- XView

  XView is an object-oriented package for creating user interfaces. It provides the programmer with the basic objects in use — windows, buttons, etc.

- `Devguide`

  SUN:s own interface builder `devguide` was used to create the user interface. `Devguide` consists of an interface *editor* and an interface *compiler*. The editor supports the programmer with interactive means of defining the interface, and the compiler creates the corresponding XView code.

- ISIS

  The ISIS toolbox, developed at Cornell University, provided us with the primitives necessary for inter process communication.

1.3 CoDe

`CoDe` is responsible for administration and coordination of the processes in ***CoDe***. While other processes are highly specialized for one or maybe a few tasks, `CoDe` handles a lot of tasks. The process is responsible for starting other processes, monitor status exchanges, handle access of shared files, control synchronous connections, etc.

`CoDe` also offers other processes a set of services such as process activation, access to `Catalogue Card`s of specified users and the connection establishment of phone calls. `CoDe` is working in the background and is never immediately visible to the user.

1.4 Sound

`Sound` is the process responsible for all audio services. Other processes are denied access of the audio device (`/dev/audio` on a SUN), and must direct their requests to `sound`. `Sound` offers a wide range of services to other processes. These are (among others):

- **Transfer** data from the workstation's microphone to other workstations' speakers. This is done digitally over Ethernet, using ISIS calls. The result is a PhoneTalk-like service, used by the ***CoDe*** process `Call Manager`.

- **Record** an audio message to file. This is used in several processes for recording messages in another user's `Answering Machine`.

- **Play** back recorded messages. This service is used by the `Displayer` to play previously recorded messages, and by other processes to play "effect"-sounds, such as dial tones when another user is calling, etc.

The requests for audio services is transfered between processes using ISIS calls. `Sound` also provides some services not currently used in **CoDe**, such as recording of conversations, etc.

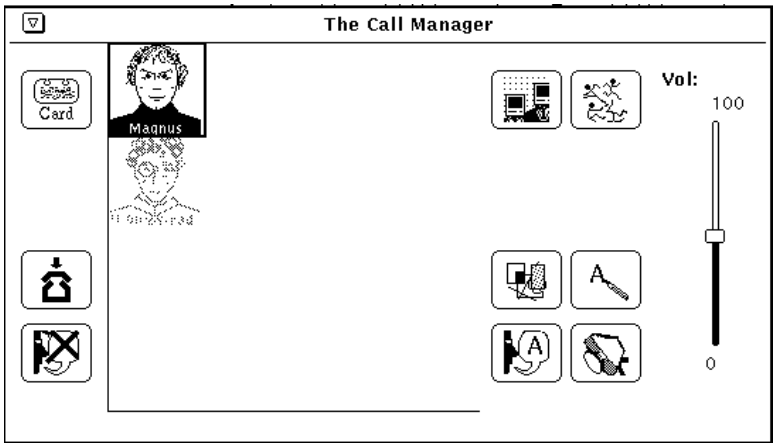


Fig.1: The Call Manager

1.5 Call Manager

The `Call Manager` is activated when a synchronous connection has been established between two or more group members. The user icons of the participants are displayed on a paint area, called the *Call Area*. `Call Manager` is able to handle four independent connections with at most five participants each. Each row in the Call Area represents a different call, and the user can switch between calls by selecting the desired row. Participants can be connected by dragging the icons from one call to another.

The `Call Manager` is also responsible for starting other synchronous tools, such as the `WhiteBoard`, the `Shared Editor` and the `Video`. This is a natural assignment since the Call Manager is the only tool used in connecting team members directly to each other.

The `Call Manager` is working intimately with `CoDe`. A specialized ISIS group is used for their internal communication.

1.6 WhiteBoard

The `WhiteBoard` offers multiple users a possibility of cooperative designing and drawing of a shared sketch consisting of simple geometric shapes such as lines, rectangles, circles and arcs. The graphical objects can be filled with different patterns, and use various line styles and thicknesses.

An object that has been placed on the paint area can be resized and moved. A set of objects can be grouped together and moved as a single object. Objects can both be copied and removed. When a user has selected an object or a set of objects, they become *his* resources. No one else can manipulate these objects before they are deselected.

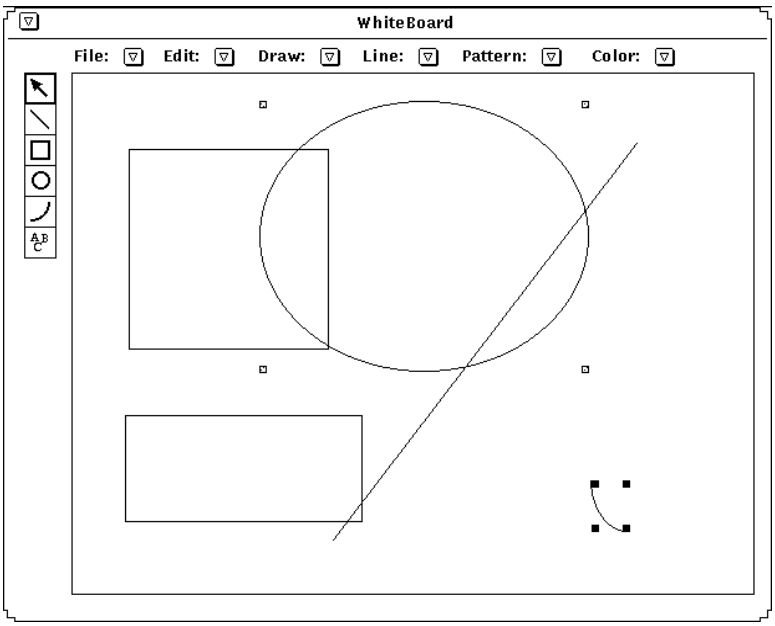


Fig.2: The WhiteBoard

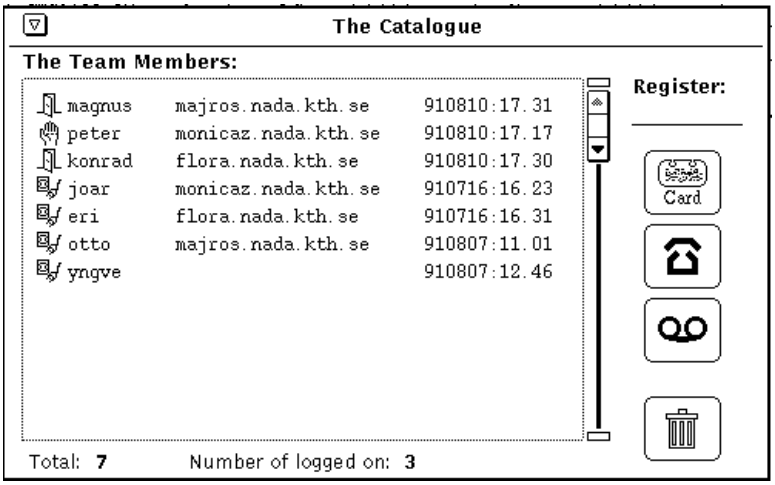


Fig.3: The Team Catalogue

1.7 Team Catalogue

The `Team Catalogue` keeps track of every CoDe member. Each member is displayed in a list along with the current status of availability, date and time of the last `login` and the name of the machine used. It is possible to select a member and give him a call, or to transfer an asynchronous message (i.e. press the record button and talk into the microphone). The `Catalogue Card` of the user can be displayed. It is also possible to enlist new CoDe members in the Team Catalogue.

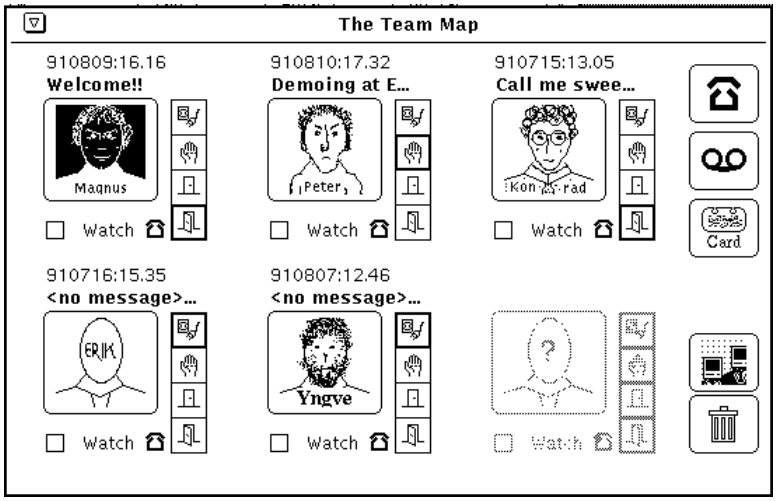


Fig.4: The Team Map

1.8 Team Map

The members that the user often want a summarized view of can be assembled in the `Team Map`. These members are probably people that the user collaborates with on a regulary basis, important contacts or co-workers in common projects. More information is displayed in the `Team Map` than in the `Team Catalogue`, as e.g. the *icon* of the member, the current *message* in the `Answering Machine` and an indication of whether the person is involved in a phone call. As in the `Team Catalogue`, the user may *call* members or send multimedia messages. The `Catalogue Card` of a specified user can be displayed.

The `Team Map` contains at most six team members. This may seem as a low limit, but it is resonable considering the amount of information displayed. Applications that cover a large part of the screen are harder to use in conjunction with other applications.

1.9 Card

All available information of a certain team member is displayed in his `Catalogue Card` along with a photograph of the user and a brief description of his function,

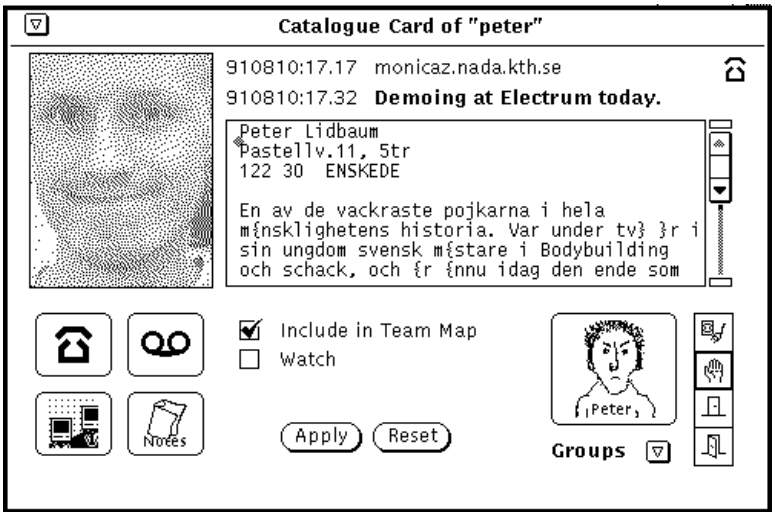


Fig.5: The Catalogue Card

current projects, knowledge, skills, etc. All information available in the Team Map about a member is also present in his card.

The user icon of the person can be defined. When the button containing the icon is pressed, the Open Windows application `iconedit` is activated, thus providing means of *editing* the icon. It is also possible to *call* the "owner" of the card or to leave a message.

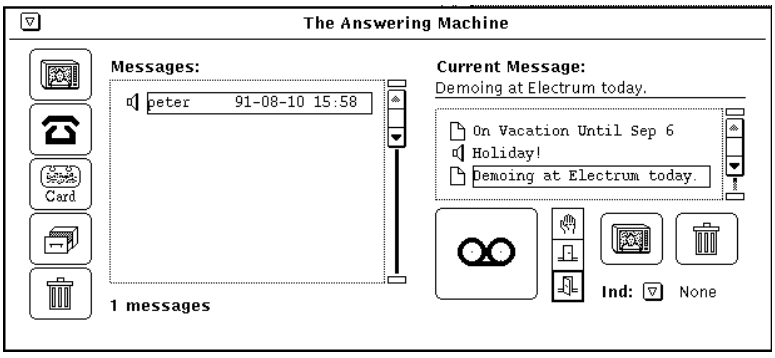


Fig.6: The Answering Machine

1.10 Answering Machine

The `Answering Machine` is the process responsible for handling messages. It consists mainly of two lists — the left list containing new, unread *incoming* messages, and the right list containing the personal, *outgoing* messages. The incoming messages can be displayed, replied to, saved and deleted, using the leftmost column of buttons.

The outgoing messages in the right list is supposed to show other **CoDe** users our *current status*. The *Current Message* is the selected item in the list of messages. A message contains mainly of the following three parts.

- The *message* itself, an audio/text file.
- The current *status*, as shown in the column next to the big recording button. The three options are (from the top):
  - **Don't Disturb**. In this mode, the user indicates that he/she would like to be undisturbed. No incoming calls are allowed.
  - **Closed Door**. A slightly more friendly alternative to the previous choice. This mode does not prevent any actions from taking place, and should be seen mainly as an indication of privacy.
  - **Welcome**. This is the default mode. The user is willing and able to be interrupted with questions or casual conversation.

  The user can associate one of the above to each message, thereby supplying other users with an indication of work stress.
- The message *string*. This is a user-supplied brief summary of the message, which is shown in other processes, such as the `Team Map` and the `Catalogue Card`. The string of the selected message is displayed in the text field `Current Message`, where it may be edited at will.

To add new messages, just press the big recording button and talk into the microphone. Text documents can be added using the Open Window's *drag 'n' drop* technique; select the file in the Open Windows *File Manager*, drag it onto the `Answering Machine` window, and drop it.

The personal list can thus be seen as a library of often-used messages, where the user quickly and smoothly can choose an appropriate item.

1.11 Displayer

The `Displayer` process is a rudimentary version of a generic MultiMedia message displayer. It currently supports only text/audio messages, but all interpretation of messages is separated from the rest of **CoDe** and handled by the `Displayer` thus making it simple and straightforward to add new message formats.

The `Displayer` has two main purposes, one is to display incoming and outgoing messages in the `Answering Machine`, and the other is to act as an alias for another user's `Answering Machine`. This is done when one attempts to call a user not currently logged in (or with his status set to `Don't Disturb`). In this situation, the `Displayer` will show the users current message, and supply a recording button for leaving a message (much in the same way as an ordinary answering machine).

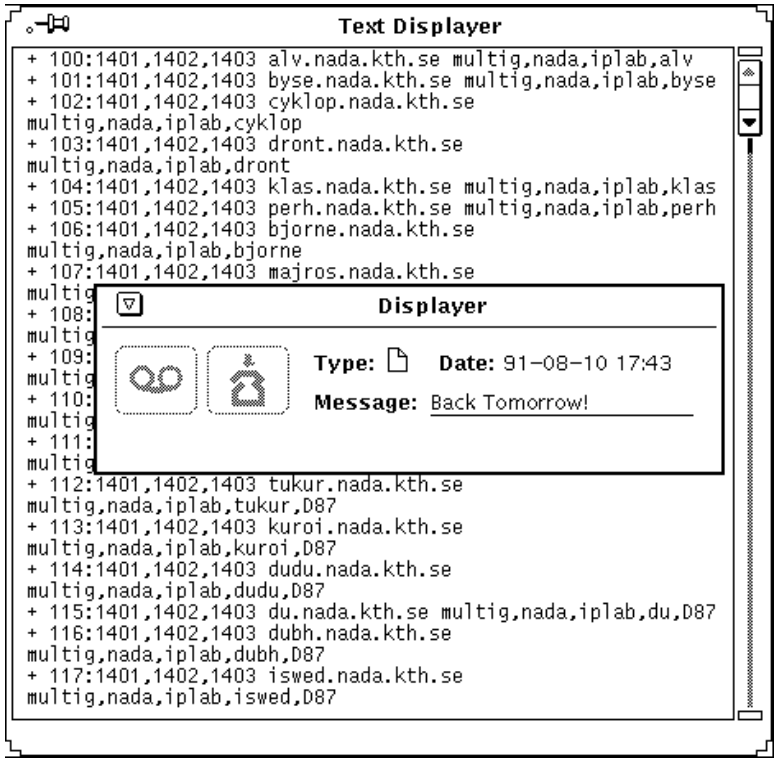


Fig.7: The Displayer

2 Inledning

Denna rapport behandlar vårt examensarbete. Med detta avslutar vi vår civilingenjörsutbildning vid linjen för Datateknik, KTH. Arbetet har utförts på institutionen för Numerisk Analys och Datalogi (NADA) vid Kungliga Tekniska Högskolan i Stockholm, som en del av ett större samarbetsprojekt kallat MultiG. Våra handledare har varit Yngve Sundblad, Hans Marmolin och Konrad Tollmar.

Av tids- och kompetensskäl har vi skrivit rapporten på svenska. För att kompensera detta har vi fyllt sidorna med försvenskade uttryck och "svengelska"[2]. Vänner av det engelska språket hittar en engelsk sammanfattning av CoDe som avsnitt 1.

Vid genomläsning av denna rapport kan valet av *nivå* verka underligt, vissa saker förklaras verkligen på djupet medan andra bara beskrivs ytligt. Detta är i alla fall *delvis* medvetet. Alla examinatorer, handledare och övriga intresserade läsare får ursäkta, men denna rapport är i *första* hand skriven för eventuella blivande exjobbare som ska gå i våra fotspår och fortsätta att utveckla CoDe. Vi vill peka på vad *vi* tyckt varit viktigt och vad vi fastnat på, så att andra inte ska behöva göra samma misstag.

2.1 MultiG

Det tvärvetenskapliga forskningsprojektet MultiG är ett samarbete mellan KTH, SICS, SISU, Televerket och Ericsson. Namnet är en akronym för *Multimedia in a Gigabit/s Network*, och som detta antyder omfattar projektet både utvecklingen av snabba nätverk och datorapplikationer som ska dra nytta av dessa. Målet är att ha ett fungerande Gigabit/s nätverk färdigt till 1993. MultiG består av flera, löst sammankopplade delprojekt. Vårt examensarbete ingår som en del i ett av dessa, *Computer Supported Cooperative Work in Design*.

Målet är att undersöka vilka nya metoder för datorstött samarbete som blir möjliga med de nya näten, och även utveckla dessa. Detta är inte bara ett självändamål, utan kan ha stor betydelse för att integrera de datormiljöer som finns vid KTH och vid Electrum. De övriga delprojekten är spridda över ett brett spektrum, från fibernivån upp till tredimensionella användargränssnitt.

2.2 TheKnowledgeNet

Grundvalen för hela vårt arbete har varit Hans Marmolins rapport *TheKnowledgeNet* [Marm91]. Denna beskriver en *vision* av hur samarbete bör bedrivas, samt ger förslag till en uppsättning generiska verktyg för att stödja detta. Det var denna rapport vi hade som förlaga då vi gjorde vår prototyp. Med tanke på den snäva tidsram den inledande prototypfasen rörde sig inom var det tur att en stor del av tänkandet redan var avklarat.

Det är svårt att kortfattat beskriva vad TheKnowledgeNet *är*. Vi kan tänka oss ett *moln* av kunskap, lite som ett bibliotek där alla författare finns på plats, beredda att svara på frågor. De användare som är anslutna till TheKnowledgeNet besitter var och en viss *kunskap*, både rent mentalt och i form av doku-

[2] Det är svårt att på svenska tala om vår process `Call Manager` i bestämd form pluralis.

ment. Tanken är att denna kunskap ska kunna *distribueras* ut i nätet, för att göra det lättare för gruppen som helhet att lösa problem. TheKnowledgeNet erbjuder de verktyg som behövs för att göra detta möjligt.

The Collaborative Desktop är helt enkelt dessa verktyg kompakt förpackade, integrerade i den vanliga datormiljön. På så sätt ska dessa alltid vara tillgängliga.

2.3 Rapportens Uppläggning

Tidigare versioner av denna rapport fick kritik för att de var "splittrade" och att det var svårt att hitta. Därför har vi här försökt dela in rapporten i tre mer eller mindre avgränsade delar:

- HyperCardprototypen
- Programmeringsverktyg
- ***CoDe*** — The Collaborative Desktop

Dessutom finns som appendix A en ingående lätt kåserande beskrivning av vår arbetsmetodik och som avsnitt 1 en engelsk sammanfattning.

Fig.8: Rapportens uppläggning

Den som bara är intresserad av en viss del av rapporten kan med hjälp av ovanstående tabell snabbt bläddra till önskad sida.

3 HyperCardprototypen

Detta avsnitt behandlar dels den inledande prototyp vi utvecklade i HyperCard, dels en del viktiga definitioner av olika begrepp som det kan vara bra att ha läst.

3.1 Grundläggande Metaforer och Begrepp

Vi inledde vår brainstorming med att försöka sammanfatta hur samarbete och kommunikation utförs idag. Det överlägset vanligaste sättet att ta kontakt med varandra över större avstånd är *telefonen*. Telefonens dominans på området är så total att det är svårt att se *förbi* dess begränsningar, vi är så färgade av telefonen att vi har svårt att hitta andra, alternativa sätt att utföra samma tjänster. Att integrera en förbättrad "supertelefon" med ett datornät och alla dess möjligheter, allt ständigt åtkomligt i desktopen borde vara en framkomlig väg. Vår allra första idé var att utforma hela miljön runt telefonen, och bygga in övriga verktyg i denna. Nu stred detta både mot de metaforer vi arbetat fram och de förutsättningar vi fått, så vi övergav idén.

3.1.1 Samtalet

Det kanske viktigaste begreppet av alla är vår definition av *samtalet*. Med ett samtal menar vi en uppkopplad förbindelse mellan två eller flera parter, genom vilken vi kan överföra data. Data kan här vara tal, bild, filer eller dokument[3], eller en kombination av dessa. I ett samtal skall de vanliga konferenshjälpmedlen finnas lätt tillgängliga. Antalet deltagare i ett samtal ska kunna variera över tiden, och samtalet ska existera så länge det har minst två parter. Om än vanligt i verkligheten, saknar enpartssamtal relevans i vår modell.

"Om än vanligt i verkligheten, saknar enpartssamtal relevans i vår modell"

3.1.2 Gruppen

En grupp är en samling användare, antingen formell eller informell. Grupper ska kunna skapas, ändras och raderas när och hur som helst. De existerar både för att kunna erbjuda en abstraktion för samarbete (människor vi ofta arbetar tillsammans med), och för att strukturera upp användare logiskt (människor vi ofta till exempel skickar samma post till). En användare kan vara med i godtyckligt många grupper, och grupper kan i sin tur vara organiserade i subgrupper.

3.1.3 Desktopen

En *desktop* är kort uttryckt en skärmyta, på vilken arbete utförs. En desktop kan vara av tre olika slag:

- en *personlig* desktop
- en *grupporienterad* desktop
- en tillfällig, *samtalsorienterad* konferensdesktop

[3] Vi använder genomgående termen dokument i en multimediell mening.

Den personliga desktopen motsvarar den desktop vi möter på en Apple Macintosh, eller motsvarande. Det är här vi utför allt personligt arbete, och även sköter det mesta av samarbetet. Den egna desktopen kan användaren naturligtvis inreda och skräddarsy som han eller hon själv vill.

Gruppens desktop är gemensam för alla gruppmedlemmar. Så fort en ny grupp skapas får den sig automatiskt tilldelad en egen, virtuell desktop (det vill säga att den inte är bunden till någon specifik maskin). I gruppens desktop finns en full uppsättning av de verktyg vi senare ska presentera; sålunda finns det en anteckningsbok (NoteBook) för varje grupp, där gruppens anteckningar om gemensamma projekt och dylikt förs. Här finns även länkar ner till varje gruppmedlems anteckningsbok, så att man från gruppnivån kan få total översikt över alla anteckningar i ämnet. På samma sätt kan meddelanden till en hel grupp skickas till *gruppens* telefonsvarare, för att därifrån distribueras ner till varje enskild grupp. Mer om detta senare.

En desktop kan även delas på ett enklare sätt, genom att "besöka" en medarbetares desktop kan denna visas på en annan skärm i ett separat fönster. Beroende på vilka rättigheter ägaren av den distribuerade desktopen tillåtit kan andra se/ändra i desktopen som om den var deras egen, och dessa ändringar återspeglas på den ordinarie desktopen.

3.2 Inledande idéer

En av förutsättningarna var att de nödvändiga samarbetsverktygen skulle vara så små, så *atomära* som möjligt. Vi diskuterade utförligt olika uppdelningar, och kom fram till att *verkligen* minimera verktygens omfattning är ett svårt problem, både datalogiskt och socialt. I stället har vi koncentrerat oss på att producera ett antal något större enheter, så pass stora att de är egna applikationer. Den möjlighet att skräddarsy sitt verktyg som små enheter ger har vi försökt ersätta med att ha *länkar* mellan verktygen, som på ett "intelligent" sätt ska kombinera de olika modulerna.

De krav vi ställde på verktygen, oberoende av hur de skulle struktureras och implementeras var följande:

- Det grundläggande sättet att kommunicera bör vara via röst/audio. Även bild/video måste finnas tillgängligt i systemet, men mer som ett sorts komplement, *tyngdpunkten* ska ligga på att få röstkommunikationen att fungera så smidigt som möjligt, utan kompromisser.
- Samtalet måste betonas som kommunikationsenhet. Det måste gå snabbt och enkelt att ta kontakt med vilken annan användare som helst.
- Det måste vara lätt att hålla flera samtal igång samtidigt och att snabbt skifta mellan dessa. Det ska även vara lätt att koppla ihop olika parter med varandra.
- Om en potentiell samtalspartner inte är tillgänglig bör det vara extremt enkelt att lämna ett meddelande.
- Under samtalets gång måste det finnas konferensverktyg som gemensamma Whiteboards, delade fönster, medel för smidig filöverföring och dylikt tillgängliga.

- Kunskapsbasen ska finnas omedelbart tillgänglig, både för konsultation och uppdatering. Anteckningar om personer ska kunna göras både privat och publikt.
- Vad som helst bör kunna *sparas* undan för senare behov. Att spara en skärmdump eller ett personligt röstmeddelande (diktafon) ska vara enkelt och intuitivt.
- Man måste snabbt kunna få en översikt över vilka användare som är inloggade och smidigt kunna nå dessa.

3.3 Verktygen

Våra inledande diskussioner ledde till en uppdelning av de tjänster vi ansåg viktiga i ett antal verktyg. Dessa presenteras i de följande avsnitten.

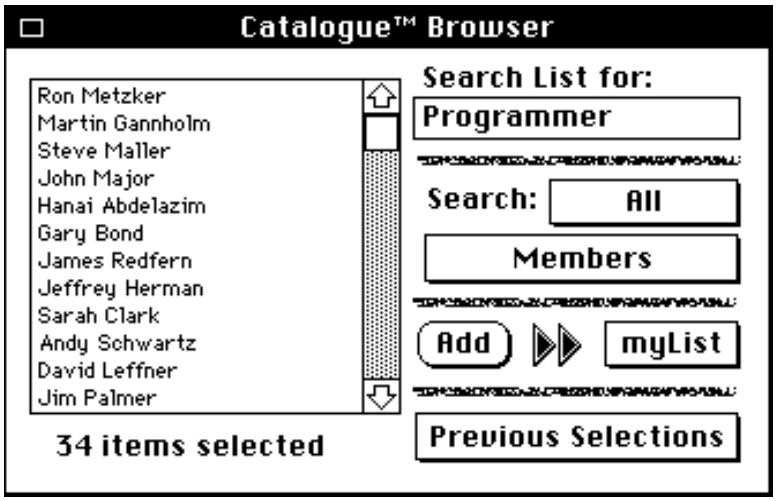
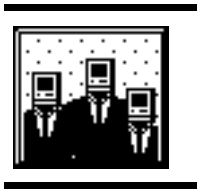


Fig.9: The Catalogue Browser

3.3.1 The Catalogue



The Catalogue (eller Katalogen[4], som vi kan kalla den på svenska) är ett generellt verktyg för att dokumentera och söka efter den kunskap som finns i systemet, både *mentalt* och *dokumenterat*. Mental kunskap finns förborgad i de olika teammedlemmarnas huvuden, medan den dokumenterade kunskapen även finns i skriftlig form.

The Browser Katalogen består grovt sett av två delar, själva browsern och kunskapskorten. Browsern visas i figur 9. Vilka som ligger i den stora listan till vänster vid sökningens början bestäms av innehållet i de popupmenyer som

[4] I vår ursprungliga totaltelefonmetafor var `The Catalogue` naturligtvis en avancerad telefonkatalog.

kommer fram om man klickar på knapparna i regionen `Search` till höger. Dessa kan användas för att sålla ut de kategorier av användare man vill titta på. *Användarmenyn* (i läge `Members` i figuren) har alternativen:

- `Members`. Visa bara de enskilda användarna.
- `Groups`. Visa enbart *grupper*.
- `Members & Groups`, som visar båda ovanstående alternativ.

Statusmenyn (i läge `All` i figuren) har alternativen:

- `All`. Visar *alla* i nätet ingående personer/grupper (beroende på hur användarmenyn är satt.
- `Logged on`. Inskränker urvalet till de som är *inloggade* för tillfället.
- `Available`. Samma som ovanstående, med det tillägget att urvalet ska gälla bara de användare som verkligen är *tillgängliga*, alltså inte har markerat sig själva som upptagna.

Det som söks igenom är de individuella kunskapskorten, och *sökningen* sker fritt (typ `grep` i UNIX). Detta eftersom den sorts kunskap som vi vill kunna söka efter svårligen låter sig formuleras i formulärform. Den sökmetafor vi använder är *sållet*. Användaren matar in ett sökbegrepp, och användarlistan till höger sållas ut, så att endast de personer som uppfyller detta blir kvar. Denna nya lista kan sedan sökas igenom på nytt, med avseende på något annat sökbegrepp, tills urvalet blivit överskådligt.

Om urvalet vid något tillfälle skulle bli för litet (eller till och med *tomt*) kan man vid vilket tillfälle som sist gå tillbaka till en tidigare söknivå, genom en popupmeny kopplad till knappen `Previous Selections` nere i högra hörnet.

Som resultat av sin sökning ska användaren kunna plocka med sig poster, genom att markera dessa och addera dem till `myList`. Denna popupmeny utgör en *container*, som användaren tar med sig till katalogkorten, för att snabbt kunna skifta mellan sina val utan att behöva gå tillbaka till browsern.

Katalogkorten Dessa utgör *kärnan* i katalogen, och är den grund på vilken hela idén vilar. Korten kan vara av två typer:

- Personliga kort
- Gruppkort

I figur 10 visas ett exempel på ett personligt kort. Kortet domineras av en bild på användaren och ett textfält där denne givit en kortfattad beskrivning av sig själv. Under bilden finns tre större knappar, `Call, Visit` och `Record`.

- `Call` är en snabbknapp för att ringa upp personen. Knappen är dimmad om personen inte är tillgänglig. På gruppkorten ska funktionen vara att man ringer upp *samtliga* gruppmedlemmar, och att de som kan svarar.

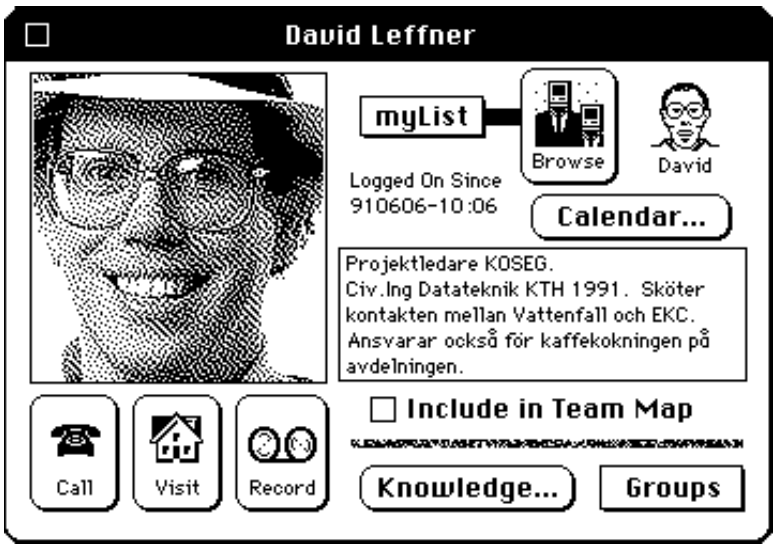


Fig.10: De enskilda katalogkorten

- `Visit` öppnar personens desktop i ett separat fönster på skärmen, om han/hon tillåter detta. På så sätt kan användare enkelt se vad andra håller på med, utan att störa dessa. På gruppkorten leder knappen till gruppens "virtuella" desktop.

- `Record` är en snabbknapp till användarens telefonsvarare. Det är bara att trycka ned knappen med musen och tala in sitt meddelande. På gruppkorten sprids meddelandet ut och hamnar i samtliga gruppmedlemmars telefonsvarare.

En ikon leder tillbaka till Browsern, medan `myList`, som vi erinrar oss var den popuplista med intressanta personer som vi tog med oss från browsern, erbjuder ett snabbt sätt att hoppa mellan dessa. En liten ikon av användaren (som han eller hon naturligtvis kan definiera om efter tycke och smak) kan dras loss från kortet och placeras på desktopen. Denna kan fungera som en sorts "pipeline" in i personens telefonsvarare[5], så att allt som släpas och släpps där flyttas över dit.

En popupmeny `Groups` innehåller alla grupper som personen är medlem i, och att välja en av dessa byter till denna grupps katalogkort. Väl i gruppkortet ersätts menyn med en meny `Members` som är länkad åt *andra* hållet, det vill säga innehåller alla gruppmedlemmar. Val ur denna byter till deras individuella katalogkort. På så sätt går det snabbt att manövrera sig fram genom grupper och deras användare.

Kryssrutan "`Include in Team Map`" avgör om personen ska vara med i min personliga TeamMap, mer om detta längre fram.

Återstår två knappar med länkar ut till andra kort, `Calendar` och `Knowledge`. `Calendar` plockar upp personens personliga kalender, så att vi där lätt kan se

[5]Eller som ett användarspecifikt "svart hål", som suger in allt vi släpper i närheten.

om denne till exempel är på semester i tre veckor. Då folk i allmänhet inte är så duktiga på att uppdatera sina kalendrar kan denna funktions nytta kanske diskuteras. Kanske behövs någon form av morot.

`Knowledge` leder till det *egentliga* kunskapskortet. Detta (som vi tyvärr inte har någon riktigt bra bild på) innehåller all den information som letas igenom vid sökningar i browsern. Kortet består av tre olika delar. I den del första delen får personen själv sammanfatta sina kunskaper och specialområden. I den andra kan jag som användare ge kommentarer om personen som jag tycker att andra bör läsa, till exempel kunskapsområden som personen själv inte tagit upp eller specifika problem som personen hjälpt mig med. Slutligen finns en area med egna kommentarer.

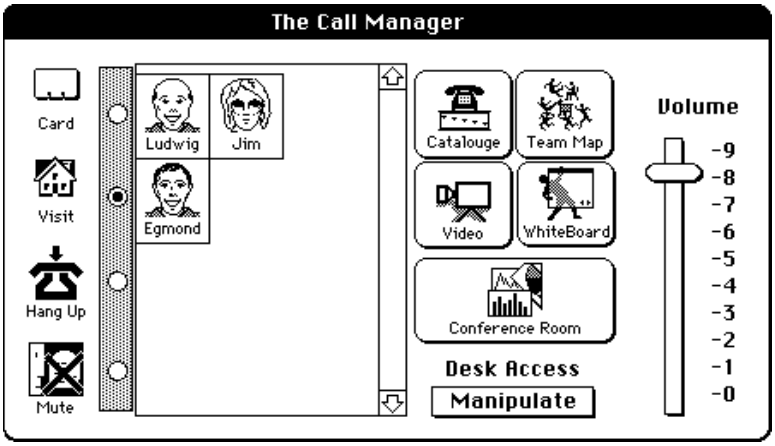


Fig.11: The Call Manager, hjärtat i kommunikationen

3.3.2 The Call Manager

The Call Manager är i princip en avancerad telefonväxel. Det går att ha ett flertal samtal uppkopplade samtidigt, och det går lätt att snabbt växla mellan dessa. Om vi studerar figur 11, ser vi att Call Manager i princip består av ett scrollande fält med samtal och en mängd ikoner, som var och en representerar möjliga operationer på det aktuella samtalet, eller på aktuell person.

Vi börjar med att studera samtalslistan. Varje rad i ett listan representerar ett samtal. I figuren har vi alltså två samtal igång, ett trepartssamtal med Ludwig och Jim, samt ett tvåpartssamtal med Egmond. Radioknapparna till vänster markerar vilket av samtalen som för tillfället är aktivt, och genom att shift-klicka på flera knappar kan flera samtal vara aktiva samtidigt.

För att koppla ihop två samtal, eller för att flytta personer mellan *olika* samtal klickar man på dessa och drar dem till önskad rad. På så sätt kan man enkelt bygga ihop sina samtal på samma sätt som ett barn leker med legobitar.

De olika ikonerna har följande funktioner.

- `Card` är en länk till aktuell persons kunskapskort, så att det lätt ska gå att undersöka en främmande persons identitet innan man svarar.

- `Visit` plockar (på samma sätt som på kunskapskortet) upp aktuell persons desktop i ett fönster på skärmen.
- `Hang Up` är ett generellt verktyg för att lägga på luren.
- `Mute` är en sekretessknapp, om vi håller den nedtryckt hör ingen av våra samtalspartners vad vi säger.
- `Catalogue` är en snabblänk till browsern.
- `Team Map` är en snabblänk till vår *Team Map*, som vi kommer att behandla utförligare några sidor längre fram.
- `Video` öppnar en videokanal mellan parterna i ett samtal. Denna öppnar automatiskt även Videoverktyget.
- `Whiteboard` är en gemensam kladdyta som delas av alla i samtalet ingående parter. I denna ska det gå att både rita och skriva, vara möjligt att klippa och klistra in vad som helst, att demonstrera idéer för varandra, samtidigt som alla har röstkontakt.
- `Conference Room` påminner mycket till sin struktur om Whiteboard, men är mer raffinerad. Ett konferensrum är en virtuell desktop, mycket lik den virtuella gruppdesktop vi beskrivit tidigare. Här kan vi demonstrera program för varandra, starta applikationer, och göra i princip allt som går att göra i en personlig desktop.
- `Desk Access` är en popupmeny som reglerar vilka *rättigheter* våra samtalspartners har att manipulera vår desktop. Valmöjligheterna är:
  - `Locked`, som innebär att ingen får öppna vår desktop.
  - `Only Look` tillåter att andra öppnar vår desktop, men tillåter inte att de utför några ändringar.
  - `Manipulate` ger våra medarbetare fulla rättigheter att göra vad som helst på vår desktop.
- `Volume` justerar volymen. Exakt vad vi menar med volym i multimediell mening är lite oklart, men preliminärt kan vi definiera det som ljudvolymen.

3.3.3 The Answering Machine

The Answering Machine, telefonsvararen, betecknar vi som en av våra stora innovationer. Det innovativa tycker vi egentligen är sättet att *tala in* meddelanden till andra[6], inte sättet att lyssna av våra egna. Telefonsvararen (som finns på bild 12) påminner starkt om en lite mer avancerad mailhanterare, av typ Slate.

I meddelandekön ligger alla nya meddelanden. På ikonen framför syns vilken *typ* meddelandet har (röst, video, dokument, etc.) Dessa kan avlyssnas[7] eller

[6] ...just press and talk...(för att citera en reklamslogan i vår video)
[7] Naturligtvis i multimediell mening, därav ögat som ikon.

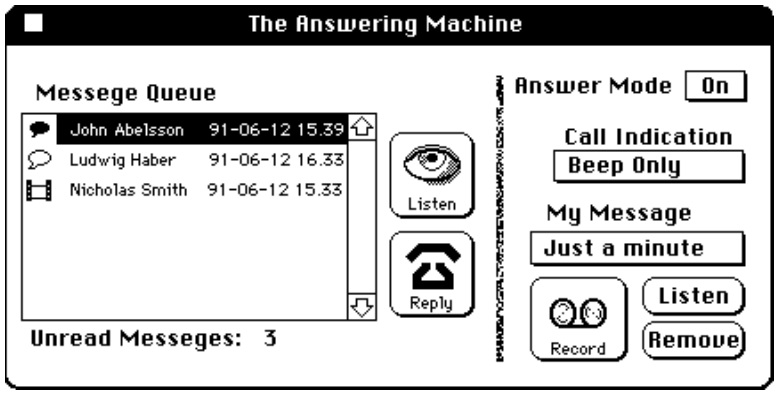
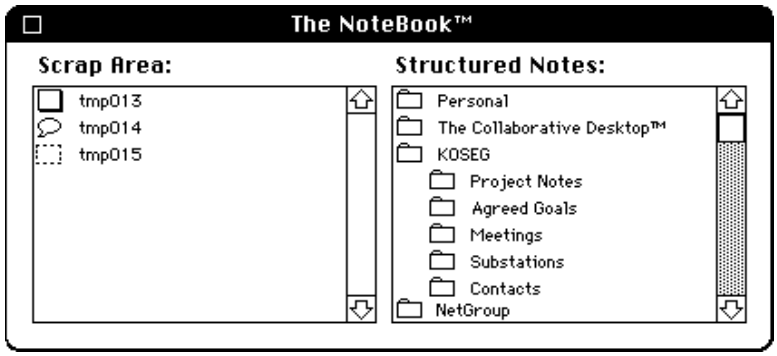


Fig.12: The Ansvering Machine, en utökad telefonsvarare

besvaras med ikonerna brevid listan. Om något meddelande skulle vara så intressant att det kan anses värt att spara drar vi bara meddelandet till en lämplig destination (vanligtvis vår NoteBook).

Den andra halvan av telefonsvararen ägnas åt personliga inställningar. En popupmeny anger om telefonsvararen ska vara på eller av (man kan vilja ha telefonsvararen på även när man är "hemma"), och två andra popupmenyer sätter några egendefinierade standardsvar (typ "Ute på lunch"), respektive väljer vilken indikation vi själva ska få när andra ringer.

Med knappen `Record` går det slutligen att snabbt tala in ett meddelande som andra ska få när de ringer oss. Alternativt ska det gå att dra dit en fil med ett multimediellt meddelande.



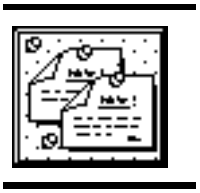


Fig.13: The NoteBook. Verktyget för generellt sparande

3.3.4 The NoteBook och The Remote Control

The NoteBook (figur 13), vår *anteckningsbok*, kan ses som ett försök att förverkliga en av Vannevar Bush's idéer [Bush45]. Bush's vision av en vetenskapsmans dagliga arbete var att denne ständigt skulle bära ett par glasögon med inbyggd kamera. Så fort vetenskapsmannen såg något som intresserade honom tryckte

han på en knapp och hans glasögon förevigade det han just tittade på. Detta har vi försökt att efterlikna elektroniskt, med hjälp av två verktyg, The NoteBook och The Remote Control.

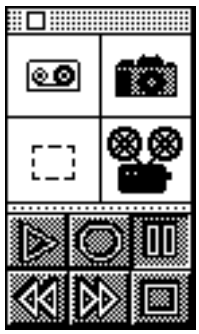


Fig.14: The Remote Control

The Remote Control (figur 14), eller *fjärrkontrollen* står för "glasögonbiten" i Bush's vision, detta att enkelt kunna inhämta information. The NoteBook erbjuder smidiga sätt att spara undan denna. Dessa två hör intimt samman, så vi ska försöka beskriva dem under samma rubrik.

Fjärrkontrollen är ett sorts "metafönster", ungefär som verktygslådan i HyperCard. Den flyter alltid ovanpå alla andra fönster och blir således aldrig dolt i bakgrunden. De verktyg som finns tillgängliga är:

- **Diktafonen** fungerar på samma sätt som när man talar in ett meddelande på någon annans telefonsvarare, bara det att meddelandet istället hamnar i ens *egen* NoteBook.
- **Markören**, som markerar ett skärmområde för senare bearbetning med kameran eller videoverktyget.
- **Kameran** tar ett snapshot på det markerade området.
- **Videon** bevakar det markerade området och spelar in allt som händer inom detta.

Resultatet av alla dessa operationer hamnar i vår NoteBook, närmare bestämt i den del av denna som kallas *The Scrap Area* (eller *slaskarean*, på svenska). Allt ligger som filer, i kronologisk ordning, och med automatgenererade namn[8]. Detta mest för att vi vill att det ska vara *extremt* enkelt att spara undan information, vi ska inte förvilla användaren med en mängd dialogrutor för filnamn, underbibliotek och dylikt, det ska bara vara att slänga in allt i slaskarean och arbeta på, medan inspirationen finns kvar!

När glöden har sinat, eller då man får tid över kan det vara dags att strukturera upp dagens tankar. Vi har vår slaskyta till vänster och våra mappar till höger, och börjar gå igenom våra anteckningar. Om vi dubbelklickar på en av

[8] ...som `tmp013`, eller något annat lika självförklarande.

dessa presenteras den på "rätt" sätt, röstmeddelanden till högtalaren, och så vidare. På så sätt kan vi enkelt associera anteckningen till rätt plats i strukturen, och flytta över det dit. Vid detta tillfälle måste vi ge användaren möjlighet att ersätta det automatgenererade filnamnet med ett nytt. Några anteckningar har kanske redan hunnit bli inaktuella, så dessa slänger vi i papperskorgen.

Detta sätt att arbeta förutsätter kanske en hel del disciplin, och har man den inte från början lär man säkert förvärva den efter ett tag, då ens slaskarea svämmar över. Fast somliga lär sig nog aldrig.

En annan finess är att varje *grupp* har sin egen NoteBook. I denna går det att anteckna saker som är specifika för gruppen, plus att det finns länkar ned i varje enskild gruppmedlems NoteBook. Sålunda kan all information om ett projekt sägas ligga samlad på ett enda ställe.

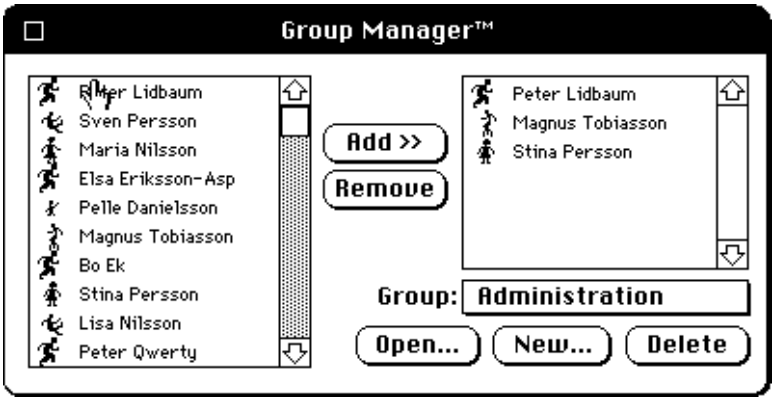


Fig.15: The Group Manager

3.3.5 The Group Manager

Som en av våra förutsättningar hade vi att det måste vara lätt att organisera människor i grupper. The Group Manager löser detta problem.

Sättet att arbeta påminner mycket om hur man installerar typsnitt på en Macintosh. Vi har alla användare i listan till vänster (figur 15) och aktuell grupp till höger. Där kan vi flytta över användare hur vi vill, och även skapa nya och ta bort grupper. Kunskapskorten uppdateras automatiskt. Vid skapandet av en ny grupp skapas även gruppens virtuella desktop och vissa andra gruppspecifika verktyg, exempelvis skapas en mapp med gruppens namn i alla gruppmedlemmars NoteBook.

3.3.6 The TeamMap

The TeamMap är tänkt som en lista över de användare som vi arbetar mycket tillsammans med, ofta pratar med och vet att vi kan behöva störa med frågor. Detta motsvarar den sociala situation där vi knackar på olika dörrar och ser efter vilka som är inne. Vilka som finns med bestäms av kunskapskorten, alla vi markerar som `Include in TeamMap` läggs till i listan. Alternativt skulle

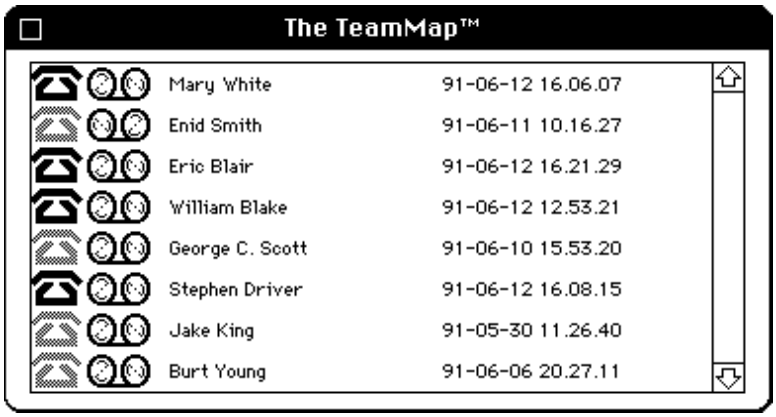


Fig.16: The TeamMap

vi kunna tänka oss automatgenerering av en lista med de personer man talat mest med. Detta skulle dock kunna upplevas som ett hot mot vår personliga integritet.

Idén är att vi ska ha våra nära och kära snabbt tillgängliga. Med telefonikonen längst till vänster (figur 16) kan vi snabbt ringa upp vilken av dessa vi vill. Om användaren inte är tillgänglig är ikonen dimmad. Telefonsvararikonerna leder på samma sätt som från kunskapskorten rakt in i telefonsvararna. Det klockslag som visas är den senaste tidpunkten då användaren visade något tecken på aktivitet.

3.3.7 The Video Controller

Som det sista av våra verktyg kommer videokontrollen (The Video Controller) Denna fungerar ungefär som en vanlig videobandspelare, med snabbspolning, in- och uppspelning. Även här kan vi ta ett snapshot av bilden. Vi kan också välja vilken kamera vi vill använda, samt om vi vill se den bild vi sänder iväg eller den vi tar emot.

Trots att vi själva inte såg videoverktyget som något nödvändigt var det det av våra verktyg som såg "proffsigast" ut i demon. Detta berodde på att det till videokortet följde med ett bibliotek med XCMD:s till HyperCard, och en exempelstack för dessa. Vi hade bara att kopiera över allt till vår stack med ResEdit, anropa en färdig funktion, så kunde vi visa video på skärmen!

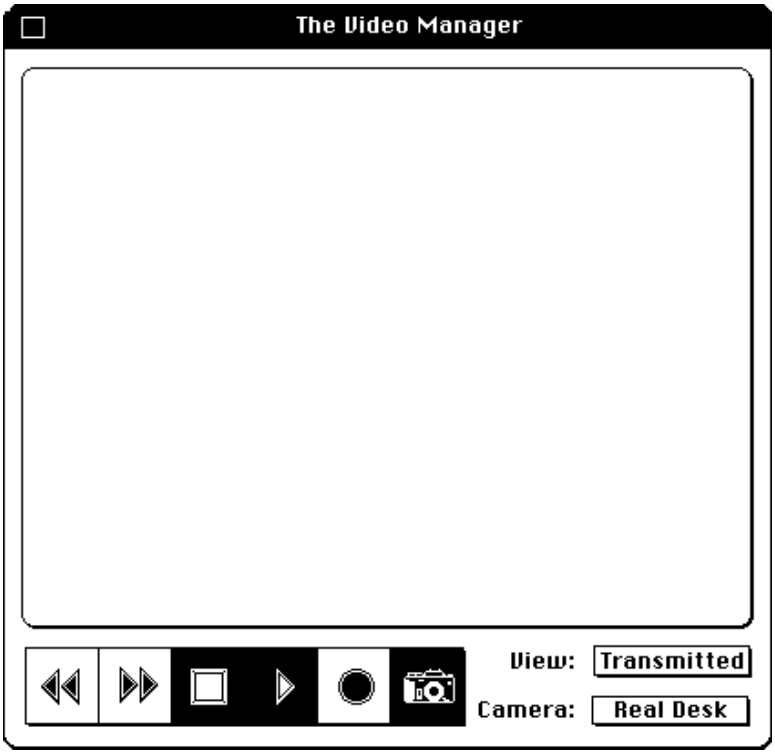


Fig.17: The Video Controller

4 Programmeringsverktyg

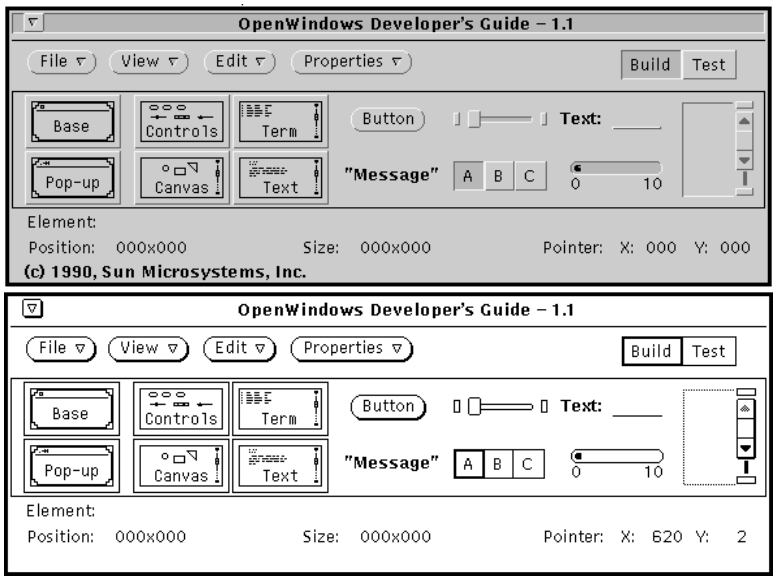


Fig.18: `Devguide` i två skepnader, i färg och i svartvitt.

4.1 Devguide — kort genomgång

`Devguide` är ett verktyg för att skapa användarinterface. Arbetssättet liknar delvis det man använder i verktyg av typ HyperCard, utan att binda programmeraren till en klumpig programmeringsmiljö med onaturliga scriptspråk. `Devguide` består av två huvudbeståndsdelar:

- En gränssnittseditor
- En gränssnittskompilator

Dessa verktyg tillsammans hjälper programutvecklaren att skapa moderna, kraftfulla och vackra användargränssnitt — utan att skriva en enda rad kod.

4.1.1 Gränssnittseditorn

Den del av `devguide` som hjälper programmeraren att skapa ett interface är *gränssnittseditorn*. Denna syns i två upplagor i figuren. Den översta av dessa visar hur ett fönster skapat i `devguide` ser ut då det körs på en färgskärm, den undre på en svartvit. Just det utseende som programmen får i färgläge, den starkt tredimensionella effekt som uppnås genom att använda olika gråskalor finner vi mycket tilltalande. Tyvärr gör dessa sig inte så bra i tryck, så vi har genomgående använt skärmdumpar från en svartvit skärm för att visa våra verktyg här i rapporten. Färgerna får ni föreställa er själva.

Fönstret domineras av en *palett*. Denna är "mittenarean" i figuren, ovanför finns några knappar och under ett statusfält som visar information om det objekt som för tillfället befinner sig under markören.

Paletten innehåller de objekt som finns att tillgå. Den vänstra delen innehåller de olika grundelement interfacet byggs upp av. Dessa är:

- **Base Window.** Ett vanligt fönster.
- **Pop-up Window.** Ett pop-up-fönster.
- **Control Area.** En kontrollarea läggs i ett fönster som "underlägg" på det ställe kontrollobjekt som knappar, listor och textfält ska ligga.
- **Canvas.** En canvas är en tom "duk", en yta att rita på med vanliga `Xlib` ritkommandon.
- **Term Panel.** Ett terminalfönster, färdigt att addera till applikationen!
- **Text Panel.** En texteditor. Om applikationen behöver använda en editor drar vi bara in den i fönstret, och vips har vi en fullt fungerande texteditor i vår tillämpning, komplett med alla funktioner, spara, sök-och-byt-ut, klipp-och-klistra — utan att själv behöva skriva en enda rad kod!

På den högra sidan av paletten finns vanliga kontrollobjekt:

- **Button.** Knappar.
- **Message.** Meddelandefält.
- **Slider.** "Skjutpotentiometrar".
- **Choice.** Motsvarar radioknappar och checkboxar i HyperCard.
- **Text.** Textinmatningsfält.
- **Gauges.** Indikatorer.
- **List.** Scrollande listor.

För att skapa ett interface *dras* objekten ut från paletten ned till tillämpningen, där de kan flyttas och skalas om interaktivt. Arbetet inleds lämpligen med att skapa själva fönstret. Klicka på `Base Window` i paletten, dra ut ett fönster och lägg det någonstans på skärmen. Dra sedan ned en `Control Area` och placera det i fönstret. På denna yta kan sedan knappar, listor och textfält placeras och flyttas runt efter behag. Varje typ av objekt i `devguide` har ett eget fönster där inställningar av objektens parametrar kan göras. För att få fram denna information dubbelklickas objektet, alternativt används menyn `Properties`. Här kan man ändra varje enskilt objekt, justera storlek, plats, namn, ikon, etc.

`Devguide` har två olika huvudmoder, *byggmod* för att skapa ett interface och *testmod* för att provköra dessa. I testmod uppför sig interfacet *nästan* som den färdiga tillämpningen kommer att göra.

För att senare kunna knyta egen kod till gränssnittet finns möjlighet att förse varje objekt med en *handler*, en funktion som kommer att anropas vid alla försök att påverka objektet. Det enda användaren behöver göra för att skapa en handler är att lägga till ett funktionsnamn på raden `Notify Handler` i objektets inställningsfönster. Vissa objekt kan även ha en `Event Handler`.

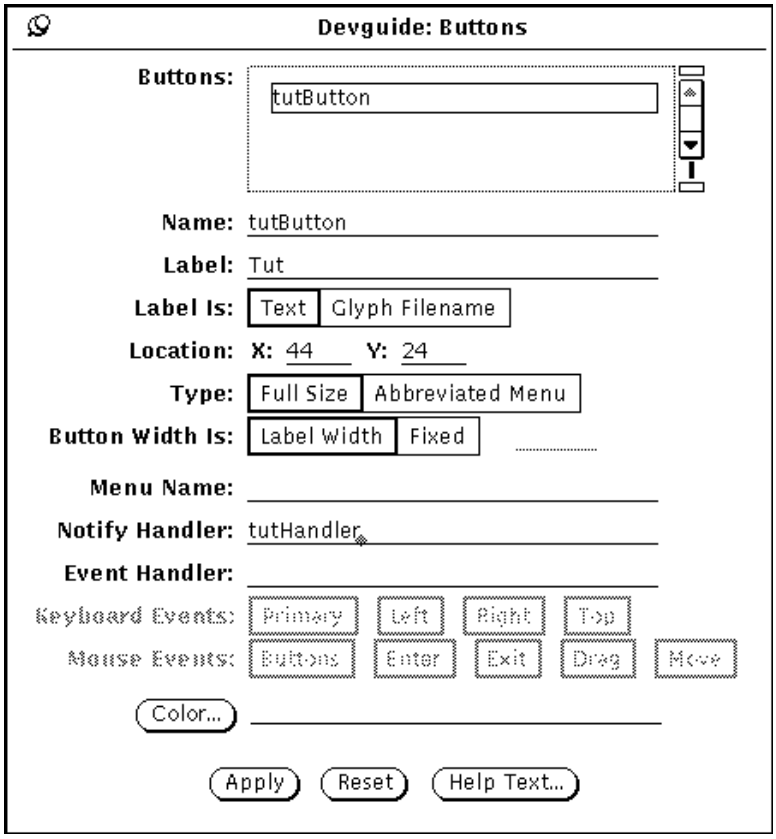


Fig.19: `Properties`, *inställningar* möjliga för en knapp

Skillnaden är (i exemplet *knapp*) att medan en **`Notify Handler`** anropas *en* gång vid event (`button_press`) anropas en **`Event Handler`** åtskilliga gånger för varje litet delevent (`mouse_down, mouse_up, mouse_enter, mouse_leave`, etc.).

Då interfacet börjar anta en behaglig form kan vi spara det på fil, och avsluta gränssnittseditorn, som nu gjort sin del av arbetet (i alla fall tills en ändring måste göras tio minuter senare).

Det gränssnittseditorn egentligen *producerar* är en grafisk beskrivningsfil (suffix `.G`) i ett språk kallat GIL (Graphical Interface Language). För varje objekt som skapats sparas en fullständig beskrivning:

```
(
   :type                  :button
   :name                  tutButton
   :owner                 controls1
   :help                  ""
   :x                     164
   :y                     8
   :constant-width        nil
```

```
    :button-type            :normal
    :width                  39
    :height                 19
    :foreground-color       ""
    :label                  "Tut"
    :label-type             :string
    :menu                   nil
    :notify-handler         tutHandler
    :event-handler          nil
    :events                 ()
    :user-data              ()
)
```

Exemplet visar beskrivningen för en knapp, kallad `Tut`. En komplett beskrivning av ett interface omfattar alltså en lista av dylika beskrivningar.

4.1.2 Gränssnittskompilatorn

Efter att ha skapat sitt interface i gränssnittseditorn är det dags att *kompilera* detta till C- och XViewkod. Kompilatorn kallas `gxv`, det finns även en version för C++, `gxv++`. Om den grafiska beskrivningsfilen heter `test.G` kompileras denna med kommandot

```
gxv test.G
```

`gxv` går då igenom filen objekt för objekt och skapar C- och XViewkod för att skapa motsvarande objekt. Exempelvis resulterar beskrivningen av knappen `Tut` ovan i C-koden

```
Xv_opaque
control_window1_tutButton_create(ip, owner)
        caddr_t         ip;
        Xv_opaque       owner;
{
        extern void     tutHandler();
        Xv_opaque       obj;

        obj = xv_create(owner, PANEL_BUTTON,
                XV_KEY_DATA, INSTANCE, ip,
                XV_X, 164,
                XV_Y, 8,
                XV_WIDTH, 39,
                XV_HEIGHT, 19,
                PANEL_LABEL_STRING, "Tut",
                PANEL_NOTIFY_PROC, tutHandler,
                NULL);
        return obj;
}
```

vilken ju i princip bara är en funktionsdeklaration med ett XViewanrop för att skapa objektet. Om koden ser obegriplig ut beror detta på att vi inte har gått

igenom XView än, det kommer i avsnitt 5.2.3. Totalt sett skapar `gxv` fem nya filer. För vår exempelfil `test.G` blir dessa:

- `test_ui.c` innehåller ovanstående funktion och motsvarande funktioner för att skapa alla i gränssnittet ingående objekt.
- `test_ui.h` är deklarationer av ovanstående.
- `test_stubs.c` innehåller "funktionsskal", tomma C-funktioner som bara skriver ut sitt eget namn, för alla `handler`-funktioner som angavs i gränssnittseditorn. Det är i dessa funktioner som den egna koden ska anropas. `test_stubs.c` innehåller även en färdig `main()`-funktion, som initierar alla i interfacet ingående komponenter.
- `test.info` innehåller texten till den on-linehjälp som kan knytas till varje objekt. Om ingen sådan har angetts är filen tom.
- `Makefile` innehåller information till programmet `make` hur de olika filerna ska kompileras, vilka beroenden som finns, vilka `#include`-filer och bibliotek som behövs för kompileringen, etc.

Just på grund av att även `make`filen automatgenereras räcker det att skriva

```
make
```

för att hela programmet ska kompileras och länkas. Det färdiga resultatet blir en exekverbar fil kallad `test`.

4.1.3 Hur man länkar in egen kod

När vi nu skapat vårt interface är det dags att knyta det till den kod som behövs för att verkligen *uträtta* något. Låt oss för enkelhetens skull anta att denna finns i en fil kallad `test.c`. För att kunna länka med den till den övriga koden måste vi ändra i `make`filen. Börja med att lägga till namnet på källkodsfilen. `Make`filens första rader ser ut som:

```
# This file was generated by `gxv' from `test.G'.

# Parameters

PROGRAM = test
SOURCES.c =
SOURCES.h =
SOURCES.G = test.G
STUBS.G = test.G

# SOURCES = \
          $(SOURCES.G) \
          $(SOURCES.h) \
          .
          .
          .
```

För att kompilera och länka in en källkodsfil är det enda en villrådig programmerare behöver göra att lägga till filens namn efter `SOURCES.c`, så att raden ser ut som:

```
SOURCES.c = test.c
```

Om flera filer ska länkas in skriver man deras namn efter varandra på raden.

Anropen till den egna koden läggs in i de tomma funktionsskalen i `test_stubs.c`. Ett typiskt sådant skal (för knappen `Tut` som vi behandlade ovan) ser ut som

```
/*
 * Notify callback function for 'tutButton'
 */
void
tutHandler(item, event)
      Panel_item item;
      Event *event;
{
  test_window1_objects *ip =
    (test_window1_objects *) xv_get(item, XV_KEY_DATA, INSTANCE);

  fputs("test: tutHandler\n", stderr);
}
```

direkt efter automatgenereringen. Varje gång knappen `Tut` klickas kommer denna funktion att anropas och strängen `test: tutHandler` skrivas ut. Om vi vill att något *annat* ska hända tar vi bort anropet till `fputs()` och lägger dit vår egen kod i stället, kanske ett anrop till någon av funktionerna i vår källkodsfil `test.c`.

En annan ändring som bör göras i `make`filen är att ändra anropet av `gxv` till att i stället anropa `gxv_merge`. Detta beror på att den vanliga `gxv` producerar en ny version av utfilen `test_stubs.c` varje gång den körs, och att alla ändringar som lagts till försvinner. `gxv_merge` löser detta genom att först spara undan den gamla versionen av `test_stubs.c`, skapa en ny, och sedan kombinera dessa på så vis att de nya funktionerna adderas till den gamla filen. På detta sätt behöver man som programmerare inte oroa sig för att alla ändringar ska försvinna från gång till gång.

4.1.4 Slutsatser

Om vi ska utvärdera `devguide` efter tre månaders hårt användande kan vi inte vara annat än starkt positiva. Sättet att arbeta passar oss perfekt, de resulterande programmen får fullt acceptabla prestanda och systemet verkar rimligt bugfritt. Det skulle förmodligen göra stor succé om det presenterades som verktyg till projekten i PIM-kursen.

Om vi jämför med programmeringsmiljön på en Macintosh kan `devguide` sägas kombinera de egenskaper som HyperCard och MacApp har. Det går dels snabbt att arbeta fram en prototyp, samtidigt som det gränssnitt som genereras blir en fristående applikation som enkelt kan kombineras med egen kod.

4.2 ISIS och distribution

ISIS är uppsättning verktyg för processkommunikation som utvecklats på Cornell University i USA. Dessa verktyg erbjuder både möjligheter till att övervaka andra processer, på den egna och på andra maskiner, samt möjlighet att överföra data mellan processer.

ISIS har utvecklats med betoning på driftsäkerhet och konsistens. Genom att använda ISIS behöver vi till exempel inte oroa oss för uppdateringsanomaliteter. Dessutom slipper vi fundera på hur information skall överföras mellan olika maskiner. ISIS används på samma sätt oberoende av om de kommunicerande processerna befinner sig på samma eller olika maskiner, oberoende av om de är varandras föräldrar eller exekverar fristående av varandra.

4.2.1 Processgrupper och broadcast's

ISIS erbjuder primitiver för att samla processer i grupper under ett gemensamt processnamn, som är en godtycklig alfanumerisk sträng. I dessa grupper kan sedan en process sända iväg ett datapaket som automatiskt distribueras vidare till samtliga i gruppen ingående processer. Om den sändande processen själv är medlem i processgruppen erhåller även den det utsända dataknippet. Denna distribution kallas *broadcast*. ISIS erbjuder ett flertal olika broadcastmöjligheter, *överföringsprotokoll*, som skiljer sig både vad avser snabbhet och funktion.

En process kan samtidigt vara medlem i flera processgrupper. Dessa grupptillhörigheter är dynamiska, en process kan under exekvering både bli medlem i en nya grupper och lämna andra. I **CaDe** finns det processer som är med i upp till ett tiotal grupper samtidigt.

4.2.2 Processövervakning

ISIS ger också möjlighet till övervakning av processers skiftande medlemskap i grupper. Denna möjlighet är ofta användbar. En process behöver till exempel inte tala om för övriga processerna i samma grupp (med en broadcast) att den skall lämna gruppen, utan lämnar bara gruppen, varvid ISIS omedelbart distribuerar information om detta till övriga processer. (Om de behöver denna information). Om en process självdör (på grund av något programmeringsfel) kan den naturligtvis inte tala om för andra processer att den lämnar ingående ISIS-grupper, detta sker automatiskt. Denna övervakningsfunktion ger oss stora möjligheter att ha korrekt och konsistent information hos olika processer.

4.2.3 Hantering av delade datastrukturer

Den sista stora tjänsten som vi flitigt använder är möjligheten till hantering av delade datastrukturer. ISIS saknar, så vitt vi vet, mekansimer för att ha gemensamma datastrukturer. Däremot finns fullt utvecklat stöd för att låta olika processer ha lika kopior av datastrukturer. När vi här talar om *delad datastruktur* menar vi alltså gemensamma kopior.

Om varje uppdatering av en delad datastruktur går via ISIS kan vi, med rätt överföringsprotokoll, garantera att alla processer får meddelanden om uppdateringar i exakt samma följd. När en process erhåller ett meddelande om en

uppdatering av data utför den omedelbart denna. När en process själv önskar ändra i den delade datastrukturen skickar den iväg ett meddelande om detta, utan att utföra ändrigen. Eftersom ISIS även skickar tillbaka meddelandet till den sändande processen (om den ingår i processgruppen) kommer vi slutligen att få tillbaka meddelande om dataförändringen. Nu kan vi utföra uppdateringen. Vi vet att alla processer i gruppen får alla meddelanden i samma ordning (med rätt överföringsprotokoll), alltså kommer samma ursprungsinformation hos varje process att genomgå samma förändring i samma ordning. Naturligtvis leder detta till att olika processer har exakt samma kopior.

En förutsättning för att det skall fungera enligt ovanstående resonemang är att samtliga processer har lika data till att börja med. Om en processgrupp skall understödja delade data, och processer både kan gå in i och lämna processgrupper dynamiskt, måste det finnas en mekanism för att överföra den *ögonblicksbild* av data som råder exakt då en process går in i en grupp. Detta kan *inte* utföras med broadcastmekanismen[9].

ISIS har dock särskilt stöd för detta. Om en processgrupp har delade data, anger den inträdande processen att den vill bli uppdaterad. Detta utförs automatiskt av ISIS samtidigt som processen registreras i gruppen, när processen är medlem har den även erhållit den ögonblicksbild som rådde vid inträdandet. ISIS utför detta genom att välja *någon* process i processgruppen och låta denna överföra sin data. ISIS garanterar att detta sker i rätt ordning i förhållande till eventuella meddelanden om uppdatering. Utan denna hjälp från ISIS hade vår kod varit betydligt svårare att framställa.

ISIS erbjuder ett flertal ytterligare möjligheter än de ovan beskrivna, som till exempel processsynkronisering mm. Dessa har vi dock varken studerat eller använt.

[9]Antag att process `A` går in i processgrupp `G`. Process `B` upptäcker detta och skickar iväg hela datastrukturen med en broadcast. Precis innan `A` går in i `G`, skickar process `C` ut begäran om uppdatering, denna uppdatering når därför aldrig `A`. `C`:s uppdatering når `B` och `C`, och de utför denna. Därefter kommer `B`:s meddelande av ögonblicksbilden av datastrukturen då `A` gick in i `G`, som alltså inte innehåller `C`:s utförda förändring. Alla processer tar mot detta meddelande, kastar sin gamla data och ersätter den med det nya. Vi har trollat bort `C`:s förändring.

Eller värre, endast `A` läser in nya data, varefter `A`, `B` och `C` har olika information.

5 CoDe — The Collaborative Desktop

5.1 Inledning

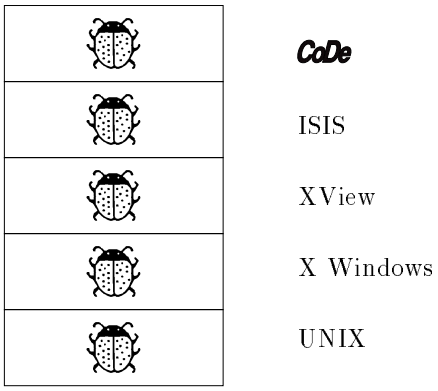


Fig.20: Bughierarkin i The Collaborative Desktop.

Den andra huvuddelen av vårt arbete var implementationen av ***CoDe***. Denna beskrivs här utförligt, först allmänt om förutsättningarna, sedan en utförlig presentation och genomgång av varje verktyg.

5.2 Plattform

Innan vi börjar vill vi ge en kort beskrivning av den utvecklingsmiljö vi använt.

5.2.1 Hårdvara

De maskiner vi använt för själva arbetet har varit två stycken SUN Sparc IPC arbetsstationer (`flora` och `majros`) med 19 tums färgskärmar, försedda med 8 respektive 12 MB primärminne. Dessutom har vi provkört vårt system både på en SparcStation2 med 32 MB minne (där det går undan värre) och på vanliga SparcStation SLC (där det går lite långsammare). På SLC-maskiner har tyvärr alla program skapade med `devguide` och XView en tendens att "hänga" sig, vilket torde bero på ofullkomligheter i XView. Klart är i alla fall att det inte är *vårt* fel.

5.2.2 X Windows

X Windows är ett nätverksbaserat fönstersystem för arbetsstationer som blivit industristandard under de senaste åren. X Windows är organiserat efter den så kallade *Client-Server*-principen. Detta innebär att all skärmhantering separeras och sköts av en serverprocess, medan de olika klientprocesserna måste låta sin skärmhantering gå via denna. Klientprocesserna kan köra på andra maskiner, och kommunicerar då med serverprocessen över nätverket via ett speciellt *protokoll*. Då protokollet är standardiserat kan klientproceserna köra på datorer av en helt annan typ än den där serverprocessen kör. X Windows kan därmed sägas separera de maskinspecifika delarna av koden från resten av applikationen.

5.2.3 XView

XView (för **X** Window System based **V**isual/**I**ntegrated **E**nvironment for **W**orkstations är ett objektorienterat verktyg för att skapa användarinterface. Det har utvecklats av SUN och är en vidareutveckling av ett tidigare system kallat SunView.

Efter att ha arbetat med `devguide` ett tag insåg vi att det inte är `devguide` som är intelligent, `devguide` är bara ett skal ovanpå XView som erbjuder ett interaktivt sätt att skapa och ändra XView-objekt.

XView innehåller en mängd fördefinierade *objekt*. Detta objekt kan vara av antal fördefinierade typer, exempelvis fönster, knappar, menyer, etc. Objekten har i sin tur en stor mängd *attribut*. För att påverka objekt och deras attribut används endast 6 generiska funktioner:

- `xv_init()` initierar XView.
- `xv_create()` skapar ett objekt.
- `xv_destroy()` förstör ett objekt.
- `xv_find()` hittar ett objekt som uppfyller vissa kriterier, och skapar det om det inte finns.
- `xv_set()` sätter värdet på ett attribut.
- `xv_get()` hämtar värdet på ett attribut.

Med hjälp av dessa funktioner kan alla i interfacet ingående komponenter skapas och påverkas efter behag.

För ett gränssnitt utvecklat i `devguide` är koden för att initiera XView och skapa alla objekt redan automatgenererad. Därför är det huvudsakligen funktionerna `xv_get()` och `xv_set()` som programmeraren behöver använda. `xv_set()` har syntaxen

```
xv_set(<objekt>, <attribute1>, <value1>,
                 <attribute2>, <value2>,
                 .
                 .
                 .
                 <attributeN>, <valueN>,
                 NULL);
```

där flera attribut kan sättas på en gång genom att parametrarna utgör en `NULL`-terminerad lista. `xv_get()` har på motsvarande sätt syntaxen

```
variable = xv_get(<object>, <attribute>);
```

Exempelvis ser koden för att "dimma" en knapp kallad `tutButton` ut som

```
xv_set(tutButton, PANEL_INACTIVE, TRUE, NULL);
```

där *objektet* `tutButton` får sitt *attribut* `PANEL_INACTIVE` satt till *värdet* `TRUE`. Det finns flera hundra olika attribut, och genom att sätta dessa till önskat värde kan gränssnittet skräddarsys i detalj.

5.2.4 Open Windows

Själva X Windows specificerar inte hur användargränssnittet ska se ut, detta sköts i stället av speciella *fönsterhanterare*. En sådan är SUN:s Open Windows, som implementerar en interfacestandard kallad Open Look. XViews objekt följer denna standard.

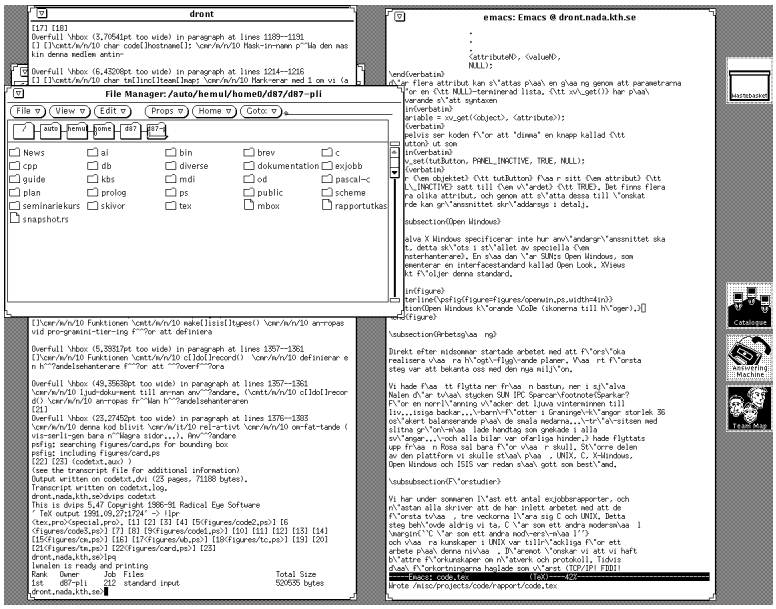


Fig.21: Open Windows körande CoDe (ikonerna till höger).

5.3 Implementationen av The Collaborative Desktop

Här följer den kompletta beskrivningen av implementationen av ***CoDe***. Vi har medvetet undvikit att ta med så mycket kod här, och koden är inte heller bifogad som bilaga. Anledningen till detta är att koden är stor, någonstans runt 150 A4-sidor. Då vi inte har behov av dessa sidor som utfyllnad (rapporten verkar bli tjock nog ändå) utelämnar vi dem för att spara papper. Den extremt intresserade läsaren får i stället skriva ut sin kod själv, allt ligger under `/misc/projects/code/CoDe/`. Där ligger även de körbara filerna, så den som vill prova ***CoDe*** kan starta systemet med kommandot

```
c peter
```

(eller `c magnus`) och prova sig fram.

Begreppet CoDe används på flera sätt i texten, och typografin får hjälpa oss att särskilja begreppen. ***CoDe*** (med den speciella logotypen) används som förkortning (macro) för *The Collaborative Desktop*, alltså hela vårt system. `CoDe` däremot (med "skrivmaskinsstil"), syftar på den speciella huvudprocessen med detta namn. Därmed hoppas vi undvika förvirring.

5.3.1 Inledning

När vi bestämt vilka av de samarbetsverktyg Marmolin föreslår i [Marm91] som skulle implementeras, återstod frågan om hur detta skulle organiseras. Vi såg två grundläggande modeller:

1. Allt byggs in i en "gigantisk" monolit. Hela ***CoDe*** implementeras i ett program.
2. ***CoDe*** utformas som en mängd fristående processer, där varje process ansvarar för en begränsad funktion. Dessa processer kommunicerar med varandra via ett skarpt definierat snitt.

Vi valde metod 2. Fördelarna i detta förslag var flera; genom modularisering av funktionerna blir processernas interna logik mindre komplicerad, olika moduler skulle relativt enkelt både kunna modifieras och bytas ut utan att programmet i stort påverkades, mm.

Tyvärr fanns även flera nackdelar; om olika processer behövde tillgång till samma data måste dessa kopieras mellan processerna, varje process behövde en egen ISIS-hanterare, och om processerna skulle interagera med användaren behövde de en egen händelse- och fönsterhanterare. Samma kod exekveras på flera ställen på samma maskin vilket minskar prestanda.

Gemensamt för alternativ 2:s nackdelar var dock att ju mer datorkraft som är tillgänglig, desto mindre störande upplevs de. Eftersom ***CoDe*** är en produkt för framtiden, och framtiden erbjuder betydligt snabbare datorer än idag bestämde vi oss för att ha överseende med nackdelarna.

Nedan beskrivs dels processorganisationen på makronivå, dels i huvuddrag enskilda processers interna organisation.

5.3.2 Processorganisation på makronivå

När hela det processknippe som utgör ***CoDe*** betraktas, är det främst följande faktorer som är intressanta; Vilka processer utgör ***CoDe***? Hur är de olika processerna organiserade i förhållande till varandra på en enskild maskin och i förhållande till processer på andra maskiner? Vilka ISIS processgrupper finns och vilken information överförs i dessa grupper? Vi avser att nedan reda ut dessa frågor.

CoDe består av ett antal verktyg som är relativt oberoende av varandra. Det var därför naturligt att låta varje verktyg utformas som en egen process. Förutom de olika verktygen fanns behov av en gemensam samordnare, en styrprocess som organiserar de övriga processerna. Vi lade även märke till att ljudhantering hade en central roll i ***CoDe***. Därför skapade vi en process vars enda uppgift var att erbjuda andra processer ljudrelaterade tjänster.

Nedan följer en kort sammanfattning av de processer som utgör ***CoDe***. Varje process kommer senare att avhandlas utförligare i egna avsnitt.

- `CoDe`

  Administratör och samordnare av övriga processer. Ansvarar för processkommunikationen både internt inom den egna maskiner och externt mot övriga maskiner.

- `Sound`

  Process som erbjuder ljudrelaterade funktioner, som överföring av samtal vid "telefonförbindelse", in- och uppspelning av ljud till fil, överföring av ljuddokument mellan gruppmedlemmar i ***CoDe***, mm.

- `Call Manager`

  Hanterare av synkron förbindelse,"telefonsamtal", mellan medlemmar i ***CoDe***. Kan hantera upp till fyra parallella samtal med maximalt fem medlemmar i varje. Personer i olika samtal kan enkelt kopplas ihop till gemensamt samtal. `Call Manager` är även ansvarig för uppstart av synkrona verktyg att användas som extra samarbetshjälpmedel av de personer som samtalar, som till exempel `WhiteBoard`, *Shared Editor*, *Video*, mm. `Call Manager` är osynlig när synkron förbindelse saknas.

- `Team Catalogue`

  Ger översikt av *samtliga* medlemmar i ***CoDe***. Visar aktuell status för dessa, som om de är påloggade, upptagna, lediga (och samarbetsvilliga). Även senaste påloggningstid och maskin visas.

- `Team Map`

  Visar utökad status över de "närmaste" medlemmarna, det vill säga de medlemmar där kontakt och samarbete ofta förekommer. Detta verktyg är i första hand avsett att ge användaren ett snabbt och översiktligt sätt att överblicka och ta kontakt med sina närmaste sammarbetspartners. Användaren avgör själv vilka personer som skall ingå i sin `Team Map`[10]. Denna rymmer upp till sex medlemmar.

- `Answering Machine`

  Tar mot och hanterar inkomna meddelanden på liknande sätt som en modern telefonsvarare. Dessa samlas i en gemensam lista och presenteras med meddelandetyp, avsändarnamn och tid. `Answering Machine` används även till att välja meddelande att lämna till uppringande personer då användaren är frånvarande eller upptagen. Med detta meddelande markerar användaren även sin egen status, det vill säga om han är tillgänglig eller upptagen.

- `Card`

  Presenterar all tillgänglig information om någon gruppmedlem, som till exempel rasterat fotografi, officiella anteckningar om personens funktion, kunskaper och historik, status, aktuellt telefonsvararmeddelande, etc. Skall innehålla del där egna, personliga anteckningar om personen kan föras, men detta har ännu inte implementerats.

[10] En tanke skulle annars vara att automatgenerera medlemmarna i `Team Map`. `CoDe` skulle kunna föra statistik över mellan vilka personer dokument och samtal ofta överförs och med detta som underlag avgöra vilka medlemmar som är "närmast". Vi anser dock att en normal människa själv föredrar att avgöra vilka som är hans/hennes närmaste vänner.

- `Displayer`

  Multimediatolkare. En förutsättning för ***CoDe*** var att de dokument som överförs mellan medlemmar skall vara fullständigt multimediella, det vill säga de skall ha möjlighet till kombination av ljud, bild, video mm. (lukt? smak? känsel?) Att utveckla och implementera format och avkodare för dylika dokument skulle i sig själv kräva åtminstonde ett par examansarbeten. Genom att utveckla en primitiv multimediaavkodare anser vi att vi smidigt gått runt detta problem. Displayermodulen är enkelt utbytbar och skall utvecklas allt eftersom framtiden förändrar förutsättningarna. Vi kan alltså abstrahera bort våra tillkortakommanden och hänvisa till att framtiden utökar våra torftliga dokumentformat. (Vår displayer understödjer enbart ljud- eller enbart text-dokument).

När ***CoDe*** startas är det `CoDe`-processen som aktiveras. Denna process startar de processer som tillsammans utgör ***CoDe***. `CoDe`-processen ansvarar för kommunikation mellan övriga processer. Om process `A` vill aktivera process `B` måsta `A` begära detta via `CoDe`. Vi har alltså efterstävat att bygga in så mycket komplexitet som möjligt i `CoDe` och på så sätt minska övriga processers ansvar. Processkommunikationen illusteras med vidstående figur.

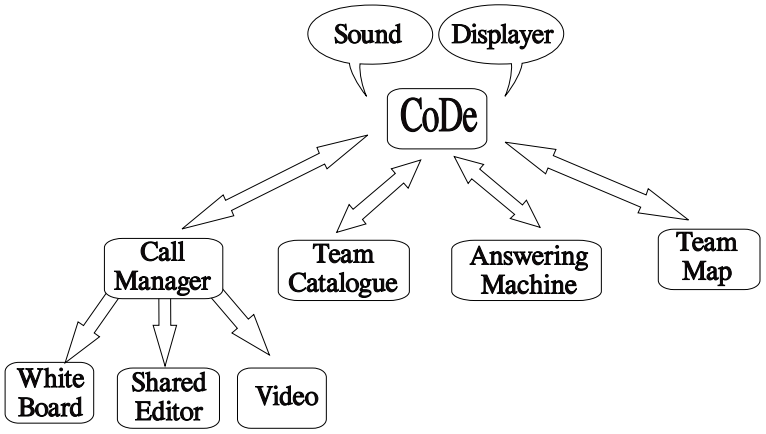


Fig.22: Den interna processkommunikationen

Förutom `CoDe` finns ett par processer som är rena stödprocesser för de samarbetsverktyg vi utvecklat. Dessa processer är `sound` och `Displayer`. Alla som behöver tillgång till de tjänster dessa processer erbjuder har möjlighet att begära dem direkt. Grundtanken är dock att ha ett klart snitt mot `CoDe` för att förenkla utveckling och integrering av nya verktyg.

När vi vill ringa upp en gruppmedlem behöver vi endast sända namnet på personen till `CoDe`, som själv kontrollerar om denna person är påloggad och "ledig" och i så fall på vilken dator. När vi vill hämta ett katalogkort för en medlem, skickar vi medlemmens namn till `CoDe`, om vi vill aktivera ett samarbetsverktyg skickar vi processens namn, osv...

Det bör alltså bli okomplicerat att integrera nya verktyg och att modifiera redan existerande.

Kommunikationen mellan processer *på olika maskiner* är relativt begränsad. Främst används detta då två eller flera personer samtalar med varanda, det vill säga när de har kopplats samman i en synkron förbindelse. Kommunikation mellan maskiner kan även gälla överföring av asynkrona dokument och meddelanden, samt naturligtvis gruppmedlemmars statusförändringar. Det är endast `CoDe`-processerna som kommunicerar. Processorganisationen illustreras i figuren.

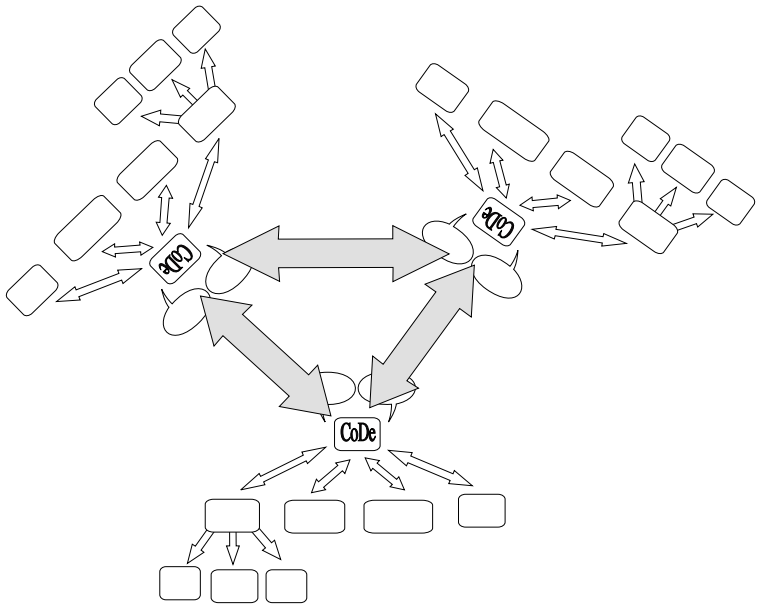


Fig.23: Den externa processkommunikationen.

Den `CoDe`-process som tillhör den första användare som loggar på ***CoDe*** har extra uppgifter. Den är bla ansvarig för att kontollera och förmedla när andra medlemmar loggar ur ***CoDe***[11] och för uppdatering av gemensamma datafiler.

Processerna kommunicerar med varandra både internt och externt med hjälp av ISIS. För att skapa gruppnamn som är unika inom den egna maskinen[12] använder vi unik information om maskinen i gruppnamnet, som till exempel maskinnamnet eller maskinens *site-nummer*, som är ett unikt heltal ISIS ger olika maskiner för att själv kunna skilja dessa åt.

Nedan beskrivs de ISIS-grupper som används av ***CoDe***, vilken data som överförs samt vilka processer som använder dessa grupper.

- `CoDe`

[11] En `CoDe`-process skall inte själv meddela om den lämnar "gemenskapen". Detta kan ju till exempel bero på ett falaktigt driftavbrott där processen automatsikt har dödats av UNIX själv. Då hinner den aldrig meddela sin urloggning och alla andra medlemmar skulle ha felaktig information om den dödade processens status.

[12] Vi vill ha namn som är unika inom den egna maskinen då vi vill sända intern infomation mellan de egna processerna. Om till exempel `CoDe` skickar ett meddelande till sin `Call Manager` vill vi garantera att det är den egna `Call Managern` som mottar meddelandet, och inte en annan maskins `Call Manager`.

Endast CoDe-processen är medlem i processgruppen CoDe. Det är via denna grupp som CoDe-processer på olika maskiner kan kommunicera med varandra. CoDe-gruppen används i första hand av CoDe-processerna till att övervaka varandra med hjälp av de av ISIS erhållna mekanismerna för processövervakning.

- CoDe/mxch

  Alla processer som behöver information om gruppmedlemmars status kan lyssna på CoDe/mxch-gruppen. Alla statusförändringar som förekommer skrivs ned till denna. CoDe/mxch (CoDe members exchange) existerar alltså utanför själva CoDe-processen. Via denna ISIS-grupp kan följdaktligen processer delvis kommunicera utan att gå via CoDe. Anledningen till detta avsteg från grundprincipen (all kommunikation går via CoDe) är rent implementationsteknisk. Det är betydligt enklare och säkrare att använda de inbyggda ISIS-primitiverna för överföring och uppdatering av information av medlemmars status än att låta CoDe hålla reda på samtliga förändringar och distribuera dessa vidare till sina barnprocesser. Dessutom är det på detta sätt enklare att integrera nya verktyg. Det enda de behöver göra för att erhålla information om andra medlemmar är att gå med i CoDe/mxch.

  Det är endast informationsblock av typen ACTIVE_MEMBER (se avsnitt 5.4.1) som överförs inom denna grupp. Här förekommer inga styrkommandon.

  Naturligtvis är CoDe själv medlem i denna grupp. Dessutom ingår Team Catalogue, Catalogue Card och Team Map. CoDe/mxch är den till antalet tveklöst största ISIS-gruppen i *CoDe*.

- <host_name>.Sound

  Endast processen Sound är medlem i <host_name>.Sound. Alla processer som vill använda ljudfuktionerna kan begära dessa via anrop till denna grupp. <host_name> skall ersättas med namnet på den egna maskinen. På flora.nada.kth.se heter denna grupp alltså flora.nada.kth.se.-Sound.

- <site_no>/CoDe

  Genom processgruppen <site_no>/CoDe kan processer begära tjänster från sin CoDe-process. Dessa tjänster är till exempel begäran om uppringning till annan gruppmedlem, begäran om aktivering av verktyg, m.m.

  Endast CoDe-processen är medlem. Övriga processer sänder enbart till gruppen.

- <site_no>/dp

  Gruppen <site_no>/dp används för att föra över multimediadokument till Displayer. Det är enbart Displayer som är medlem i gruppen. Alla processer som önskar visa ett multimediadokument (vanligtvis ett telefonsvararmeddelande, eget eller annans) kan skicka detta dokument till denna grupp. Displayer tar över ansvaret och presenterar dokumentet allt efter bästa förmåga.

- `<site_no>/CoDe/cm`

  Gruppen `<site_no>/CoDe/cm` används av `CoDe` och `Call Manager` för intern kommunikation. Detta är en tvåvägskommunikation varför både `CoDe` och `Call Manager` är medlemmar. Med denna grupp kan till exempel `CoDe` begära att `Call Manager` hanterar ett uppkopplat samtal, `Call Manager` kan begära att `CoDe` kopplar samman två samtal till ett gruppsamtal m.m.

- `<site_no>/cm.<number>`

  Detta är en grupp `CoDe` skapar för att temporärt hantera kommunikationen i en tillfällig synkron förbindelse mellan olika användare. `CoDe` och `Call Manager` är medlemmar, och överför bland annat information om vilket samtal som är aktivt, det vill säga vilket samtal (av upp till fyra stycken) som användaren för ögonblicket deltar i. Notera att `Call Manager` på olika maskiner på detta sätt direkt kan tala om för varanda vilket samtal som de har aktivt, utan att gå via `CoDe`. `CoDe` använder även denna grupp till att överföra information till `Call Manager` (på olika maskiner) om de användare som deltar som till exempel användarnamn, användarikon m.m.

  `<number>` är ett löpnummer som `CoDe` lägger till för att göra varje gruppnamn unikt.

- `<site_no>/cm.<number>.S`

  `Sound`-processen behöver en egen grupp för att överföra ljud mellan de användare som samtalar. `Call Manager` tar gruppnamnet för det aktuella samtalet och lägger till suffixet `.S` för att namnge denna grupp. Därefter överlämnas namnet till `Sound` som i sin tur skapar gruppen och omedelbart börjar överföra ljud mellan maskinerna. Detta är alltså, vad avser mängen överförd data, en *tungt* belastat grupp.

5.3.3 Processers interna organisation

I ***CoDe*** hittar vi två grundläggande typer av processer, processer som interagerar direkt med användaren och processer som enbart interagerar med andra processer. Här behandlas endast processer av den första typen[13]. Grundstrukturen för en användarinteraktiv process i ***CoDe*** beskrivs enklast av en figur.

Processen består något förenklat av tre lager, XView, ISIS och den egenutvecklade koden. Vår ursprungliga förutsats var att separera dessa lager från varandra så mycket som möjligt. Möjligheten att enkelt kunna byta ut gränssnittet mot användaren, XView, var tilltalande. Likaså skulle det vara trevligt att enkelt kunna modifiera processkommunikationen den dag ett bättre verktyg än ISIS framställs.

Den senare förutsatsen, att isolera ISIS, visade sig bli för komplicerat. ISIS inbyggda primitiver ger automatiskt koden ett särskilt utseende och struktur. Denna struktur, som enklast kan beskrivas som en kraftig intern modularisering av koden, kändes både naturlig och fördelaktig. Behovet av att betrakta

[13] Strukturen för en process som *inte* interagerar med en användare är inte särskilt intressant, utan är, organisationsmässigt betraktat, som vilken vanlig process som helst.

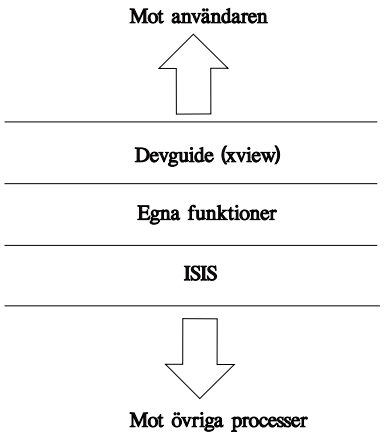


Fig.24: Interaktiva processers grundstruktur.

ISIS som en ersättningsbar modul blev därför inte särskilt starkt, vilket snart ledde till att ISIS-anrop blev utspridda tvärs genom koden. Idag skulle det knappast vara en lätt uppgift att ersätta ISIS med en annan verktygslåda för processkommunikation.

Med användargränsnittet gällde däremot andra förutsättningar. Redan i projektets inledande skede betonade vår uppdragsgivare, Hans Marmolin, vikten av att gränssnittet skulle vara enkelt utbytbart. Det finns alltid flera möjliga sätt att presentera ett program för en användare, varar de flesta kan göras bättre. Vår huvuduppgift var snarare att få *CoDe* att fungera än att söka det mest intuitiva och lättförståerliga gränssnittet. Därför har vi till så stor del det varit möjligt separerat de rutiner som driver gränssnittet från programmets övriga delar. Tyvärr går det inte att helt separera dessa uppgifter. Vi kan ta följande som ett exempel: när två gruppmedlemmar samtalar i "vår" *CoDe* presenteras och hanteras detta av en fristående process, `Call Manager`. I den prototyp som Hans Marmolin utvecklat parallellt med vårt arbete hanteras samtal istället som en del av `Team Map`. Det är inte trivialt att låta vårt program bete sig på detta sätt enbart genom en transformation av gränssnittet.

5.4 Datastrukturer

Här följer en kort genomgång av samtliga datastrukturer som används i *CoDe*:s olika processer. Detta har separerats från beskrivningarna av processerna då flera olika processer ofta använder samma datastrukturer.

5.4.1 Strukturerna `MEMBER/ACTIVE_MEMBER`

Datastrukturer för att hålla reda på gruppmedlemmar sönderfaller naturligt i två beståndsdelar, en del som handhar beständig information om medlemmen, som till exempel namn, och en del som handhar tillfällig status. Vi har därför skapat två datastrukturer, `MEMBER` och `ACTIVE_MEMBER`, där `ACTIVE_MEMBER`-strukturen innehåller en `MEMBER`-struktur. De ser ut som följer, först `MEMBER`:

```
typedef struct {
  int        code_id;
  char       code_username[USER_SLEN + 1];
  char       code_realname[REAL_SLEN + 1];
  char       code_hostname[HOST_SLEN + 1];
  MM_MSG     code_currmsg;
  DATE       code_logged_on;
  char       code_access;
} MEMBER;
```

där de enskilda fälten är:

- `int code_id;` Unikt heltal för denna gruppmedlem.
- `char code_username[];` Användarnamn.
- `char code_real_name[];` Används ej ännu.
- `char code_hostname[];` Maskinnamn på den maskin denna medlem antingen är inloggad på nu eller var inloggad på senast.
- `MM_MSG code_currmsg;` Aktuellt meddelande i denna persons `Answering Machine`. Se mer om denna datastruktur nedan.
- `DATE code_logged_on;` Datum för senaste påloggning till ***CoDe***.

`ACTIVE_MEMBER` ser på motsvarande sätt ut som:

```
typedef struct {
  MEMBER   tm_memb;
  char     tm_status;
  char     tm_inc_team_map;
  char     tm_watch;
  char     tm_phone_active;
  char     tm_do_apply;
  address  tm_address;
} ACTIVE_MEMBER;
```

där de enskilda fälten är:

- `MEMBER tm_memb;` Se ovan.
- `char tm_status;` Aktuell status: `ej påloggad`, `stör ej`, `upptagen` eller `ledig` (0-3).
- `char tm_inc_team_map;` Markerar med `TRUE` om vi (aktuell maskins användare) vill infoga denna medlem i vår `Team Map` (annars `FALSE`).
- `char tm_watch;` Markerar med `TRUE` om vi (aktuell maskins användare) vill bli uppmärksammade vid denna medlems statusförändring (annars `FALSE`).
- `char tm_phone_active;` Markerar med `TRUE` om personen har synkron förbindelse med annan medlem (telefonsamtal).

- `char tm_do_apply;` Kan ignoreras.
- `address tm_address;` ISIS-adress till denna gruppmedlems `CoDe`-process.

`MEMBER`-strukturen sparas på fil. `ACTIVE_MEMBER`-strukturen existerar däremot endast som lokala kopior i primärminnet på de olika maskinerna som är inkopplade på nätet. När första personen startar **CoDe** initieras samtliga användares `ACTIVE_MEMBER`-strukturer, det vill säga samtliga `MEMBER`-strukturer läses in från filen och alla statusvariabler "nollställs". När nya medlemar loggar på erhåller de automatiskt denna information och skickar därefter ut meddelande om uppdatering av sin egen datapost.

Två av parametrarna, `tm_inc_team_map` och `tm_watch`, är personliga inställningar för den egne användaren. Dessa två tal kan alltså skilja sig mellan olika arbetsplatser. Alla andra parametrar skall vara identiska.

5.4.2 Meddelandestrukturen

Det format som används för att skicka över meddelanden idag kallas `MM_MSG`, för *MultiMedia Message*. C-strukturen är uppbyggd enligt:

```
typedef struct {
  int        mm_type;
  char       mm_name[MSG_NAME_SLEN + 1];
  char       mm_sender[USER_SLEN + 1];
  char       mm_file[MSG_FILE_SLEN + 1];
  DATE       mm_date;
  char       mm_status;
} MM_MSG;
```

Fälten är i tur och ordning:

1. `mm_type` är meddelandets *typ*. Det kan i nuläget vara `SOUND, DOCUMENT` eller `MAIL`, där de två sistnämnda behandlas likvärdigt, enda skillnaden är den ikon som associeras till meddelandet. Typen avgör hur meddelandet ska presenteras. Ljudfiler spelas upp i högtalaren, medan dokument visas i ett speciellt popupfönster, `Text Displayer`.
2. `mm_name` är den meddelandesträng som knyts till varje meddelande.
3. `mm_sender` är namnet på den användare som skickat meddelandet.
4. `mm_file` är namnet på den fil som innehåller själva data.
5. `mm_date` är den tidsstämpel som varje meddelande förses med. Även denna är en separat C-struktur:

```
typedef struct {
  int       year;
  char      mon;
  char      day;
  char      hour;
  char      min;
} DATE;
```

Datumet visas i anslutning till det aktuella meddelandet i flera tillämpningar, bland annat katalogkorten och i Team Map.

6. `mm_status` är den status som kopplas till varje meddelande. Kan anta värdet `STS_LOGGED_OFF, STS_DONT_DISTURB, STS_BUSY` eller `STS_WELCOME`.

5.5 Filorganisation

Filorganisationen för ***CoDe*** bör redas ut. Detta är av särskilt intresse för de som avser leda en eventuell vidareutveckling av programsystemet. När vi tilldelades diskutrymme för projektet gav systemgruppen oss ett *projektkonto*, som placerades under `/misc/projects/code`. Vi valde att utveckla programkoden i underbiblioteket `CoDe`. Här återfinns all källkod och alla datafiler till ***CoDe***.

5.5.1 Processernas filorganisation

Både koden för `CoDe`-processen och den exekverbara applikationen ligger överst i biblioteket. Fördelat på varsitt underbibliotek finns källkod för övriga verktyg.

Det är viktigt att denna organisation inte förandras. `CoDe` utgår från att ***CoDe***:s samarbetsverktyg återfinns på dessa platser då de skall startas. Om filorganisationen förändras krävs att stora delar av `CoDe`-processen skrivs om.

Nedan en kort sammanfattning av var verktyg och stödprocesser återfinns. Denna utgår från `/misc/projects/code/CoDe`.

Bibliotek	Process	Applikation
`./CoDe`	`CoDe`	`c`
`./CoDe/CARD`	`Catalogue Card`	`card`
`./CoDe/CM`	`Call Manager`	`cm`
`./CoDe/DP`	`Displayer`	`dp`
`./CoDe/SOUND`	`Sound`	`sound`
`./CoDe/TC`	`Team Catalogue`	`tc`
`./CoDe/TM`	`Team Map`	`tm`
`./CoDe/WB`	`WhiteBoard`	`wb`

Fig.25: Filorganisationen

5.5.2 Datafilernas organisation

I ***CoDe*** finns data som måste sparas även om ingen `CoDe`-process är aktiv, som information om medlemmar och inkomna brev. Denna information kan inte lagras som delade datastrukturer i primärminnet utan måste sparas på fil.

Alla datafiler till ***CoDe*** lagras i underbiblioteket `./MEMBERS`[14]. Filen `members.dat` innehåller information om samtliga gruppmedlemmar. `members.dat` är en heap bestående av `MEMBER`-strukturer. Dessa ligger alltså sekvensiellt utan någon särskild organisation. Det finns inga indexfiler att ta hänsyn till.

[14] Alltså i `/misc/projects/code/CoDe/MEMBERS`.

Varje gruppmedlem har ett eget bibliotek i `./MEMBERS` på liknande sätt som UNIX har ett hembibliotek till varje användare. Biblioteket har samma namn som gruppmedlemmens användarnamn. Magnus bibliotek heter följdaktligen `./MEMBERS/magnus`.

I användarbiblioteket återfinns i fyra "viktiga" filer:

- `<username>.icon`

  Inneåller användarens ikonfil. Denna skapas enklast med XViews medföljande program `iconedit` som anropas från `Card`.

- `<username>.ras`

  Innehåller användarens rastrerade forografi. Om denna fil ligger i biblioteket kommer `Card` automatiskt att presentera bilden. Om filen saknas ignoreras detta, fotografiet i `Card` blir ofyllt.

  Vi har skapat dessa filer med det verktyg som medföljer VideoPixkortet, `vfctool`. I **CoDe** saknas stöd för att framställa bilderna; de måste tillverkas "utanför" systemet.

- `<username>.txt`

  Innhåller den textinformation som anges i `Card` till användaren. Detta är en ren textfil i ASCII-format.

- `<username>.tm`

  Innehåller information om vilka medlemmar som ingår i användarens `Team Map`.

I användarbiblioteket finns även biblioteket `mail` där användarens post lagras. Se mer om detta i avsnittet om `Answering Machine`.

5.5.3 Nödvändiga åtgärder för att flytta CoDe

Även om filorganisationen inte bör ändras kan det finnas goda skäl att vilja lägga *utgångspunkten*, `/misc/projects/code/CoDe`, på annan plats[15]. Vi har försökt att ta hänsyn till detta. Varje process som behöver åtkomst till applikationer eller datafiler har en global variabel, `CoDe_Dir`. Denna initieras (i senare versioner av **CoDe**) genom att läsa av en *environmentvariabel*. Innan **CoDe** startas måste variabeln `CODEHOME` sättas att peka på det bibliotek där **CoDe** ligger. Detta sker genom att ute i UNIX skriva

```
setenv CODEHOME /misc/projects/code/CoDe
```

eller var nu **CoDe** placerats. Om det inte redan är gjort bör även variabeln `OPENWINHOME` initieras. Detta för att **CoDe** ska kunna hitta de av Open Windows egna applikationer den använder (främst `iconedit`).

[15] Detta var ett allvarligt problem när **CoDe** flyttades till SICS.

5.6 CoDe

Den avgörande skillnaden mellan `CoDe` och övriga processer i ***CoDe*** är att `CoDe` har ett stort antal arbetsuppgifter medan övriga processer är starkt specialiserade att utföra en eller högst några få uppgifter. Det är därför lämpligt att inleda med en beskrivning av `CoDe`s arbetsuppgifter.

5.6.1 Arbetsuppgifter för CoDe

`CoDe` har en mängd uppgifter. Allt eftersom ***CoDe*** utvecklas kommer mer och mer uppgifter tillföras `CoDe`. I nuläget kan följande huvuduppgifter främst urskiljas:

- **Påloggning**

  De procedurer som måste utföras då en användare aktiverar ***CoDe*** är omfattande. Aktuellt användarnamn (namn på den som "kör", chauffören) anges på kommandoraden. Det första `CoDe` gör är att kontrollera att användaren är registerad ***CoDe***-användare. Om så inte är fallet avbryts exekveringen omedelbart. `CoDe` kontrollerar även att denne användare inte samtidigt använder ***CoDe*** på en annan maskin[16] och att inte en annan användare har ***CoDe*** aktivt på samma maskin[17].

  `CoDe` går in i ISIS-gruppen `CoDe/mxch` och erhåller information om andra medlemmars status. Information om den egna statusen distribueras vidare till övriga medlemmar. Denna information består främst av *användarnamn, maskinnamn, påloggningstid, tillgänglighetstatus* och *ISIS-adress*[18]. Därefter aktiveras de processer som utgör ***CoDe***. Det enda som återstår är att gå med i ISISgruppen `CoDe`.

  CoDe är uppstartat och aktivt.

- **Distribution av personliga inställningar för gruppmedlemmar**

  I den datastruktur som beskriver en gruppmedlems status finns vissa parametrar som är unika för den egna användaren, det vill säga att deras värden kan skilja mellan olika användare. När vi använder ISIS till att överföra en delad datastuktur automatiskt vid gruppinträde i `CoDe/mxch`, kan vi inte garantera att samtliga processer blir uppdaterade av "sin" `CoDe`-process, utan denna uppdatering kan lika gärna komma från annat håll. överföringen av personliga inställningar måste utföras på annat sätt. `CoDe` övervakar därför inträde i `CoDe/mxch`-grupppen; när en process som exekverar på den egna maskinen går in i gruppen översänder `CoDe` omedelbart sin information om gruppmedlemmarna med hjälp av ett antal broadcasts. Denna metod är inte alltid konsistent, det finns risk för förlorade uppdateringar (se avsnittet om ISIS). Dock går alla processer, som skall vara med i `CoDe/mxch`, in i gruppen omedelbart vid uppstart och stannar där. Denna riskfyllda överföring sker därför enbart en gång och

[16] Egentligen borde väl detta tillåtas. Det skulle dock ge *oss* som programmerare stora problem; hur skall vi till exempel göra när någon försöker ringa upp oss?

[17] Främsta anledningen till att ***CoDe*** måste vara *unik* på en och samma maskin är att *ljudresursen* är exklusiv; fler än en kan inte använda denna samtidigt.

[18] Varje process erhåller av ISIS en egen, unik adress.

nästan alla av **CoDe**:s processer startas omedelbart (`Card` undantaget). På denna tid, mellan aktivering av **CoDe** och överföring, hinner användaren trots allt knappast påverka sina egna inställningar.

- **Statusövervakning**

  Varje `CoDe`-process ansvarar för att meddela egen påloggning på nätet enligt beskrivningen ovan. Däremot behöver ingen process meddela egen urloggning. Denna statusförändring upptäcker `CoDe`-processerna på andra maskiner med hjälp av ISIS och distribuerar vidare. Det är viktigt att korrekt information om alla medlemmars status är tillgänglig när till exempel en användare skall uppringas.

- **Processaktivering**

  När en process önskar aktivera något av **CoDe**:s verktyg går denna begäran till `CoDe`. Alla verktyg är processbarn till `CoDe`, som därför har tillgång till deras processnummer. När `CoDe` vill aktivera ett verktyg skickar den UNIX-signalen `SIGCONT` till det aktuella verktyget. Alla **CoDe**:s verktyg har en särskild hanterare för denna signal, där de öppnas (om de är ikoniserade) och läggs överst på skärmen.

- **Filhantering**

  Eftersom flera interna datafiler som används av **CoDe** är delade mellan samtliga arbetsplatser behövs någon mekanism för att undvika krockar i filhanteringen. Den traditionella metoden med fillåsning och särkilda behörighetsflaggor vid filöppning valdes bort till förmån för att låta en och endast en process sköta all delad filhantering. Den äldsta `CoDe`- processen av alla aktiverade får även ansvaret för uppdatering av filer. Det är med ISIS enkelt att avgöra vilken process som är äldst. Om denna äldsta process dör kommer ISIS omedelbart att rappportera detta varvid en ny filhanterare automatiskt utses (den process som nu råkar vara äldst).

- **Samtalshantering**

  `CoDe` är ansvarig för etablering av synkron förbindelse mellan användarna. När `CoDe` får en begäran om uppringning kontrolleras först om personen som skall kontaktas är anträffbar, det vill säga om han är påloggad och ledig. Om så inte är fallet skickar `CoDe` denna persons aktuella telefonsvararmeddelande till `Displayer`.

  Om personen *är* anträffbar skapas en temporär ISIS-grupp. Namnet på denna grupp skickas dels vidare till den `CoDe`-process som exekverar hos den person som skall kontaktas, dels till sin egen `Call Manager`. Den `CoDe`-process som får meddelande om att samtal väntar med ett visst gruppnamn, förmedlar namnet vidare till *sin* `Call Manager`. När `Call Manager` erhåller namnet går den omedelbart med i gruppen.

  Då `CoDe` (den uppringande) upptäcker att en `Call Manager` (den egna eller den uppringandes) gått med i gruppen skickar den över information om de personer som samtalar (namn, användarikon mm). Därefter låter den `Call Manager` själv hantera samtalet.

Slutligen, när båda arbetsplatsernas `Call Manager` har lämnat gruppen (lagt på luren) plockar `CoDe` bort den temporära ISIS-gruppen.

Om `CoDe` erhåller en begäran om hopkoppling av samtal distribuerar den helt enkelt vidare gruppnamnet till de `CoDe`-processer som skall in i samtalet.

5.6.2 Tjänster som CoDe erbjuder (ISIS-nivå)

`CoDe` är framför allt en styrprocess för övriga processer, men tjänar också som en serviceinrättning där processer kan begära olika tjänster. Tjänsterna begärs med broadcasts till processgruppen `<site_no>/CoDe`. Följande tjänster kan vara av intresse för den som önskar integrera nya verktyg till ***CoDe***:

- `REQ_GET_CARD`

  Begär katalogkort till namngiven användare. Medsänd användarnamn samt en `int` som är `TRUE` om kortet skall skapas i det fall det inte existerar, annars `FALSE`. Om kortet redan har hämtats och finns aktivt, möjligen ikoniserat, hämtas det nästan omedelbart. Annars startas `Card`-processen vilket brukar ta en hel del tid.

- `REQ_APPLICATION`

  Begär aktivering av samarbetsvertyg. Medsänd namn på önskad process:

  ```
  Team Catalogue:          tc
  Team Map:                tm
  Answering Machine:       am
  ```

  Om verktyget är ikoniserat eller aktivt men dolt av andra fönster öppnas det och placeras främst på skärmen.

- `REQ_CALL`

  Begär telefonförbindelse till namngiven användare. Översänd enbart användarnamnet. `CoDe` kontrollerar om användaren är aktiv och på vilken maskin denne befinner sig. Om användaren ej är tillgänglig visas användarens telefonsvararmeddelande, annars uppkopplas gemensam förbindelse.

- `XCH_MM_MSG`

  Sätt eget aktuellt telefonsvararmeddelande. Översänd meddelandet (är av typen `MM_MSG`, se beskrivning av `Displayer`). Denna tjänst bör *inte* användas av andra processer än `Answering Machine` själv.

- `SIGNAL_AM`

  Aktiverar namngiven användares `Answering Machine`. Översänd användarnamnet. Denna tjänst är framför allt användbar då en process (eller snarare människan som styr processen) lämnat ett meddelande till någon gruppmedlem och vill göra mottagaren uppmärksam på detta.

5.6.3 CoDes uppbyggnad

Nedan följer en kortfattad genomgång av innehåll och funktion i de källkodsfiler som tillsammans utgör `CoDe`.

- `c.c`

  Innehåller främst de rutiner som "kör igång" ***CoDe***, det vill säga initierar ISIS, startar övriga processer till ***CoDe*** och distribuerar information om egna uppstarten. Här ligger även funktionen `CoDe_group_change()` som övervakar de `CoDe`-processer som exekverar på övriga maskiner. Denna funktion gör tre saker: den avgör om den egna `CoDe`-processen är äldst och därmed ansvarig för filhantering mm, den distribuerar information om andra `CoDe`-processers terminering, den kontrollerar att enbart en version av ***CoDe*** exekverar på samma maskin.

- `c_mb.c`

  Här ligger de rutiner som handhar hantering av gruppmedlemmar, vad gäller inläsning från och nedskrivning till fil. Datastrukturen för hantering av gruppmedlemmar byggs upp, den dynamiska hantering av detta sköts sedan om av funktionerna i `c_mxch.c`.

  Funktionen `mb_init_user()` anropas vid `CoDes` uppstart och hämtar all information om gruppmedlemmarna som finns lagrad på fil och initierar och bygger upp datastrukturer. Om fler `CoDe`-processer finns aktiva kommer dessa datastrukturer senare uppdateras med de eventuella dynamiska förändringar som inträffat. Funktionen `mb_register_change()` registrerar ovan nämnda dynamiska förändringar; funktionen väljer ut de förändringar som gäller den egna användaren och skriver, om nödvändigt, ned dessa till fil.

- `c_mxch.c`

  Implementerar hanteringen av gruppmedlemmars status, så som övervakning och överföring av data till barnprocesser. Med funktionen `mxch_init_watch()` går `CoDe` in i gruppen `CoDe/mxch` och inhämtar aktuell status för samtliga gruppmedlemmar. Funktionen `mxch_group_change()` övervakar förändringar av processers medlemskap i `CoDe/mxch` och ser till att uppdatera "sina" processbarn (med aktuell användares personliga inställningar för gruppmedlemmarna).

  Här ligger även rutinen `mxch_get_card()`, som anropas när någon process begärt ett katalogkort för en användare, samt `req_application()`, som anropas när någon process begärt en annan process (till exempel om `Team Catalogue` begärts från `Team Map`).

- `c_cm.c`

  Innehåller rutiner som implementerar `CoDe`-processens del i hanteringen av "telefonsamtal". Dessa kan grovt indelas i tre delar: uppkoppling, övervakning och nedkoppling. Funktionen `req_call()` erhåller intern begäran om uppringning, `req_connect_call()` anropas när annan `CoDe`-process begär uppkoppling, det vill säga när vi har blivit uppringda av

annan person. `req_call()` kontrollerar om den som skall uppringas är anträffbar, och kopplar i så fall upp samtalet. Funktionen `CoDe_inform_cm()` informerar `Call Manager` att samtal finns med visst gruppnamn

Funktionen `pc_monitor_phcall()` övervakar pågående samtal; om ny person tillträder distribueras information om denne till övriga deltagare, om samtliga deltagare har lämnat samtalet plockas samtalets ISIS-grupp bort.

Funktionen `pc_xfer_out()` används av ISIS för att delge nytillkomna deltagare i samtalet aktuell information, som vilka de övriga deltagarna är, deras samtalsstatus mm.

- `c_proc.c`

  Håller reda på verktygens olika processnummer och aktiver dessa processer vid behov.

- `c_conv.c`

  Funktionen `make_isis_types()` anropas vid programinitiering för att definiera nya ISIS-typer att användas vid *broadcasts*.

- `c_lib.c`

  Ursprungligen var tanken att samtliga "generella" rutiner skulle samlas här i form av ett egenutvecklat (litet) funktionsbibliotek. Dock visade det sig snart att behovet av egenutvecklade "generella" rutiner var tämligen begränsat, C-biblioteket erbjuder själv de mesta som kan önskas[19]. Dock finns några enklare rutiner för att hämta tiden och skriva in denna i *vårt* format (UNIX eget format är onödigt omständigt och fylligt), att konvertera vårt tidsformat till en textsträng samt slutligen en rutin för att jämföra otypat minne. (generisk jämförelse av godtyckliga typer).

5.7 Call Manager

5.7.1 Teori / funktion

När synkron förbindelse upprättas mellan gruppmedlemmar används `Call Manager`. Detta verktyg presenterar samtalsparternas användarikoner på en rityta som kallas `Call Area`. `Call Manager` kan hantera upp till fyra oberoende samtal med upp till fem deltagare i varje. Användaren växlar mellan olika samtal genom att "klicka" på önskat samtal. Olika samtalsparter kopplas samman genom att ikonerna dras till varandra. Detta direktmanipulationsgränssnitt har utvecklats utan direkt stöd från XView. Istället har X Windows grundläggande ritprimitiver används. Detta har lett till att koden för `Call Manager` är mer komplicerad (snårig) än för övriga processer i ***CoDe***.

`Call Manager` hanterar samtal i intimt samarbete med `CoDe`. En särskild ISIS-grupp används enbart för deras interna kommunikation. Även varje samtal

[19] Dessutom finns det tyvärr vissa rutiner utspritt i koden som är att betrakta som "generella" men som, mest av slöhet, inte placerats i `c_lib.c` utan istället i den fil man råkat vara då de behövts.

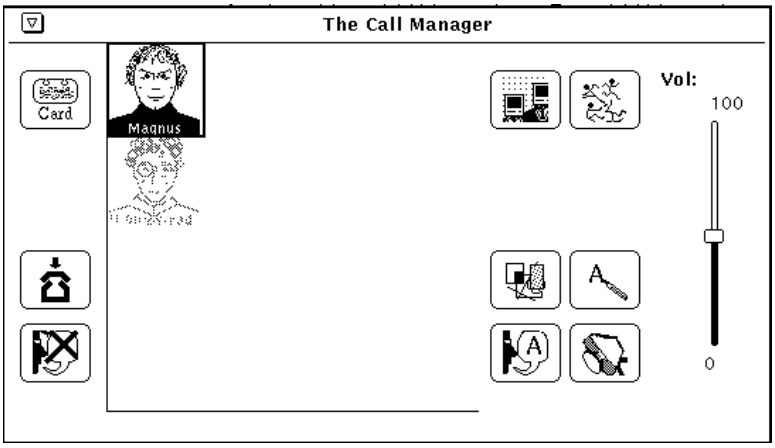


Fig.26: Call Manager

utgör en egen ISIS-grupp, som både skapas av `CoDe` vid uppkoppling och plockas bort av `CoDe` vid nedkoppling. Till stor del är det alltså egentligen `CoDe` som verkar som telefonväxel, inte `Call Manager`.

5.7.2 Funktionsbeskrivning

När `Call Manager` startas initeras alla datastrukturer och medlemskap ingås i aktuella ISIS-grupper. Därefter gör programmet inget, det fortsätter osynligt för användaren.

När `Call Manager` behövs för att hantera ett samtal skickar `CoDe` ett meddelande som bara innehåller gruppnamnet på den ISIS-grupp som skapats för att hantera samtalet. Nu aktiveras `Call Manager`, den visas på skärmen. Nästa meddelande från `CoDe` innehåller information om vilken person samtalet är uppkopplat mot. `Call Manager` hämtar nu användarikonen från användarens ikonfil och presenterar denna i `Call Arean`. När användaren väljer samtalet skickar `Call Manager` ett meddelande till `Sound`-processen att börja överföra ljud.

Om flera samtal inkommer innan det första är avklarat hanteras de på samma sätt som i beskrivningen ovan.

När användaren avslutat samtliga samtal plockar `Call Manager` automatiskt bort sig själv från skärmen och återgår till sitt ursprungliga dvalatillstånd.

Programmet byggs upp av följande källkodsfiler:

- `c_is.c`

  Går in i nödvändiga ISISgrupper samt etablerar kontakt med ljudprocessen.

- `c_cm.c`

  Detta är huvudfilen för `Call Manager`. Här hanteras all kommunikation till `CoDe` och `sound`.

- `c_proc.c`

Ett flertal synkrona verktyg skall kunna aktiveras utifrån `Call Manager` (se nedan). Funktionen `proc_fork_off()` i denna fil implementerar detta. Om det önskade verktyget redan är aktivt (det vill säga om funktionen redan tidigare anropats med aktuella parametrar) skickas UNIX-signalen `SIGCONT` till processen. Verktyget bör då omedelbart aktivera sig!

- `c_xv.c`

  Innehåller de funktioner som driver gränssnittet. Eftersom XView's möjligheter inte räckt för att implementera det direktmanipulativa gränssnittet vi önskade, och vi därför tvingats implementera detta direkt i X Windows, är denna kod ganska stökig och komplex. Funktionen `pt_paint_handler()` (som utvecklades för `WhiteBoarden`) administerar händelsekön och ser till att rätt funktioner anropas som svar på användarens initiativ. Funktionen `pc_display_phcall()` anropas varje gång `Call Arean` måste ritas om. Dessutom finns här funktioner för att läsa in ikonfiler mm.

5.7.3 Synkrona samarbetsprocesser

`Call Manager` är det enda verktyg som upprättar en direkt synkron förbindelse mellan olika användare. Därför är denna process en naturlig utgångspunkt att aktivera andra synkrona samarbetsprocesser från.

Det är lätt att tänka ut ett flertal intressanta delade applikationer, allt från program som ger stöd för slutna omröstningar till direkta spelprogram. Vår avsikt var att ge några grundläggande exempel av dylika tillämpningar.Vi fastnade för `WhiteBoard` — en delad riteditor, `Shared Editor` — en coeditor och `Video` — möjlighet till videoöverföring. Tyvärr fick vi bara tid att implementera en av dessa, ett rudiment till en delad riteditor, `WhiteBoard` har framställts. Denna har dock för många brister i funktionalitet för att vara praktiskt användbar.

5.8 WhiteBoard

`WhiteBoard` är det verktyg där vi lagt ned mest arbete, samtidigt som det är det verktyg som är minst fullständigt. Det är tyvärr inte mer än ett halvfabrikat.

5.8.1 Teori / funktion

Verktyget har en intressant historia. Ursprungligen utvecklades `WhiteBoard` inom `Call Manager`. Båda verktygen låg i samma process. Då `Call Manager` kan hantera fyra oberoende samtal parallellt, måste detta även gälla `WhiteBoard`. När samtal byttes skulle `WhiteBoard` automatiskt följa med och visa det nya samtalets `WhiteBoard`. Efter att ha arbetat med dessa förutsättningar i åtminstonde ett par veckor insåg vi två saker.

1. Källkoden blev *mycket* stökig, komplex och fylldes alltmer av *fixar*.

2. Det skulle (ofta) kunna uppstå behov av att samtidigt studera två olika samtals `WhiteBoard`.

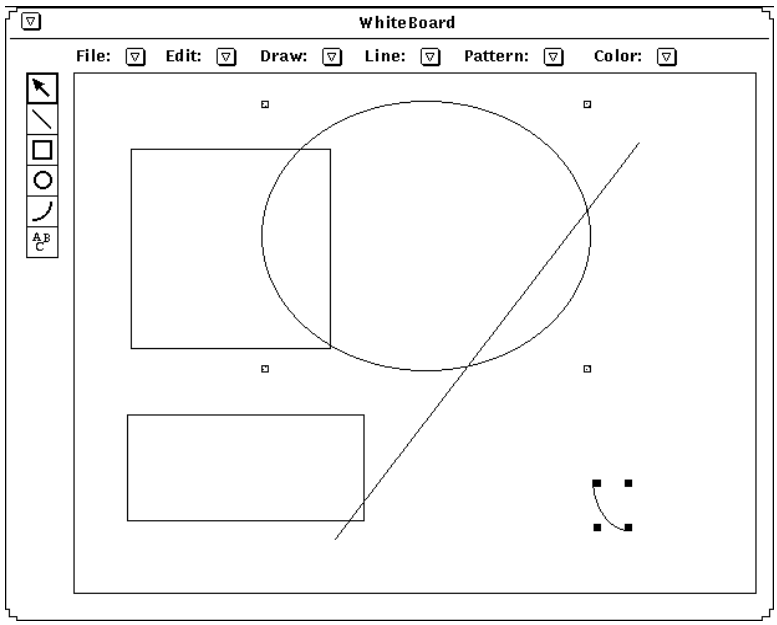


Fig.27: WhiteBoard

Vi var tvungna av att utforma `WhiteBoard` som en egen process. Det skulle ge betydliga förenklingar avseende programlogik och skulle dessutom tillåta flera samtals `WhiteBoard` att vara aktiva samtidigt. Det var bara ett problem. All framtagen kod var utformat med förutsättningen att flera oberoende `WhiteBoard` skulle kunna hanteras och dessutom styrdes detta till stor del från samtalhanteringen i `Call Manager`. Det var ett omfattande arbete att omforma koden och olyckligtvis blev resultatet inte riktigt rent och vackert. Tiden räckte inte för att göra en användbar produkt.

Enda anledningen till att `WhiteBoard` fortfarande finns kvar är att den är trevlig att visa vid demonstrationer. Detta är det enda synkrona samarbetsverktyg vi skapat och duger åtminstonde till att illustrera hur en sådan applikation passar in i ***CoDe***.

`WhiteBoard` ger flera användare möjlighet att tillsammans utforma en skiss bestående av enkla geometriska former som streck, rektanglar, cirklar och cirkelbågar. Objekten kan fyllas med olika mönster och använda olika linjetjocklekar och linjetyper. Det fanns även planer på att kunna visa objekt med olika färger, men det implementerades aldrig.

Ett objekt som lagts till på ritytan kan omskalas och flyttas. Ett flertal objekt kan markeras i en grupp och flyttas tillsammans. Objekt kan även kopieras och plockas bort. När en användare har markerat ett eller flera objekt tillhör de honom. Ingen annan har möjlighet att påverka dem förrän de har avmarkerats.

Ett flertal tänkta funktioner implementerades aldrig. Vi avsåg till exempel att kunna spara ett dokument samt att kunna exportera denna i andra format, som `PostScript`, `MacPaint` mm. Dessutom ville vi ha möjlighet infoga text i dokumentet, gärna med olika storlekar och typsnitt. Saker blir inte alltid som

man vill.

5.8.2 Funktionsbeskrivning

`WhiteBoard` aktiveras av `Call Manager`. `Call Manager` medsänder som argument en textsträng som `WhiteBoard` använder som gruppnamn i den ISIS-grupp där ritkommandona överförs. I övrigt saknas all koppling mellan `WhiteBoard` och ***CoDe***.

När `WhiteBoard` startas går processen omedelbart in i ovan nämnda ISIS-grupp. Om annan `WhiteBoard` har aktiverats till samma grupp översänder den de objekt som redan finns på ritytan. Till objekten finns information om position, linjetyp och -mönster samt objektets ägare, om objektet har markerats av någon. Denna information presenteras på ritytan.

Därefter lyssnar `WhiteBoard` på sin ISIS-grupp och utför samtliga mottagna kommandon samtidigt som användarens förändringar rapporteras till gruppen.

Nedan ges en *kort* beskrivning av de ingående källkodsfilerna till `WhiteBoard`. Denna hålls framför allt kort eftersom vi utgår från att ingen kommer att vidareutveckla detta program. Det skulle förmodligen löna sig bättre att bygga om programmet från början.

- `c_is.c`

  Initierar ISIS-hanteringen.

- `c_pt.c`

  Utformar ett skal mellan den interna representationen av de grafiska objektet till ritprimitiverna i X Windows. Här finns troligen ett flertal "godbitar" för den som själv vill ha möjlighet att rita på en *canvas* i XView. Det ligger en del grundforskning bakom dessa rutiner. Vi har gått genom ett flertal demoprogram i XView för att kunna utforma dessa. Det är främst funktionen `pt_init_paint_struct()` som bör studeras. Här initieras nödvändiga parametrar för att kunna rita på en *canvas*.

- `c_wb.c`

  Innehåller de funktioner som tolkar externa användarmanipulativa händelser, som musförflyttningar och knapptryckningar, och ansvarar för att dessa resulterar i, för användaren, önskade resultat.

- `c_dw.c`

  Översänder samtliga ritkommandon till övriga processer. Mottager även dylika meddelanden som avkodas och överlämnas till rutinerna i `c_go.c`.

- `c_go.c`

  Manipulerar de datastrukturer som hanterar beskrivningen av ritytans grafiska objekt. Detta innefattar allt från att lägga till, markera, storleksförändra, flytta och plocka bort objekt. Rutinerna ansvarar även för att bilden som visas på ritytan överensstämmer med underliggande datastrukturer.

- `c_xv.c`

  "Hack" för att anpassa ISIS och XView till varandra.

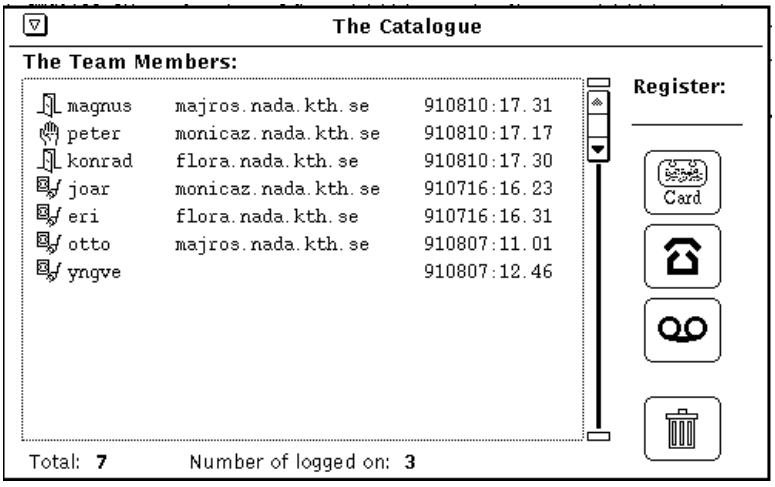


Fig.28: Team Catalogue

5.9 Team Catalogue

5.9.1 Teori / funktion

`Team Catalogue` håller reda på och presenterar samtliga medlemmar i ***CoDe***, deras status, på- eller utloggningstid och maskinnamn. Här erbjuds möjlighet till att ringa upp gruppmedlemmar, sända dem asynkrona meddelanden samt att hämta deras `Catalogue Card`, vilken ger en utökad statusbeskrivning.

I `Team Catalogue` kan även nya medlemmar registreras. Detta utförs genom att ange ett användarnamn vid `Register` och trycka `<Enter>`. `Team Catalogue` begär tjänsten `REQ_GET_CARD` av `CoDe` för användare med angivet namn, med tillägget att kortet skall skapas om det inte existerar. Om användaren redan existerar hämtas därför dennes kort; vi kan alltså inte skapa två medlemmar med samma namn (vilket väl är bra?).

Den `Team Catalogue` vi har skapat är tämligen primitiv. Vad som främst saknas är sökmöjligheter. Här skulle det vara önskvärt med möjlighet till att söka efter personers specialkunskaper, pågående projekt mm. Vår stora tidsbrist har tyvärr omöjligjort många önskade utökningar.

5.9.2 Funktionsbeskrivning

`Team Catalogue` börjar med att gå in i gruppen `CoDe/mxch` och inhämta information om samtliga personer och presenterar detta för användaren. Därefter läser den och visar alla förändringar av medlemmars status.

Programmet tillåter användaren att markera godtycklig person, påloggad eller ej, och utföra någon av funktionerna *uppringning*, *meddelandeöversändning*

eller *hämta katalogkort*. Detta realiseras enkelt genom att skicka vidare aktuell begäran till `CoDe`-processen.

`Team Catalogue` består av följande filer:

- `c_mxch.c`

  `c_mxch.c` innehåller huvuddelen av `Team Catalogues` funktionalitet. Här ligger koden för att gå in i `CoDe/mxch`, få medlemmars status översänd samt att bevaka förändringar.

  Funktionen `mxch_init()` startar ISIS, går in i `CoDe/mxch` och får därgenom information om gruppmedlemmarna. Huvudfunktionen i filen är `mb_register_change()` som tar in en statusförändrad gruppmedlem och uppdaterar både gränssnittet och interna datastrukturer. I denna fil finns även de rutiner som begär tjänster av `CoDe`, som `mxch_req_get_card()`, `mxch_req_call()` och `mxch_record_event()`. Funktionsnamnen beskriver själva (nåja) vad de utför.

- `c_cnv.c`

  Funktionen `make_isis_types()` anropas vid programinitiering för att definiera nya ISIS-typer att användas vid *broadcasts*.

- `c_rec.c`

  Funktionen `c_do_record()` definierar en händelsehanterare för att överföra ljuddokument till annan användare. (`c_do_record()` anropas från händelsehanteraren till den XView-knapp som skall handha funktionen `record`.)

- `c_xv.c`

  "Hack" för att anpassa ISIS och XView till varandra.

- `c_xview.c`

  Innehåller i stort samtliga funktioner som direkt manipulerar gränssnittet. I detta programs fall innebär det två funktioner; att *lägga till* statusrader i listan av medlemmar, `xview_addmember()`, samt att *förändra* dessa, `xview_changemember()`.

  Datastrukturer och inställningar för gränssnittet initieras vid programstart genom anrop till `tc_init()`.

5.10 Team Map

5.10.1 Teori / funktion

I `Team Map` kan de medlemmar som användaren ofta vill kunna få en snabb översikt av samlas. Dessa medlemmar är rimligtvis de personer som användaren ofta arbetar med, som till exempel viktiga kontaktpersoner och deltagare i gemensamma projekt. I `Team Map` visas betydligt mer information om medlemmarna än i `Team Catalogue`. Här visas medlemmarnas ikonbilder, tillgänglighetsstatus, senaste inloggningstid, namn på aktuellt telefonsvararmeddelande

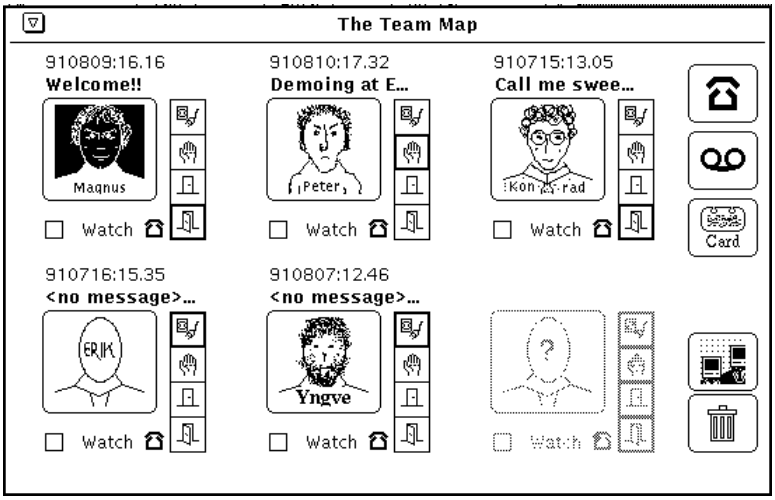


Fig.29: Team Map

samt en indikation på om medlemmen för ögonblicket har ett pågående samtal. Det finns dock ingen möjlighet att avgöra med vem medlemmen samtalar, detta skulle vara ett för stort inbrott i den personliga integriteten.

Här finns även en särskild statusflagga, `watch` som användaren kan aktivera för varje medlem. När `watch` är satt på en gruppmedlem kommer denne medlems status att bevakas av ***CoDe***. Då förändring inträffar, som när medlemmen loggar på ***CoDe***, byter tillgänglighetsstatus mm, kommer medlemmens katalogkort att hämtas. Om användaren vill få kontakt med en gruppmedlem så fort möjlighet ges kan han alltså aktivera `watch` och automatiskt bli underrättad då tillfälle ges.

För att infoga en medlem i `Team Map` skall den önskade medlemmens katalogkort hämtas. I kortet finns flaggan `Include in Team Map`. När denna aktiveras läggs omedelbart medlemmen in i `Team Map`.

`Team Map` rymmer upp till sex medlemmar. Detta är en låg gräns, men med tanke på den mängd information som presenteras är det ändå en rimlig begränsning. En applikation som täcker alltför stor del av skärmen är svårare att använda tillsammans med andra applikationer.

5.10.2 Funktionsbeskrivning

`Team Map` är funktionsmässigt en enkel applikation. Den inleder med att initiera datastrukturer för att driva gränssnittet. I `Team Map` används en stor mängd av XViews möjligheter. Därför har ett "skal" utvecklats ovanpå XView för att strukturera gränssnittshanteringen.

`Team Map` går därefter med i `CoDe/mxch` och erhåller information om samtliga gruppmedlemmar. Endast de medlemmar som har flaggan `Include in Team Map` aktiva sparas och presenteras. Övriga medlemmar ignoreras.

När denna initieringsfas är avklarad väntar `Team Map` på medlemmars statusförändringar. Omedelbart då status förändras för en medlem som ingår i

`Team Map` presenteras detta. Om en ingående medlem plötsligt inkommer med `Include in Team Map` inaktiv plockas denna bort från `Team Map`. Om icke ingående medlem inkommer med flaggan aktiv infogas denna istället.

Förutom bevakning av statusförändringar erbjuder `Team Map` användaren vissa tjänster, som till exempel uppringning och meddelandeöversändning.

- `c_mxch.c`

  I denna fil ligger de rutiner som handhar kommunikationen via `CoDe/mxch`-gruppen. All uppdatering av information om gruppmedlemmar sköts här. Detta är huvudfilen för `Team Map`.

  `mxch_init()` anropas vid programstart. Denna rutin initierar samtliga ISIS-rutiner och får `Team Map` att gå in som medlem i `CoDe/mxch` där information om gruppmedlemmarna överförs. Huvudfunktionen i filen är `mb_register_change()`. Den tar mot information om medlems statusförändring och avgör om medlemen skall läggas till `Team Map` eller plockas bort, eller om den enbart behöver uppdateras.

- `c_cnv.c`

  Funktionen `make_isis_types()` anropas vid programinitiering för att definiera nya ISIS-typer att användas vid *broadcasts*.

- `c_rec.c`

  Funktionen `c_do_record()` definierar en händelsehanterare för att överföra ljuddokument till annan användare. (`c_do_record()` anropas från händelsehanteraren till den XView-knapp som skall handha funktionen `record`).

- `c_xv.c`

  "Hack" för att anpassa ISIS och XView till varandra.

- `c_xview.c`

  Innehåller i stort samtliga funktioner som driver användargränssnittet. `Team Maps` gränssnitt är något mer komplicerat än övriga verktygs, varför denna kod blivit *relativt* omfattande (visserligen bara några sidor...). Användare kan både läggas till och plockas bort från `Team Map` vilket ger omfattande förändringar av gränssnittet.

  Funktionen `xview_init()` initierar några datastukturer för att enklare kunna stryra gränssnittet. `xview_insert_member()` lägger till en ny medlem till `Team Map` och `xview_delete_member()` plockar bort medlem. Dessa operationen kan medföra stora förändringar av gränssnittet.

  Funktionen `xview_show_member()` anropas när en medlems status har förändrats. Denna rutin uppdaterar gränssnittet med den nya informationen.

  Eftersom även användarikoner visas i `Team Map`:en finns här kod för att läsa in ikonfiler.

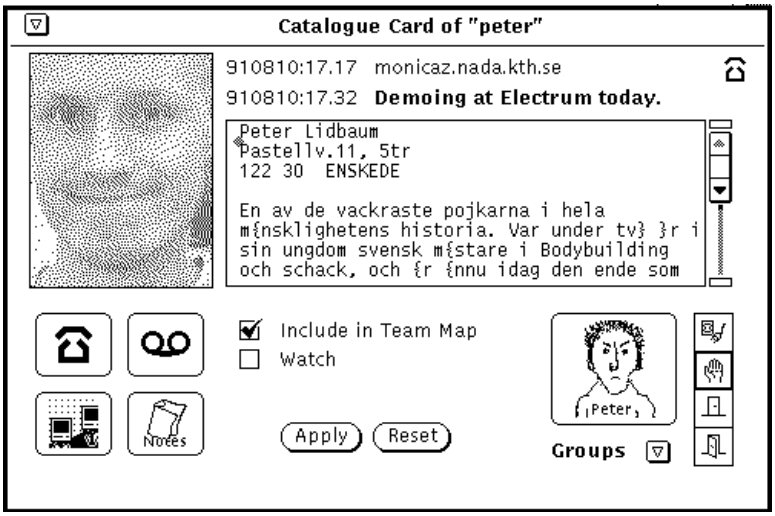


Fig.30: Catalogue Card

5.11 Catalogue Card

5.11.1 Teori / funktion

I stort sett all information som finns tillgänglig om en gruppmedlem visas i `Card`. Här finns ett rastrerat fotografi av personen och en textuell presentation av funktion, uppgifter, aktuella projekt, speciella kunskaper och färdigheter och annat av intresse. Den textuella presentationen utformas valfritt och har implementerats genom att använda en inbyggd möjlighet i XView; vi har helt enkelt infogat en texteditor. Personens användarikon definieras i `Card`. Användarikonen visas i en XViewknapp. När denna nedtrycks startar `Card` automatiskt programmet `iconedit` med filnamnet på ikonen som argument. Ikonen kan editeras och sparas utan att användaren behöver ange filnamn. När editeringen är avklarad skall knappen `Apply` i `Card` nedtryckas. `Card` läser åter in ikonen från fil och meddelar samtidigt alla andra processer i `CoDe/mxch` att göra det samma. Plötsligt har den nya ikonen slagit genom överallt i ***CoDe***.

I `Card` visas även tid för medlemmens senaste påloggning (tid och maskin) och meddelandebyte i telefonsvararen (tid[20] och meddelandets namn). Även tillgänglighets- och telefonstatus (samtalar personen med någon?) presenteras.

I `Card` finns de vanliga möjligheterna att ringa upp personen eller att lämna ett meddelande. Här finns även knappen `Notes` som är avsedd att hämta ett, för den aktuelle avändaren, personligt anteckningblock där denne skall kunna föra anteckningar om kortets person, synbara endast av användaren själv. Detta är tyvärr inte implementerat.

[20]Om meddelandet till exempel säger `"Jag är tillbaka inom fem minuter"` kan det vara mycket intressant att kunna avgöra *när* detta meddelande lämnades.

5.11.2 Funktionsbeskrivning

- `c_init.c`

  Rutinerna i denna fil används i princip endast vid uppstart och initiering av programmet. Här ligger ett stort knippe korta funktioner vars enda uppgift är att sätta parametrar som anger *var* och under vilka namn olika filer ligger, som till exempel `init_iconfilename()`, `init_canvasfilename()` och `init_textfilename()`.

  De "viktiga" funktionerna är dock `card_init()` och `init_create_user()`. `card_init()` anropar rutiner för att initiera "allt": ISIS, gränssnitt, interna datastrukturer mm. Funktionens främsta uppgift är dock att läsa in programmets argument, att avgöra vilken användare kortet skall presentera och se till att denna information inhämtas från nödvändiga filer. Om användaren inte är registrerad som medlem i **CoDe** och programmet har anropas för i så fall att skapa användaren, tar rutinen hjälp av `init_create_user()` för att initiera och registrera en ny användare.

- `c_mxch.c`

  Här ligger huvudrutinerna vad avser kommunikation med andra processer. Funktionen `make_isis_types()` anropas vid programinitiering för att definiera nya ISIS-typer att användas vid *broadcasts*. `mxch_init()` startar ISIS och låter `Card` gå med i gruppen `CoDe/mxch` där information om gruppmedlemmar erhålls.

  Under drift är främst funktionen `mxch_team_memb_xch()` intressant. Funktionen tar mot meddelande om statusförändring hos medlemmar. Eftersom den erhåller information om *samtliga* medlemmar inleder den med att kontrollera om meddelandet gäller den aktuella personen. `Card` samlar enbart information om *en* gruppmedlem. Om meddelandet gäller den egna personen tas det om hand och behandlas. Detta innebär i stort sett endast uppdatering av skärmen.

  Även de rutiner som begär tjänster av `CoDe` återfinns här. Detta gäller `CoDe_get_application`, `mxch_req_call()` samt `mxch_record_event()`.

- `c_handle.c`

  Denna fil var ursprungligen avsett att innehålla de rutiner som hanterar kopplingen mellan processens interna datastrukturer och användargränssnittet. Allt eftersom applikationen utvecklades visade det sig att denna koppling i stort sett alltid kunde överföras direkt från datastrukturerna till användaren. Det fanns inget behov av särskilda rutiner för detta. `c_handle.c` är därför närmast att betrakta som historiens vingslag.

- `c_lib.c`

  Här samlas "generella" rutiner som inte riktigt hör hemma i övriga filer. Filen är främst avsedd att innehålla rutiner som kan tänkas vara användbara i flera olika situationer. Här finns till exempel funktionen `c_copy()` som implementerar UNIX-kommandot `cp`, det vill säga kopierar en fil. Även `c_val()` som tar in ett ASCII-tecken och returnerar värde, tolkat som hexadecimalt värde, det vill säga för 'A' returneras 10 osv.

Funktionen `c_iconeditor()` startar upp processen `iconedit`. Det kan kanske vara ett ämne för diskussion huruvida detta är att betrakta som en "generell" och allmänt användbar rutin.

- `c_rec.c`

  Funktionen `c_do_record()` definierar en händelsehanterare för att överföra ljuddokument till annan användare. `c_do_record()` anropas från händelsehanteraren till den XView-knapp som skall handa funktionen `record`.

- `c_xv.c`

  "Hack" för att anpassa ISIS och XView till varandra.

- `c_xview.c`

  Innehåller de rutiner som driver användargränssnittet. Filen består av ett flertal korta funktioner som var och en ansvarar för någon parameter på katalogkortet. Funktionen `xview_changemember()` ansvarar för att dessa "småfunktioner" anropas, det vill säga ser till att kortet fylls med rätt information. Varje gång den aktuella medlemmens status förändras registeras detta med `xview_changemember()`.

  Filen innehåller även rutiner för att läsa och presentera användarikonen. En annan intressant funktion är `xview_set_canvas()` som läser in användarens rastrerade fotografi och visar detta i kortet.

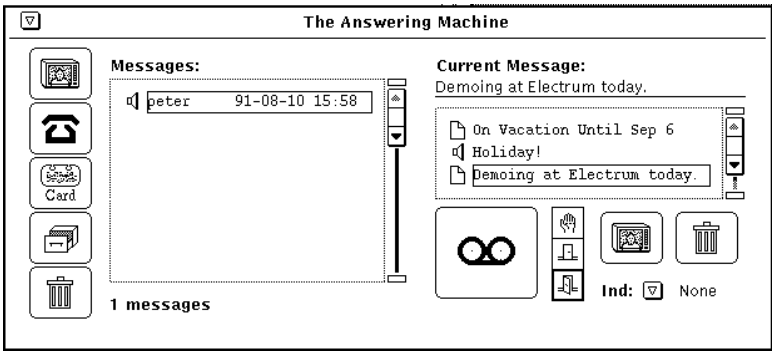


Fig.31: Answering Machine

5.12 Answering Machine

5.12.1 Teori / funktion

Telefonsvararen, `Answering Machine`, är en av de enklare applikationerna i *CoDe*. Den har få egna funktioner, utan lutar sig tungt mot tjänster som erbjuds av andra moduler, främst `sound`. Telefonsvararen har två huvuduppgifter, dels att samla in och strukturera *inkommande* meddelanden, och dels att organisera *egna* meddelanden. På så vis är analogin med en vanlig telefonsvarare fullständig.

Om vi studerar figuren ser vi att fönstret domineras av två *listor*. I den större till vänster ligger inkommande meddelanden, medan den lite mindre till höger innehåller användarens *personliga* meddelanden, som ska återspegla hans eller hennes nuvarande status. Den information som visas i listorna är:

- Vilken *typ* meddelandet har (ljud ( ) / dokument ( ) / mail ( )), visas som ikon längst ut till vänster för varje meddelande)
- För inkommande meddelanden:
  - Avsändare.
  - Datum då meddelandet sändes.
- För egna meddelanden:
  - Ett meddelande, en sträng som knyts till varje fil.

Inkommande meddelanden i listan till vänster är både sådana som andra användare skickat, dels vanlig `e-mail`. För att manipulera inkommande meddelanden används de fem knapparna längst ut till vänster. Dessa är (uppifrån och ned):

- **Visa meddelande** ( ). Skickar meddelandet vidare till `Displayer`, som tolkar det och presenterar det på rätt form. Telefonsvararen själv gör ingen som helst tolkning av *innehållet* i ett meddelande, den bara "tjuvkikar" på vilken *typ* meddelandet har för att kunna sätta rätt ikon i listan.
- **Svara på meddelande** ( ). Ringer upp avsändaren. Om denne inte är inloggad når vi avsändarens telefonsvarare, får höra dennes meddelande och kan lämna ett eget.
- **Visa katalogkort** ( ). Presenterar avsändarens katalogkort.
- **Spara meddelande** ( ). Ska spara meddelandet i NoteBook. Inte implementerat i nuvarande version av ***CoDe***.
- **Släng meddelande** ( ). Ta bort meddelande ur listan.

Listan till höger innehåller alltså *egna* meddelanden. Ljudmeddelanden talas in genom att trycka ned inspelningsknappen (den stora märkt med). Allt som sägs under tiden knappen är nedtryckt lagras undan på fil, och så fort knappen släpps upp dyker meddelandet upp överst i listan. Dokument läggs in genom att dra in dem från Open Windows *File Manager* och släppa filen var som helst på fönstet. Även här adderas filen direkt till listan.

Till varje ljud- eller textmeddelande kan en *sträng* knytas. Denna definieras av användaren själv, men bör naturligtvis återspegla innehållet i meddelandet. Från början innehåller strängen filnamnet, men detta kan ändras genom att *markera* detta i listan. Strängen visas då i textfältet `Current Message` ovanför listan och där kan texten editeras efter behag.

Till varje meddelande knyts även en *status*. Till höger om inspelningsknappen finns en kolumn med tre ikoner som representerar de tre olika grader av

önskad avskildhet som användaren kan markera (den fjärde statusen, *utloggad* kan inte påverkas manuellt). Dessa är uppifrån och ned:

- **Don't Disturb ( )**. I detta läge markerar användaren att han/hon är upptagen och inte vill bli störd. Andra användare kan inte heller ringa upp, utan når istället telefonsvararen, på samma sätt som när användaren inte är påloggad.

- **Closed Door ( )**. Lite mildare version av ovanstående. I detta läge *går* det att ringa upp användaren, och statusen är mer att se som en markering att inte störa om det inte är absolut nödvändigt.

- **Welcome ( )**. I detta läge visar användaren att han är villig att bli avbruten och uppringd. Detta läge är det normala, och är det som knyts till varje meddelande om inget anges.

Det viktigaste begreppet här är det *aktuella meddelandet*. Detta meddelande är det som är *markerat* i listan och vars sträng syns i `Current Message`-fältet. Detta meddelande kan sägas vara användarens ansikte utåt.

Det aktuella meddelandet är det meddelande som spelas upp för andra användare som försöker ringa upp när användaren inte är påloggad eller har markerat sig upptagen. Den sträng och den status som knutits till meddelandet kommer även att visas som information om användaren i andra användares `Team Map`.

Genom att alla meddelanden sparas kan användaren enkelt bygga upp ett litet personligt *bibliotek* av standardmeddelanden, och bara snabbt välja ett av dessa då hans/hennes status förändras.

De två ikonerna nere i högra hörnet är ikoner for att *visa meddelande* och *slänga meddelande*. Dessa fungerar analogt med sina bröder på vänstersidan, med den skillnaden att dessa två påverkar de meddelanden som ligger i den "egna" listan.

Under dessa två ikoner finns den sista detaljen som förtjänar att kommenteras i telefonsvararen; en meny för att välja hur användaren ska uppmärksammas på att ett nytt meddelande anländer. Valmöjligheterna är:

- **None**. Ingen indikation på att nytt meddelande inkommer.

- **Beep**. Användaren uppmärksammas på nyanlända meddelanden med en ljudsignal.

- **Pop Up**. Förutom samma ljudsignal som i ovanstående exempel aktiveras telefonsvararfönstret, och detta lägger sig överst på skärmen.

Dessa val påverkar naturligtvis endast de meddelanden som inkommer medan användaren är inloggad.

5.12.2 Koppling XView/data

Telefonsvararen är som tidigare sagt en *enkel* applikation. Strukturen hos de meddelanden den hanterar är helt transparent ur telefonsvararens synvinkel,

därför beskrivs den nuvarande meddelandestrukturen `MM_MSG` (för multimedia-message) under avsnittet om `Displayer`.

Telefonsvararen är även den tillämpning där separationen mellan interface och underliggande kod/data är minst. För att lagra den information om varje meddelande som finns i de båda meddelandelistorna används ett standardattribut i XView, `PANEL_LIST_CLIENT_DATA`. Detta attribut knyter till varje rad i listan 32 bitar data, som kan associeras till vad som helst. Här används detta till att låta attributet peka på rätt meddelandestruktur. Detta ger ett vackrare sätt att koppla rätt meddelande till XViewlistan, jämfört med att lägga denna information separat.

Detta har vi tillåtit oss med tanke just på att telefonsvararen är så pass enkel. Om denna skulle flyttas till en annan miljö vore det förmodligen lättare att programmera om dess funktionalitet från början i den nya miljön än att försöka flytta över vissa delar.

5.12.3 Funktion

Varje användare i ***CoDe*** har ett separat directory där nya meddelanden lämnas. Dessa ligger under `CoDe/MEMBERS/`*username*`/mail`. Vid uppstart går telefonsvararen igenom de nya meddelanden som anlänt sedan senaste inloggningstillfälle, dessa filer har suffix `.ur` (för UnRead). Allt andra processer som vill lämna ett meddelande behöver göra är att skriva ned en fil på rätt format i mottagarens brevdirectory. Med rätt format menas att:

- Filen inleds med en *header*, bestående av den struktur (`MM_MSG`) som beskriver meddelandet, dess typ, status, associerad sträng, datumstämpel, m.m.
- Efter headern följer själva filen, ljud eller dokument.
- Suffixet är `.ur`.

När telefonsvararen vid uppstart går igenom alla filer som slutar på `.ur`, strippar den bort headern och sparar data i en ny fil med suffix `.au` eller `.tx`, beroende på typ. Därefter tas `.ur`-filen bort.

Efter att ha gått igenom de nya filerna tittar telefonsvararen i `/usr/spool/mail`. Om det finns en fil där med användarens namn (namn i UNIX, inte i ***CoDe***) så innehåller denna användarens post. Denna adderas då till övriga nyinkomna meddelanden.

För att initiera listan med egna meddelanden går telefonsvararen lite annorlunda tillväga. Data om dessa ligger lagrade i en separat fil kallad `am.dat`. Vid uppstart går telefonsvararen igenom denna fil och initierar listan med egna meddelanden. Även det *aktuella meddelandet* sparas, i en fil kallad `myMessage`. Detta både för att telefonsvararen ska minnas mellan varje uppstart vilket meddelande som är aktuellt, dels för att andra användare alltid ska hitta rätt meddelande till Team Map.

`Answering Machine` består av följande källkodsfiler (i bokstavsordning):

- `am_cnv.c` definierar egna meddelandetyper som används i ISIS `bcast()`.

- `am_dir.c` är telefonsvararens huvudfil. Här finns rutiner för att initiera meddelandelistorna;
  - `am_init()` initierar i största allmänhet.
  - `am_init_messageList()` initierar listan med inkommande meddelanden.
  - `am_init_mail()` lägger till vanlig e-mail till listan.
  - `am_init_myMessageList()` initierar listan med egna meddelanden.
- `am_is.c` innehåller rutiner för att via ISIS kommunicera med andra processer.
- `am_rec.c` innehåller en enda rutin, `am_do_record()`, som sköter inspelning av telefonsvararmeddelanden.
- `am_stubs.c` innehåller de rutiner som anropas vid manipulation av gränssnittet, härifrån anropas alltså all annan kod. Här ligger även `main()`-rutinen.
- `am_ui.c` är helt automatgenererad från `devguide` och innehåller XViewkod för att skapa alla i gränssnittet ingående komponenter.
- `am_xv.c` innehåller den nya huvudloopen som kombinerar XView med ISIS plus rutiner för att aktivera telefonsvararen då den ligger i ikoniserat läge.

5.12.4 Vidareutveckling

En naturlig vidareutveckling är att göra kopplingen till vanlig e-mail ännu starkare. I dagsläget *läser* telefonsvararen bara e-mail, det borde inte vara alltför svårt att även kunna *skicka* post, för att öppna dörrarna och göra det möjligt att kommunicera med även icke-CoDe-användare.

5.13 Displayer

5.13.1 Teori / funktion

`Displayer` (eller kanske *presentatören* på svenska) är den applikation som sköter all *presentation* av meddelanden. Anledningen till att denna tjänst har separerats är att i framtiden göra det möjligt att skapa nya meddelandeformat (multimedia!), utan att behöva ändra i den övriga koden.

Även `Displayer` är ett enkelt program, mest ett skal ovanpå ljudprocessen. På skärmen representeras den av två fönster, ett litet kontrollfönster (kallat `Displayer` på bilden), där meddelandets typ, datumstämpel och eventuell meddelandesträng visas. I det fall då meddelandet är av typ *ljud* är det det enda som visas, om det är ett textdokument öppnas filen i ett popupfönster (`Text Displayer` på bilden). I detta fönster kan delar av texten markeras, klippas ut och klistras in i andra Open Windows-tillämpningar.

`Displayer` har egentligen två huvudanvändningsområden och två ansikten utåt. Det ena är det som syns i figuren. Här används `Displayer` för att visa

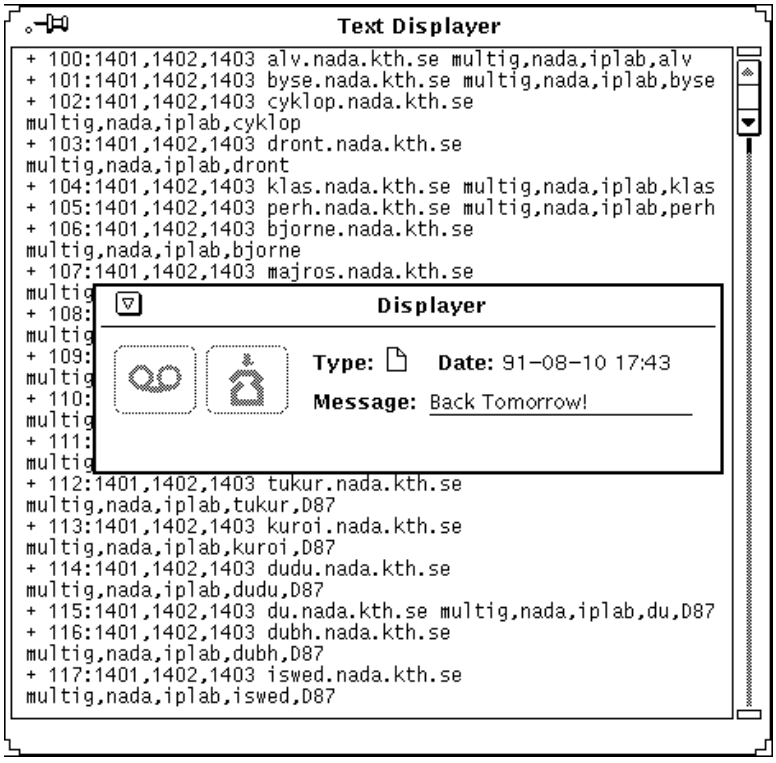


Fig.32: Displayer

meddelanden i telefonsvararen, både inkommande och egna. Det andra sättet är att verka som en *bakdörr in i andra användares telefonsvarare*.

Vid försök att ringa upp en användare som inte är inloggad, eller har markerat sig som upptagen träder dennes telefonsvarare i aktion. Detta är en sanning med modifikation, ty själva applikationen *Answering Machine* är inte inblandad alls. Det som sker är i stället att `Displayer`n aktiveras för att visa den uppringde användarens aktiva meddelande.

I detta läge antar `Displayer` en annan skepnad. Fönstrets titel ändras från `Displayer` till `The Answering Machine Of` *namn*, och de båda knapparna längst till vänster avdimmas. Dessa är en inspelningsknapp och en lägg-på-luren-knapp. Inspelningsknappen erbjuder det som vanliga telefonsvarare *brukar* erbjuda frustrerade vänner — en chans att tala in ett meddelande. Lägg-på-luren-knappen talar för sig själv.

5.13.2 Funktion

När `Displayer` körs igång vid uppstart av ***CoDe*** händer ingenting, processen ligger passivt och väntar på att någon annan process ska behöva dess tjänster. För att anropa den är en separat ISIS-kanal, `DISPLAYER_MSG` avdelad. När en annan process behöver använda `Displayer`:s tjänster skickar den ett ISIS-meddelande på denna kanal. Detta meddelande innehåller

- det **CoDe**-meddelande som ska visas
- användningsmod, `OWN` eller `REPLY`. Talar om för `Displayer` om den används för att visa egna meddelanden (då ska fönstret heta `Displayer` och knapparna vara dimmade) eller visa någon annans aktiva meddelande (då ska fönstret heta `The Answering Machine of` *username*, och knapparna avdimmas).
- användarens namn, så att de meddelanden som "talas in efter pipet" får rätt avsändare.
- mottagarens namn. Detta används bara i fallet då `Displayer` används i rollen som en inte inloggad teammedlems telefonsvarare. Om detta namn är "Kalle" blir fönstrets titel `The Answering Machine of Kalle`.

Utifrån dessa parametrar utför `Displayer` sitt arbete. Innehållet i **CoDe**-meddelandet studeras och i fönstret visas ikonen för meddelandets typ, tidsstämpeln och meddelandesträngen.

Efter detta är det dags att ta itu med själva meddelandefilen, som pekas ut av `mm_file`. Om `mm_type` är av typ `SOUND` anropas ljudprocessen, och denna får spela upp `mm_file`. Annars, om typen är `DOCUMENT` eller `MAIL` öppnas ett popupfönster med en texteditor för `mm_file`.

Så fort användaren har tillgodogjort sig meddelandet har `Displayer` slutfört sitt arbete. Denna kan då klickas ned i stängt, *ikoniserat* läge, så att den inte tar upp någon plats tills den behövs igen.

`Displayer` består av följande källkodsfiler (i bokstavsordning).

- `dp_cnv.c` innehåller även den rutiner för att definiera olika meddelandetyper till ISIS.
- `dp_is.c` initierar ISIS.
- `dp_rec.c` innehåller en rutin som sköter inspelning av meddelanden.
- `dp_stubs.c` består av *callback*-rutiner för de olika objekten plus `main()`.
- `dp_ui.c` är automatgenererad av `devguide` och innehåller XViewkod för att skapa de objekt som ingår i gränssnittet.
- `dp_xv.c` får XView och ISIS att trivas ihop.

5.13.3 Vidareutveckling

Möjligheterna till vidareutveckling är många. Det nuvarande meddelandeformatet är fattigt, och ett multimediellt meddelandeformat bör utvecklas.

5.14 Sound

5.14.1 Teori / funktion

All ljudkommunikation i **CoDe** sköts av en egen process, `sound`. Detta har sin grund i att datorns ljuddevice, `/dev/audio`, är en *exklusiv resurs*, det vill säga

att endast en process han öppna den samtidigt. På så sätt föll det sig naturligt att lyfta ut all ljudhantering till en speciell process och låta de övriga processerna styra denna genom att skicka förfrågningar med hjälp av ISIS.

Soundprocessen erbjuder andra processer en uppsättning tjänster. För att ifrån en annan process kunna utnyttja dessa behöver dessa rader läggas till programkoden:

```
#include "sound.h"
    .
    .
    .
address *soundServer;
int soundAns;
    .
    .
    .
soundServer = pg_lookup([hostname].Sound);
```

Sedan kan soundprocessen styras med nedanstående kommandon:

- `soundSetOutputGroup(char *gName)` sätter den ISIS-grupp som vi pratar med. Anropas i ***CoDe*** till exempel varje gång vi skiftar samtal i `Call Manager`.
- `soundOutputToFile(char *fName)` talar om att våra utdata ska skickas till en fil med namnet `fname` i stället för till högtalaren. Används i ***CoDe*** vid telefonsvararmeddelandeinspelningar och andra långa ord.
- `soundOutputToSpeaker()` återställer situationen till det normala efter föregående kommando.
- `soundPlayFileOnce(char *fName)` spelar upp en fil med namnet `fName` på den *egna* högtalaren i stället för att sända ut ljuddata på nätet. Efter att ha spelat filen en gång tas indata återigen från mikrofonen. Används i ***CoDe*** för att spela upp ljudmeddelanden och effektljud.
- `soundPlayFileLooping(char *fName)` fungerar precis som ovanstående, med den subtila skillnaden att `soundPlayFileLooping()` gör detta om och om igen i evigheters evighet eller tills `soundInputFromMicrophone()` (se nedan) anropas, vilket som nu kommer först. Används inte i ***CoDe***, men var till stor nytta vid utprovningen.
- `soundInputFromMicrophone()` ordnar så att indata återigen tas från mikrofonen. Detta sker "med våld" om så behövs, och uppspelning av ljud från fil avbryts tvärt. Behövs inte efter anrop till `soundPlayFileOnce()`
- `soundGetVolume(int *vol)` läser av vilken uppspelningsvolym som ljuddevicen är inställd på (en `int` mellan 0 och 100). Kan vara bra vid initieringar av ljudstyrkekontrolleringsskjutpotentiometrar.
- `soundSetVolume(int vol)` *sätter* istället uppspelningsvolymen till angivet värde (även här en `int` mellan 0 och 100).

- `soundShutDown()` gör att ljudprocessen stänger öppna filer och terminerar.
- `soundMakeMeCurrentSender()` gör att processen "får ordet" om antalet personer i gruppen är fler än två. Bör inte anropas om antalet medlemmar i gruppen är mindre än 3.
- `soundBeQuiet()` stänger av högtalaren. Denna sätts lämpligen på igen med `soundOutputToSpeaker()` när så behövs.
- `soundRecordEverythingIHear(char *fName)` startar en frenetisk aktivitet i filsystemet. Allt som skrivs till högtalaren skrivs även ned till fil med namnet `fName`. Detta i en takt av cirka en halv MB per minut. Systemansvarigas mardröm! Var tänkt att användas för att spela in samtal, men eftersom det inte går att mixa ljud och det därför bara är möjligt att höra endera parten i ett samtal, såg vi inget användningsområde. Används därför inte i dagens **CoDe**.
- `soundStopRecordingEverythingIHear()` avslutar ovanstående. Används följdaktligen inte *heller* i **CoDe**.

Dessa utgör alltså snittet mellan ljudprocessen och resten av **CoDe**. Den som vill implementera sin egen ljudprocess behöver bara se till att den reagerar på dessa kommandon. Alla de ovanstående är för övrigt implementerade som macron som sänder meddelanden till ljudprocessen med hjälp av ISIS `bcast()`.

5.14.2 Funktion

Datorns ljudenhet, `/dev/audio`, kan bara öppnas av en process för läsning och en för skrivning samtidigt. Läsning och skrivning sker båda i en takt av 8 kB per sekund. Detta innebär att bandbredden är 64kbit per sekund, det vill säga av modern telefonkvalitet.

Både läsning och skrivning är buffrade. Eftersom läsning och skrivning sker i samma takt kommer varje avbrott i utmatningen (som orsakas då systemet är tungt lastat) kommer att medföra en eftersläpning som aldrig kan "arbetas igen". Den maximala eftersläpningen beror på storleken på buffrarna. En stor del av arbetet har därför lagts ned på att utforska hur och när dessa buffrar ska *flushas*.

Ljudet styckas upp i delar om 1024 bytes och skickas ut via ISIS. Detta innebär att den *minimala* eftersläpningen vid denna bufferstorlek blir en åttondels sekund.

Ett problem vid samtal är att det inte går att mixa ihop två ljud från olika källor och spela upp på samma högtalare. En tänkbar lösning till detta vore att ta emot data från alla sändare, omvandla dessa till linjär form och på något sätt addera dem, samt skicka ut allt till högtalaren. Problemet här är dels att detta tar *tid*, samt att vi måste hålla reda på varifrån olika data kommer, och *när* de skickades. På det sätt vi löst problemet i **CoDe** tillåter vi bara en person åt gången att prata i samtal mellan fler än två personer. På så vis uppkommer aldrig problemet, men någon speciellt *snygg* lösning är det ju inte. Ljudprocessens kan *mycket* förenklat beskrivas såhär (i psuedokod):

```
send loop:

   do {
     read buffer from input device;
     if (outputToFile)
       write buffer to file;
     else
       send buffer via ISIS to output group;
   } while (running);

recieve loop:

   do {
     recieve buffer via ISIS;
     write buffer to speaker;
   } while(running);
```

Dessa ligger och kör hela tiden, och gör ingen som helst tolkning av de ljuddata de skickar över. Det våra styrkommandon (`soundInputFromMicrophone()`, `soundOutputToFile()`, etc) gör är att ändra på olika parametrar i huvudloopen så att till exempel vid uppspelning av en fil, sätts `input device` att peka på *filen* istället för mikrofonen. Motsvarande gäller för `output device`.

Då `sound` inte har något gränssnitt (i alla fall inget visuellt) och till stora delar är uppbyggd som en homogen enhet har alla funktioner placerats i en fil, `sound.c`. Huvudrutinerna är `main_service()` som läser in data och skickar iväg det, och `recieve_data()` som tar emot data.

I `sound.h` ligger definitioner av en del konstanter plus ett stort antal macron som implementerar de macron som övriga processer använder för att kommunicera med ljudprocessen (`soundBeQuiet()`, med flera).

5.14.3 Vidareutveckling

För att höja prestanda och göra det möjligt att utnyttja de nya protokoll för överföring av ljuddata som utvecklats vid SICS, bör den del i ljudprocessen som sköter själva ljudöverföringen bytas ut. I dag sänds detta med hjälp av ISIS, men detta är enkelt att byta ut till effektivare lösningar. En del ansträngningar borde även göras för att försöka mixa ihop ljud från olika källor (se ovan).

5.15 CoDe, igen?

Många som läst denna rapport har frågat oss: *om ni skulle göra om* **CoDe** *idag, vad skulle ni då göra annorlunda?* Vi skall nedan försöka redogöra för detta.

Grunden till **CoDe** lades under två intensiva veckor då en protoyp av systemet utformades. Protoypen skulle därefter användas vid en demonstration på Electrum. Den begränsade tiden tvingade oss att nå snabba resultat.

Vad vi egentligen *borde* ha ägnat oss åt i inledningsskedet var grundläggande studier av samarbetsprocesser, intervjuer med tänkta användare och utvärder-

ingar av de första idéerna. Vi borde ha hittat ett antal personer att konfrontera med våra tankar. Dessutom skulle vi redan från början studerat andra lösningar och verktyg som erbjöd stöd för samarbete. Nu slumpade det sig istället så illa att samtliga våra uppdragsgivare och handledare var starkt upptagna med andra uppgifter. Vi hade i stort sett endast oss själva att utvärdera våra tankar mot och våra erfarenheter av samarbete var begränsade. Dessutom blir den interna självkritiken kraftigt nedsatt när man har arbetat 10-12 timmar om dagen flera dagar i sträck, och tiden rusar mot "deadline".

Vad prototypen saknade var alltså en kritisk granskning av användbarheten. Hur gör människor egentligen när de samarbetar? Vilka verktyg finns det verkligen behov av? Vi borde även ha formulerat en konceptuell modell av systemet där viktiga samband och begrepp förtydligats, som till exempel hur olika verktyg skall interagera och hur användare och grupper av användare förhåller sig till varandra.

Även efter att prototypen var avklarad gick vi tillväga på ett annat sätt än vi skulle göra idag. När vi bestämt att **CoDe** skulle utvecklas under UNIX och OpenWindows kastade vi oss in i en jakt på möjliga utvecklingsverktyg. När `devguide` och ISIS utsetts till vinnare inledde vi omedelbart utvecklingen av **CoDe** *samtidigt* som vi försökte lära oss verktygen. Detta var ett misstag.

Vi skulle först ha avsatt en begränsad tid för studier av verktygen. Därefter borde en stor tid ha investerats i grundläggande systemering, resulterande i en specifikation av vad som skulle ingå i **CoDe** både avseende funktionalitet och moduluppdelning, med olika modulers uppgifter och tjänster, samt snittet moduler mellan. Specifikationen skulle naturligtvis även definiera ett genomtänkt och homogent användargränssnitt. Detta tillvägagångssätt hade medfört att tid endast återstått för implementation av en smärre delmängd av **CoDe**. Förmodligen hade dock detta varit att föredra; nu kommer en eventuell vidareutveckling att innebära vissa svårigheter eftersom en "röd tråd" saknas i systemet. Det är inte alltid självklart hur vi har resonerat under programutvecklingen (ibland inte ens för oss själva).

För att kort sammanfatta ovanstående utläggning; vi borde ha tänkt mer och programmerat mindre. Skillnaden att tänka jämfört med att programmera är dock att eventuella framsteg är betydligt mer märkbara för det senare. Frestelsen är därför alltid stor (och alltför ofta oemotståndlig) att kasta sig framför datorn och "köra igång".

Som avslutning vill vi istället ge några exempel på vad vi *inte* skulle ha gjort annorlunda idag.

- Vi skulle *inte* ha valt en annan uppgift

  Arbetet med **CoDe** har varit både stimulerande och utmanande. Vi är övertygade om att detta område är en viktig del av framtiden och är tacksamma att vi fått vara med och "smaka". Dessutom har vi uppskattat den fria och självständiga form arbetet utförts under.

- Vi skulle *inte* valt en annan plattform

  Att arbeta under X Windows och UNIX är att kombinera det bästa av två världar. Moderna fönsterhanteringssystem kan skapas parallellt med

enklare textbaserade program. Vi är dock inte övertygade om att X Windowsapplikationer är enklare för *användaren* än jämförbara Macintoshprogram.

- Vi skulle *inte* ha valt andra utvecklingsverktyg

  ISIS är ett utmärkt verktyg för processkommunikation. Vi beklagar att liknande verktyg inte finns under Macintosh och PC för lokala nätverk (oj, vilka program man skulle kunna göra!). `Devguide` kombinerar det bästa av verktyg som HyperCard och MacApp. Det är enkelt att snabbt utforma enkla prototyper, som i stort sett omedelbart är färdiga applikationer, exekverbara utan stödmiljöer av det slag HyperCard kräver (Open Windows krävs ju i och för sig). Dessutom slipper vi konstiga scriptspråk som HyperTalk utan kan direkt använda ett kraftfullt högnivåspråk. Den enda nackdelen är möjligtvis att applikationen inte kan förändras "under drift", vilket ofta görs med HyperCard-prototyper.

5.16 Kopplingen till användargränssnittet

Vi ska här diskutera kopplingen mellan användargränssnittet och övrig kod i **CoDe**. Vi har redan tidigare behandlat XView:s funktion och uppbyggnad i stora drag i avsnitt 5.2.3, så här kommer vi att lägga tyngdpunkten på hur *vi* utnyttjar systemet i **CoDe**. Dessutom ska vi försöka bedöma vilka problem som kan uppkomma vid försök att byta ut XView mot något annat, motsvarande system.

Att här utförligt funktion för funktion redovisa *hela* snittet mellan interface och kod skulle uppta åtskilliga sidor och dessutom vara av ringa intresse för andra än de som direkt ska *programmera* under **CoDe**. Därför utelämnar vi detta och hänvisar istället till studier av programkoden, företrädelsevis filerna `*_stubs.c`. Där finns som vi tidigare nämnt (sidan 37 och framåt) de rutiner som anropas vid manipulation av gränssnittet.

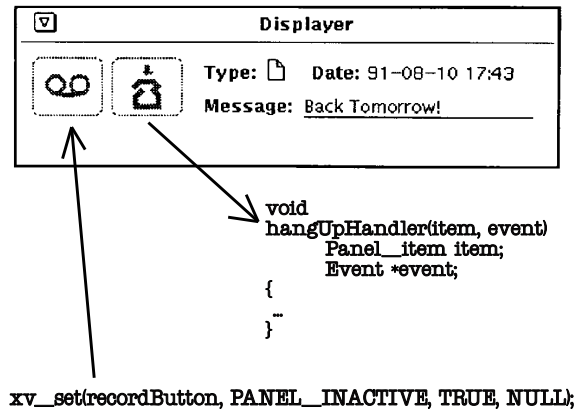


Fig.33: Kopplingen till XView

Kopplingen till XView sker åt två håll (se figur 33);

1. Från gränssnittet och ned mot den underliggande koden genom att XView hanterar användarens manipulation av knappar och anropar motsvarande funktioner.

2. Åt andra hållet, då applikationens interna tillstånd kräver att detaljer i gränssnittet uppdateras (som till exempel att knappar som inte har någon effekt dimmas). Detta sker genom anrop av XView:s standardfunktioner `xv_set()` och `xv_get()`. Dessutom använder vi X Windows direkt för att utföra den interaktiva grafiken (för att flytta om ikonerna i `Call Manager` och för allt ritande i `WhiteBoard`).

Det finns alltså ingen möjlighet att under vår utvecklingsmiljö *verkligen* separera programmet från dess gränssnitt . All grafisk manipulation utförs genom funktionsanrop som logiskt betraktat ingår i programmet. Vår föresats har istället varit att så långt som möjligt separera den *kod* som direkt manipulerar gränssnittet från den övriga koden i olika filer. En funktion som direkt påverkar gränssnittet skall aldrig förändra interna variabler och tillstånd. Den skall enbart presentera given information, och kan därför enkelt bytas ut mot nya funktioner som visar en annan bild för användaren.

Även om vi *ansträngt* oss för att separera gränssnittet från den övriga koden behöver inte detta innebära att det vore *lätt* att byta ut XView mot något annat system. Det skulle förmodligen *gå*, förutsatt att den alternativa miljön fungerar på ungefär samma sätt som XView. Att flytta den existerande **CoDe** till en miljö som grundas på en helt annan metafor, som till exempel *MultiCard* vore troligen svårt. Risken finns att man i denna situation skapar fler nya problem än man löser, och det lättaste bör vara att skriva om allt från grunden, kanske med nuvarande **CoDe** som inspirationskälla.

Nya problem torde även uppkomma vid försök att strukturera om *funktionaliteten* i systemet. Den uppdelning vi gjort i olika verktyg (`Call Manager`, `Answering Machine`, etc.) är *statisk*, det vill säga att de olika applikationerna specialiserats att sköta varsin del av arbetet. Att ändra denna uppdelning kan exempelvis innebära att låta `Team Map` sköta hanteringen av samtal. Det är svårt för oss att exakt bedöma hur svårt detta skulle vara att genomföra i praktiken, men klart är att det *inte* är trivialt.

6 Förslag till fortsättning / nya examensarbeten

Nu när vi har klarat av "vår" del i det hela, ska vi försöka blicka framåt och se vilka vägar det borde gå att fortsätta vandra för framtida exjobbare. Vi är ju trots allt de som bäst känner till vårt systems brister och förtjänster, och utifrån denna vetskap har vi sammanställt en lista med 6 stycken delvis överlappande potentiella inriktningar.

1. **Dokumentation och presentation av kunskap.** En viktig del av The-KnowledgeNet[Marm91] är just dokumentationen av den kunskap som olika medlemmar besitter. Här har vi bara skrapat på ytan i den nuvarande **CoDe**, så det finns mycket att göra för framtida exjobbare. Ett sätt är det vi föreslog i vår prototyp — att till katalogkorten koppla ett separat fönster med plats för personens anteckningar om sig själv och andras anteckningar om honom. Tillsammans med en browser för att söka igenom denna information utgör detta en möjlig lösning. Browsern bör även kunna söka igenom informationen i nästa förslag:

2. **Distribuerad text / hyperTexthantering.** Detta examensarbete skulle omfatta utveckling av det verktyg som vi i vår prototyp kallade NoteBook. Detta var tänkt att vara någon form av hypertexttillämpning för att organisera information, skapa länkar mellan dokument, etc.

3. **Synkrona verktyg i en distribuerad miljö.** Här gäller det att skapa flera synkrona verktyg redo att användas i anslutning till det aktuella samtalet. I nuläget har vi bara implementerat `WhiteBoard`, en distribuerad riteditor, men detta examensarbete skulle kunna omfatta studier av vilka olika verktyg det finns behov av, samt implementation av dessa. Exempel på synkrona verktyg kan vara:

   - En *CoEditor*, ett verktyg för att gemensamt skriva text. Här måste en mängd problem beaktas, om texten ska delas upp i en privat och en offentlig del, på vilken nivå texten ska "låsas", hur samtidiga uppdateringar ska hanteras, etc. En mängd mer eller mindre lyckade CoEditorer har utveclats, och en del av arbetet vore att studera dessa och dra lärdom av deras brister. En lustig bieffekt här är att man genom att använda CoEditorn vid rapportskrivandet samtidigt kan utvärdera denna.
   - *Video.* Arbetet är här bara att integrera de *existerande* rutiner som utvecklats på SICS för att överföra videoinformation. I dagens läge klarar dessa att överföra bild mellan två punkter (1:1), men vidareutveckling pågår för att även klara 1:N och M:N. Sålunda kommer **CoDe** med hjälp av dessa rutiner att kunna erbjuda fullständiga videokonferensmöjligheter.
   - *Telepresence.* Detta är ett annat projekt utvecklat på SICS, som **CoDe** kan integrera. Tanken är att till varje samtal koppla en *virtuell värld* i 3D, där de olika parterna i samtalet kan gå omkring och manipulera olika objekt (och varandra!).

4. **Asynkrona verktyg.** Det asynkrona verktyg vi främst saknar är någon form av Bulletin Board / anslagstavla. Denna skulle kunna anta en mängd olika former och vara mer eller mindre multimediell, detta borde bli föremål för omfattande förstudier.

5. **Distribuerad Multimedia.** Dagens CoDe understödjer bara två olika typer av meddelanden, ljud och dokument (egentligen tre om man räknar in mail, men detta är bara ett specialfall av dokument). All hantering av dessa meddelanden är separerad från dess tolkning, så att det ska vara lätt att i framtiden ändra formatet till att kunna föra över generella multimediadokument. Målet med detta exjobb vore att specificera och implementera detta nya meddelandeformat. Vid överföring av tillämpningsfiler måste även information om *vilken* tillämpning som skapat dessa föras över, så att motsvarande program kan startas i andra änden för att tolka data. Problemen här är många. Hur ska mottagaren reagera om han saknar den tillämpning som behövs för att tolka de data han just fått? Ska även tillämpningen skickas över tillsammans med data, eller ska den köras på sändarens maskin och presentera sig på mottagarens skärm?

6. **Utvärdering av CoDe.** Ett lite "mjukare" examensarbete, tänkt att studera CoDe och utvärdera om / hur det används. Tanken är att installera en förfinad (enligt ovanstående punkter) version av CoDe i en verklig arbetsmiljö (av typ SICS) och undersöka hur olika personer utnyttjar de tjänster som erbjuds, om de anser sig ha haft nytta av CoDe i arbetet, vad de har saknat etc.

Fler inriktningar är naturligtvis tänkbara. Många av de ovanstående låter intressanta, och vi är nästan ledsna att vi inte kan utföra dem själva.

References

[Bush45] Bush, Vannevar, "As We May Think", *Atlantic Monthly* 176(1):101-108, June 1945.

[Lidb91] Lidbaum, Peter, och Tobiasson, Magnus, "Rapport till projektet i Översättarteknik", Stockholm, Mars 1991.

[Marm91] Marmolin, Hans, "TheKnowledgeNet — An Environment for Distributed Design", In *Proceedings from the 2nd MultiG WorkShop*, Stockholm, Juni 1991.

[Hell90] Heller, Dan, "XView Programming Manual", O'Reilly & Associates, Inc., 1990

[Jone88] Jones, Oliver, " Introduction to The X Window System", Prentice-Hall, Inc., 1988

[Open90] "Open Windows Developer's Guide 1.0 Users Manual", Sun Microsystems, June 1990

[ISIS90] "The ISIS System Manual, Version 2.1", The ISIS Project, Cornell University, Sept 1990

[Piri85] Pirinen, Joakim "Socker-Conny", Tago Förlag, 1985

Bilagor

A Personliga erfarenheter

Examensarbete.

Vi smakar på ordet och väger det på tungan. En ödesmättad stämning sprider sig långsamt ut i kroppens alla delar. Det är *hit* alla vägar har lett, alla sidospår genom atomfysik, ekonomi och optimeringslära, alla projekt och tentor. Kort sagt har *allt* vi gjort de senaste åren gjorts enbart för att förbereda oss för detta. Tanken svindlar.

Vi har tillsammans skrivit ett tiotal mer eller mindre omfattande rapporter, alla i den lättsamt kåserande stil som blivit något av vårt kännemärke. Trots att vissa av dessa har ägnat fler sidor åt ämnen som *livets mening* eller *kvinnan*[21] än åt det de *borde* handla om, har ingen blivit direkt dåligt mottagen, varken av lärare eller elever. Vi har alltid försökt att leva upp till vårt motto, *Information by Entertainment*, och stödja vår tes om att inget budskap blir tydligare av att framföras torrt och tråkigt.

Detta är alltså historien om vårt examensarbete. Den har de flesta ingredienser en bra historia *ska* innehålla, spänning, passion, svek och hämnd och är ett försök till utförlig redogörelse för vad vi gjort under de fyra månader vårt examensarbete pågått. Vi har försökt vara så sanningsenliga som möjligt och varken förtiga fakta eller framstå som intelligentare än vi är.

A.1 Arbetets uppläggning

Det totala arbetet sönderföll i två delar. Först att göra en prototyp av hur vi skulle vilja realisera "helheten", idéer på hur de olika samarbetsverktygen skulle kunna utformas. Som ett andra steg skulle delar av dessa idéer implementeras under UNIX, på SUN:s SparcStation.

Det var bråttom att starta det första steget; redan efter två veckor skulle vår prototyp visas som en demo på den andra MultiG-workshopen i Electrum. Vi hade alltså fjorton dagar på oss, att från ingenting både kläcka idéer *och* implementera dem, för att kunna ha något att visa upp.

A.2 Den första prototypen

Vår del av arbetet inleddes måndagen den tredje juni med att vi sammanträffade med Hans Marmolin för att få riktlinjer inför de första två veckorna. Egentligen hade vi tjuvstartat tidigare, genom att läsa igenom några av de rapporter vi fått vid ett tidigare besök. Dessutom hade vi lördagen innan spenderat två timmar efter en tenta med att över varsin öl utföra grundläggande brainstorming i ämnet samarbete. Detta för att inte vara *totalt* nollställda.

Vi hade fått ett eget rum med två mac:ar uppe på vinden i Nalen, i ett rum

"Vi hade fått ett eget rum, som med intern NADA-jargong kallas bastun"

[21] ...eller andra ämnen som vi inte *heller* behärskar.

som med intern NADA-jargong kallas *bastun*[22]. Hans monterade upp ett stort kladdblock på två spikar och förklarade sina idéer lite mer ingående. Det han tryckte hårdast på var att vi skulle betona *verktygsmodellen* och att vi inte bara skulle skapa "ännu ett konferenssystem"[23]. Likt unghingstar på vårbete satte vi genast lyckligt igång att skapa ännu ett konferenssystem. Lyckligtvis insåg vi det själva, och styrde in vår brainstorming på ett plan där våra idéer inte stred mot våra förutsättningar.

På ett tidigt stadium började vi utforska HyperCard och de möjligheter vi skulle ha att realisera våra idéer. Vi hade innan vi började utgett oss för att *kunna* HyperCard, men när vi började insåg vi att det vi egentligen kunde var att titta på när *andra* använder HyperCard, vilket inte är riktigt samma sak. Nu är HyperCard lyckligtvis inte speciellt svårt, så efter en kvälls laborerande och en natt med varsin HyperTalkmanual ansåg vi oss behärska ämnet till fullo.

Ett av problemen var att inse vad vi skulle kunna modellera med HyperCard och vad som måste fejkas. Därför lade vi ner en halv dag på att utforska det arkiv med XCMD:s som finns på Elevarkiv. Vi hittade rutiner för att göra menyer (både popup- och pulldown-) och för att sköta andra uppgifter vi ansåg nödvändiga. Däremot var den support vi fann för att göra olika *fönster* så gott som obefintlig. Det sätt att ha olika stackar i skilda fönster som HyperCard 2.0 erbjuder verkade lovande, tills vi upptäckte att det var omöjligt att dra objekt *mellan* olika fönster. Problemet tycktes svårlöst, så vi sköt det åt sidan tills vidare, och använde HyperCard mer som ett sorts kreativt klotterplank för att fästa våra idéer om verktygen på pränt.

Vi inledde med att specificera vad vi ville att vårt system skulle uppfylla (se avsnitt 3.2). Med dessa lösa krav (och några till) började vi implementera våra idéer. Detta blev en tung och tidskrävande process. Vi tog några verktyg var och gjorde designutkast, som vi bollade fram och tillbaka mellan varandra.

Efter att ha grälat några dagar om hur verktygen skulle vara utformade och hur arbetsuppgifterna skulle fördelas dem sinsemellan kom vi fram till en designlösning av verktygen som vi var nöjda med. Vi presenterade dessa lite informellt för Yngve och Konrad uppe i bastun, och fick klartecken att kombinera ihop dem till en demo.

A.3 Vår demo

Efter en veckas arbete hade vi idéer till en full uppsättning verktyg, och av arbetet återstod att sätta ihop dessa till en fungerande enhet. Tiden hade runnit iväg så långt att vi aldrig skulle hinna få klar en *fungerande* prototyp, utan vårt enda hopp var att kunna fejka en demo på ett så trovärdigt sätt som möjligt.

Vi fick en diskett från Hans med posten med en färdig skärmbild av ett videoverktyg utvecklat vid FOA, som vi kanske skulle kunna använda. Denna låg i en SuperCardstack, och vi letade reda på en version av SuperCard 1.5 för att undersöka det vi fått. Vi hade redan tidigare övervägt att använda SuperCard istället för HyperCard för hela demon, detta på grund av att SuperCard ger

[22] Vi behövde inte arbeta länge för att förstå varför. Molniga dagar kunde temperaturen sjunka under trettio grader. Om vi hade tur.

[23] Yet Another Conference System?

utökade möjligheter att själv skapa egna objekt. Då blev vi avrådda, eftersom SuperCard inte ansågs så stabilt som ett programmeringssystem *bör* vara, men nu beslöt vi att ge det en chans.

Efter installation provkörde vi några av de exempelstackar som följer med. Den dator vi provkörde på hängde sig mer eller mindre okontrollerat fyra gånger på tio minuter, den sista gången så totalt att hela skärmen svartnade, underliga ljud strömmade ur högtalaren och resetknappen upphörde att fungera. Efter det beslöt vi att hålla oss till HyperCard.

Lösningen verkade vara — en *video*! Konrad ordnade fram ett RasterOps videokort som gjorde det möjligt att spela in skärmsignalen med en vanlig videobandspelare. På så vis borde vi kunna "hårdkoda" vår demo utan att det skulle märkas på samma sätt som vid en levande demonstration.

Vi började skissa på ett scenario, där vi på ett okonstlat sätt skulle få visa så många av våra verktyg som möjligt. När vi dragit upp riktlinjerna i stort började vi tillverka den serie kort som skulle implementera dessa. Då vi väl svalt vår stolthet fejkade vi allt. Allt.

Hela vår demo består av ett hundratal HyperCardkort. När vi ska visa att ett fönster öppnas byter vi helt enkelt kort till ett där en bild av fönstret visas. När fönstret stängs byter vi tillbaka igen. Denna datalogiska harakiri är ingenting vi är *stolta* över, men under den tid vi hade till vårt förfogande var det den enda metod vi kunde använda.

A.3.1 Förberedelserna fortsätter

Mitt i arbetet bättrade Kai-Mikael (Jää-Aro, NADA) på våra mindervärdeskomplex genom att ge oss en demonstration av *Slate*, ett existerande system för avancerad dokumentförmedling och samarbete. Detta gjorde oss lite nedstämda, många av de idéer vi värkt fram fanns redan i sinnevärlden, implementerade och klara! Vi arbetade oförtrutet vidare.

"Mitt i arbetet bättrade Kai-Mikael på våra minder-värdeskomplex"

Sakta men säkert blev vår stack större och större. Slutligen var den över en halv MegaByte. Vi började anse oss klara, och visade i ett ögonblick av högmod upp vårt alster för Kai-Mikael, som ville vara snäll och visa intresse. Hans första kommentar var "ni har stavat fel, det heter *message* och inte *messege*". Vilket han ju hade alldeles rätt i. Problemet var bara att texten i fråga låg som *bild* på varje enskilt kort, och att ändra detta skulle innebära att vi måste gå in på ett femtiotal ställen och ändra utan att en enda pixel fick hamna fel. Eftersom det var två dagar kvar till vår deadline beslöt vi att strunta i det. Nu efteråt inser vi att det var rätt, ingen annan verkar ha sett på videon så noga att de upptäckt något.

A.3.2 Den första videon

Tiden kom att bli videoproducenter. Vi utarbetade en engelsk dialog som vi skulle framföra samtidigt som vi spelade in videon, för att förhoppningsvis göra demon självförklarande. Allt verkade bra; då slog ödet till igen.

Två meter bakom den Mac vi var tvungna att använda för inspelningen monterades en DECConcentrator 500, en stor burk som hade något att göra med fiberoptikledningen till Kista. Denna var utrustad med ett stort antal fläktar som alla tävlade inbördes om att låta högst och mest. Hela Nalen förvandlades

"Hela Nalen förvandlades till ett ljudtekniskt katastrof-område"

till ett ljudtekniskt katastrofområde. Vi gjorde några försök att spela in tal i normal samtalsnivå, men det som fastnade på bandet lät som en TV-reporter på reportage bland bortblåsta hustak i höststormarna. Vi frågade lite försynt om det inte skulle gå att *stänga av* apparaten en liten stund medan vi gjorde vår inspelning, men det enda svar vi fick var att den gick loss på tre kvarts miljon, och att ju mindre vi rörde den, desto bättre.

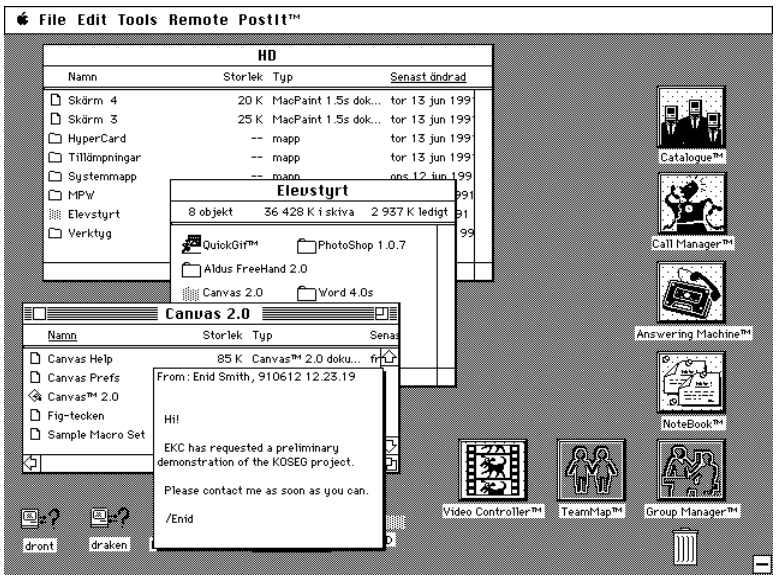


Fig.34: The Collaborative Desktop. Bild ur filmen med samma namn

Trots problemen offrade vi fredag natt till att spela in. Vi väntade till alla andra hade gått hem innan vi började repetera. Vårt engelska manus var omfångsrikt, sju A4-sidor, och vi läste igenom det högt *många* gånger innan vi vågade oss på en inspelning. Svårigheten var att koordinera allting, att hålla ordning både på all utrustning och på vårt manus. När vi tittade på vad vi åstadkommit märkte vi att vi knappt hörde ljudet. Till nästa tagning trängde vi ihop oss, höjde volymen till det maximala och skrek ut våra repliker rakt in i mikrofonen. Denna gång lät det högre. Tyvärr är det *väldigt* svårt att låta seriös när man skriker.

"Tyvärr är det väldigt svårt att låta seriös när man skriker"

Även bildkvaliteten lämnade mycket övrigt att önska. Dels syntes inte hela skärmytan på TV:n, dels flimrade vissa delar så kraftigt att det var svårt att urskilja detaljer.

Efter några försök lyckades vi prestera en tagning som vi var nöjda med. Tunnelbanan hade sedan länge slutat gå, så vi promenerade hemåt i försommarnatten med kassetten i handen.

A.4 Installationen på Electrum

På söndag, dagen innan vi skulle presentera vår demo på konferensen var det dags att hjälpa till att frakta upp all hårdvara som skulle användas. Vi beslöt att

komplettera videon med en Macintosh, för att kanske kunna visa saker lättare om någon mot all förmodan skulle vara intresserad och fråga. Vi packade ihop en av våra datorer och bar ner den i delar till en inhyrd skåpbil, tillsammans med ett par SUN arbetsstationer värda några årslöner.

Vid framkomsten till Kista upptäckte vi vid uppackningen att vi saknade *skärmen* till vår Macintosh. Vi bråkade om vem som burit ut den till bilen, men ingen ville riktigt kännas vid detta. Vi gick jättenervösa på lunch med Yngve och Ragge och diskuterade människor som blivit överkörda av spårvagnar. Några oroliga timmar följde, innan vi på telefon fick veta att vi glömt den kvar uppe i bastun.

Fig.35: Vi hade svårt att få tiden att räcka till

Videon kunde vi ju iallafall montera ihop, så vi anslöt alla kablar till TV:n och tryckte in kassetten. Så fort vi startade bandet samlades en folkmassa runt och alla fällde kommentarer av typ *Vicka ere som skriker?* och *Gårente å stänga av ljude?* Känsliga som vi är för yttre intryck beslöt vi att stänga av ljudet under demonstrationerna. En hel natts arbete, förgäves. Fast vi ska kanske vara tacksamma att ingen bad oss att stänga av bilden också ("kan ni lägga en filt över teven, grabbar, så kan vi servera kaffe ovanpå?").

"Känsliga som vi är för yttre intryck beslöt vi att stänga av ljudet under demonstrationerna"

Konrad tyckte inte heller om ljudet. Dessutom påpekade han att vi kanske borde göra iordning något litet faktablad som vi skulle kunna dela ut till nyfikna. Detta hade vi förbisett (vi trodde inte att vi skulle träffa några nyfikna), så det var bara att åka tillbaka till skolan och författa.

Vi skrev snabbt ihop varsin sida och satte i PageMaker. När vi skulle kopiera upp våra alster fastnade pappren i kopieringsapparaten, så att vi var tvungna att skriva ut 100 exemplar på laserskrivaren. (Några dagar efteråt upptäckte vi att dessa papper innehöll vissa halsbrytande tryckfel, bland annat kallade vi snapshot för det betydligt burleskare *snopshot*. Det är tur att ingen läser de papper de får)

A.5 Konferensen

Dagen D. Vi anlände tidigt för att ställa upp all utrustning på utställningsplatsen. Allting fanns på plats, den försvunna skärmen hade kommit tillrätta, så vi hade bara att koppla ihop alla sladdar. Sedan "checkade vi in" som konferensdeltagare och fick en massa papper, en tygkasse och varsin T-shirt. För detta hade NADA betalat 1875:-.

Efter några inledningstal var det dags för kaffe och demonstrationer. Vi gruvade oss. Ett tag hade vi övervägt att uppträda incognito med papperspåsar över huvudet. Detta för att undvika att bli svartlistade från *alla* jobb i framtiden[24].

Vi ställde oss vid vårt bord och inväntade anstormningen. Vi blev förvånade när det verkligen *kom* människor, som dessutom verkade *intresserade*. Vi körde våran åttaminutersvideo framlänges och baklänges, och förklarade för förbipasserande vad som utspelade sig. En grupp japaner smög försynt fram och tog med varsitt exemplar av vår lilla stencil hem till *Nippon*.

Vi fick även ett kvitto på att vår fejkade miljö var verklighetstrogen. När vi gått iväg för att få oss varsin kopp kaffe vågade sig en konferensdeltagare fram till datorn och försökte klicka på några av de fönster som låg som bilder i bakgrunden, och han såg *väldigt* fundersam ut när det inte gick.

Den engelska varianten av vårt föredrag verkade uppskattat. Att höra *oss* berätta någonting på engelska verkar vara som att höra någon dra en fräckis på finska — man missar kanske poängen men skrattar ändå.

Flera föredrag, lunch med demonstrationer på rasten, och sedan var det dags för en *tutorial* på ämnet Computer Supported Cooperative Work. Här höll *Jonathan Grudin* en föreläsning i ämnet[25] och visade en video i vilken ett flertal system för samarbete visades. Vårt mod sjönk mer och mer i takt med att videon fortskred, för i princip *alla* beståndsdelar i vår prototyp fanns här redan som delar i existerande applikationer, inte bara som bilder i en HyperCardstack. System som *Rapport, Cruiser* och Apple's *Spider* hade alla stora likheter med vår modell.

Vi genomled en kortare depression och grät ut hos Konrad på rasten. Han försökte trösta oss genom att påpeka att "allt det som dom har lagt ner hundratals manår på att utveckla, det har ni kommit på på två veckor". Det är kanske så vi måste se på saken för att klara oss från självmordstankar.

"Allt det som dom har lagt ner hundratals manår på att utveckla, det har ni kommit på på två veckor"

Efter en sista sejour vid videon var det dags att packa ihop och fara hem. Vi försökte se till att vi hade med oss mer prylar hem än vi hade med oss dit.

Dagen efter konferensen fick vi visa vår prototyp för Jonathan som tack och lov var alltför diplomatisk för att kritisera den öppet. även Hasse Haitto och Donald Broady på IPLab ville ta en titt. Vi började fundera på att hålla ett symposium och låta alla som ville titta betala 1875:- för att se vår åttaminuters video. Men då kunde de få varsin T-shirt på köpet.

[24] Var det *ni* som var på MultiG-workshopen förra året? Ring inte oss, vi ringer er.

[25] Han pratade även en hel del om ämnets *historia*. Uppenbarligen har något stort hänt inom området ungefär var tionde år. 1965 med multianvändarsystemen, -75 med utvecklingen av persondatorn och enanvändarsystem och -85 med GroupWare och nätverk. Nu har vi skådat in i framtiden och förutspår att trenden från 1995 och framåt kommer att vara *NoWare*, programvaran ingen använder. Kom ihåg var ni hörde det först.

A.6 Vidare inriktning

Dagen efter konferensen bjöd Yngve alla inblandade på tårta för att fira att allt gått bra (och framför allt att det var över). Vi skulle komma överens om hur vi skulle tillbringa den återstående tiden. Efter att ha arbetat i stort sett dygnet runt under den inledande tvåveckorsperioden kändes det inte riktigt som om vi bara uträttat två veckor av våra tre månader, men så var det.

Konrads förslag var att vi skulle lämna Macintosh för att gå över i UNIX-miljön och implementera delar av våra idéer där. Detta tyckte vi om, delvis för att vi insåg att det skulle innebära att vi skulle få lämna bastun för arbetsstationerna en trappa ner. Tyvärr hade Konrad också idéer om att vi skulle arbeta i Smalltalk. Dessa tog vi snabbt ur honom.

Vi ville helst arbeta i C eller C++, eftersom detta är den miljö vi känner oss mest hemma i, och som vi bedömde skulle innebära kortast inlärningsperiod. Även om Smalltalk säkert har sina förtjänster var vi rädda att den tid vi skulle vara tvungna att lägga ned på att lära oss klassbiblioteken lätt skulle kunna skena iväg.

Yngve kom in tillsammans med Jonathan då samtalet var som hetast, och förklarade för honom att "de har den vanliga plattformsdiskussionen".

Då vi väl fått vår vilja igenom ställde Konrad upp en lista med programmeringsverktyg som vi skulle kunna använda, både till användargränssnittet, ljud- och bildöverföringen och distributionen. För gränssnittet var utbudet av verktyg nästan överväldigande. HyperNeWS, Guide, Andrew Toolkit, Interviews och Ioffice, alla verkade lovande och vi kände oss som småbarn i en godisbutik.

Hans anmärkte att det fanns någon sorts bestämmelse "uppifrån", att användargränssnitten bör baseras på någon sorts *kortmetafor*, som i HyperCard, och att detta tydligen var fallet i HyperNeWS. Han rekommenderade därför att vi borde inleda med att titta på det, men att vi inte skulle känna oss tvungna att använda att använda det om något av de andra skulle visa sig vara bättre.

För att distribuera vår programvara hade vi däremot ett självklart val — ISIS. Konrad hade arbetat en hel del med ISIS under Smalltalk, och sade sig behärska ämnet. Dessutom fanns det för en gångs skull rikligt med dokumentation.

Exakt *vad* vi skulle implementera blev också föremål för diskussion. Vi hade hoppats slippa undan med en mindre delmängd av våra verktyg, men Konrad verkade helt inställd på att vi skulle implementera allt. Det är ett helvete att vara visionär om man måste *förverkliga* sina idéer.

A.7 Litteraturstudier

Då ett examensarbete bör innehålla någon form av litteraturstudier bad vi Konrad att rekommendera några böcker i ämnet. Vi delade upp litteraturen mellan oss, så att Peter tog de teoretiska volymerna om CSCW och samarbete och Magnus fick några lite mer praktiska volymer om X Windows och Distribuerade Applikationer.

Detta plötsliga intresse för *teori* berodde inte på att vi blivit omvända utan på att vi märkt att midsommarhelgen närmade sig. Vi tänkte ta några dagar ledigt, och skulle kunna göra det med bättre samvete, om vi kunde kalla

ledigheten för "litteraturstudier i hemmet". Nu var vi givetvis beredda att i nödfall tillbringa midsommarhelgen i Nalen, men det kan bli så äckligt med öl i tangentborden.

A.7.1 CSCW i litteraturen

Om CSCW (Computer Supported Cooperative Work, om nu någon missat det) har det skrivits mycket. Och länge. Vid genomläsning av en hel bok med artiklar i ämnet slås man av att de blir bättre ju äldre de är. De enda *riktigt* intressanta artiklarna var just Bush artikel från 1945 som vi tidigare nämnt och Douglas Engelbarts artikel om sitt Augmentprojekt från 1960-talet.

Efter att arbetet med **CoDe** inletts försvann hux flux all tid för litteraturstuidier, så vi hann aldrig fullborda dessa. Men tanken var god.

A.8 Slutsatser om Hypercard

Hypercard är en underbar miljö att utveckla prototyper i — till en viss gräns. Det är synd att programmerarna på Apple inte har lagt ned den där sista tiondelen arbete, det är så lite som saknas för att allt skulle vara perfekt.

Det vi först och främst saknar är möjligheten att *gruppera* objekt. Om femtio knappar ska flyttas en centimeter till höger vill man inte behöva göra detta knapp för knapp. Det är möjligt att denna inskränkning har gjorts av prestandaskäl, men när ett program som PageMaker klarar motsvarande borde det gå även för Hypercard.

Vidare borde det i HyperTalk gå att bilda egna objekt och strukturer. HyperTalk i stort känns ganska påklistrat, och Apple borde erbjuda ett alternativt interface, mer avpassat för programmerares behov.

Till sist tycker vi inte om uppdelningen i separata knapp- och fältmoder. Det borde räcka att som i `devguide` ha en bygg- och en testmod.

A.9 Att hitta rätt Interface Builder

Då förutsättningarna väl var spikade satte vi igång att utforska alla hjälpmedel på Konrads lista. Detta tog några veckor i anspråk, men det var väl investerad tid, och vi såg det som "grundforskning".

De krav vi hade på det verktyg som vi skulle använda för användargränssnittet var att det skulle vara:

- lättanvänt
- lätt att skräddarsy efter våra önskemål
- enkelt att länka in vår egen kod
- estetiskt tilltalande
- väldokumenterat
- grundat på någon slags kortmetafor

där det sista kravet mer var ett önskemål än ett krav. Det är med ett bra användarinterface som med en kvinna — utseendet är viktigare än man kanske vill erkänna, men det *verkligt* viktiga är det som finns under. Med detta i åtanke började vi beta av listan.

A.9.1 HyperNeWS

HyperNeWS verkade vara det mest lovande alternativet, i alla fall innan vi provkört. Så här efteråt är det svårt att sätta fingret på vad som var fel, men vi har enats om att det var kravet på estetik som satte stopp för HyperNeWS kandidatur. Programmet är helt enkelt *fult*. Turinginstitutet som utvecklat systemet borde offra lite pengar på att anställa en grafisk designer till nästa version. Färgerna är gräll a (aprikos mot laxrött) och fönstren "klumpiga". Dessutom tyckte vi inte om dominansen av PostScript som programmeringsspråk.

Fördelarna var kortmetaforen och det faktum att all dokumentation låg online.

A.9.2 Devguide

Efter bakslaget med HyperNeWS vände vi oss mot `devguide` (där `guide` står för **G**raphical **U**ser **I**nterface **D**evelopment **E**nvironment), som är SUN's eget verktyg för att utveckla applikationer under OpenWindows.

Det blev kärlek vid första ögonkastet, allt kändes rätt, det var så här vi ville jobba. Som vid alla förälskelser blev vi kanske blinda för vår tillkommandes brister och ofullkomligheter. Det fanns ingen dokumentation. Ingen alls. Vi var uppe hos Patrik Fältström på systemgruppen och letade igenom hans bokhyllor två gånger. Båda gångerna kom vi lyckliga därifrån med famnarna fyllda av pärmar som vid närmare granskning visade sig vara helt ovidkommande.

Det vi fäste oss vid hos `devguide` var samma sak som fällde HyperNeWS. Utseendet. En sorts tredimensionell känsla i olika gränivåer måste upplevas, det går inte att beskriva. En guideapplikation blir helt enkelt så vacker att man som programutvecklare blir helt mållös — utan att man behöver trolla med knäna.

När vi väl bestämt oss för `devguide` höll vi fast vid detta beslut. En mer detaljerad beskrivning av `devguide` finns i avsnitt 4.

A.10 Arbetet med CoDe

Direkt efter midsommar startade arbetet med att försöka realisera våra högtflygande planer. Vårt första steg var att bekanta oss med den nya miljön.

Vi hade fått flytta ner från bastun, ner i själva Nalen där två stycken SUN IPC Sparcar[26] hade flyttats upp från Rosa sal bara för vår skull. Större delen av den plattform vi skulle stå på, UNIX, C, X Windows, Open Windows och ISIS var redan bestämd.

[26]Sparkar? För en norrlänning väcker det ljuva vinterminnen till liv...isiga backar...barnfötter i Graningekängor storlek 36 osäkert balanserande på de smala medarna...träsitsen med slitna grönmålade handtag som gnekade i alla svängar...och alla bilar var ofarliga hinder.

A.10.1 Förstudier

"C är som ett andra modersmål"

Vi har under sommaren läst ett antal exjobbsrapporter, och nästan alla skriver att de har inlett arbetet med att de första två, tre veckorna lära sig C och UNIX. Detta steg behövde aldrig vi ta, C är som ett andra modersmål och våra kunskaper i UNIX var tillräckliga för ett arbete på denna nivå. Däremot önskar vi att vi haft bättre förkunskaper om nätverk och protokoll. Tidvis då förkortningarna haglade som värst (TCP/IP! FDDI! ST-2!) kände vi oss *väldigt* bortkomna, även om vi var noga med att inte *visa* det.

Just våra kunskaper om UNIX kan jämföras med de kunskaper vi har om bilar; Vi kan gasa, växla och styra och vi kommer dit vi vill, men när vi ska titta under huven blir vi förvirrade. Vi kan tanka, byta tändstift och fylla på olja, men så fort något går *riktigt* snett måste vi tillkalla experthjälp. I dessa lägen brukade vi gå till systemgruppen och gråta ut. Varje gång gick vi därifrån lite klokare och med några luckor i vårt kunnande tilltäppta[27].

A.10.2 Programmeringsverktyg

Eftersom vi redan bestämt os för att använda `devguide` som Interface Builder, blev vi tvungna att ägna en del tid till att lära oss behärska detta verktyg.

Även `devguide` utgjorde trots sin enkelhet problem för oss i början. Det fanns nämligen inga som helst manualer, `man`blad eller liknande. Till och med on-line-hjälpen var borta, så vi var helt hjälplösa. Vår enda chans var att prova oss fram med trial-and-error. De exempel som fanns till hands var också mycket få och knapphändiga, men vi studerade dessa och började göra egna experiment. Stora milstolpar på vägen var första gången vi lyckades knyta en meny till en knapp och det första strecket vi lyckades rita på en *canvas*.

"Vår enda chans var att prova oss fram med trial-and-error"

Just hur vi skulle kunna använda grafik var ett stort frågetecken. Vi hade gjort ett första (av många) besök hos systemgruppen och lånat några böcker om XView. Här var allt utförligt beskrivet, sida upp och ned med funktioner och attribut för knappar, fönster, listor och menyer. Det enda XView verkade tillhandahålla för grafik var just en *canvas*, en duk att rita på. Till slut provade vi att använda vanliga `Xlib`:s ritkommandon (`XDrawLine` m.fl.), och det fungerade! Så här i efterhand verkar detta inte alltför otroligt, men vid tidpunkten kändes det som en stor seger.

Vid det första besöket hos systemgruppen fick vi också ett eget projektkonto, där vi fick mer utrymme för våra filer än på våra vanliga hemdirectoryn. Peter visade sig här vara ökänd under sitt användarnamn (`d87-pli`) för att vara den D-teknolog som hade absolut mest på sitt vanliga användarkonto, 16MB mot tillåtna 4MB. När vi fick vårt projectkonto råkade vi även mynta akronymen ***CoDe*** (för The Collaborative Desktop, om nu någon missat det).

Vi lärde oss snabbt hur sårbara vi var. I och med att vi använde så många olika programpaket hade vi brett ut oss över filsystemet (hembibliotek på `Hemul`, projektfiler på `Dront`, ISIS och `devguide` på `Alv`), så att så fort *någon* av datorerna gick ned kunde vi inte arbeta.

"Så fort någon av datorerna gick ned kunde vi inte arbeta"

Efter att ha bekantat oss med XView och `devguide` under några dagar gick vi över till att titta på ISIS. Här fanns en hel pärm med dokumentation som vi

[27] Ragnar Andersson försökte en gång trösta oss med ett citat: *UNIX is the New Jersey of operating systems — It was once a beautiful place, now it's a dump.*

även hittade på fil, så vi skrev ut 400 sidor på laserskrivaren för att få varsitt exemplar. Tyvärr verkade manualen ursprungligen vara skriven för en tidigare version av ISIS och inte hundraprocentigt uppdaterad till den nuvarande, för vi hittade många sakfel, funktioner som inte fanns, parametrar som försvann, etc. Stor nytta hade vi av ett antal små exempelprogram.

A.10.3 XView och ISIS — Att ena och förena

Det tog ett tag innan vi lyckades para ihop XView med ISIS. Båda systemen vill "ta kontrollen" genom sina repektive huvudloopar (`isis_mainloop()` repektive `xv_main_loop()` och därifrån själv anropa *våra* rutiner. Då det endast går att ha *en* huvudloop var vi tvungna att lösa det på annat sätt.

Den första tanken var att XView borde innehålla någon form av "idlefunktion", som den anropar när den inte har annat för sig. Vi sökte igenom den dokumentation vi hade utan att hitta någon indikation på att en sådan funktion skulle existera. Därför löste vi det den hårda vägen (och här menar vi verkligen *hårda*). Vi lät XView ta kontrollen, men installerade en UNIX-timer som genererade en `SIGNAL` med jämna intervall. Denna fick anropa ISIS stötvis, med funktionen `isis_accept_events(ISIS_ASYNCH)`. Här fick vi dock vara noggranna så att UNIX-signalerna inte kom så tätt att det tidigare ISIS-anropet inte hunnit exekvera klart, då genererade ISIS ett runtimefel med en spydig anmärkning om att den inte ville bli rekursivt anropad, tack. Detta ledde till att vi var tvungna att göra alla ISIS-anrop "ömsesidigt uteslutande", så att endast *ett* sådant anrop tilläts samtidigt. Lösningen var inte bra och *definitivt* inte vacker, men den fungerade. Arbetet fortsatte[28].

"Lösningen var inte bra och definitivt inte vacker, men den fungerade"

Efter ett tag när vi fått mer dokumentation om XView [Hell90] hittade vi ett avsnitt om hur `xv_main_loop` kunde skrivas om, så att vi där kunde stoppa in vårt ISIS-anrop utan att behöva gå omvägen via signaler. Här gjorde vi vår egen huvudloop, `xv_idle_loop`, där kärnan ser ut som:

```
while (1) {
  for (j = 0; j < 50; j++)
    notify_dispatch();
  XFlush(dpy);
```

[28]En dag tog Magnus med sig Dicken, sin hund. Dicken hade en ledande roll i en av våra tidigare rapporter [Lidb91], och han sprang runt i Nalen hela dagen och viftade på svansen. Caroline (Nordquist, NADA) fick se honom, och brast glatt ut i ett "*Nu* går det upp ett ljus, var det *ni* som skrev den där rapporten i Översättarteknik, åt Hasse?". Vi kunde bara bekräfta detta. Hon uttryckte viss bitterhet att vi i denna avslöjade mördaren i *Twin Peaks*. Vi kanske måste revidera vår uppfattning att inga läser våra rapporter.

Carin (Sjöholm, NADA) hade inte hört talas om Dicken tidigare, men verkade mycket förtjust. Däremot hade hon träffat *oss* förut. Detta skedde under vårt projekt i PIM året innan. Vi gjorde ett program för att styra en CD-ROM skiva med tjugofem olika ordböcker på tolv olika språk. Det fanns ett fungerande program på en IBM PC, och vår uppgift var att göra ett motsvarande program till Macintosh.

En vinterdag gick vi upp i Nalen för att titta på det fungerande programmet. Vi började genast slå upp våra vanligaste fula ord (och vi vet alla vilka vi menar) och översätta dessa till de elva andra språken. Efter hand som vi kom vi på nya ord som vi ville ha översatta skrek vi dessa till den av oss som satt vid tangentbordet. När vi gick därifrån märkte vi att vi inte varit så ensamma som vi trodde, att Carin hela tiden suttit bakom en skärm och arbetat. Hon gav oss ett glatt leende.

Vi kunde bara hoppas att hon glömt bort oss.

```
    timer_idle();
}
```

I denna (busy-wait-)loop betar `notify_dispatch()` av XViewevents i den mån det finns några, medan `timer_idle()` i sin tur låter ISIS köra en bit. Med denna rutin fungerade allt exemplariskt, i alla fall då inga prestandakrav fanns. Vi kunde ta bort all kod för att garantera ömsesidig uteslutning, då detta problem inte längre kunde uppkomma. Senare ändrade vi av prestandaskäl även denna rutin, mer om detta senare.

A.10.4 Arbetet inleds, allvaret börjar

Efter den första inledande grundforskningen började det bli dags att få lite arbete utfört. Vi delade upp oss så att Peter fortsatte forska med inriktning på ljud och med mål att utveckla en speciell ljudserver som skulle erbjuda övriga processer de ljudtjänster de behöver. Magnus skulle under tiden fortsätta att utforska grafikmöjligheterna i XView och försöka göra en distribuerad riteditor, ett första utkast till vår `WhiteBoard`.

Peter inledde arbetet med ljudprocessen med att hitta så mycket information som möjligt om Sparcens ljudanordning, denna stod att finna i `man`blad och i dokumentationen till SUN OS. På filsystemet under `/usr/demo/SOUND` ligger både en mängd färdiga ljudfiler som det bara är att spela upp, till exempel genom att göra

```
cat *.au > /dev/audio
```

samt en applikation för att spela in egna ljud, `x_soundtool`. Dessutom fanns två för Peter mycket viktiga filer, `play.c` och `record.c`. Dessa filer innehåller källkoden till två exempelprogram, ett som spelar in ljud till fil och en som spelar upp samma filer igen. Ni får själva lista ut vilken som gör vad. Peter satte genast igång att dissekera dessa för att göra ett program som läser från mikrofonen och skriver till högtalaren direkt utan att gå omvägen via filer. Detta gick efter lite filande, så nästa steg var att skriva ett enkelt PhoneTalkprogram som läser från mikrofonen på en dator och skriver ut på högtalaren på en annan.

Peter lade en hel del överskottsenergi på att tala in små verbala förolämpningar på fil och försöka skicka över dessa till kamrater som satt i andra datorsalar och arbetade.

För att skyffla ljuddata mellan datorerna använde Peter det enda sätt han kunde — ISIS. Innan han tog steget ut till ett riktigt PhoneTalk använde han en rutin som spelade upp en ljudfil om och om igen och försökte spela upp detta på olika datorer. Detta gjordes med hjälp av ett gammalt exempelprogram för ISIS som gjordes om till att överföra ljud istället. Efter att ha fått *det* att fungera togs steget fullt ut och ett mycket enkelt PhoneTalkprogram utvecklades.

A.10.5 The American Way of Life

När det började bli allmänt känt att vi förde över ljud med ISIS blev vi plötsligt uppmärksammde från en rad håll, både från SICS och från Amerika, där vi fick brev från Olof Hagsand på SICS, som vid tillfället arbetade på Cornell University, där ISIS utvecklats. Vi kommunicerade några varv fram och tillbaka

över Atlanten och fick en del goda tips hur vi *egentligen* borde ha kombinerat ISIS med XView. Tyvärr råkade vi uttrycka oss lite slarvigt i ett av breven och stötte oss med SICS, men det klarade vi snabbt ut. Problemet med `e-mail` är att en del ironi försvinner med den smala bandbredden. Om man som vi har varsin giftig tunga och ännu giftigare pennor[29] bör man vara försiktig med vad man skriver. Vi är vana med att prata med folk som känner oss och vet att vi inte ska tas på allvar.

"Vi är vana med att prata med folk som känner oss och vet att vi inte ska tas på allvar"

När vi väl sett att grundkonceptet höll började Peter på den generella ljudprocessen. Den struktur denna borde ha hade klarnat fram under arbetet på PhoneTalkprogrammet, och den första versionen av `sound`, som den nya processen finurligt nog kom att heta var snart klar. Peter utvecklade också en XView-applikation, `control` som skulle användas enbart under utvecklingsarbetet för att styra `sound`.

Arbetet plågades av en viss långsamhet, samt av en del oförklarliga incidenter. Peter fick två dagar förstörda på grund av en bug i den äldre C-kompilatorn `cc`. Denna klarade inte av att expandera macron på rätt sätt, trots att C-standarden föreskriver att macron inte ska expanderas in i strängkonstanter gjorde `cc` just detta. Sålunda expanderades

```
#define test(s) printf("%s", s);
```

vid anropet `test("kalle");` till

```
printf("%"kalle"", "kalle");
```

i stället för till

```
printf("%s", "kalle");
```

`gcc`, den lite "modernare" C-kompilatorn klarade detta med glans, men den kunde å andra sidan inte kompilera ihop XView- och ISIS-kod. `gcc` är *betydligt* kinkigare vad gäller typkontroll, och klagar redan på `#include`-filerna. Gamla goda `cc` kontrollerar nästan inga typer alls och kompilerar koden utan att gnälla.

Peter fortsatte finputsa på ljudet, och tillbringade en hel vecka med att försöka få bort den delay som uppkom ibland. Ett flertal flushar av både ISIS och XView vid strategiska lägen löste problemet tillfredsställande.

A.10.6 Titta Vi Demonstrerar

Medan Peter harvade på med sin ljudmodul gick Magnus från klarhet till klarhet. Han hann utveckla två olika versioner av Call Manager med tillhörande WhiteBoard tillsammans med en tidig primitiv version av Team Map innan Peter var klar med ljudet. Vi beslöt koppla ihop Call Manager med ljudprocessen för att prova om det gick. Vi behövde bara lägga till några få anrop till ljudprocessen i Call Manager och se — allt fungerade perfekt!

Lyckorusiga letade vi reda på Yngve, Hans och Konrad för att få visa vad vi åstadkommit. Peter tog med sig den ena mikrofonen och gick ned i Rosa Sal för att visa hur fint allt fungerade. Vi startade upp systemet, fick ljudkontakt, öppnade WhiteBoarden och började rita. Då hängde sig allt. Ingenting

[29] Eller borde det så här i dagens informationssamhälle kallas giftiga *tangenter*?

fungerade, och vi lyckades inte få igång det igen. Våra handledare verkade inte imponerade.

Efteråt visade det sig att ISIS simultant hängt sig på samtliga sju maskiner där vi hade det igång. Vi vet än idag inte vad det berodde på, men vi misstänker att någon varit inne och ändrat i ISIS *sites*-fil, den förteckning över de datorer som ingår i nätet.

Magnus fortsatte oförtrutet och utvecklade en tredje version av Call Manager, denna gång komplett med vår favoritidé med riktiga användardefinierade ikoner som går att dra runt hur som helst. Att konvertera ikonerna från XViews ikonformat till `XBitmap`format var också en stötesten. Formaten var sinsemellan totalt bak och fram och ett bra tag nöjde vi oss med att låta dem vara spegelvända. Efter att ha fullbordat denna vände Magnus blicken mot Team Catalogue, Catalogue Card och Team Map, medan Peter började arbeta på Answering Machine och Displayer.

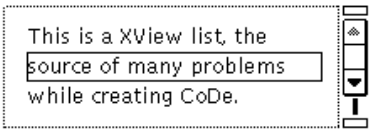


Fig.36: En XViewlista, källa till mycken förtret.

A.10.7 Nya program, nya buggar

Under arbetet med telefonsvararen hittade Peter ännu en bug, denna gång i XView. Det visade sig vara problem att ta bort element i XViews *listor*, ibland dog hela applikationen. Lösningen var att markera listorna som `CHOISE_REQUIRED`, det vill säga att de alltid måste ha ett element valt. Då försvann alla problem.

Arbetet med de kvarvarande applikationerna gick snabbt, nu när vi behärskade alla de verktyg vi använde och visste exakt vad vi ville göra. Likt Carl Hamilton, hjälten i Jan Guillous böcker om Coq Rouge blev vi även med tiden avtrubbade och tvingades acceptera allt mer våldsamma lösningar på våra problem.

A.10.8 Prestationsångest

"När vi började försöka använda våra verktyg drabbade ödet oss igen med full kraft"

Nu kunde vi börja provköra *hela* CoDe. Huvudrutinen `fork`ade igång alla delprogram, och dessa startades precis som de skulle. När vi började försöka *använda* våra verktyg drabbade ödet oss igen med full kraft. Allt gick *extremt* långsamt. Extremt. Det var så illa att det kunde gå tjugo sekunder mellan en knapptryckning och att systemet reagerade på denna. Ljudet styckades upp i småbitar, 75% försvann och resten spottades ut i högtalaren på ett sätt som var allt annat än njutbart.

Vi hade helt enkelt gjort misstaget att inte förutse denna prestandaexplosion. När vi utvecklat våra verktyg och provkört dem körde vi som mest två

åt gången. Nu, när samtliga program var igång blev processorkraven annorlunda. Vi kontrollerade belastningen (med det utmärkta programmet `top`, rekommenderas varmt) och upptäckte att alla processer tog cirka 10% CPU var, oavsett om de körde eller inte. Vi gjorde helg, *mycket* bedrövade. Det var bara en vecka till vi skulle visa upp det färdiga systemet på Electrum. Som programmet betedde sig skulle det *inte* gå att visa upp.

Efter en dag av tårar kom vi ihåg ett av de brev vi fått av Olof Hagsand. Vid det tillfället hade vi varit nöjda med vår egen busy-wait-loop, men som läget var nu var vi beredda att prova vad som helst. Hade inte Olof föreslagit en *alternativ* huvudloop?

Vi tog tunnelbanan in en sen lördagkväll och började leta igenom vår `mbox` efter rätt brev. Och där var det! Olof föreslog att XViews huvudrutin `notify_dispatch()` skulle installeras som en `isis_input`, för att sedan låta ISIS arbeta. Detta skulle ge en möjlighet att hantera alla events direkt när de kommer in. Efter en del experimenterande blev den slutgiltiga versionen av huvudloopen:

```
isis_input(dpy->fd, notify_dispatch, 0);
for (;;) {
  isis_accept_events(ISIS_BLOCK);
  if (!(j++ % 50)) XFlush(dpy);
}
```

Döm om vår förvåning när det fungerade! På första försöket! Vissa nya problem uppkom då en del nya `XFlush`:ar behövde läggas till för att skärmen skulle uppdateras ordentligt, men detta var ett billigt pris. Prestanda förbättrades nämligen drastiskt, det var ett helt nytt program som stirrade tillbaka på oss från skärmen. Med `top` kontrollerade vi CPU-lasten, och denna var nu 0.0% för de processer som inte körde. Jämfört med de cirka 10% per process som vår tidigare huvudloop krävde är det en procentuell förändring på...på...på *jättemycket!* Även ljudet lät njutbart igen. Vi sände en tacksamhetens tanke över Atlanten.

Sålunda mentalt uppiggade utförde vi en sista finputs, inte med inriktning på att allt skulle bli perfekt, utan med tonvikt att det skulle fungera tillfredsställande under demonstrationen.

A.11 CoDe. Arbetet klart — Ur askan i elden

A.11.1 Electrum — tillbaka på brottsplatsen

Så var det åter dags att genomföra en demo på Electrum. Det kändes inte som om två och en halv månad hade förflutit sen förra gången. Nu kunde vi för första gången titta tillbaka på det arbete vi utfört. Vi hittade ett exemplar av det papper vi skrivit ihop och delat ut till den första demon innan midsommar. Till vår förvåning upptäckte vi att vi faktiskt hunnit implementera större delen av de idéer vi hade redan då, så att vi nu skulle kunna visa ett *fungerande* system som utförde i stort sett samma saker som vi bara antytt i vår första demo. Undantaget är att vi rensat bort många av de mest högtflygande planerna ur prototypen — främst möjligheten att gå in i varandras desktopar. I övrigt är överenstämmelsen mellan hypercardprototypen och vårt färdiga system bitvis nästan skrämmande.

"Nu kunde vi för första gången titta tillbaka på det arbete vi utfört"

Vi hade gjort klart de sista bitarna sent på tisdag kväll, och hela onsdag förmiddag hade vi avsatt till att få allt att fungera ute i Kista. Vi hade fått två rum uppe på SICS, med varsin Sparc IPC av samma typ som vi använt på KTH. Mikrofoner var vi däremot tvungna att ta med själva. SICS' systemansvarige Per Danielsson hjälpte oss att flytta över **CoDe** från KTH och installera det.

Den stora kalldushen kom när vi inte lyckades få igång ISIS. Att starta igång ISIS på en ny dator är lite krångligt, maskinen måste läggas till i en speciell tabell, vissa portar måste reserveras, m.m. Vi gick tillväga på samma sätt som vi gjort på KTH två månader tidigare, men ingenting ville fungera. Till slut fick vi hjälp, och plötsligt fungerade allting.

Lite problem uppkom också när vi märkte att vi hårdkodat in vissa filnamn (bland annat tutsignalen som spelas upp när någon ringer), och dessa filer inte fanns på SICS filsystem. Problemen förstärktes av att vi inte kunde kompilera om på SICS, eftersom vissa programpaket och bibliotek inte fanns där. Därför var vi tvungna att göra `rlogin` mot KTH:s datorer, kompilera *där* och efter det skicka upp den nya filen till Kista. Detta omständiga sätt att arbeta tog tid, och vi blev klara bara någon timme innan demonstrationerna skulle börja klockan sex. Vi hann precis klämma i oss vår andel av buffén (en tunnbrödsklämma, en lättöl, en sliskig bakelse och åtskilliga koppar kaffe) innan det var dags.

"I vanliga fall brukar allting strula och misslyckas när vi ska visa något för någon utomstående"

Demonstrationerna förflöt helt problemfritt under de två timmar vi höll på. I vanliga fall brukar allting strula och misslyckas när vi ska visa något för någon utomstående. Vi satt i två separata rum och pratade med varandra över datornätet, ritade små figurer på den gemensamma WhiteBoarden, skickade meddelanden mellan varandra och talade in egna telefonsvararmeddelanden. Allt gick som smort. Vissa åskådare verkade uppriktigt intresserade, andra lätt blasé. Ingen erbjöd oss arbete. Magnus tog med sig mikrofonerna hem för att ta med dem till skolan dagen efter.

A.11.2 Framtiden — finns den?

Dagen efter vår demonstration var det stort *MultiG*-möte på Stora Skuggans Värdshus. På förmiddagen presenterades alla olika delprojekt översiktligt och på eftermiddagen delades deltagarna upp i smågrupper. En av dessa var **CoDe**.

Vi hade stämt möte på skolan en timme före utsatt tid för att åka ut till Stora Skuggan i samlad trupp. Magnus kom med andan i halsen och förklarade att han tappat bort mikrofonerna. Antingen kom de aldrig med hemifrån eller också blev de kvar på tunnelbanan.

Magnus åkte tillbaka för att leta och lämnade Peter att på egen hand leta sig ut till mötesplatsen. Den närmaste timmen blev mörk. Vad *kostar* ett par mikrofoner? Kommer det att dras från lönen? Viktigare, kommer det att dras från *Peters* lön?

Magnus anlände till Kista station exakt i rätt ögonblick för att se en spärrvakt med väskan med mikrofonerna i handen på väg att kliva in i ett tåg, förmodligen på väg till Rådmansgatan och hittegodsavdelningen. Magnus förklarade att väskan var hans och fick efter en beskrivning av innehållet även tillbaka denna[30].

[30] Mardrömsscenariot hade varit att vid ankomsten finna hela stationen och halva Kista Centrum avspärrat. Poliser i full kravallmundering, skottsäkra västar och bombhundar håller nyfikna borta medan en radiostyrd robot rullar fram och skjuter skott efter skott med ett

Efter detta letade han sig tillbaka ut till Stora Skuggan och anlände en och en halv timme efter utsatt tid, vilket inte gjorde så mycket eftersom mötet då ännu inte kommit igång.

Under (gratis-)lunchen belastade Peter NADA:s ekonomi genom att *beställa* halstrad gravlax för 88 kronor, men vid disken av misstag (i ett anfall av hunger-hallucinationer) ta med sig en dagens rätt, stekt forell, som bara kostade 55 kronor. Han insåg inte sitt misstag förrän han hunnit rensa forellen och äta en fjärdedel. När källarmästaren kom ut med den halstrade gravlaxen gömde sig Peter bakom en blomvas, och källarmästaren gick flera varv runt matsalen innan han retirerade in i köket.

Eftermiddagens diskussion om **CoDe** blev en spännande tillställning. Vår grupp blev den som samlade överlägset flest intresserade och det var runt tjugo personer i rummet. Först föste Peter fram Magnus, som helt oförberedd fick förklara **CoDe**:s inre uppbyggnad och struktur för en förstummad publik. Sedan bröt den allmäna diskussionen lös. Diskussionsklimatet var delvis förbluffande för den ovane. Allt liknade en Tom-och-Jerry-film, där alla pucklade på varandra med full kraft, även om det alltid verkade finnas *någon* sorts hjärta i botten. Alla var i alla fall väldigt snälla mot *oss*, vilket var tur eftersom vi inte är så starka rent psykiskt.

"Allt liknade en Tom-och-Jerry-film, där alla pucklade på varandra med full kraft"

Ur vår synvinkel var det också underligt att höra andra människor diskutera *vår* **CoDe** och svänga sig med just termen **CoDe** lika naturligt som med UNIX, SPARC och andra datortermer. Det kändes lite som att ha satt ett barn till världen. För att ytterligare öka på *den* känslan träffade vi också de tre nya exjobbare som skulle ta vid där vi slutat och föra **CoDe** vidare. Just deras framtida uppgifter var föremål för mycket diskussion, även om denna kanske inte kom fram till något konkret resultat.

Bland de mer fantasieggande tankarna var Lennart Fahléns (SICS) förslag att inkludera Telepresence, den virtuella värld som utvecklats på SICS i **CoDe** *och vice versa*. På så vis skulle vi kunna erbjuda deltagarna i ett samtal möjlighet att dela en virtuell värld på samma sätt som de idag kan dela en WhiteBoard. Från andra hållet kan **CoDe** inkluderas i den virtuella världen så att den som promenerar omkring där plötsligt kan träffa på **CoDe**. Denna kan därifrån användas till att ringa upp andra, och i den kan man starta upp en *ny* virtuell värld, där man kan hitta en **CoDe**, där man kan starta *ännu* en virtuell värld, och så vidare i all oändlighet. Tanken svindlar.

A.11.3 Filmstjärnor igen

Direkt efter att vi diskuterat färdigt åkte vi tillbaka till KTH för att spela in vår andra video (kallas på branschspråk för en *uppföljare*). Den här gången behövde vi inte göra allt själva, utan vi hade professionell hjälp av Bengt Ahlström (NADA, f.d. FOA) som gjort ett flertal videor förut. Videon var avsedd att visas vid CSCW-konferensen i Amsterdam. Ursprungligen var tanken att vi skulle följa med och demonstrera allt "live", men demonstrationerna ströks från programet, så videobandet blev det bästa (och billigaste) alternativet .

grovkalibrigt hagelgevär genom väskan. Magnus funderar om det finns något som kan binda den vid honom.

Först skulle Hans Marmolin visa *sin* SuperCardprototyp över vissa verktyg som han utvecklat själv parallellt med vårt arbete, sedan skulle vi visa vår bit. För att förbereda oss på vårt framträdande satte vi oss ned och skrev ett kort scenario, komplett med manus och allt. Detta tog någon timme.

I väntan på att det skulle bli vår tur framför kameran satt vi inne i fikarummet och drack kaffe, kopp efter kopp. För att hålla Magnus vaken berättade Peter en mängd episoder från sin barndom, hur han uppfostrats av lejon[31], hur han i ungdomen blev nära personlig vän till Michail Gorbatjov och hur han blev mytoman.

Klockan tre på natten blev det vår tur. Vi övade några gånger utan att spela in för att hitta rätt kameravinklar och nöta in replikerna. Sedan körde vi. När vi väl kom igång gick allt fort, inte så mycket för att allt flöt fint, utan för att tolerensnivån på vad man accepterar sjunker när klockan passerar fyra på morgonen. Halv fem var vi klara, och det var fortfarande en halvtimme kvar tills tunnelbanan skulle börja gå. Sakta gick vi genom stan ner mot T-Centralen — den konstruktiva delen av vårt examensarbete var över.

"Tolerensnivån på vad man accepterar sjunker när klockan passerar fyra på morgonen"

A.12 Att skriva en rapport

Rapportskrivningen har vi skjutit upp till sist, inte för att vi gruvat oss, utan för att det är svårt att skriva om ett program medan det fortfarande är i förändring. Det är lätt hänt att det som är sant för tillfället kan ha förändrats helt till nästa vecka. Å andra sidan är det svårt att komma ihåg precis vad man gjorde för tre månader sedan, så att föra någon typ av dagbok kan vara ett bra tips.

Själva skrivandet *brukar* aldrig utgöra något problem, vi tycker båda om att skriva. Ibland har det gått så långt att vi sett själva arbetet mest som ett nödvändigt ont för att samla stoff till rapporten. Den här gången har vi emellertid bitvis drabbats av skrivkramp. Detta är med största sannolikhet en bieffekt av det vi berörde redan i förordet — stundens allvar.

I vanliga fall brukar vi vara helt ohämmade och skriva precis vad som faller oss in utan hänsyn till smak och god ton, den här gången har vi försökt tygla oss. Eventuella blivande arbetsgivare ska kunna läsa det här utan att rodna. Vi vill dock inte göra något alltför strömlinjeformat och utslätat heller — bra exjobbsrapporter finns det gott om, rapporter som är både bra och underhållande står kanske ut från mängden.

A.12.1 Författande — teknik och praktik

Rent arbetstekniskt går vi tillväga med en metod vi kallar *itererat kaos*, som vi tror att vi är ensamma om. Först gör vi en grov disposition och delar upp

[31] Den historien kan vara värd att återberätta även här. Som ung lämades Peter att dö långt ute på savannen, men en lejonhona som just förlorat sin unge adopterade honom och uppfostrade honom som sin egen. Peter minns med tårar i ögonen hur Elsa (hans lejonmamma) lärde honom jaga buffel. Taktiken var att springa ikapp buffeln bakifrån, bita sig fast i dennes könsorgan och hänga där tills buffeln segnade ihop och dog av blodförlust. Peter var inte *helt* övertygad om att detta var *hans* sätt att jaga, men Elsa buffade på honom med nosen tills han gav med sig. Att springa ifatt buffeln var svårt. Att hänga sig fast var ännu värre. När det var dags att *bita till* veknade dock Peters hjärta och han blev bara hängande där dinglande fram och tillbaka när buffeln sprang iväg. Inte förrän de nådde civilisationens yttersta rand släppte Peter taget. Han såg aldrig Elsa mer.

de olika rubrikerna mellan oss. Sedan åker vi hem var och en till sitt och skriver på våra respektive bitar några dagar. Sedan exporterar vi våra Word-filer till ASCII-format och skickar över dem till KTH:s datanät via modem. Väl där lägger vi till formateringskommandon (LaTeX), TeXar allt och tar ut varsin papperskopia som vi studerar noggrannt. I denna första version inleder Magnus raskt med att stryka större delen av Peters sexuella anspelningar (de han förstår). Peter kontrar med att lägga till nya, ofta inne i Magnus del av texten, där han aldrig hittar dem.

Vid nogrannare genomläsning märker vi alltid att vi glömt stora delar och att vi båda två har skrivit om andra bitar. Dessa fel rättas till, och vi har båda en hel del synpunkter på varandras alster. En andra version arbetas så fram, och så håller det på några gånger tills rapporten börjar nå en angenäm form.

Från början krigade vi även om i vilken *miljö* vi skulle typsätta våra alster. Magnus förordade Macintosh och PageMaker, medan Peter på ett närmast fanatiskt sätt framhöll TeX:s fördelar. De senaste åren har dock Magnus insett sig besegrad, kanske därför att det gjort att Peter ensam fått ta på sig hästjobbet att formatera texten.

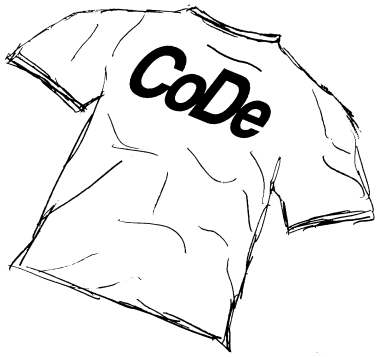
A.13 Vad vi lärt oss och hur vi blivit bättre människor

I detta avsnitt ska vi försöka summera de slutsatser vi dragit utöver de rent datatekniska, med inriktning på kommunikationen människa - människa och även ge några allmäna råd.

- Försök att i möjligaste mån undvika någon som helst form av arbete under sommarmånaderna. Dessa bör ägnas åt sol och bad, inte till att sitta inomhus.
- Var väldigt försiktig med vad ni skriver i `e-mail`. Om ISIS kan sägas vara ett bra verktyg om man vill hänga så många datorer som möjligt simultant, så är `e-mail` bra om man vill bli missförstådd av så många som möjligt.
- Tänk på vilodagen så att ni helgar den. Ingen människa mår bra av att arbeta 7 dagar i veckan, i alla fall inte under någon längre tid.
- Ett gott råd till alla som ska skriva rapporter är att alltid se till att göra det under stark tidspress — om man har gott om tid på sig tenderar man ändå att skjuta upp arbetet till de sista dagarna, så sätt tidsramen snävt till att börja med så får ni mer gjort.

You've read the report...
You've seen the video...
Now...BUY THE T-SHIRT!

Detta speciella meddelande riktar sig till alla **CoDe**-användare jorden över — nu har NI chansen att visa er särart genom att beställa vår speciella **CoDe**-T-shirt. Denna exklusiva tröja i 100% bomull har trycket "**CoDe**" i relief på framsidan och devisen *Framåt i tapperhet mot den ärorika döden* i blodrött på ryggen. Priset för denna unika tröja är endast 345:- plus moms och porto — ett tillfälle som aldrig återkommer!. En krona per såld tröja går till Föreningen för Utbrända Programmerares omskolningsverksamhet i Bollmora. Ni beställer den (eller något av våra andra exklusiva erbjudanden) enklast genom att fylla i tabellen nedan och lägga er beställning på brevlådan.



Våra andra specialerbjudanden är:

- **CoDe Avenger**. Spännande fullmatat actionspel för alla åldrar. Följ med Peter och Magnus på äventyr bland drakar och demoner och meja ner de onskefulla hindren på deras väg mot examen med strålgevär och laserbomber. Finns för Atari, Amiga och Nintendo.
- **CoDe-Man**. He-Man-liknande docka av Peter i hårdplast. Mycket naturtrogen. Två hastigheter. Lätt att rengöra.
- **CoDeomer**. Exklusiva presentkondomer i specialdesignad förpackning med texten **CoDe** *users do it together* på baksidan. Endast 90-pack. Den perfekta julklappen?

Ja tack! Jag accepeterar ert generösa erbjudande och beställer:

__ st. **CoDe T-shirt** à 345 SEK
__ st. **CoDe Avenger** à 299 SEK
__ st. **CoDe-Man** à 99 SEK
__ st. **CoDeomer** à 149 SEK

CoDe Trading Inc.
IPLab, Nada, KTH
Osquars Backe 27
100 44 STOCKHOLM