

# Verktyg för grafiska användargränssnitt

---

RAPPORT NR 20

PER BERGSTEN  
MARIE BERN  
PEETER KOOL  
ULF WINGSTEDT

SVENSKA INSTITUTET FÖR SYSTEMUTVECKLING

**SISU**

# INNEHÅLL

---

Förord .....	1
1. Sammanfattning .....	3
2. Introduktion .....	5
3. CASE:W .....	11
4. Craftman .....	19
5. DevGuide .....	29
6. Easel .....	41
7. ezX .....	49
8. HyperCard .....	59
9. Interface Builder .....	67
10. Open Interface .....	75
11. Plus .....	81
12. TeleUse .....	89
13. Visual Basic .....	101
14. VUIT .....	111
15. XVT .....	123
16. Verktögsleverantörer .....	129





# FÖRORD

---

**Syfte:** Denna rapport ger en inblick i några av de olika typer av verktyg för konstruktion av grafiska användargränssnitt som finns på marknaden. Vi gör en detaljerad genomgång av ett antal typiska sådana verktyg, i syfte att belysa hur programutveckling stöds av verktygen.

Denna rapport är den andra i en serie rapporter som behandlar standarder och verktyg för grafiska gränssnitt. Den första rapporten i serien, SISU Rapport nr 19, *Standarder för grafiska användargränssnitt*, ger en beskrivning och jämförelse av olika förslag till standarder, samt förklarar vanliga begrepp i gränssnittssammanhang.

**Målgrupp:** Rapporten är teknisk till sin karaktär och riktar sig främst till personer som ansvarar för utvärdering och val av verktyg för systemutvecklingsprojekt där program med interaktiva grafiska gränssnitt utvecklas.

**Läsanvisning:** Rapporten inleds med en kort sammanfattning. Därefter följer en introduktion som översiktligt beskriver en möjlig klassificering av verktyg samt de olika klassernas typiska egenskaper. Denna översikt innehåller också några tabeller som bland annat visar hur de beskrivna verktygen kan grupperas enligt klassificeringen.

Vi har provat verktygen mot en gemensam fallstudie där två prototyper utvecklats. Fallstudien beskrivs sist i introduktionskapitlet.

Därefter följer beskrivningar av 13 olika verktyg där varje kapitel kan läsas fristående.

Efter verktygsbeskrivningarna kommer kapitlet Verktygsleverantörer som innehåller kortfattad information om ett stort antal verktyg, bland annat namn på och adress till tillverkaren.

Observera att rapporten inte ger en totalbild av marknaden utan endast beskriver verktyg som utgör representativa exempel. Att ett verktyg förekommer i denna rapport innebär inte heller att SISU rekommenderar detta verktyg.

De beskrivna verktygen har studerats från våren 1992 till vintern 1992. SISU reserverar sig för att nya versioner av programmen kan ha släppts sedan dess.





# 1. Sammanfattning

---

I rapporten har 13 verktyg för konstruktion av grafiska gränssnitt av olika typ provats och beskrivits. Verktygen har klassificerats enligt följande klasser:

- Layout-redigerare
- UIMS, User Interface Management Systems
- Flerplattformswerktyg
- Totalverktyg
- Lekmannaverktyg

*Layout-redigerare* är förmodligen den vanligaste verktygstypen på marknaden. Speciellt kännetecknande för dessa verktyg är att de framför allt används för att konstruera de statiska delarna av gränssnittet, d v s dess grafiska utseende.

*UIMS*, User Interface Management Systems, eller system för hantering av gränssnitt, som de kan kallas på svenska, karaktäriseras av att det program som skapas delas upp i en gränssnittsdel och en programdel. Gränssnittsdelens har direkt att göra med gränssnittets utseende och beteende. Programdelen innehåller olika beräkningsalgoritmer och databasrutiner.

*Flerplattformswerktyg* gör det möjligt att flytta ett program från en hård- och mjukvaruplattform till en annan genom att göra gränssnittskoden flyttbar. Dessa verktyg ser också till att programmets gränssnitt översätts till den standard som gäller den nya plattformen.

*Totalverktygen* kännetecknas av att de kan användas för att utveckla kompletta program. I teorin behövs alltså inga ytterligare verktyg. Både användargränssnitt och programkod kan utvecklas med de inbyggda funktionerna.

*Lekmannaverktygen* är tänkta först och främst för personer som behöver utveckla mindre program utan att för den skull vara erfarna programmerare.

**gränssnittets utseende**

**uppdelning av  
gränssnitts- och  
programdel**

**flyttbar gränssnittskod**

**kompletta program**

**mindre program**



I rapporten beskrivs följande verktyg:

- CASE:W
- Craftman
- DevGuide
- Easel
- ezX
- HyperCard
- Interface Builder
- Open Interface
- Plus
- TeleUse
- Visual Basic
- VUIT
- XVT

# 2. Introduktion

---

För rapporten har 13 verktyg för konstruktion av grafiska gränssnitt av olika typ provats och beskrivits. Verktygen har klassificerats enligt en klassificering som beskrivs kort nedan.

Verktygen har provats mot två gemensamma utvecklingsuppgifter. Dessa beskrivs sist i detta kapitel.

## 2.1 Klassificering

När vi i denna rapport talar om verktyg för grafiska gränssnitt menar vi endast verktyg som speciellt är avsedda för att stöda gränssnittskonstruktion. Idag innehåller som bekant många andra typer av verktyg stöd för gränssnitt, t ex 4GL, kalkylprogram, CASE o s v. Dessa tas dock inte upp i denna rapport.

Verktyg för grafiska gränssnitt kan delas in i följande grupper:

- Layout-redigerare
- User Interface Management Systems, UIMS
- Flerplattformswerktyg
- Totalverktyg
- Lekmannaverktyg

*Layout-redigerare* är förmodligen den vanligaste verktygstypen på marknaden. Speciellt kännetecknande för dessa verktyg är att de framförallt används för att konstruera de statiska delarna av gränssnittet, d v s dess grafiska utseende.

En layout-redigerare kan liknas vid ett ritprogram som består av en palett med färdiga interaktionsobjekt. Från paletten kan utvecklaren välja objekt med hjälp av direktstyrning och placera dem på ritbordet där gränssnittets layout byggs upp. Interaktionsobjekten i gränssnittet kopplas sedan till programkod som är utvecklad i konventionella programmeringsspråk genom så kallade Callback-rutiner. Från den layout som byggs upp kan verktyget skapa kod som vid körning av programmet ritas upp på gränssnittet på skärmen.

Verktyg av denna typ kan ses som användarvänliga åtkomstgränssnitt till interaktionsobjekt som finns realiserade i de kodbibliotek (toolkits) som verktyget stöder.

**gränssnittets  
utseende**

**interaktionsobjekt**



## **kodbibliotek**

Den största nackdelen med layout-redigerarna är att de kräver att utvecklaren behärskar det underliggande kodbiblioteket i detalj. Stora delar av gränssnittet måste fortfarande utvecklas direkt i något programmeringsspråk utan stöd av layout-redigeraren. Kod för utmatning av värden samt visualisering av data som är specifika för programmet (t ex databasvärden) måste till exempel uttryckligen kodas.

## **uppdelning av gränssnitts- och programdel**

*User Interface Management Systems, UIMS*, eller system för hantering av gränssnitt, som de kan kallas på svenska, karaktäriseras av att det skapade programmet blir uppdelat i en gränssnittsdel och en programdel. Gränssnittsdelens är den del som direkt har att göra med gränssnittets utseende och beteende. Programdelen innehåller t ex olika beräkningsalgoritmer och databasrutiner.

Denna typ av verktyg stöder utvecklingen av såväl de statiska som de dynamiska delarna av gränssnittet, d v s både utseendet och de delar av programmets beteende som återfinns i själva gränssnittet. Ett UIMS-verktyg hanterar även kopplingen mellan gränssnittet och programkoden under körning av ett program.

I bästa fall ger utveckling med hjälp av ett UIMS-verktyg ett program där gränssnittet är oberoende av programkoden och där det är enkelt att förändra användargränssnittet utan att programkoden påverkas. I praktiken kan det dock vara svårt att särskilja de olika kodkomponenterna.

## **flyttbar gränssnittskod**

*Flerplattformsverktyg* gör det möjligt att flytta ett program från en hård- och mjukvaruplattform till en annan genom att göra gränssnittskoden flyttbar. Dessa verktyg översätter även programmets gränssnitt till den standard som gäller på den nya plattformen.

## **kompleta program**

*Totalverktygen* kännetecknas av att de kan användas för att utveckla kompletta program. I teorin behövs alltså inga ytterligare verktyg. Både användargränssnitt och programkod kan utvecklas med de inbyggda funktionerna. Ofta ingår en layout-redigerare för den statiska layouten samt ett speciellt programmeringsspråk för gränssnittets dynamik och programkod.

## **prototyputveckling**

*Lekmannaverktygen* är tänkta för personer som behöver utveckla mindre program utan att för den skull vara erfarna programmerare. De liknar totalverktygen i det avseendet att de kan användas för att utveckla kompletta program. De är dock enklare uppbyggda och har också färre funktioner. En vanlig användning för denna typ av verktyg är utveckling av prototyper i ett tidigt skede av ett systemutvecklingsprojekt.

## 2.2 Verktøyen

Denna rapport innehåller beskrivningar av verktyg som kan hänföras till någon eller några av de typer som vi diskuterat ovan. Klassificeringen visas i tabell 1.

Verktygen stöder programutveckling enligt en eller flera olika industristandarder för grafiska fönstermiljöer. I tabell 2 visas vilka fönstermiljöer verktyget stöder.

Tabell 1: Klassificering av verktygen

	Layout-redigerare	UIMS	Flerplattform-verktyg	Total-verktyg	Lekmannaverktyg
CASE:W	•				
Craftman				•	
DevGuide	•				
Easel				•	
ezX	•				
HyperCard				•	•
Interface Builder	•				
Open Interface	•		•		
Plus				•	•
TeleUse	•	•			
Visual Basic				•	•
VUIT	•				
XVT	•		•		



Tabell 2: Verktygens stöd för olika fönstermiljöer

	Windows	Presentation Manager	Motif	Macintosh	Open Look	Nextstep
CASE:W	•					
Craftman						•
DevGuide					•	
Easel	•	•				
ezX			•			
HyperCard				•		
Interface Builder						•
Open Interface	•	•	•	•	•	
Plus	•	•		•		
TeleUse			•			
Visual Basic	•					
VUIT			•			
XVT	•		•	•		

## 2.3 Prototyper

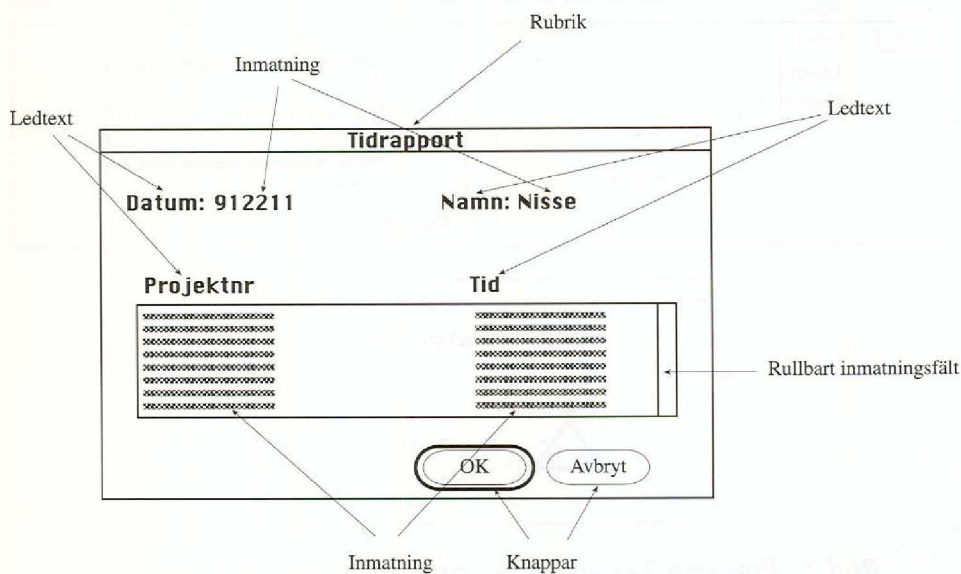
Vi utvecklade två prototyper med avsikten att genom dem dels prova de olika verktygens egenskaper, dels lättare jämföra de olika verktygen. I möjligaste mån har vi försökt realisera prototyperna fullständigt i alla provade verktyg. Där detta inte varit möjligt redovisas det.

### 2.3.1 Prototyp 1 – tidrapport

Den första prototypen är ett enkelt formulär för tidrapportering och är ett exempel på en vanlig typ av program där formulär med relativt begränsat dynamiskt beteende används. Prototypen utnyttjar några enkla interaktionsobjekt men kräver också att interaktionsobjekt ska kunna läggas till i gränssnittet under körning av programmet.

Formuläret innehåller statiska inmatningsfält med beskrivande ledtexter, knappar och en tabell med två inmatningsfält och rullningslistor, (se bild 1).

enkelt formulär



**Bild 1.** Prototyp 1 är ett enkelt formulär för tidrapporter.

De två inmatningsfälten ska ha ledtexterna Namn respektive Datum. Därunder ska en inmatningsruta med två kolumner finnas, en för projektnummer och en för timmar med respektive ledtext.

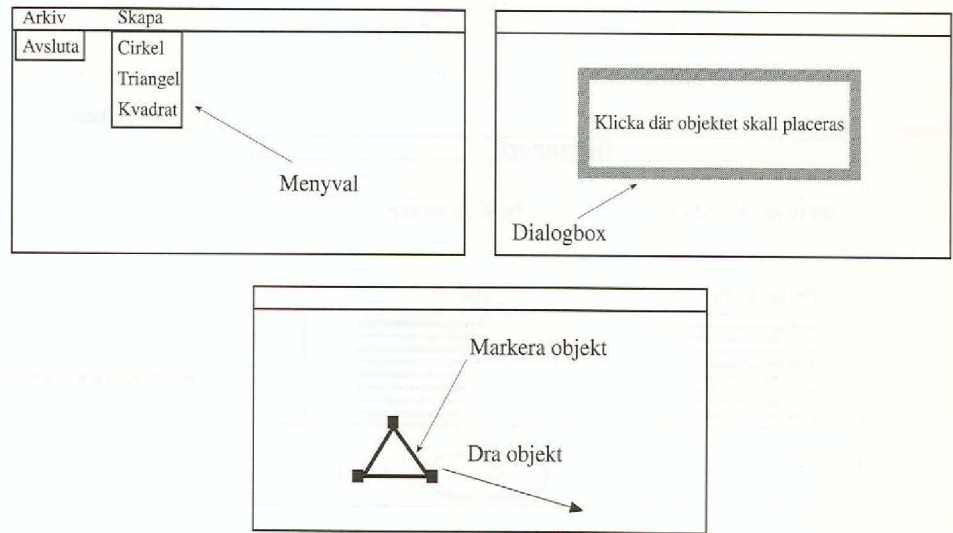
I de två kolumnerna ska nya rader dynamiskt kunna läggas till under körning genom att användaren gör en radmatning med markören placerad på den understa raden. Kolumnerna ska rullas samtidigt med hjälp av rullningslist.

Underst i formuläret ska två knappar för OK respektive Avbryt finnas. De behöver dock inte leda till något annat än att programmet avslutas.

### 2.3.2 Prototyp 2 – ritverktyg

Den andra prototypen innebär utveckling av ett enkelt interaktivt ritprogram. Här provas möjligheten att dynamiskt skapa nya objekt som ska kunna visas och manipuleras i gränssnittet.

**ritprogram**



*Bild 2. Prototyp 2 är ett enkelt ritprogram som bl a utnyttjar direktstyrning.*

**grafiska objekt**

Från en meny ska användaren kunna välja mellan tre olika typer av grafiska objekt: en kvadrat, en cirkel samt en triangel. Sedan ett alternativt valts ska en meddelanderuta visas med texten "Placera objektet!". Meddelanderutan ska vara flyttbar samt icke-modal. När användaren klickar på ritytan ska meddelandet försvinna och det valda objektet placeras i den valda punkten.

**direktstyrning**

De grafiska objekten ska kunna markeras och flyttas med hjälp av direktstyrning.

Ritytan ska vara rullningsbar och menyn ska innehålla alternativet "Avsluta".



# 3. CASE:W

---

**Klassificering:** Layout-redigerare

**Gränssnittsstandard:** CUA

**Miljö:** IBM PC/AT, PS/2 eller motsvarande

Case:w skapar kod för C, C++, XVT eller Cobol.

**Tillverkare:** Legato

**Leverantör:** Software Selection Sweden AB

Case:w kan beskrivas som en utvecklingsmiljö snarare än som ett utvecklingsverktyg. Denna miljö tillhandahåller inte en komplett uppsättning verktyg för utformning av grafiska användargränssnitt. Istället är den en ram inom vilken utvecklaren kan samordna användningen av redan existerande verktyg, såsom resurs-redigerare och kompilatorer. Case:w binder sålunda samman dialogboxar och kodavsnitt, designade och realiserade av externa verktyg, och skapar Windows-kod för det uppbyggda programmet.

För att få ett fungerande program måste utvecklaren så gott som alltid modifiera stora delar av den med Case:w skapade Windows-koden. Case:w döljer inte komplexiteten utan ger snarare en första kodstomme att arbeta från. Detta medför att utveckling med Case:w är en uppgift för programmerare som har stor erfarenhet av Windows-program.

Case:w ansluter sig till CUA (Common User Access) och innehåller en funktion för att kontrollera att utvecklaren följer denna.

**utvecklingsmiljö**

**för erfarna programmerare**

**CUA**

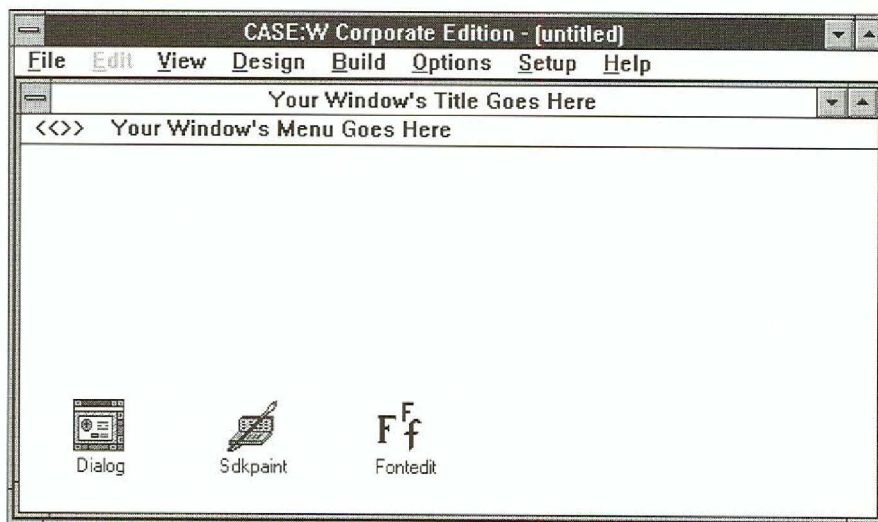
## 3.1 Layout

Utvecklaren kan inte skapa egna interaktionsobjekt, som knappar och fält, med hjälp av Case:w. På denna punkt är utvecklaren beroende av en extern resurs-redigerare, som till exempel Microsoft SDK (Software Development Kit) eller Borlands Resource Workshop. Den enda del av designen av det statiska gränssnittet för programmet som understöds av Case:w är designen av Windowsfönstren och dess menystruktur. Fönster som skapas med hjälp av Case:w kan även vara MDI-fönster

**extern resurs-redigerare**

(Multiple Document Interface), d v s huvudfönster med ett eller flera dotterfönster.

Idén är att utvecklaren ska ha direkt tillgång till de resurs-redigerare som han valt att använda för design av sina interaktionsobjekt och dialogboxar. De valda resurs-redigerarna och de andra verktygen nås lätt via menyer och ikoner på arbetsytan (se bild 3). Det finns möjlighet att konfigurera Case:w till att innehålla 15 olika användardefinierade verktyg.



*Bild 3. Arbetsytan med ikoner för de fördefinierade externa verktygen*

## Windows-kod

Den huvudsakliga funktionen i Case:w är att skapa Windows-kod för dialogdelen i det skapade programmet. För detta ändamål innehåller Case:w en kunskapsbas. Informationen i kunskapsbasen består av kodavsnitt som används som byggklossar i uppbyggnaden av Windows-programmet och regler för hur dessa kodavsnitt ska sammanfogas.

## 3.2 Beteende och koppling till kod

Det finns begränsade möjligheter att med hjälp av Case:w beskriva interaktionsobjektens funktion och interaktion. För att skapa avancerade program är det meningen att utvecklaren ska ändra och lägga till kod i den genererade C-koden.

## kommenterad kod genereras

Den genererade koden är kommenterad. Användaren kan själv välja hur utförligt koden ska vara kommenterad. Det finns tre olika nivåer, låg, medel och hög. Kommentarer är utförliga och lättbegripliga.

Exemplet nedan visar ett kodavsnitt från den av Case:w skapade koden. Nivån på kommentarerna är hög.

```

/*****
/* Main Window Procedure */
/*
/* This procedure provides service routines for the Windows
events */
/* (messages) that Windows sends to the window, as well as the
user */
/* initiated events (messages) that are generated when the user
selects */
/* the action bar and pulldown menu controls or the correspon-
ding */
/* keyboard accelerators.
*/
/*
/* The SWITCH statement ...*/
/*
/*****

LONG FAR PASCAL WndProc(HWND hWnd, WORD Message, WORD wParam, LONG
lParam)
{
    HMENU      hMenu=0;          /* handle for the menu
    */
    HBITMAP    hBitmap=0;       /* handle for bitmaps
    */
    HDC        hDC;             /* handle for the display device
    */
    HANDLE     hPen=0;          /* handle for the current pen
    */
    PAINTSTRUCT ps;             /* holds PAINT information
    */
    int        nRc=0;           /* return code
    */

    switch (Message)
    {
        case WM_COMMAND:
            /* The Windows messages for action bar and pulldown menu
items */
            /* are processed here.
            */
            switch (wParam)
            {
                case IDM_A_NYTT:
                    /* Place User Code to respond to the */
                    /* Menu Item Named "&Nytt" here. */
                    break;

                case IDM_A_AVSLUTA:
                    /* Place User Code to respond to the */
                    /* Menu Item Named "&Avsluta" here. */
                    break;

                case IDM_REDIGERA:
                    /* Place User Code to respond to the */
                    /* Menu Item Named "&Redigera" here. */
                    break;
            }
        }
    }
}

```



**sammanfogning av  
gammal och ny fil**

Case:w kan sammanfoga en ny och en tidigare skapad fil som innehåller en manuellt inlagd kod. Det grafiska gränssnittet kan ändras och koden genereras på nytt utan att den manuellt inlagda koden försvinner. Ändringar kan dock bli svåra eftersom stora delar av programkoden lagts in i den skapade stommen. Det är då omöjligt att garantera att den egna koden inte refererar till objekt som tagits bort i gränssnittet.

### 3.3 Arbetssätt

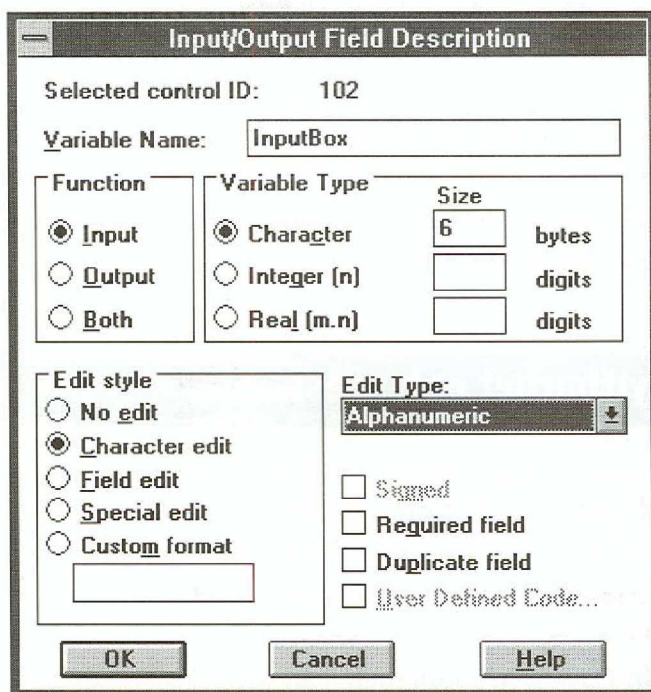
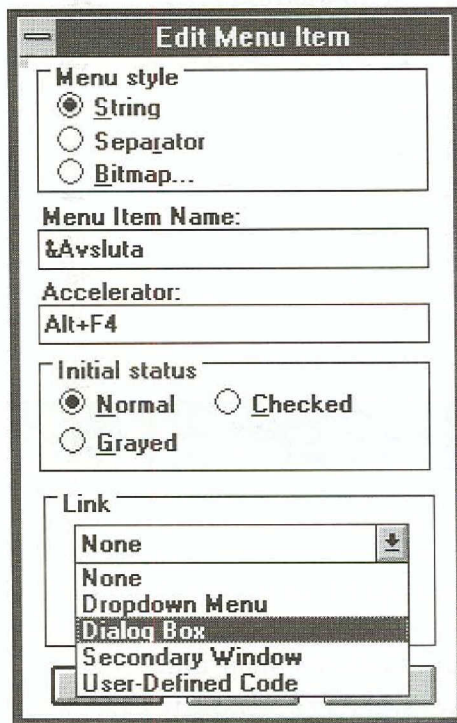
Innan Case:w används vid utveckling av ett program, bör det statiska gränssnittet hos dialogboxarna i gränssnittet vara definierat. Designen av dessa dialogboxar sker helt fristående från Case:w i en extern resursredigerare som t ex Microsoft SDK.

**huvudfönster  
medmenybalk**

Case:w ska binda samman dialogboxarna och koden till ett program. När utvecklaren startar Case:w visas ett huvudfönster med en menybalk överst. I menybalken bygger utvecklaren upp programmets menystruktur. För varje menyobjekt definieras menyobjektets egenskaper med ett speciellt verktyg (se bild 4). Menyobjektet kan bland annat länkas till en dialogbox, ett nytt fönster i Case:w eller till programkoden.

#### *Bild 4. Verktyg för menyobjekten*

För de vanligaste interaktionsobjekten i dialogrutorna kan utvecklaren specificera grundläggande samband och restriktioner. Till exempel kan man definiera att ett visst textfält i en dialogruta måste innehålla sex tecken för att dialogrutan ska få stängas (se bild 5).



*Bild 5. Värdet för textfält som går att definiera i Case:w*

**interaktionsobjekt**

Det finns möjlighet att foga in interaktionsobjekt direkt i ett fönster. För dessa objekt kan dock inga samband eller restriktioner bestämmas

med hjälp av Case:w. All interaktion mellan dem måste definieras och skrivas in i den genererade koden. Interaktionsobjekt som ligger direkt på arbetsytan i ett fönster betraktas i den genererade koden som egna fönster. Dessa fönster är barn (child windows) till det fönster på vars arbetsyta de ligger.

#### **C eller C++**

När ett gränssnitt är definierat, skapar Case:w hela den kod som behövs för att beskriva gränssnittet. Koden är i C eller C++, beroende på vilken kodgeneratormodul som används. Koden översätts sedan till ett körbart program med en valfri kompilator.

### **3.4 Hjälp**

Case:w har utförliga hjälpfunktioner om programmets funktioner. För alla menykommandon finns till exempel hjälp att få genom att markera menyalternativet och välja F1-tangenten. Det finns även en hjälpmeny där det ges en mer översiktlig redovisning av de olika kommandona. Hjälpen finns inlagd i Windows hjälpfunktion.

#### **CUA-kontroll**

Case:w innehåller en aktiv CUA-kontroll. Om användaren försöker designa ett gränssnitt som inte följer CUA, kommer en meddelanderuta upp på skärmen. Denna aktiva kontroll kan ignoreras eller stängas av då användaren vill gå utanför CUA.

När programkod skapas, kan utvecklaren välja om han vill att kod som stöder hjälpfunktionerna ska infogas i kodstommen eller inte. Hjälptexterna läggs in manuellt i Windows hjälpfunktion.

Den tryckta dokumentationen för Case:w är ostrukturerad och svåränvänd och ger därför föga hjälp.

### **3.5 Flyttbarhet och prestanda**

#### **flyttbar**

Ett program som är skapat med Case:w beskrivs i sin helhet av ett C-program. Detta innebär att programmet lätt kan flyttas mellan olika plattformar och att underhåll kan utföras med andra verktyg än Case:w. Det behövs ingen licens för att distribuera program som är utvecklade med hjälp av Case:w.

#### **flera språk**

En fördel med Case:w är möjligheten att köpa kodgeneratorer för flera olika språk och kodbibliotek, t ex C, C++ eller XVT. För C++ finns även flera olika klassbibliotek tillgängliga, t ex Commonview från Glockenspiel, Borlands OWL och Microsofts MFR.



Om programkoden har manuellt inlagd kod i den genererade kodstommen, kan programkoden inte översättas till ett annat språk utan arbete. Den nya kodgeneratoren genererar en helt ny kodstomme utifrån det statiska gränssnitt som är specificerat med Case:w utan att ta hänsyn till den tidigare programkoden. Detta medför att manuellt inlagd kod måste läggas in på nytt i den nya kodstommen.

## 3.6 Prototypstöd

Case:w stöder arbetet under hela utvecklingsfasen, från utvecklingen av prototypen till det färdiga programmet. Case:w lämpar sig dock inte för traditionell prototyputveckling. Programmens skelett och grundläggande utseende kan visualiseras relativt snabbt, men när en mer komplex funktion ska realiserars måste C-kod skrivas. Att skriva C-kod är ofta alltför tidskrävande på prototypstadiet.

**ej prototyputveckling**

Case:w innehåller en lättanvänd testfunktion som gör det möjligt att kontrollera gränssnittets funktioner under arbetets gång. Programkod anropas dock inte.

**enkel testfunktion**

Huvuddelen av funktionerna hos de båda prototyperna rör interaktionen mellan knappar och fält. Interaktionsobjektens utseende skapas framförallt utanför Case:w, i en valfri resurs-redigerare. Objekten kopplas till dialogdelen som är skapad av Case:w. Ett kodskelett skapas för dialogdelen och dess interaktionsobjekt. Det mesta av objektens funktioner läggs till för hand i den skapade koden.

### 3.6.1 Tidrapporten

Tidrapporten designade vi i en extern resurs-redigerare. Vi inkluderade den i Case:w som ett dialogfönster och definierade några av textfältens attribut, bl a restriktioner för textfältens värden, med hjälp av Case:w.

**extern  
resursredigerare**

Övriga funktioner som t ex interaktion mellan textfält och rullningslist realiserades i den av Case:w skapade C-koden.

### 3.6.2 Ritverktyget

Med Case:w skapades lätt den meny i vilken användaren väljer det objekt som ska ritas ut.

Ritytan importerades från en extern resurs-redigerare. All grafikhantering i ritytan, såsom att skapa, rita, välja och flytta symboler, måste

**grafikhantering i  
C-kod**

skrivs i C-kod. När C-funktioner väl fanns, var det dock lätt att koppla dem till olika menyalternativ med hjälp av Case:w.

Det statiska utseendet hos dialogfönstret skapades även det i en extern resursredigerare. I Case:w ges dock en möjlighet att koppla tryckningen på OK-knappen till dialogfönstrets stängning.

### 3.7 Slutsatser

- Case:w skiljer sig en hel del från övriga verktyg i den här rapporten eftersom det i egentlig mening inte är ett utvecklingsverktyg utan snarare en utvecklingsmiljö. Möjligheten att kombinera olika utvecklingsverktyg inom en gemensam ram ger utvecklaren en flexibel utvecklingsmiljö som är möjlig att anpassa till olika uppgifter. Kommunikationen och samspelet mellan de olika komponenterna blir dock mycket små. Case:w kan till exempel enbart stödja de mest grundläggande dynamiska funktionerna för det statiska gränssnitt som skapats i de externa resursredigerarna.
- Case:w erbjuder ingen genväg till konsten att utveckla program för Windows. Utvecklaren måste förstå hur menyer och fönster skapas. En designer av användargränssnitt i Windows som använder Case:w måste fortfarande behärska konsten att programmera i Windows. I och med att utvecklaren måste gå in och ändra och modifiera i den skapade koden måste han förstå vad som sker.

ingen genväg



# 4. Craftman

---

**Klassificering:** Totalverktyg

**Gränssnittsstandard:** Nextstep

**Miljö:** Nextstep

**Version:** 1.0

**Tillverkare:** Xanthus

**Leverantör:** Xanthus AB

Craftman är ett programmeringsverktyg med stöd för gränssnitt enligt standarden i Nextstep. Verktöget började utvecklas inom ett samarbetsprojekt mellan SISU och SICS (Swedish Institute for Computer Science) men utvecklas nu vidare som en produkt av ett fristående svenskt företag, Xanthus AB.

Craftman består huvudsakligen av två delar:

- En *layout-redigerare* där man från ett antal paletter kan välja mellan ett stort antal grafiska interaktionsobjekt som med musen kan placeras på en valfri plats i ett fönster.
- Ett *objektorienterat språk*, Craftscript, som ger möjlighet att definiera beteendet för interaktionsobjekten.

Craftman har inspirerats av bl a Apples Hypercard men ger betydligt större möjligheter, framför allt tack vare det kraftfulla objektorienterade språket. Arbetssättet påminner också mycket om hur man arbetar med Interface Builder, som är Nexts egna utvecklingsverktyg. Däremot är de fördefinierade interaktionsobjekten mer sammansatta och har fler funktioner i Craftman. Craftscript är även ett språk på högre nivå än Objective C som används i Interface Builder, vilket gör Craftman mera effektivt än Interface Builder som prototypverktyg.

När Craftman startas visas två fönster; ett tomt som är huvudfönster för det program som ska utvecklas, samt ett verktygsfönster (Toolbox). Verktygsfönstret innehåller paletter med de olika interaktionsobjekt och andra objekt för att bygga upp gränssnitt, som finns tillgängliga i Craftman.

Craftman är helt objektorienterat. Objekten har därför inte bara statiska egenskaper som storlek, färg och placering, utan också dynamiska egenskaper, d v s beteenden, som definieras av metoder knutna till objekten.

**stöd för Nextstep**

**verktygsfönster**

**statiska och  
dynamiska egenskaper**



Metoderna kan antingen användas i det grundutförande de har i Craftman eller specialiseras till egna metoder av utvecklaren. Metoderna programmeras i Craftscript.

#### klassbläddrare

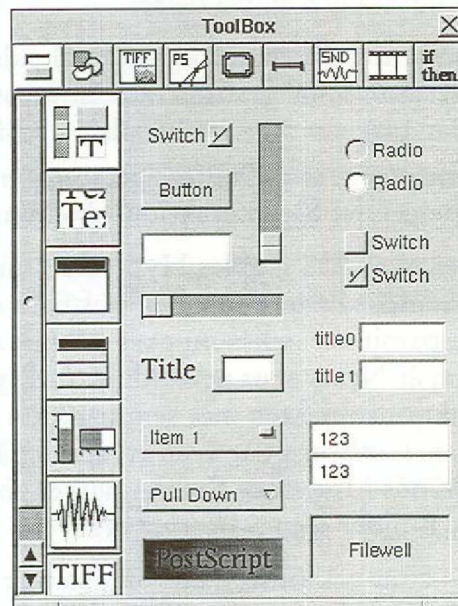
Craftman ger god hjälp att hitta bland de tillgängliga objekten och deras metoder. Hierarkin av klasser ligger i en traditionell Smalltalk-influerad klassbläddrare (class browser). Direktansluten hjälp finns hela tiden tillgänglig och Craftman kan själv ge förslag till vilka metoder som kan användas i ett givet sammanhang. Kan man bara början på ett metodnamn kan Craftman fylla i det som fattas eller presentera en lista över alla metoder vars namn börjar på samma sätt.

#### prova kommandon

I ett Console-fönster (liknar ett terminalfönster) kan man interaktivt skriva Craftscript-kod som omedelbart tolkas och utförs. Fönstret är användbart för att prova olika Craftscript-kommandon.

## 4.1 Layout

Verktygsfönstret, som öppnas när man startar Craftman, innehåller flera olika paletter med objekt. Paletterna nås via knappar i verktygsfönstrets topp.



*Bild 6. Den första paletten med knappar, matriser och menyer. Längst ner till vänster finns ett objekt som kan presentera Postscript-bilder.*

#### interaktionsobjekt

Den första paletten (se bild 7) består i sin tur av flera paletter som nås via en vertikal rullningslist. Dessa paletter innehåller alla grundläggande interaktionsobjekt som finns i Craftman, t ex knappar, menyer och

matrisobjekt. Där finns även objekt med ett mer avancerat beteende. Exempel på sådana är textfält för ordbehandling, videoobjekt och bildspel. Med hjälp av dylika interaktionsobjekt är det lätt att snabbt bygga exempelvis interaktiva presentationer.

De andra paletterna som nås via knapparna i verktygsfönstrets topp innehåller bl a flera olika typer av dekorativa symboler, t ex utsirade ramar och olika bilder. Symbolerna är lagrade i olika bibliotek, t ex bild-, video- och ljudbibliotek. En stor samling bibliotek finns med från början men det är lätt att lägga till egna bibliotek med bilder och ljud för användning i Craftman-program.

**stor variation**

De interaktionsobjekt som finns i verktygsfönstret har ett visst standardutförande som enkelt kan anpassas genom att ändra egenskaperna för enskilda interaktionsobjekt i gränssnittet med verktyget Inspector. Detta verktyget består av dialogboxar där man definierar utseende och beteende för interaktionsobjekten. Detta gör det möjligt att tillverka enklare program i Craftman utan att skriva någon Craftscript-kod.

**objektens egenskaper**

Interaktionsobjektens utplacering i gränssnittet görs helt interaktivt med hjälp av direktstyrning. I Inspector-verktyget finns även möjlighet att sätta objektens koordinater numeriskt och att definiera hur objekten ska ändras om fönstrets storlek ändras vid körning av programmet.

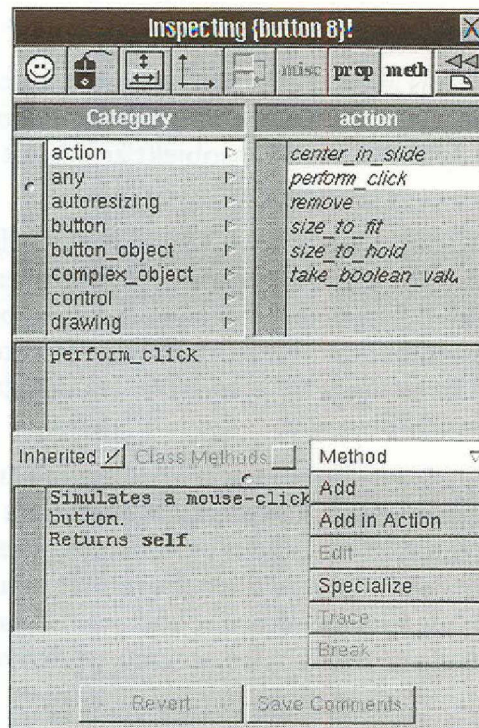
**direktstyrning**

## **4.2 Beteende och koppling till kod**

Alla objekt har förutom egenskaper som gäller utseende också dynamiska egenskaper, ett beteende. Beteendet för ett objekt bestäms av de metoder som är definierade för objektet. Beteendet kan ärvas från dess superklass eller specificeras i egna metoder för enskilda objekt (d v s instanser av klasser). Metoderna definieras i Craftscript-kod.

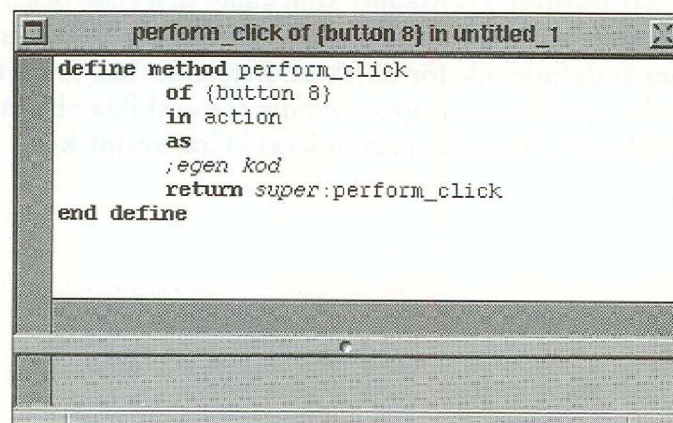
**objektens beteende**





**Bild 7.** I Inspector-verktyget kan man se beskrivningar av ärvda metoder samt välja vilka av dessa som ska specialiseras till egna metoder.

Från Inspector-verktyget har utvecklaren tillgång till alla metoder för ett objekt (se bild 8), både de som ärvs och de som utvecklaren har lagt till själv. Det finns också möjlighet att specialisera ärvda metoder. I kodstommen för de metoder som specialiseras är ett anrop förvalt till superobjektets metod med samma namn. Detta anrop kan dock tas bort.



**Bild 8.** Metoden perform-click specialiseras i redigeraren.

För att i Craftscript-koden referera till andra objekt än det aktuella har man från Interface Builder lånat möjligheten att definiera referensen grafiskt. Genom att hålla ner control-tangenten och samtidigt dra en "kabel" med markören från koden till det objekt som ska refereras



infogas objektets identifierare automatiskt i koden. Ett mycket smidigt tillvägagångssätt.

## 4.3 Arbetssätt

För att bygga upp ett gränssnitt i Craftman används dra-och-släpp. Utvecklaren tar interaktionsobjekt och andra byggstenar, som t ex ljudfiler från verktygsfönstret, och placerar i ett Craftman-fönster. Objektens egenskaper kan sedan förändras, dels med direktstyrning i huvudfönstret, dels via verktyget Inspector.

För att illustrera arbetssättet beskriver vi införandet av en knapp med ikon och ljud som spelas upp då knappen aktiveras. Utvecklaren drar en knapp från verktygsfönstret och släpper i det Craftman-fönster där den ska placeras. Knappens storlek ändras med musen genom direktstyrning och utseendet ändras med hjälp av Inspektor-verktyget. Den bild som ska fungera som ikon dras från verktygsfönstret och släpps på knappen, varvid knappens utseende förändras. Genom att på samma sätt dra ett ljud till knappen kopplar man det till knapptryckningen.

Under uppbyggnaden av ett Craftman-program är det lätt att växla mellan redigering och körning av programmet. Det krävs ingen kompilering av programkoden för att köra programmet eftersom Craftscript är ett tolkat (interpreterat) språk.

I standardfallet behöver man inte programmera om metoderna för t ex en specifik knapp, utan dessa ärvs från den generella klassen knapp. Om man inte är nöjd med knappens standardbeteende, som definieras av knappens metoder, kan man enkelt specialisera en viss metod eller skriva en ny metod för knappen för att uppnå det önskade beteendet.

Är det dessutom önskvärt att den nya metoden också ska gälla för andra knappar skapar man istället en ny subklass till standardklassen knapp, till vilken den nya metoden knyts. Från denna subklass kan sedan nya knappar skapas, där alla ärver den specialiserade metoden. Den nya specialiserade knappen kan sedan infogas i en palett för egna klassobjekt. Objekten i denna palett kan föras in i gränssnittet precis som de fördefinierade interaktionsobjekten, dvs med direktstyrning. Paletten kan dessutom användas i andra program.

Sammansatta objekt kan också införas i en palett. Om man definierat ett standardformulär som, med smärre justeringar, användas i flera program, kan hela formuläret infogas i paletten. Med formuläret som grund kan man sedan göra önskade förändringar och skapa nya program. Denna egenskap ger ett gott stöd för återanvändning av objekt och kod.

**dra-och-släpp**

**metoden ärvs**

**egna klassobjekt**

**bra stöd för återanvändning**

Vissa av de fördefinierade interaktionsobjekten på paletterna i verktygsfönstret ansluter till Nextstep-standarden, men det finns andra som inte gör det. T ex mätare av olika slag och de sammansatta objekten för t ex bildspel. Craftman ger inget stöd för layout enligt Nextstep, dvs placering av menyer mm, även om många av interaktionsobjekten liknar dem som används i Nextstep.

Craftman-program kan enkelt kommunicera med andra program på Next-datorer. Via dessa kan sedan program på andra typer av datorer nås, t ex stordatorer, men detta måste programmeras utanför Craftman på traditionellt vis i exempelvis C eller Objective C.

## 4.4 Hjälp

### klassbläddrare

Craftman innehåller flera funktioner som hjälper utvecklaren att hitta bland klasser och deras metoder. I en klassbläddrare av Smalltalk-typ kan utvecklaren navigera i hierarkin av klasser. Klasserna i klassbläddraren kan inspekteras med Inspector-verktyget. Där ges bl a kortfattade beskrivningar av alla fördefinierade metoder för klasserna.

### stöd för egen kod

I verktygsfönstret finns stöd för utveckling av egen Craftscript-kod. Ur listor med olika språkliga konstruktioner och metoder kan man genom att klicka med markören infoga dessa i sin egen kod i redigeraren.

### ger förslag på metod

Ett problem är ofta att komma ihåg namn och parametrar för den metod som ska användas. I redigeraren finns det stöd för att hitta rätt metod och utvecklaren får på begäran ett förslag på metoder utifrån det som skrivits. Om utvecklaren t ex tror att namnet på metoden för att rotera ett objekt börjar på "rotate", kan Craftman visa en lista över alla tillämpbara metoder som börjar på "rotate". Ur listan kan utvecklaren sedan välja den metod som söks, varefter den automatiskt infogas i koden.

Vad gäller hjälpfunktioner i det egna programmet finns inget uttryckligt stöd för att skriva sådana. Det är dock mycket enkelt att lägga till en egen hjälpmetod för alla objekt som användaren får tillgång till.

## 4.5 Flyttbarhet och prestanda

### prestandaproblem i stora program

På grund av att Craftscript-koden tolkas vid körning av Craftman-program kan det eventuellt bli problem med prestanda vid krävande program med mycket Craftscript-kod. Vid vår realisering av ritprogrammet var dock detta inget problem, trots att realtidsflyttning-



en av grafiska objekt hanterades direkt i Craftscript-kod. Craftmans inbyggda rutiner är alla tillräckligt effektiva för detta.

Att Craftscript-koden tolkas innebär också att ett Craftman-program inte kan köras utanför Craftman-miljön. Man behöver dock inte hela utvecklingsmiljön, utan det räcker med en enkel variant kallad Craftman Engine som kan distribueras fritt av utvecklaren. I denna kan inte programmet förändras utan endast köras. Craftman Engine kan hantera flera olika aktiva program samtidigt.

Craftman kan endast användas på datorer som kan köra Nexts operativsystem, vilket i dagsläget är Next-datorer och PC med Intel 486 processor. Craftman-program underhålls så som de utvecklas, d v s via verktygen i Craftman.

**ej utanför  
Craftman-miljö**

**endast  
Nexts operativsystem**

## 4.6 Prototypstöd

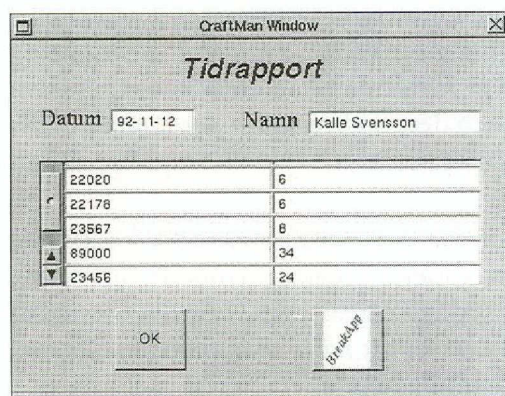
Craftman ger ett mycket gott stöd för utveckling av prototyper. Tack vare att programmen i sin helhet utvecklas i Craftman, d v s både utseende och beteende, kan de också provas i sin helhet.

För en utvecklare med god kunskap om vilka klasser och metoder som finns tillgängliga, är Craftman ett mycket effektivt verktyg. Avancerade funktioner från fördefinierade och egna objekt kan enkelt återanvändas tack vare att Craftman utnyttjar objektorientering.

**bra stöd för  
prototyper**

### 4.6.1 Tidrapporten

Craftman lämpar sig väl för utveckling av denna typ av program. Textfält och knappar är lätta att hantera. Det rullningsbara inmatningsfältet kan lätt byggas upp av ett matrisobjekt och ett rullningsbart fält.



*Bild 9. Tidrapportprototypen*





# 5. DevGuide

---

**Klassificering:** Layout-redigerare

**Gränssnittsstandard:** Open Look

**Miljö:** SunOs 4.1.1 med Open Windows Version 3. Verktøget kan även användas mot andra realiseringar av X-Windows men kräver stöd av viss Sun-specifik mjukvara. Skapade program kan även användas mot andra realiseringar av X-Windows om de skapas med hjälp av Golit.

**Version:** 3.0

**Tillverkare:** Sun Microsystems

**Leverantör:** Sun Microsystems AB

Devguide (Developers Graphical User Interface Design Editor) tillåter utvecklaren att interaktivt bygga ett programs gränssnitt. Verktøget skapar den programkod som behövs för att få ett fungerande skal samt en specifikation för systemgenerering, makefile, som används för att kompilera och länka det körbara programmet. Funktionerna som aktiveras av gränssnittet måste programmeras i C eller C++.

Verktøget består av två delar:

- En *konstruktionsdel*, Devguide, som utvecklaren använder för att bygga och delvis prova gränssnittet. Sedan sparas en specifikation av gränssnittet i en datafil där specifikationen är uttryckt i ett speciellt språk, GIL (Graphical Interface Language).
- En *kodgenerator* som läser den lagrade specifikationen och producerar källkodsfiler och makefil. Kodgeneratøren föreligger i tre olika versioner, Gxy, Golit och Gnt, beroende på vilken verktygssats (toolkit) utvecklaren vill stöda sig på. En verktygssats är ett antal färdiga konstruktionselement som kan kombineras på några olika sätt.

**bygga gränssnitt  
interaktivt**

## 5.1 Layout

Då Devguide startas presenteras ett huvudfönster med ett antal funktioner åtkomliga via menyer och en palett med konstruktionselement. Under paletten finns ett informationsfönster.

### dra-och-släpp

Nya objekt skapas interaktivt med principen dra-och-släpp, d v s man tar tag i ett original på paletten och drar iväg med en kopia av denna till den plats där man vill ha det.

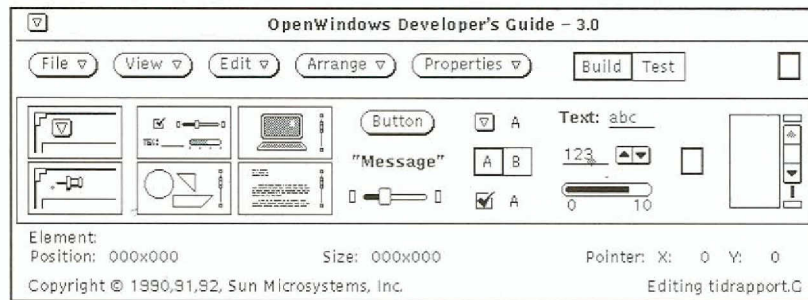


Bild 11. Devguides palett

I linje med Open Looks definition av gränssnittet tillhandahåller Devguide två grundtyper av dialogfönster: basfönster (Base Window) och dyk-upp-fönster (Popup Window). Ett program kan ha godtyckligt antal av varje. Som namnen antyder skapas och visas basfönster direkt då programmet startar, medan dyk-upp-fönster visas efter behov.

### fyra sorters fönster

I dialogfönstren kan man placera fönster av fyra typer:

- *Text Pane* kan användas för redigering av fri text av godtycklig storlek ungefär som i enkla ordbehandlingsprogram.
- *Term Pane* är en terminalemulator där man har tillgång till systemets kommandotolk.
- *Canvas Pane* är en primitiv rityta i vilken programmet kan rita godtyckliga bilder.
- *Control Area* är en behållare för s k kontrollelement.

### kontrollelement

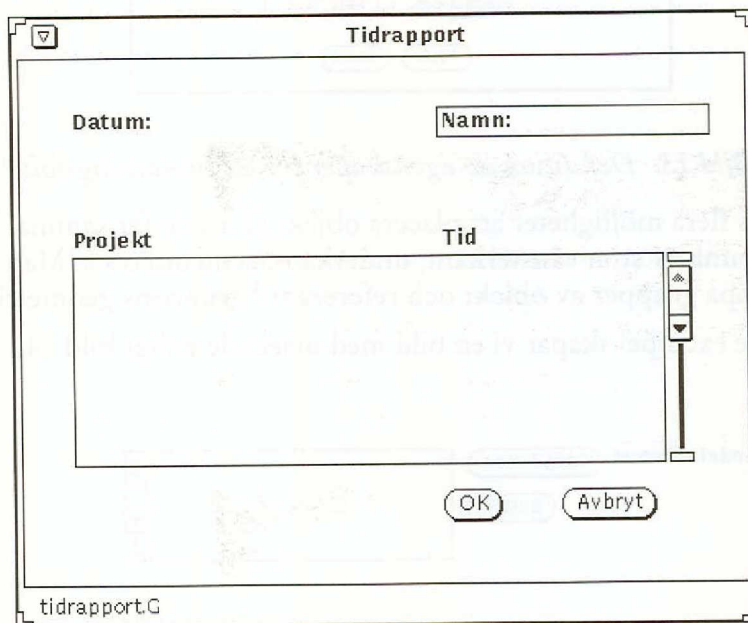
Kontrollelementen utgör de primitiver som utvecklaren använder sig av för att realisera olika typer av interaktion. Dessa är de tillgängliga typerna:

- *Button* är en knapp som aktiverar en funktion eller en meny i programmet.
- *Message* används för att visa ledtexter o dyl.



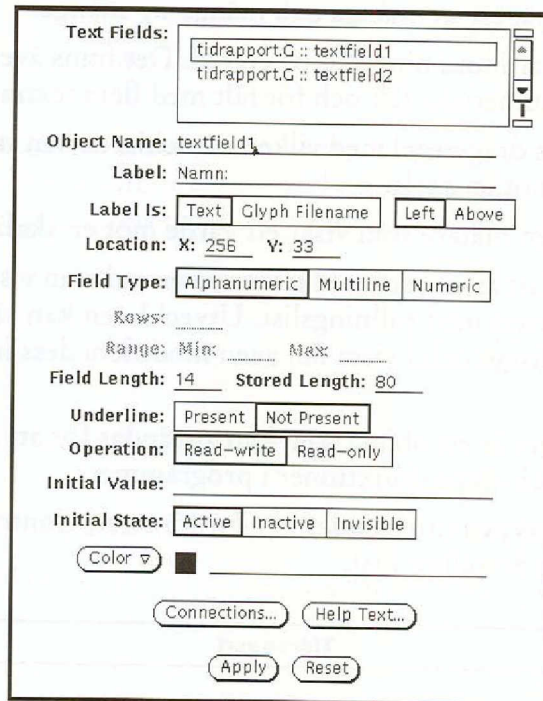
- *Setting* är en samling knappar som kan växla tillstånd såsom av-på, ett-av-många och många-av-många.
- *Text Field* är inmatningsfält på en rad. Det finns även varianter för numeriska fält och för fält med flera textrader.
- *Slider* är en drag-regel med vilken utvecklaren kan ställa in ett värde mot en skala.
- *Gauge* är en mätare som visar ett värde mot en skala.
- *Scrolling List* är en lista med texter på en rad som visas i ett fönster försett med rullningslist. Utvecklaren kan välja element i listan och i vissa fall även modifiera dess innehåll.
- *Drop Target* är ett objekt som kan användas för att realisera dra-och-släpp-funktioner i programmet.

I tidrapportprototypen utnyttjades bl a basfönstret, Control Area, Text Field, Button och Scrolling List.



**Bild 12.** Tidrapporten under uppbyggnad med inmatningsfältet "Namn" utvalt för modifiering.

För varje kontrollelement kan man i ett formulär sedan välja värden för de viktigaste egenskaperna (se bild 13).



**Bild 13.** Definition av egenskaper för ett inmatningsfält.

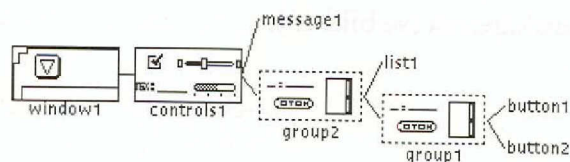
**referenspunkt**

Det finns flera möjligheter att placera objekt så att de får samma referenspunkter som vänsterkant, underkant, centrum o s v. Man kan också skapa grupper av objekt och referera till gruppens geometrier. I följande exempel skapar vi en bild med utseende enligt bild 14,



**Bild 14.** Formulär med text, knappar och lista

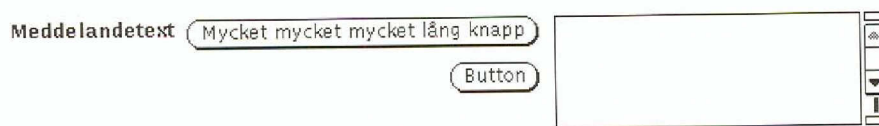
och med struktur enligt bild 15:



**Bild 15.** Strukturen hos formuläret i bild 14

Knapparna är placerade i höjded och bildar en högerjusterad grupp. Dess övre högra hörn är förankrad i listans övre vänstra hörn. Gruppen och listan bildar i sin tur en grupp förankrat i högerkanten av meddelandetexten.

Om meddelandetexten ändrar längd kommer gruppen med listan och knappgruppen att följa efter. Om knapparna ändrar storlek kommer listan att följa efter. Om vi t ex gör texten i den långa knappen längre får strukturen följande utseende:



**Bild 16.** Formuläret med lång knapp

Som framgår av figuren behåller knapparna sin placering i förhållande till listan.

Detta möjliggör att gränssnittet anpassar sig till innehållet i t ex ledtexter så att formulär och fönster ser snygga ut även om texter presenteras i annat typsnitt eller om texterna får annat innehåll p g a att man byter språk.

Devguide tillhandahåller grundprimitiverna i Open Look och *enbart* dessa. Möjligheterna att strukturera grundprimitiverna är begränsade så till vida att man enbart kan konstruera ett antal formulär med statisk form.

Man kan inte utvidga mängden primitiver. Följaktligen kommer programmet att följa Open Look med avseende på innehållet i gränssnittet så länge man enbart utnyttjar Devguide. Medaljens baksida är dock att om programmet fordrar någon form av dynamik som inte finns inbyggd i grundprimitiverna så måste de dynamiska delarna *i sin helhet* kodas i programmet.

**gränssnittet  
anpassar sig**

**grundprimitiverna  
i Open Look**

## 5.2 Beteende och koppling till kod

Händelser kan överföras mellan olika konstruktionselement genom att man beskriver *kopplingar (connections)* vilket är en generalisering och paketering av begreppen "event" respektive "action" i X-Windows terminologi.

Händelseöverföringen kan ses som ett meddelande som sänds från en källa (source) till ett mål (target). Meddelandet sänds när en viss händelse inträffar i källan och utlöser en viss aktivitet i målet. Kopplingen specificeras i ett formulär som man kan aktivera på tre olika sätt:

**koppla händelser**



- Genom att interaktivt dra en kabel från källan till målet. När man avslutat kabeldragningen visas formuläret med lämpliga initialvärden ifyllda.
- Genom att i formuläret för ett kontrollelement klicka på knappen "connections". Kopplingsformuläret visas då delvis ifyllt.
- Genom att i Devguides allmänna menyer välja "connections". I kopplingsformuläret får man sedan välja de kontrollelement som ska kopplas.

Ett exempel på en koppling kan vara att programmet ska starta ett formulär då användaren klickar på en knapp. Detta realiserar genom att man skapar en knapp K (Button) och ett formulär F (dyk-upp-fönster) samt specificerar kopplingen:

```
K(Notify) -> F(Show)
```

vilket åstadkommes genom att man i kopplingsformuläret väljer K som källa och F som mål samt genom att man specificerar "Notify" som händelse i källan och "Show" som aktivitet i målet.

#### **anropa programmet**

För att anropa programmet finns två speciella aktiviteter, Call Function och Execute Code. Call Function är ett anrop av en namngiven procedur vilket innebär att kodgeneratorn kommer att konfigurera händelsehanteraren för att utföra proceduranropet samt skapa en tom procedurkropp (stub). Executive Code innebär att utvecklaren direkt i verktyget skriver in önskad procedur som sedan av kodgeneratorn placeras in i en procedurkropp.

Utvecklaren kan inte definiera nya typer av händelser eller aktiviteter.

#### **integrerar gammal och ny version**

Devguide tillåter utvecklaren att modifiera den skapade koden (inom vissa gränser) och ändå fortsätta arbetet utan att förlora förändringarna. Efter att ha kört t ex Gxv och erhållit ett fungerande skal kan man således modifiera delar av den skapade koden. När man nästa gång kör Gxv sparar den först en aktuell version av den skapade koden innan en ny version skapas. Därefter integrerar Gxv skillnaderna mellan den gamla och den nya versionen. De förändringar man själv gjort på den skapade koden kommer därför automatiskt med även i den nya versionen. Eftersom de delar av koden man inte kan förändra utanför verktyget är tydligt markerade uppstår vanligtvis inga problem.

## 5.3 Arbetssätt

Arbetsgången blir i huvuddrag:

1. Bygga eller modifiera gränssnittet interaktivt med hjälp av Devguide. Vissa funktioner kan provas direkt under konstruktionsfasen.
2. Verktøget genererar C-kod (och ev makefile) med hjälp av Gxv, Golit eller Gnt.
3. Verktøget genererar binärkod, kopplad med hjälp av "make".
4. Prova gränssnittet med programmets funktioner, identifiera felaktigheter i eller olämplig utformning hos gränssnittet och börja om från början.

## 5.4 Hjälp

I Devguide finns direkt tillgång till hjälptexter genom att utvecklaren pekar på önskat objekt och trycker på tangentbordets Hjälp-knapp.

Verktøget stöder också att hjälptexter knyts till olika delar av det skapade programmet på samma sätt, d v s de egna hjälptexterna kommer att kunna aktiveras av den ordinarie hjälpfunktionen.

direkt hjälp

## 5.5 Flyttbarhet och prestanda

Det finns tre olika kodgeneratorer i Devguide: Gxv, Golit och Gnt.

Gxv skapar skal som är anpassade för XView Toolkit som bl a kan kombineras med äldre program ursprungligen konstruerade för Sunview. XView har samma abstraktionsnivå som Athena Toolkit från X-Consortium eller som OSF/Motif Toolkit, d v s byggstenarna har likartad funktion men annorlunda form och uppbyggnad. Det finns även en variant av Gxv som kan användas tillsammans med C++ kod.

Gxv

Golit skapar skal som är anpassade för Open Look Intrinsic Toolkit som i princip är likvärdigt med Athena Toolkit eller OSF/Motif Toolkit, d v s byggstenarna har samma eller snarlika funktion och med samma eller snarlika sätt att realisera. Program skapade med Golit kan därför användas mot andra realiseringar av X-Windows, t ex över ett nätverk.

Golit

Gnt skapar ett skal för News Toolkit som kan kombineras med News, en tidigare produkt från Sun som erbjuder skärmhantering



**Gnt**

baserad på Postscript. Den abstraktionsnivå som News Toolkit erbjuder är likvärdig med den som XView eller Olit erbjuder men programmet blir potentiellt mycket starkare i hanteringen av grafik med möjligheten att använda PostScript.

**passar inte ihop**

En fördel med denna uppdelning är naturligtvis att utvecklaren kan skapa gränssnitt med olika utseende och egenskaper från en och samma specifikation. Tråkigt nog leder uppdelningen också till att komponenterna inte riktigt passar ihop. Layout-redigeraren kan alltså skapa specifikationer som kodgeneratorerna inte riktigt klarar av.

**kompilering  
och länkning**

Det skapade gränssnittet måste naturligtvis kompileras och länkas på en maskin med ett bibliotek som innehåller stöd för XView, Olit eller News Toolkit. Programmet kan köras mot en annan X-realisering om den skapats med hjälp av Gxv eller Golit. Program skapade med Gnt kräver naturligtvis en realisering av News vilket endast Sun tillhandahåller.

**normal prestanda**

Devguide fungerar också mot andra X-realiseringar men såväl Devguide som de skapade programmen har naturligtvis Open Look-utseende vilket kan verka främmande i t ex OSF/Motif-miljö.

**GIL-syntax**

Snabbheten hos Devguide och de skapade prototypprogrammen föreföll helt normal. Eftersom prototyperna är ganska små ger de inte någon indikation om vad som händer i ett verkligt fall.

Det skapade programmets gränssnitt beskrivs helt och hållet av en text i GIL-syntax som Devguide sparar och läser. Detta gör det möjligt att utföra vidare bearbetning och/eller underhåll på beskrivningen med andra verktyg än Devguide, t ex en ordbehandlare. Det är också möjligt, åtminstone i teorin, att översätta specifikationen till något annat verktygs specifikationsspråk.

## 5.6 Prototypstöd

**enkel provning**

Devguide kan simulera alla viktiga egenskaper hos det färdiga gränssnittet som inte beror på aktiviteter i programmet. Utvecklaren kan enkelt växla mellan konstruktion och provning av gränssnittet genom att klicka på en knapp i Devguide som byter arbetsläge.

I tidrapporten kunde vi således direkt prova inmatningsfälten, projektlistan och knapparna.



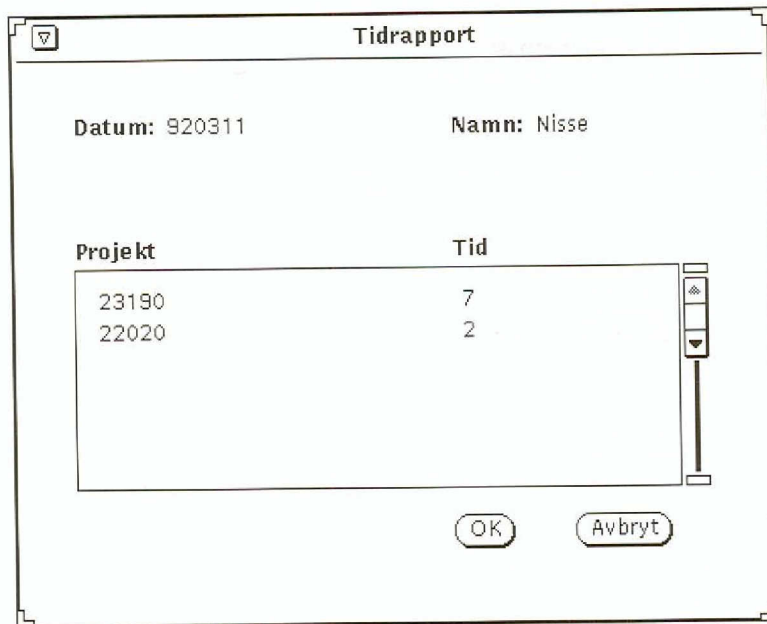


Bild 17. Tidrapporten under provning

### 5.6.1 Tidrapporten

Alla komponenter utom tabellen kunde vi direkt realisera och prova med konstruktionselementen i Devguide.

Tabellen erbjöd ett visst bekymmer eftersom det inte finns några struktureringsmöjligheter i Devguide. Det bästa vi kunde göra i detta fall var att välja en sk Scrolling List. Ett sådant objekt kan skapa, modifiera och ta bort ingångar i tabellen samt navigering via rullningslist. Däremot innehåller detta objekt *inte* parvis ordnade inmatningsfält som prototypen föreskrev utan snarare textrader med en viss färdig redigeringsmöjlighet. Programmet måste i detta fall självt realisera de funktioner som behövdes för att separera projektnummer från tid på varje textrad vid bearbetningen. Det fanns naturligtvis inte heller någon kontroll av att de inmatade värden följde formen för projektnummer eller tid.

**problem med tabellen**

### 5.6.2 Ritverktyget

Menyn, ritytan och dialogboxen innebar inga problem. Vi kunde enkelt definiera menyn och med hjälp av "connections" koppla menyningångarna till funktioner i programmet och till att visa dialogboxen. Vidare kunde vi koppla ett klick på ritytan till aktiviteten att dölja dialogboxen.

All grafikhantering måste vi realisera i programmet, alltså: skapa symbol, rita symbol på begäran, välja symbol, flytta symbol, radera symbol. När dessa funktioner väl fanns kunde vi dock lätt koppla dem till olika händelser.

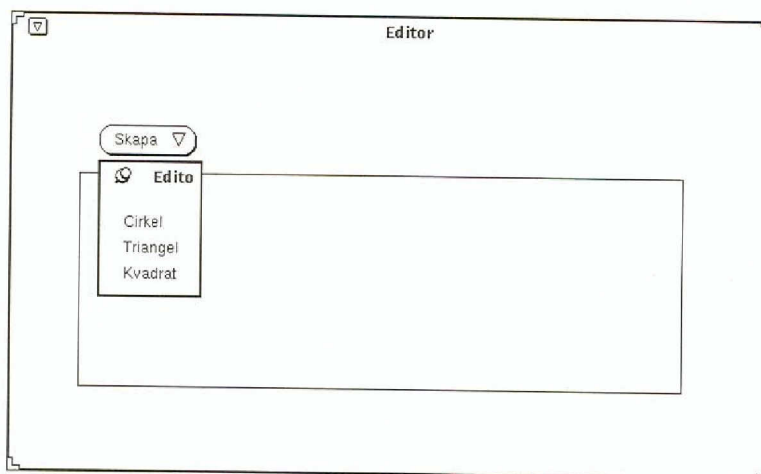


Bild 18. Ritverktyget under körning

## 5.7 Slutsatser

- Eftersom Devguide stöder dra-och-släpp som specificeras i Open Looks blir interaktionen med verktyget mycket enkel under uppbyggnadsfasen.
- Möjligheterna att navigera i en specifikation av ett gränssnitt är i stort sett väl utformade. Man kan via menyer och formulär inspektera objekt av olika kategorier. Det finns även möjlighet att se den hierarkiska strukturen hos de olika komponenterna i ett fönster.
- Det finns inte fler formulär än att man snabbt lär känna dem. Det enda som möjligen saknas är sökning på namngivet objekt.
- Det är svårt eller omöjligt att enbart med Devguide bygga program som kräver dynamik i sitt användargränssnitt.
- Devguide saknar möjligheter att direkt skriva ut eller på annat sätt spara en bild av det gränssnitt som skapats. Man måste gå omvägen över skärmdumpprogram för att erhålla en sådan.

- Uppdelningen i konstruktionsdel och kodgenerator skapar ett visst glapp.
- Utseendet hos gränssnitt som är skapade med de olika kodgeneratorerna kan skilja sig åt oväntat mycket.
- Den funktion hos kodgeneratorerna som integrerar förändringar från tidigare versioner kan ibland misslyckas. Om man skapar ett gränssnitt med t ex ett dyk-upp-fönster, kod genereras och man sedan bestämmer sig för att ta bort fönstret, kommer Devguide att lagra en korrekt beskrivning av den nya versionen. Däremot kommer t ex Gxv att integrera den kod från den föregående versionen som skapade det dyk-upp-fönster man tagit bort ur specifikationen! Enda sättet att undvika detta är att radera samtliga gamla versioner av den skapade koden innan man skapar en ny. Då förlorar man dock samtidigt eventuella förändringar man gjort utanför Gxv. Det krävs således en viss arbetsdisciplin hos utvecklaren för att undvika problem.





# 6. Easel

---

**Klassificering:** Totalverktyg

**Gränssnittsstandard:** CUA

**Miljö:** OS/2, Windows och DOS

**Tillverkare:** Easel Corporation

**Leverantör:** WM-Data

Easel är en miljö för programutveckling med stöd för OS/2, Windows och DOS som utvecklas av Easel Corporation i USA. Företaget anses stå IBM nära vilket märks genom att Easel stöder många av IBMs protokoll.

Easel består huvudsakligen av tre delar:

- Ett *utvecklingsverktyg*, Easel Workbench, som bl a innehåller en layout-redigerare.
- Ett *programmeringsspråk*, som också heter Easel.
- Ett *kodbibliotek* (run time library) till de miljöer som Easel kan utnyttja.

Easel har funnits på marknaden i några år och blivit ett av de mest kända verktygen för OS/2-plattformen. Framför allt har det blivit känt som ett verktyg för att renovera gränssnitt. Det innehåller funktioner för att lägga grafiska användargränssnitt på textbaserade terminalgränssnitt. Det finns t ex funktioner för att analysera innehållet i 3270 bilder. Easel har möjlighet att kommunicera enligt följande terminalstandarder: IBM 3270, IBM 5250 och VT100.

Verktyget innehåller även funktioner för att hämta data ur SQL-databaser samt för att utnyttja program till programkommunikation av typen APPC (Advanced Program-to-Program Communication, en IBM-standard för programkommunikation).

Easel Workbench levereras med en hårdvarunyckel som måste sättas i skrivarporten på utvecklingsmaskinen för att det överhuvudtaget ska gå att köra Easel Workbench.

**stöd för OS/2,  
Windows och DOS**

**renoverar gränssnitt**

**hårdvarunyckel**

## 6.1 Layout

som ett ritprogram

I Easel Workbench ingår det en layout-redigerare (se bild 19). Den fungerar enligt samma principer som de flesta ritprogram. I layout-redigeraren finns det en palett med interaktionsobjekt. På paletten finns även de typer av fönster (regions) som är möjliga att skapa. För att t ex placera ut en knapp väljer man först knappverkytet på paletten för att sedan placera ut knappen i fönstret.

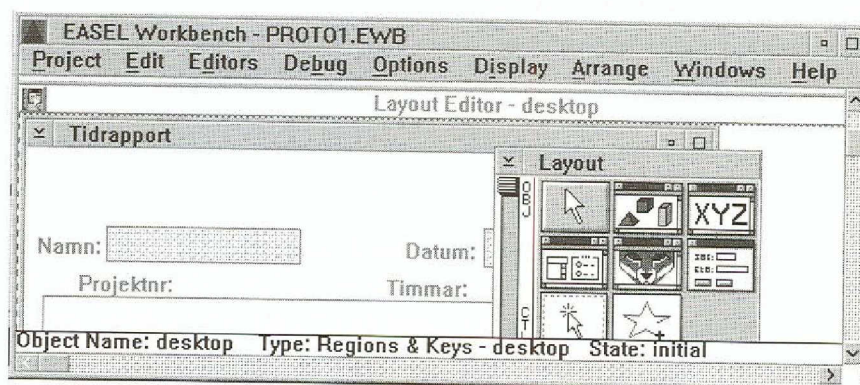


Bild 19. Layout-redigeraren i Easel Workbench

CUA 89

De interaktionsobjekt som finns tillgängliga i Easel är de som finns definierade i CUA 89. Det finns t ex knappar, radioknappar, kombinationsboxar. Easel innehåller också möjlighet att utnyttja grafiska objekt som speciella interaktionsobjekt. Detta gör det möjligt att skapa t ex klickbara kartbilder. Layout-redigeraren innehåller ritverktyg för att skapa dessa grafiska objekt.

Till skillnad mot de flesta andra verktyg som finns på marknaden idag delar Easel upp fönster i olika typer med avseende på vad som kan placeras i dem. Det är till exempel inte möjligt att placera ut knappar i andra fönster än de som är sk dialogfönster.

grundegenskaper

Interaktionsobjekten har ett antal grundegenskaper som kan förändras. Till grundegenskaperna hör till exempel färg, typsnitt och placering. Egenskaperna kan förändras i en egenskaps-redigerare som ingår i layout-redigeraren.

Till grundegenskaperna för inmatningsfält hör även viss kontroll av inmatningen, t ex datum, exakt sex tecken o s v. Det finns däremot ingen möjlighet att lägga till egna inmatningskontroller, utan man får nöja sig med de fördefinierade.

Layout-redigeraren saknar funktioner för att bygga upp menyer och menybalkar. Detta får istället göras med hjälp av programkod.



Easel Workbench innehåller en aktiv CUA-kontroll (Common User Access, IBMs standard för användargränssnitt) som kontinuerligt verifierar om användargränssnitt som skapas följer CUA-standarderna. Kontrollen är av enkel typ, t ex kontrollerar den om det är felaktiga kortkommandon till ett standardmenyalternativ. Det är inte möjligt att lägga till eller förändra regler för kontrollen av användargränssnittet. Däremot är det möjligt att stänga av CUA-kontrollen.

## 6.2 Beteende och koppling till kod

Easel innehåller ett eget programmeringsspråk som även det heter Easel. Språket är av en helt egen typ men liknar till viss del de vanligaste programmeringsspråken. Trots det är språket så pass speciellt att det kräver en viss inlärningsperiod. Dessutom är dokumentationen som följer med rörig och svår genomtränglig.

Programkoden kopplas till interaktionsobjekten genom att man skriver så kallade responsrutiner. Där anges det till vilket interaktionsobjekt denna rutin hör. I responsrutinen är det möjligt att ta hand om de händelser i användargränssnittet som rör interaktionsobjektet t ex en musttryckning. Nedan visas kod för en responsrutin som tar nedtryckningar på tangentbordet och matar in dessa i en textregion:

```
response to TextRegion
make TextRegion cursor visible
begin guarded
response to char from keyboard
add to TextRegion insert input
end
```

Däremot saknas möjligheter att hantera musens rörelser i responsrutiner. Detta medför att det är omöjligt att enbart med hjälp av kod skriven i Easel realisera program som ska utnyttja direktmanipulation av typen dra-och-släpp.

För att dynamiskt skapa interaktionsobjekt i programkod finns det tillgång till klasser av grafiska objekt. I en klass har de grafiska objekten samma programkod. Ur denna klass kan sedan nya instanser skapas dynamiskt.

**responsrutiner****skapa objekt  
dynamiskt**

**kod-redigerare**

För att kunna skriva kod i Easel Workbench ingår det en kod-redigerare. Den gör det möjligt att redigera all kod på en gång eller att redigera kod som hör till ett specifikt interaktionsobjekt. Detta ger en bra överblick av programkoden.

**ineffektivt arbetssätt**

Tyvärr är inte layout-redigeraren och kod-redigeraren i Easel speciellt väl integrerade med varandra. Det är möjligt att se koden för ett interaktionsobjekt i layout-redigeraren, men det är inte möjligt att redigera koden. Detta medför att det blir en hel del byten mellan kod- och layout-redigeraren vilket minskar effektiviteten i utvecklingsarbetet.

## 6.3 Arbetssätt

För att realisera ett program i Easel Workbench börjar man i layout-redigeraren med att skapa de fönster som behövs i programmet. I dessa fönster placeras de interaktionsobjekt som ska utnyttjas. Även grafiska objekt skapas och ritas med hjälp av layout-redigeraren.

Kod knyts till interaktionsobjekten i kodredigeraren med hjälp av responsrutiner, där de händelser i gränssnittet som är intressanta i programmet kodas.

**provkörningsläge**

Det är möjligt att i alla lägen provköra ett program direkt eftersom det finns ett speciellt provkörningsläge. I detta läge finns det möjligheter att utnyttja s k brytpunkter i programkoden för att stoppa exekveringen. Det är möjligt att kontrollera och ändra variabelvärden vid en brytpunkt. Det är sedan möjligt att stegvis köra programmet. Sammantaget ger detta ett gott stöd för stegvis (inkrementell) programutveckling i Easel Workbench.

**körbar mot målmiljön**

När programmet är färdigt kan Easel Workbench skapa en fil som är körbar mot målmiljön. Det krävs ett kodbibliotek för att köra det färdiga programmet. Kodbiblioteket är inte kostnadsfritt och licensavgift krävs beroende på hur många installationer som finns.

## 6.4 Hjälp

**saknar hjälp för programmeringsspråket**

Easel Workbench saknar direkt hjälp för programmeringsspråket, vilket känns besvärande eftersom de medföljande manualerna inte är lätt-använda. Däremot finns det hjälp för själva Easel Workbench. Hjälpen är av standardformat enligt OS/2.



Det finns inget stöd för att realisera hjälpfunktioner för program i Easel Workbench. Det är upp till programutvecklaren att skapa dessa.

## 6.5 Flyttbarhet och prestanda

Flyttbarheten för program som är realiserade i Easel varierar efter vilka målmiljöer som utnyttjas. Easel-program kan köras under OS/2, Windows och DOS. Flyttbarheten begränsas av att Easel inte stöder flera fönster i DOS. De program som tagits fram för DOS kan köras i alla miljöer, men de som tagits fram för Windows och OS/2 kan inte lika enkelt flyttas till DOS om de utnyttjar flera fönster.

Prestanda för program som är realiserade i Easel Workbench är tillräckliga för förhållandevis stora program. Är målmiljön OS/2 eller Windows finns det möjlighet att utnyttja DLL-bibliotek (Dynamic Link Library) för att göra delar av ett program snabbare. Man realiserar då programmet i t ex C. I dessa miljöer är det även möjligt att utnyttja DDE (Dynamic Data Exchange).

**flyttbarhet  
beror på målmiljön**

**tillräckliga prestanda**

## 6.6 Prototypstöd

Easel Workbench fungerar bra för att utveckla enklare prototyper eftersom den stöder stegvis utveckling. För mer komplexa prototyper kan däremot indelningen i olika typer av fönster vara ett hinder. Dessutom innebär mer komplexa prototyper en hel del programmeringsarbete.

Om direktmanipulation av typen dra-och-släpp ska ingå i prototypen så är det inte möjligt att använda Easel för detta.

**bra för enklare  
prototyper**

**ej dra-och-släpp**

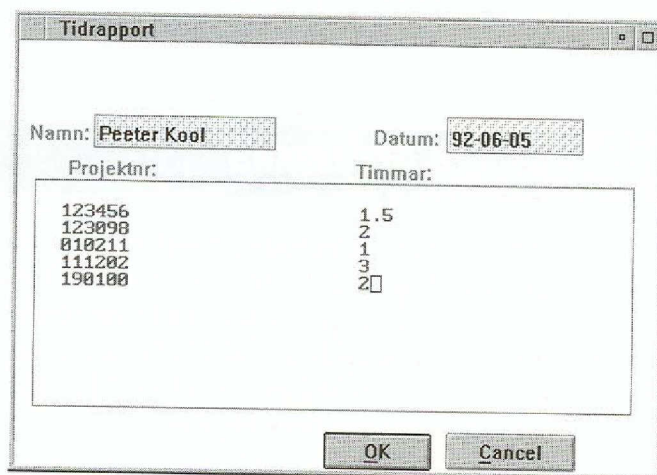
### 6.6.1 Tidrapporten

Tidrapporten realiserades med hjälp av två fönster. Vid körning ser det dock ut som ett fönster. Fönstren är dels ett dialogfönster som innehåller inmatningsfält och knappar, dels ett textfönster som innehåller projektnummer och tid som matas in (se bild 20).

Indatakontrollen i inmatningsfälten är av enkel typ och realiserades med hjälp av Easels inbyggda kontroller. Datumet som matas in kontrolleras o s v.



extraarbete med programkod



*Bild 20. Tidrapporten realiserad med Easel Workbench*

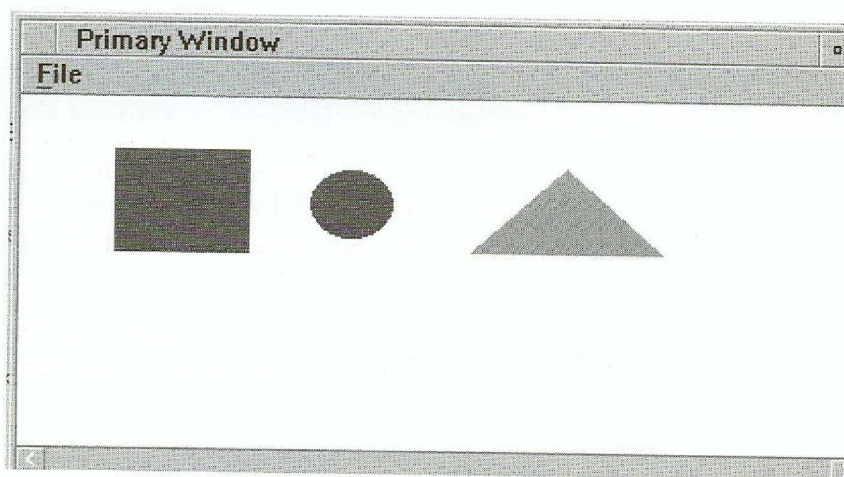
Besvärligast är att prototypen innehåller två fönster, vilket medför extra arbete i form av programkod. Båda fönstren måste t ex flyttas samtidigt. För övrigt är det lätt att realisera prototypen tack vare de inbyggda indatakontrollerna.

## 6.6.2 Ritverktyget

statisk prototyp

Som tidigare nämnts stöder inte Easel dra-och-släpp, vilket medför att det inte är möjligt att realisera ritverktyget fullt ut. Istället blev denna prototyp helt statisk.

Cirkeln, triangeln och kvadraten realiserades som grafiska objekt och ritades med hjälp av ritverktyget i Easel Workbench (se bild 21).



*Bild 21. Ritverktyget realiserat med Easel Workbench*

Menybalken med menyer är helt realiserad i programkod eftersom Easel Workbench saknar meny-redigerare.

Det var enkelt att realisera detta ritverktyg utan dra-och-släpp, men enkelheten berodde i hög grad på att prototypen blev statisk.

## 6.7 Slutsatser

- Easel är ett verktyg som framför allt passar blandade IBM-miljöer eftersom det stöder många av IBMs protokoll.
- Easel stöder stegvis utveckling på ett bra sätt, vilket gör att det passar för utveckling av prototyper. Att det saknas stöd för dra-och-släpp får anses vara en nackdel. Till nackdelarna hör också uppdelningen i olika fönstertyper, vilket kan medföra avsevärt extraarbete i en prototyp.
- Easel innehåller ett eget programmeringsspråk som kräver en tids inläring. Dessutom saknas det direkt hjälp för språket. Detta medför att verktyget har en förhållandevis hög inläringströskel.
- För att distribuera program skrivna i Easel krävs det runtime-licenser. Dessutom krävs det en hårdvarunyckel för att använda Easel Workbench.





# 7. ezX

---

**Klassificering:** Layout-redigerare

**Gränssnittsstandard:** OSF/Motif

**Miljö:** VMS 5.4, Ultrix 4.0, SunOS 4.1.1, AIX 3.1, SCO UNIX.  
Verktyget och skapade program kan även användas mot andra realiseringar av X-Windows.

**Version:** 3.1

**Tillverkare:** Sunrise Software International, USA

**Leverantör:** Sunrise Software International, USA

Ezx tillåter att utvecklaren interaktivt bygger ett programs gränssnitt. Verktyget kan skapa ett skelett till den programkod som behövs för att få ett fungerande skal. Funktionerna i programmen som aktiveras av gränssnittet måste programmeras i C eller Ada.

Verktyget består av två delar:

- En *konstruktionsdel*, Ezxdraw, som utvecklaren använder för att bygga och delvis prova gränssnittet.
- En *presentationsdel*, Ezxpresents, som kan simulera delar av gränssnittet. Då man byggt sitt gränssnitt sparar man en specifikation av detta i en datafil där specifikationen är uttryckt i ett speciellt internt språk. Denna interna representation laddas när programmet körs.

Under uppbyggnaden av ett gränssnitt kan många av egenskaperna provas genom verktygets provkörningsläge. Då simuleras de egenskaper som är inbyggda i gränssnittets komponenter.

Ezxpresents kan användas för att demonstrera ett tänkt program. Det simulerar de inbyggda komponenternas beteende och kan även användas för att förändra vissa egenskaper hos komponenterna i gränssnittet. Däremot kan det inte förändra gränssnittets uppbyggnad.

**provkörningsläge**

**demonstrera  
program**

## 7.1 Layout

När Ezxdraw startas presenteras ett huvudfönster samt tre hjälpfönster (se bild 22). I huvudfönstret finns ett antal menyer för att skapa ett program och dess dialoger. I fönstret "Tools" finns alla de konstruktionselement utvecklaren har tillgång till, i "Files and Workspaces" visas de gränssnitt som har öppnats och/eller skapats. I "Logo" visas leverantörens logotyp och adress.

Ett gränssnitt skapas genom att utvecklaren först skapar en fil för gränssnittet och sedan skapar ett antal arbetsytor i denna fil. Nya objekt i arbetsytan skapas interaktivt genom att välja objekttyp i "Tools" och sedan klicka på arbetsytan där det ska placeras. Objekten kan sedan flyttas och töjas med direktstyrning.

Ezxdraw specificerar ett gränssnitt, eller en del av det, i en fil som innehåller specifikationen för ett antal arbetsytor (workspaces). En arbetsyta motsvarar närmast ett toppfönster eller dialog. På en arbetsyta kan utvecklaren sedan placera konstruktionselement som t ex knappar, menyer, behållare och inmatningsfält.

arbetsyta

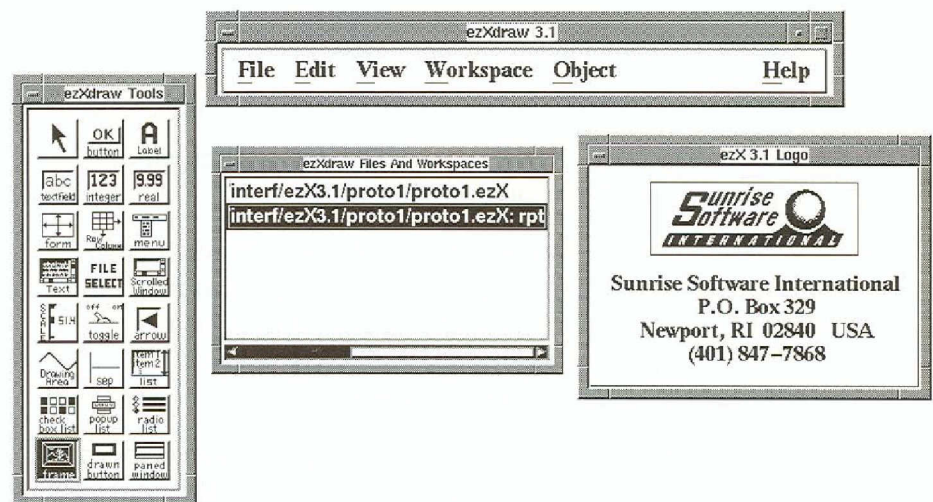
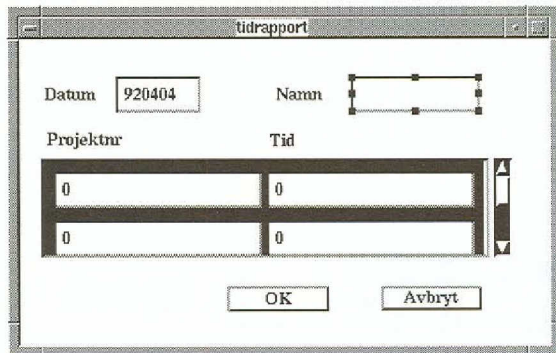


Bild 22. Ezxdraws huvudfönster

Ezxdraw tillhandahåller många av standardkomponenterna i OSF/Motif Toolkit samt några utvidgningar, såsom inmatningsfält som endast accepterar heltal.

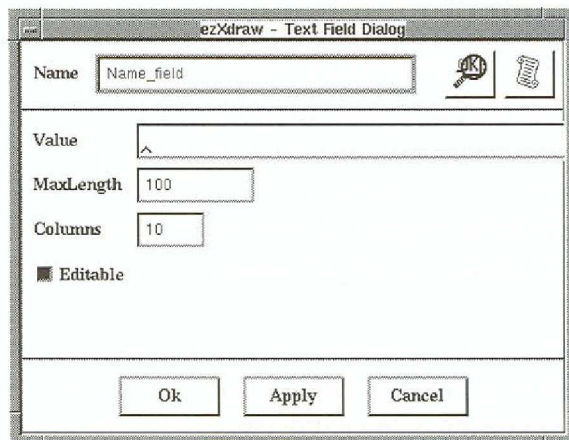
En arbetsyta, som är av typen Form, kan fyllas med olika behållare, menyer, knappar, olika inmatningsfält, ledtexter, ramar, skiljelinjer och liknande.

OSF/Motif Toolkit



**Bild 23.** Tidrapporten under uppbyggnad.  
Inmatningsfältet "Namn" är utvalt för modifiering.

Varje objekttyp har ett formulär där utvecklaren kan ange värden på vissa egenskaper som är användbara på denna typ.



**Bild 24.** Formulär för inmatningsfältet "Namn" i tidrapporten

Utseendet hos gränssnittet styrs i någon mån av vilka objekt man väljer och hur man anger objektens resurser. I princip kan man endast placera objekten på ritytan och där bestämma deras storlek. Det finns ytterst små möjligheter att få komponenterna att anpassa sig till varandra då programmet körs, vilket begränsar användbarheten om man har komplicerade formulär.

Ezxdraw tillhandahåller de flesta grundprimitiverna i OSF/Motif Toolkit. Det går inte att definiera egna primitiver eller att modifiera speciellt många egenskaper för de existerande primitiverna. Många av primitiverna i OSF/Motif Toolkit har en rik flora av resurser som styr

**kan ej anpassas  
under körning**



**måste överlista  
verktyget**

deras utseende och beteende. Endast ett fåtal av dessa resurser kan modifieras med Ezxdraw. För att komma åt övriga resurser tvingas man överlista verktyget.

Konsekvensen av detta är att det skapade programmet kommer att följa innehållet i OSF/Motifs gränssnitt så länge utvecklaren inte börjar kombinera primitiver på nya sätt. Verktyget kan inte ge någon information om huruvida en viss kombination av primitiver följer OSF/Motif.

## 7.2 Beteende och koppling till kod

Beteendet hos gränssnittets komponenter kan specificeras i ett speciellt språk, Ezxtalk, vilket medger att utvecklaren kopplar händelser (reasons) till aktiviteter (commands). Ezxtalk definierar 31 händelsetyper och nio aktivitetstyper.

**händelser från  
X Windows**

De händelser man kan ange är de generella varianterna av de vanligast förekommande händelserna i X Windows, t ex `ACTIVATE` som ungefärligen står för "användaren-klickade-på-knapp" eller "användaren-tryckte-på-enter-tangenten", `VALUE_CHANGED` som står för "användaren-ändrade-värde-på-något" eller `DISPLAYWORKSPACE` som står för "dialogen-kommer-strax-att-visas".

**aktiviteter**

De olika aktiviteter man kan ange är t ex `doCALLBACK`, vilket innebär att en namngiven procedur i programmet anropas, `doOPENWS`, vilket innebär att en namngiven dialog startas eller `doQUIT`, vilket innebär att programmet stoppas. Vidare kan man koppla flera olika aktiviteter till en händelse genom att omgärda aktiviteterna med parenteser.

**måste skriva  
programkod**

Detta förfaringssätt lyfter gränssnittsbygget ett steg från ren X11/Motif-programmering men förefaller ändå något trubbigt eftersom man inte närmare kan precisera vilka händelser man är intresserad av och inte heller kan överföra (propagera) en händelse till ett annat objekt. För alla former av datamanipulering är man dessutom hänvisad till att skriva programkod. Exempelvis kan man för ett inmatningsfält specificera:

```
ACTIVATE doSHELLCOMMAND "ls"
```

vilket innebär att systemkommandot "ls" utförs varje gång användaren trycker "enter" eller "return" medan inmatningsfältet har fokus. Dock kan man inte på något enkelt sätt ange att parametern till `doSHELLCOMMAND` ska hämtas från den textsträng som användaren skrivit i inmatningsfältet. Redan mycket enkla funktioner kräver att man lämnar Ezxtalk och skriver programkod.

## 7.3 Arbetsätt

Arbetsgången blir i huvuddrag att:

1. Bygga gränssnittet interaktivt med hjälp av Ezxdraw. Vissa funktioner kan provas direkt under konstruktionsfasen.
2. Prova gränssnittet med Ezxpresents, identifiera felaktigheter eller olämplig utformning och gå tillbaka till 1.
3. Verktøyet genererar C-kod till programskelett.
4. Skapa en fil för systemgenerering (Makefile) för hand.
5. Verktøyet genererar binärkod till programmet.
6. Prova gränssnittet med programmets funktioner, identifiera felaktigheter eller olämplig utformning hos gränssnittet, modifiera gränssnittet med Ezxdraw och börja om från 3.
7. Utvidga skelettet med erforderlig programkod och börja om från 4.

Det skapade programmets gränssnitt beskrivs helt och hållet av den text som Ezxdraw sparar och läser. Detta gör det möjligt att underhålla beskrivningen med andra verktyg än Ezxdraw, t ex en ordbehandlare. Emellertid är språket inte dokumenterat så detta förefaller inte vara avsikten. Det finns ett verktyg som kan översätta Ezx-specifikationen till UIL (User Interface Language) som är Motifs standard för representation av gränssnitt.

**stöd för UIL**

Ett problem med Ezxdraw är att det inte finns något bra sätt att kombinera programkod med det programskelett som Ezxdraw skapar. Verktøyet kan på begäran skapa ett komplett programskelett för programmet som innehåller stommar till de rutiner programmet självt ska stöda. Detta skelett kan direkt kompileras och köras. Då programkoden ska integreras med skelettet måste stommarna utvidgas med anrop till programkoden. Om man i efterhand introducerar nya anrop från gränssnittet till programmet står man inför valet att antingen själv lägga till nya stommar, eller skapa skelettet på nytt och därvid förlora (eller för hand importera) de utvidgningar man redan gjort.

**kan inte sammanfoga  
ny och gammal version**

## 7.4 Hjälp

I Ezx finns direkt tillgång till allmänt hållna hjälptexter för de objekt som verktøyet känner till.

**hjelptexter  
för objekt**



Verktyget stöder också utvecklaren att knyta hjälptexter till olika delar av det skapade programmet, bl a tack vare att HELP är en av de händelser som kan användas när kopplingar specificeras. För att visa hjälptexten till en viss knapp skulle man således med hjälp av Ezxtalk kunna specificera:

```
HELPdoOPENWS”HelpWs”
```

”HelpWs” är en dialog som definierats med Ezxdraw. Den innehåller ett textfält med önskad hjälptext samt ytterligare en knapp för att avsluta dialogen. Detta kanske inte är det mest naturliga förfaringsättet men det fungerar tillfredsställande och kräver ingen ytterligare programlogik

## 7.5 Flyttbarhet och prestanda

### kompilering och länkning

Det skapade gränssnittet måste naturligtvis kompileras och länkas på en maskin med bibliotek som innehåller stöd för såväl Ezx som OSF/Motif Toolkit. Båda erhålles från leverantören. Programmet kan köras mot annan X-realiserings.

Ezxdraw fungerar också mot andra X-realiseringar men såväl Ezxdraw som de skapade programmen har OSF/Motif-utseende, vilket kan verka främmande i en annan miljö.

### normala prestanda

Snabbheten hos Ezxdraw och de skapade prototypprogrammen föreföll normal. Eftersom prototyperna är ganska små ger de inte någon indikation om vad som händer i ett verkligt fall.

## 7.6 Prototypstöd

Ezxdraw kan simulera vissa av de egenskaper i det färdiga gränssnittet som inte beror på aktiviteter i programmet. Man kan enkelt starta provkörning av gränssnittet från en meny i verktyget. I tidrapporten kunde vi således prova inmatningsfälten och de färdiga ingångarna i projektlistan (se bild 25). Däremot kunde vi inte prova modifieringen av projektlistan eftersom det inte finns något egentligt tabellbegrepp med lämpliga fördefinierade operationer.



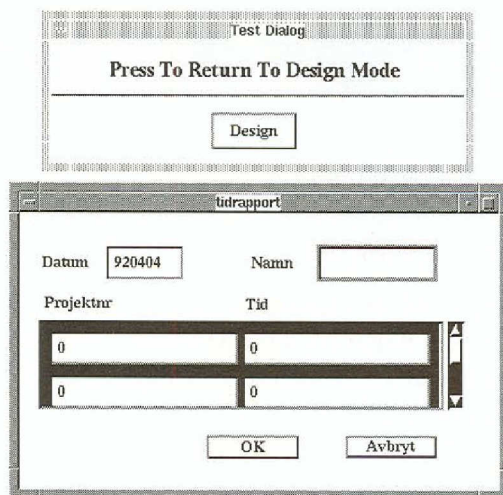


Bild 25. Tidrapporten under provning

### 7.6.1 Tidrapporten

Alla komponenter utom tabellen kunde direkt realiseras och provas med konstruktionselementen i Ezxdraw.

Tabellen var ett svårt problem eftersom det i Ezx, i likhet med Motif, inte finns något bekvämt sätt att dynamiskt utöka strukturen i ett användargränssnitt. Istället skapade vi en tabellstruktur med de befintliga primitiverna genom att lägga två inmatningsfält i en tabellrad. Sedan lades tabellraden i en tabell-kolumn som slutligen placerades i ett fönster med rullningslist.

Det finns dock inget sätt att skapa nya tabellrader med hjälp av funktionerna i Ezx. Vi kunde inte heller inifrån programmet komma åt den struktur som Ezx använder för att representera tabellraderna. Det vore möjligt att med hjälp av Ezxdraw skapa en UIL-specifikation av gränssnittet och sedan via Motifs specialfunktioner (MrmFetchWidget m fl) komma åt tabellstrukturen. I praktiken innebär detta emellertid att verktyget saknar stöd för dynamiska gränssnitt.

Ett alternativ var att representera tabellen i form av en lista av textsträngar. Dessa kunde inte redigeras när de stod i listan. Istället skapade vi dels en funktion som kopierar en textsträng till två inmatningsfält, dels en funktion som kopierar de två inmatningsfälts innehåll till en given plats i listan. Detta krävde två funktioner i programmet som, när vi väl hade dem, enkelt kunde kopplas till lämpliga händelser i listan och inmatningsfälten. Detta alternativ kan eventuellt fungera tillfredsställan-

**problem med tabellen**

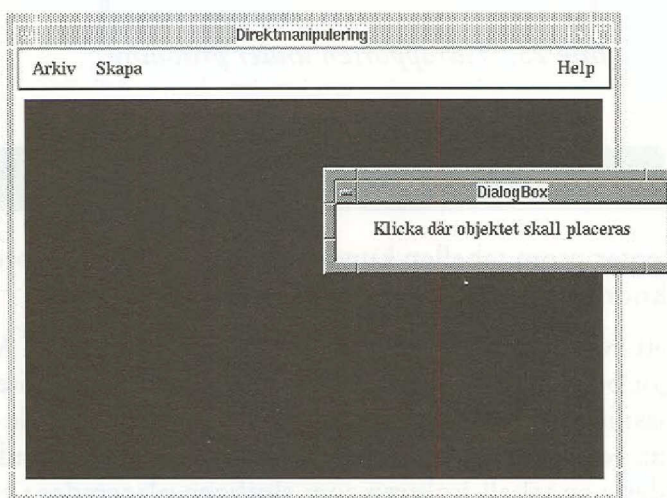
de även om det inte följer specifikationens krav på parvis rullbara inmatningsfält.

## 7.6.2 Ritverktyget

Menyn, ritytan och dialogboxen innebar inga problem. Vi kunde enkelt definiera menyn och koppla menyningångarna till funktioner i programmet och till att visa dialogboxen. Vidare kunde vi koppla ett klick i ritytan till aktiviteten att dölja dialogboxen.

**grafikhantering  
i egen kod**

All grafikhanteringen måste realiseras i programmet, alltså: skapa symbol, rita symbol på begäran, välja symbol, flytta symbol och radera symbol. När dessa funktioner väl fanns kunde vi dock tämligen enkelt koppla dem till olika händelser.



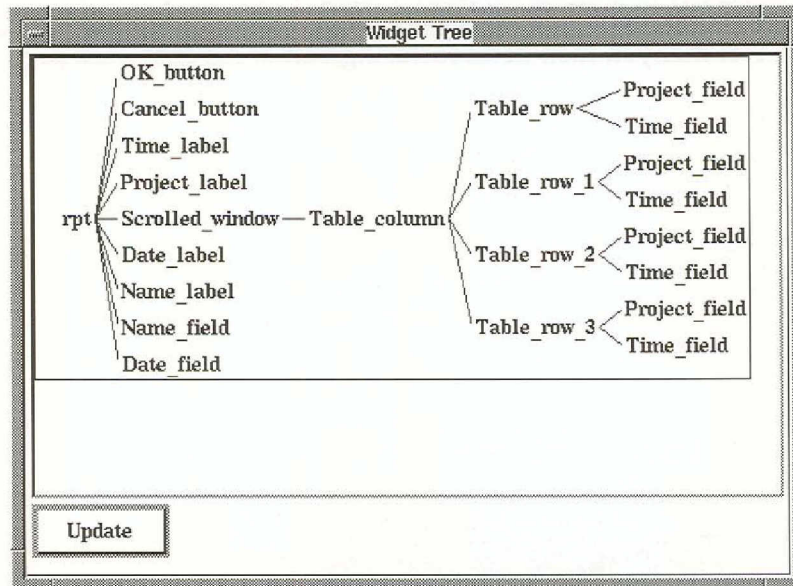
*Bild 26. Ritverktyget under körning*

## 7.7 Slutsatser

- Ezxdraw är ett enkelt verktyg. Det går snabbt att lära sig använda det och att inse dess begränsningar. Språket Ezz-talk, trots de problem som diskuterats ovan, förenklar vissa uppgifter som ofta förekommer i bygget av gränssnitt.
- Det finns möjlighet att se den hierarkiska strukturen hos de olika komponenterna i ett fönster (se bild 27). Tråkigt nog hålls detta fönster inte aktuellt av Ezxdraw utan



utvecklaren måste begära uppdatering i stort sett varje gång det behövs. Vidare kan detta fönster inte användas för att välja objekt.

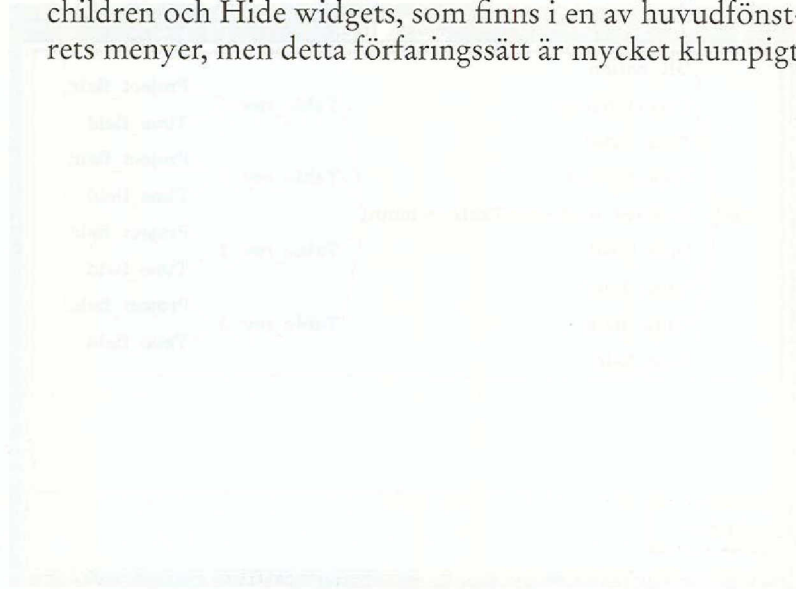


*Bild 27. Tidrapportens hierarkiska struktur*

- Det är i stort sett omöjligt att bygga program som kräver dynamik i sitt användargränssnitt enbart med Ezx.
- Verktøget saknar möjligheter att direkt skriva ut eller på annat sätt spara en bild av det gränssnitt som skapats. Utvecklaren måste gå omvägen över skärmdumpprogram för att erhålla sådana.
- Dokumentationen är bristfällig. Somliga begrepp förklaras inte alls och dessutom saknas register.
- Ezxtalk är helt otillräckligt för att frigöra utvecklaren från detaljer i gränssnittets konstruktion även om själva idén att använda ett speciellt specifikationspråk för att koppla händelser till aktiviteter är sund.
- Några av konstruktionselementen har fel beteende, t ex erhölls i tidrapporten en OK-knapp vars storlek inte var densamma i det färdiga programmet som den vi angivit i konstruktionsfasen. En annan överraskning är att man för "IntegerField" och "RealField", speciella varianterna av inmatningsfält, kan ange tillåtna gränser för det inmatade värdet, men värdet kontrolleras aldrig mot de angivna gränserna!



- Utvecklaren kan navigera i gränssnittet endast genom att välja objekt på en rityta. Det finns ingen möjlighet att få tag i objekt som inte syns. Visserligen går det att få fram alla objekt genom att applicera funktionerna Display all children och Hide widgets, som finns i en av huvudfönstrets menyer, men detta förfaringssätt är mycket klumpigt.



# 8. HyperCard

---

**Klassificering:** Lekmannaverktyg och Totalverktyg

**Gränssnittsstandard:** Macintosh

**Miljö:** Macintosh

**Version:** 2.1

**Tillverkare:** Apple

**Leverantör:** Apple

Hypercard är ett verktyg för att utveckla program för Macintosh. Det utvecklades till en början av Apple men senare tog Claris över utvecklingsansvaret. Hypercard lanserades av Apple som ett verktyg för slutanvändare och det levererades till en början kostnadsfritt med alla nya Macintosh-datorer.

Hypercard var det första sk kort-baserade verktyget som kom ut på marknaden. Dessa karaktäriseras av att de bygger på en kort-metamor, man talar om buntar, kort, bakgrund osv. Dessutom återuppväckte Hypercard intresset för hypertextsystem i och med att det finns funktioner för att bygga dessa.

Trots att målgruppen för Hypercard ursprungligen var slutanvändare upptäckte snart programutvecklare verktyget. Det har blivit ett populärt prototypverktyg på Macintosh och det har också utvecklats en hel del riktiga program med hjälp av verktyget.

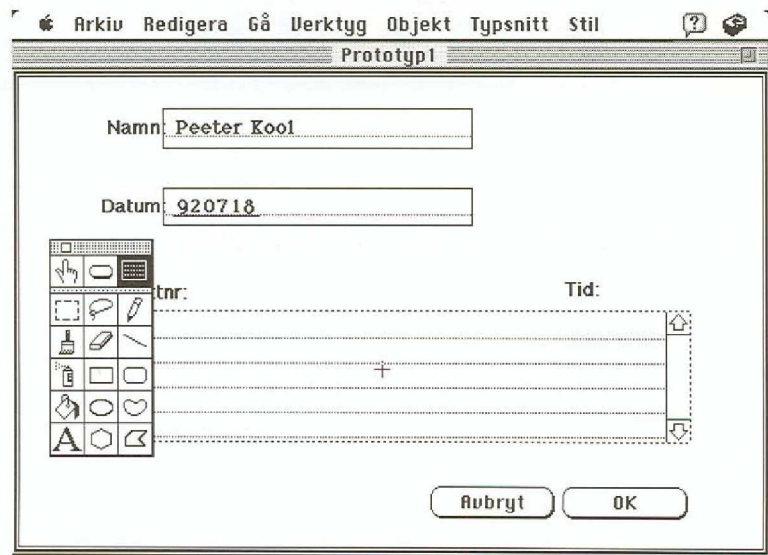
Hypercard har fått många efterföljare t ex SuperCard, Plus och MetaCard, som också utnyttjar kort-metamoren och i vissa fall också samma programmeringsspråk.

Hypercard består huvudsakligen av två delar:

- En *slutanvändarmiljö* som också är *utvecklingsmiljö*. Med andra ord skiljer inte Hypercard på utveckling och körning.
- Ett *programmeringsspråk*, Hypertalk, som är speciellt i den bemärkelsen att det försöker efterlikna engelskt tal-språk.

**verktyg för  
slutanvändare**

**många efterföljare**



*Bild 28. Hypercards utvecklingsmiljö med verktygspalette*

## 8.1 Layout

### kort i buntar

Ett användargränssnitt byggs upp genom att utvecklaren organiserar kort i buntar. Det är enbart möjligt att se ett kort i taget från en bunt. Däremot är det möjligt att ha flera buntar öppna samtidigt. Kortet blir användargränssnittet och där placeras interaktionsobjekten.

### förgrunds- och bakgrundsskikt

Korten är i sin tur uppdelade i två skikt lagda på varandra, förgrunds- och bakgrundsskikt. Bakgrundsskiktet kan vara gemensamt för flera kort och det utnyttjas för interaktionsobjekt och grafik som ska vara gemensamma.

### knappar och textfält

Interaktionsobjekten i Hypercard skapas genom ett menyval. Hypercard innehåller två typer av interaktionsobjekt, knappar och textfält. Detta kan tyckas vara en begränsning, men dessa två objekt finns i flera varianter: en knapp kan t ex vara en knapp, radioknapp eller checkbox. När ett nytt interaktionsobjekt skapats får det ett standardutseende. Till interaktionsobjekten hör ett antal egenskaper, t ex utseende, namn och typsnitt.

Det finns inga funktioner för att justera interaktionsobjekt inbördes. Utvecklaren är hänvisad till att justera på fri hand.

### verktygspalette

Det finns tillgång till en verktygspalette (se bild 28) som innehåller ritverktyg. Dessa används bl a för att skapa statisk grafik i användargränssnittet. Knapp- och fältverktygen, som också finns på paletten, används då egenskaperna hos befintliga knappar och textfält ska ändras.



Det finns även möjlighet att skapa interaktionsobjekt som inte följer standard. Med ritverktyget kan utvecklaren skapa det önskvärda interaktionsobjektet och sedan placera en osynlig knapp på samma plats. Handverktyget på paletten används för att övergå till körning.

## 8.2 Beteende och koppling till kod

Hypercard innehåller ett eget språk för programmering, Hypertalk. Syntaxen i Hypertalk försöker efterlikna engelskt talspråk. Om t ex variabeln x ska tilldelas värdet tio, uttrycks det i Hypertalk-syntax som "put 10 into x". Detta försök till likhet har gjort Hypertalk rörigt och pratigt, vilket medför att det krävs tid för inläring för att kunna utnyttja Hypertalk effektivt. En fördel är dock att Hypertalk är ett otypat språk, vilket gör det enkelt att realisera generella funktioner.

Beteendet hos interaktionsobjekt bestäms genom att utvecklaren skriver Hypertalk-kod som knyts till interaktionsobjektet. För interaktionsobjekten finns det ett antal meddelanden (händelser) som kan specialiseras, t ex mouse up, mouse move, mouse down, enter key.

Naturligtvis finns det funktioner i Hypertalk för att förflytta sig mellan olika kort, bakgrunder och buntar eftersom Hypercard bygger på en kort-metafor.

Förutom att knyta kod till interaktionsobjekt är det möjligt att även knyta kod till kort, bakgrunder och buntar. I Hypercard finns det en specificerad väg som meddelanden går. När t ex musknappen släpps upp på ett interaktionsobjekt, skickas meddelandet "mouse up" först till objektet. Om inte objektet tar hand om meddelandet fortsätter det till kortet för att därifrån gå vidare till bakgrunden och sist till buntan.

Ett problem som uppstår på grund av att kod kan knytas till olika objekt är att överblicken förloras. Hypercard innehåller ingen hjälp för att hitta var en specifik funktion finns realiserad. I större Hypercard-program är det därför svårt att hitta var ett visst givet kodavsnitt finns, vilket medför att programmet blir svårt att underhålla.

För att skriva kod i Hypercard finns det tillgång till en kodredigerare. Kodredigeraren är av enkel typ och innehåller funktioner av typen sök. Det görs även en automatisk inskjutning (indentering) av kod.

Det är möjligt att bygga upp en menybalk med menyer i Hypercard. Detta kan dock endast göras i Hypertalk-kod och är förhållandevis besvärligt. Dessutom uppstår det problem i och med att utvecklings- och slutanvändarmiljön är den samma. Som utvecklare vill man använda Hypercards egna menyer men om man utvecklar ett program som

**liknar engelskt  
talspråk**

**otypat  
objektens  
beteende**

**dålig överblick**

**kod-redigerare**

ersätter dessa med programmets egna menyer kan det uppstå situationer där utvecklaren själv inte har tillgång till Hypercards menyer.

**skapa nya  
objekt dynamiskt**

Att dynamiskt skapa nya interaktionsobjekt i Hypercard är enkelt. För att t ex skapa en ny knapp anropar man menykommandot "skapa ny knapp". Alla interaktionsobjektens egenskaper, inklusive kod, kan förändras under körning från Hypertalk-kod. Dessutom är det möjligt att utnyttja alla ritverktyg i Hypertalk-koden.

**XCMD**

Det finns också möjligheter att bygga ut Hypertalk med egna funktioner. Det finns ett gränssnitt mot s k XCMD (External Commands External Functions). XCMD är program som skrivits t ex i C och sedan kompilerats. När väl ett XCMD är installerat i Hypercard så fungerar det som ett vanligt funtionsanrop.

Det finns tillgång till en mängd XCMD-bibliotek på marknaden som kan köpas från från oberoende leverantörer. Till exempel finns det XCMD-bibliotek som gör det möjligt att styra och presentera Macro-Mind Director-filmer i Hypercard.

**svårt rätta till  
misstag**

Hypercard sparar kontinuerligt programmet både under utveckling och körning. Detta kan leda till att misstag blir svåra att rätta till. Om man t ex raderar en knapp av misstag och inte direkt väljer menyalternativet ångra, finns det ingen möjlighet att rätta knappen och dess kod.

En annan effekt av att Hypercard sparar kontinuerligt är att innehållet i textfält inte försvinner då programmet avslutas utan finns kvar nästa gång programmet startas. Detta gäller dock inte variabelvärden.

## 8.3 Arbetsätt

Ett program i Hypercard motsvaras av en eller flera buntar. Därför måste bunten eller buntarna skapas först. Sedan gäller det att bygga upp de kort och bakgrunder som bildar användargränssnittet. Interaktionsobjekten placeras på korten och bakgrunderna och objektens uteseende ställs in. Den statiska grafiken byggs upp med hjälp av ritverktygen.

**kod knyts till  
olika nivåer**

Interaktionsobjektens beteende skapar utvecklaren genom att skriva kod för objekten med hjälp av kod-redigeraren. Mer generell kod som ska gå att nå från flera olika ställen knyts antingen till kortet, bakgrunden eller bunten. De funktioner som definieras på ett kort kan endast anropas direkt från detta kort. De funktioner som definierats på ett bakgrundsskikt kan anropas från de kort som använder denna bakgrund och de funktioner som knyts till en bunt kan nås från alla kort i bunten.

**kan provköras  
i alla lägen**

Det är möjligt att i alla lägen provköra programmet i och med att utvecklingsmiljön och slutanvändarmiljön är densamma. När handverktyget används körs programmet.



För att avlusa programmet finns det brytpunkter att sätta i kodredigern. Förutom brytpunkter är det också möjligt att spåra funktionsanrop och variabelvärden.

Det är inte förvånande att Hypercard har blivit ett populärt utvecklingsverktyg eftersom det med sin integrerade utvecklings- och slutanvändarmiljö är ett typexempel på ett verktyg som stöder stegvis (inkrementell) utveckling.

**stegvis utveckling**

## 8.4 Hjälp

Med Hypercard följer två hjälpbuntar, en för själva Hypercard och en för programmeringsspråket Hypertalk. Dessa buntar är uppbyggda med funktioner för hypertext d v s det går att följa referenser mellan ord i texten o s v. Hjälpbuntarna är användbara särskilt då de medföljande handböckerna är förhållandevis röriga.

För program som utvecklas i Hypercard finns inga färdiga funktioner för att skapa hjälp. Dessa måste programutvecklaren själv realisera.

## 8.5 Flyttbarhet och prestanda

I och med att Hypertalk tolkas (interpreteras) vid körning av Hypercard-program kan det bli problem med prestanda. Redan i våra små testprototyper märktes detta. Speciellt i ritverktyget, där dra-och-släpp ingår, blev det eftersläpning mellan markörens position och objektets. Överlag får lite större program skrivna i Hypercard problem med prestanda.

**dåliga prestanda**

Ett sätt att undvika dessa problem är att flytta ut delar av programmet i XCMD-funktioner. Många av de större program som realiserats i Hypercard har just utnyttjat denna möjlighet för att få godtagbara prestanda.

**flytta ut delar av programmet**

För att distribuera ett Hypercard-program behövs de buntar som realiserar programmet och själva verktyget, om inte användarna redan har tillgång till det.

I och med att Hypercard enbart finns i Macintosh-miljö saknas möjlighet att flytta programmet till en annan plattform med hjälp av Hypercard. Dock finns det andra produkter, som t ex PLUS, som klarar av att läsa Hypercard-buntar med vissa begränsningar och kan konvertera dessa till ett eget flyttbart format.

**enbart Macintosh-miljö**



passar för  
prototyper

## 8.6 Prototypstöd

Hypercard passar utmärkt för att utveckla prototyper av flera orsaker. Slut användarmiljön är densamma som utvecklingsmiljön, vilket ger ett gott stöd för ett stegvist arbete. Dessutom är det möjligt att i programkod utnyttja alla funktioner som finns i utvecklingsmiljön, t ex ritverktygen, skapa knappar och ta bort knappar. Ytterligare en egenskap som är värdefull vid arbete med prototyper är att innehållet i textfälten inte försvinner då programmet avslutas. Det medför att det är enkelt att lägga upp och förändra testdata.

Hypercard har också en fördel i och med att programmeringsspråket är otypat. Detta sparar en hel del tid eftersom det inte behövs typkonverteringar.

### 8.6.1 Tidrapporten

Tidrapporten byggde vi upp med hjälp av ett kort med knappar och fält. Ledtexterna byggdes upp av textfält där vi först skrev in ledtexten för att sedan låsa den. Indatafälten byggdes upp av textfält som inte var låsta. Indatakontrollen i textfälten görs när användaren trycker på OK-knappen. Hypercard saknar funktioner för att på ett enkelt sätt göra indatakontroller på annat sätt.

I och med att innehållet i textfält inte försvinner mellan körningar av Hypercard-program, innehåller ritverktyget en funktion som raderar textfältens innehåll när programmet startas.

Sammanfattningsvis var det mycket enkelt att realisera tidrapporten i Hypercard.

Prototyp1	
Namn: <u>Peeter Kool</u>	
Datum: <u>920718</u>	
Projektnr:	Tid:
129219	2,0
299829	83,0
935383	6,5
Avbryt    OK	

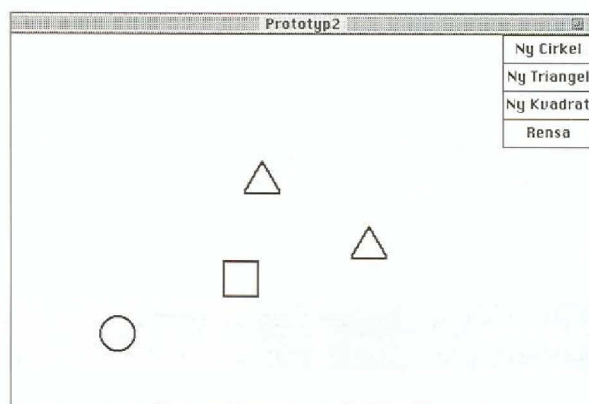
Bild 29. Tidrapporten under körning

## 8.6.2 Ritverktyget

Ritverktyget var betydligt besvärligare att realisera. Detta berodde till stor del på att Hypercard inte har färdiga funktioner för direktstyrning av typen dra-och-släpp. Koden för att skapa dra-och-släpp i ritverktyget (se koden nedan) blev förhållandevis omfattande och gav upphov till prestandaproblem.

Cirkeln, kvadraten och triangeln realiserade vi med hjälp av knappar (se bild 30). Knapparna fick olika ikoner beroende på vilken typ av objekt de var.

ej dra-och-släpp



*Bild 30. Ritverktyget under körning*

Som tidigare nämnts är det besvärligt att skapa menyer i Hypercard. Istället realiserade vi menyer som knappar.

Sammanfattningsvis var ritverktyget svårt att realisera och arbetet krävde ingående kunskaper om Hypercard. Dessutom var prestanda dåliga. Det blev en ordentlig eftersläpning när vi drog ett objekt.

Nedan ges ett kodexempel på hur direktstyrning kan realiseras i Hypercard:

```
on MouseDown
  global gSelected, gDX, gDY
  set the hilight of me to true
  -- Spara positionerna för musklicket
  put item 1 of loc of button short name of me - item 1 of
the clickloc into gDX
  put item 2 of loc of button short name of me - item 2 of
the clickloc into gDY
end MouseDown
-- ...and drag object
```

```

on MouseStillDown
-- Dra objektet
  global gDX, gDY
  put the mouseLoc into ml
-- Normalisera positionen
  add gDX to item 1 of ml
  add gDY to item 2 of ml
-- se till att objektet inte dras utanför kortet
  put max(item 2 of ml,59) into item 2 of ml
  put min(item 2 of ml, 316) into item 2 of ml
  put max(item 1 of ml,32) into item 1 of ml
  put min(item 1 of ml, 470) into item 1 of ml
-- Flytta objektet
  set loc of button short name of me to ml
end MouseStillDown

on mouseUp
  set the hilight of me to false
end mouseup

```

## 8.7 Slutsatser

- Hypercard är ett verktyg som passar bra för att realisera prototyper tack vare att utvecklingsmiljön och slutanvändarmiljön är integrerade. Dessutom ger kort-metaforen möjligheter att snabbt bygga upp olika alternativa lösningar för gränssnitten.
- Till nackdelarna med Hypercard hör programmeringsspråket Hypertalk. Det är så pass speciellt att det tar tid att lära sig det ordentligt.
- Hypercard sparar kontinuerligt programmet under utveckling och körning. Detta kan leda till att misstag inte går att ångra i efterhand.
- Att använda Hypercard för annat än prototyper och små program är förenat med vissa svårigheter. Dels uppstår det problem med underhållet eftersom det är svårt att överblicka en större mängd Hypertalk-kod, dels är det svårt att få godtagbara prestanda i större program. Trots dessa svårigheter har en del större program realiserats i Hypercard, t ex SISUs grafiska gränssnitt för databaser, Hybris, men då har stora delar av programmet flyttats till XCMD-funktioner.



# 9. Interface Builder

---

**Klassificering:** Layout-redigerare

**Gränssnittsstandard:** Nextstep

**Miljö:** Nextstep

**Version:** 3.0

**Tillverkare:** Next

**Leverantör:** Computer Nextcenter

Interface Builder är en del av den miljö för utveckling av program som medföljer operativsystemet Nextstep. Detta system är unikt så till vida att det är helt objektorienterat. Nextstep levereras och utvecklas av Next Inc. i USA och används först och främst på datorer av egen tillverkning. Dessa kommer dock att sluta tillverkas och ersättas med Intel 486-PC-datorer.

**objektorienterat**

Nextstep har lanserats av Next framförallt som en effektiv objektorienterad programutvecklingsmiljö. Nextstep har också till viss del blivit förebilden för Cairo (en objektorienterad Windows) och Taligent (ett samarbete mellan IBM och Apple som ska resultera i ett objektorienterat operativsystem).

Programutvecklingsmiljön i Nextstep består huvudsakligen av fem delar:

- Layout-redigeraren Interface Builder
- Application Kit, ett klassbibliotek som bl a innehåller interaktionsobjekt
- Database Kit, ett klassbibliotek som innehåller databasgränssnitt
- Project Builder som hanterar utvecklingen av program, ser till att rätt källkod kompileras vid uppdateringar osv
- En Objective-C-kompilator

Detta kapitel behandlar layout-redigeraren Interface Builder. Övriga komponenter i utvecklingsmiljön i Nextstep kommer enbart att behandlas översiktligt.

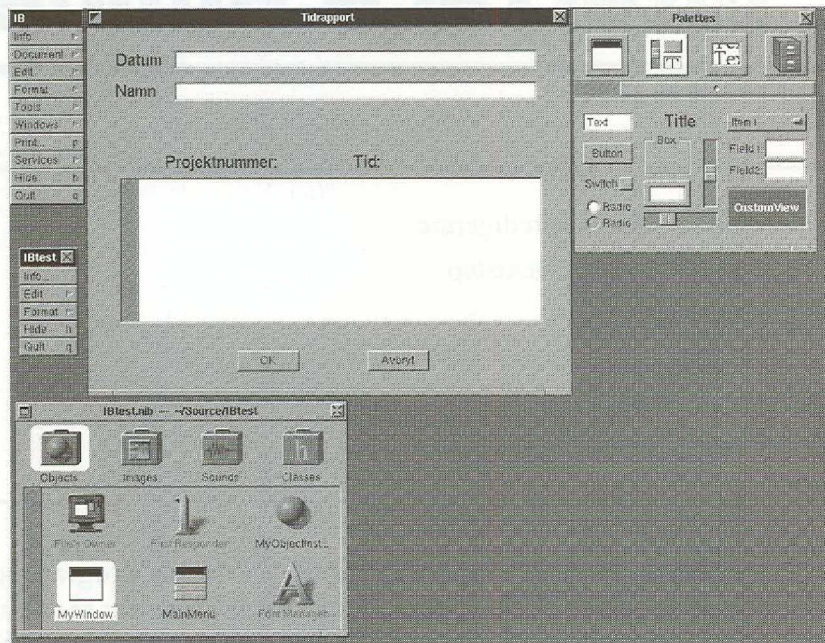


Bild 31. Interface Builder

## 9.1 Layout

### standardobjekt för Nextstep

I Interface Builder finns det tillgång till ett palettfönster som innehåller ett antal paletter med interaktionsobjekt (se bild 32). Alla standardobjekt för Nextstep finns med, även fönster av olika typer. Förutom dessa finns även mer avancerade interaktionsobjekt som ordbehandlare, postscript-objekt och databasobjekt på paletterna.

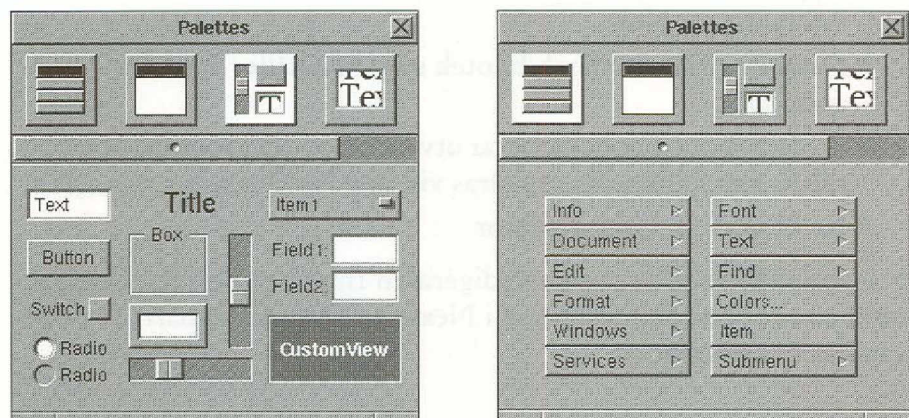


Bild 32. Paletter i Interface Builder



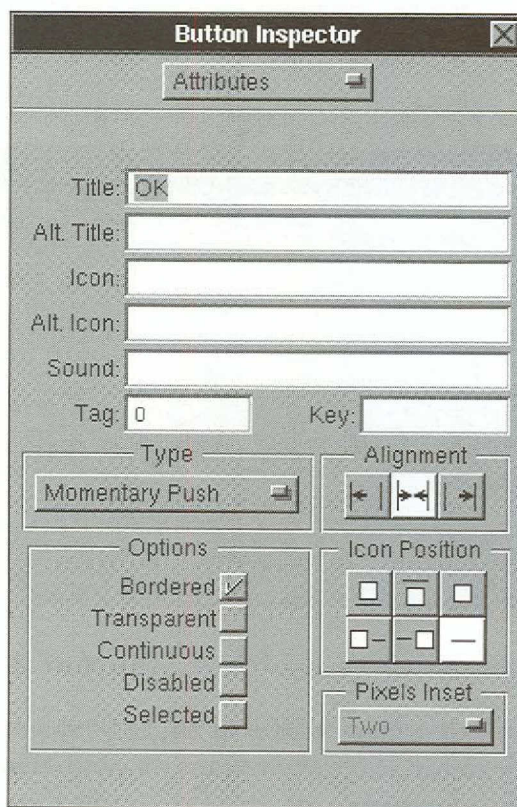
**dra-och-släpp**

**objektens  
egenskaper**

Utplaceringen av interaktionsobjekt sker helt enligt dra-och-släpp. För att skapa en knapp i ett fönster tar utvecklaren en knapp från paletten och drar den till dess plats i fönstret.

Alla interaktionsobjekt som skapas i Interface Builder har ett antal egenskaper som kan förändras med Inspector-verktyget (se bild 33). Vilka egenskaper som kan ändras beror på vilket interaktionsobjekt det rör sig om. Några typiska egenskaper är storlek, position och namn.

Det är också möjligt att definiera hur interaktionsobjektet ska förändras när det omgivande fönstrets storlek ändras. Det är mycket enkelt att få ett textfält att alltid ligga i nedre kanten av ett fönster.



*Bild 33. Inspektör, Interface Builders egenskaps-redigerare*

I Interface Builder finns det funktioner för att justera interaktionsobjekt gentemot varandra. Detta medför att det är enkelt att få ett symmetriskt användargränssnitt. Det finns även funktioner för att justera interaktionsobjektens storlek gentemot varandra.

För att testa ett användargränssnitt finns det ett enkelt gränssnittstest som aktiveras genom ett menyval. I detta test uppför sig de interaktionsobjekt som finns i gränssnittet med de funktioner de har. Det är t ex möjligt att utnyttja ordbehandlaren. Däremot fungerar inte de objekt som inte tillhör de fördefinierade.

**enkelt att få  
symmetri**

**gränssnittstest**



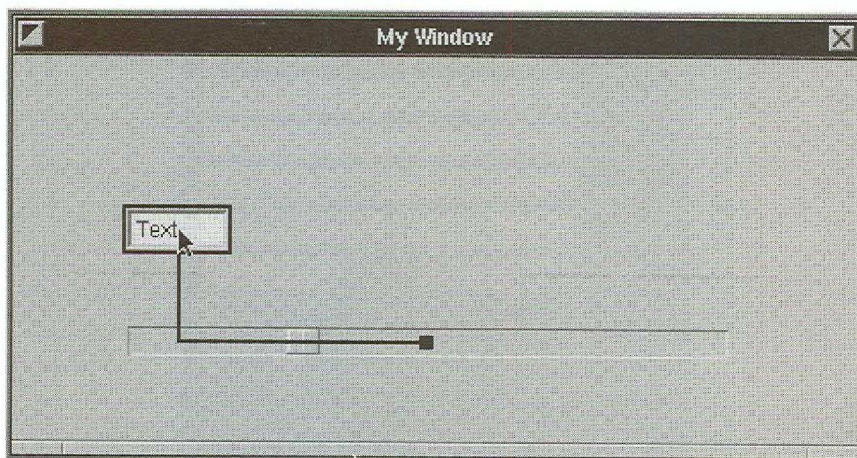
### lägga till egna objekt

Det finns även möjligheter att lägga till egna interaktionsobjekt på paletterna. Då krävs det dock att utvecklaren skriver extra kod som hör till objektet så att Inspector-verktyget kan hantera objektets egenskaper. I och med detta fungerar dessutom objektet när gränssnittstestet körs. Det finns också färdiga interaktionsobjekt att köpa från fristående leverantörer.

## 9.2 Beteende och koppling till kod

### ärver kod och beteende

I och med att miljön är helt objektorienterad ärver interaktionsobjekten en hel del kod och beteende, t ex krävs det ingen programmering för att få ordbehandlingsobjektet att fungera, utan beteendet ärvs.



*Bild 34. Skapande av objektlänk i Interface Builder*

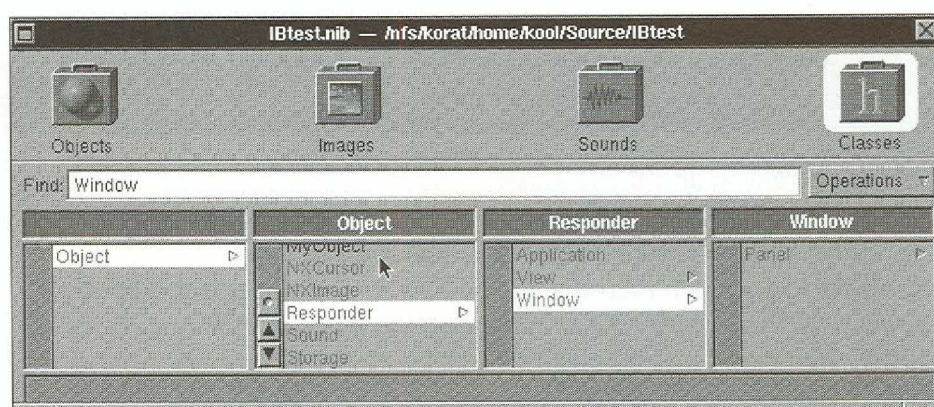
### objektlänkar

Ett centralt begrepp i Interface Builder är något vi valt att kalla för objektlänkar. Utvecklaren kan enkelt skapa en objektlänk mellan två objekt i Interface Builder genom att grafiskt koppla ihop dem på skärmen (se bild 34). Sedan väljs vilket meddelande som ska skickas längs den skapade objektlänken, t ex "getValueFrom". Poängen med detta är att varje gång ett interaktionsobjekt förändras, skickas meddelandet längs objektlänken. Exempelvis är det enkelt att realisera ett textfält som avspeglar värdet av ett skjutreglage. Det som krävs är en objektlänk från skjutreglaget till textfältet med meddelandet "SetValue". På de objektlänkar som definierats fungerar även gränssnittstestet. Det är också enkelt att infoga objektlänkar i de klasser som utvecklaren definierar själv. Att utnyttja objektlänkar innebär att en stor del av initieringsarbetet i programmet försvinner och, framför allt, att det blir enklare att återutnyttja objekt i andra program.



Interface Builder innehåller en klassbläddrare (class browser) som visar klasshierarkin (se bild 35). Trots detta är det svårt att få en överblick av klasshierarkin. Klassbläddraren innehåller också funktioner för att skapa subklasser till befintliga klasser. Det är t ex fullt möjligt att skapa en knapp med ett speciellt beteende genom att skapa en subklass till klassen knapp. För de nya subklasserna är det också möjligt att definiera metoder. Det finns dock inga funktioner i Interface Builder för att skriva programkod men det är möjligt att skapa mallar för programmeringsspråket Objective C till den nya klassen.

Det är också möjligt att i klassbläddraren läsa in definitioner av egna fördefinierade objekt för att sedan använda dem i Interface Builder. För att kunna använda objekt som skapats tidigare behövs en s k "header"-fil som innehåller definitioner och format för metoodanrop.



*Bild 35. Klassbläddraren*

I klassbläddraren finns funktioner för att skapa nya instanser till en klass. Saknar den nya instansen grafisk representation i användargränssnittet finns den trots allt representerad med en symbol i den s k Objects för att det ska vara möjligt att skapa objektlänkar till denna.

## 9.3 Arbetsätt

För att realisera ett program i Interface Builder och den övriga programutvecklingsmiljön i Nexstep börjar utvecklaren med att definiera det användargränssnitt som ska användas. Först placeras fönster ut och i dessa placeras de interaktionsobjekt som ska ingå.

De klasser med tillhörande metoder som saknas för programmet definieras med hjälp av klassbläddraren. Finns det tillgång till fördefinierade

**objektlänkar håller  
samman programmet**

klasser läses dessa definitioner in med hjälp av klassbläddraren. Från dessa klasser skapas de instanser som behövs i programmet.

Interaktionsobjekten och andra instanser kopplas sedan ihop med objektlänkar. Länkarna bildar det sammanhållna programmet. Det är möjligt att i detta läge testa användargränssnittet. En begränsning är dock att endast de objekt som finns fördefinierade i Interface Builder fungerar. De fördefinierade objekt som lästs in med hjälp av klassbläddraren fungerar inte.

**Objective C-mallar**

För att få ett fungerande program måste Objective C-mallar skapas för de nya klasser som ingår i programmet. Naturligtvis måste sedan mallarna fyllas i med Objective C-kod. Detta görs utanför Interface Builder med hjälp av en kod-redigerare.

**kompilering**

Kompileringen av programmet görs i den del av utvecklingsmiljön som kallas Project Builder. Den ser till att de förändrade källkodsfilerna kompileras och sätts samman till ett körbart program.

**dålig överblick**

## 9.4 Hjälp

Det finns direkt hjälp till alla klasser i Application Kit och om Interface Builder. Denna hjälp är identisk med de manualer som medföljer utvecklingsmiljön. Trots denna dokumentation är det besvärligt att överblicka klasser och metoder för de klasser som finns definierade i Application Kit. Till viss del avhjälpas detta av klassbläddraren i Interface Builder, men problemet kvarstår.

Interface Builder innehåller inga färdiga funktioner för att knyta hjälp till det program som utvecklas.

**bra prestanda**

## 9.5 Flyttbarhet och prestanda

Program som är realiserade med hjälp av Interface Builder och programutvecklingsmiljön i Nextstep är inte flyttbara till andra plattformar. Dock ska Nextstep bli tillgängligt på Intel-plattformen och programmen ska då gå att flytta till denna plattform enligt tillverkaren Next.

Prestanda är inget stort problem för program realiserade i Interface Builder, eftersom de är kompilerade. Deras prestanda kan jämföras med program utvecklade i andra jämförbara programutvecklingsmiljöer, t ex Microsoft C.



## 9.6 Prototypstöd

ej avsett för  
prototyper

Interface Builder är inte huvudsakligen avsett för att realisera prototyper med, utan är tänkt för att bygga fullständiga program i Nextstep-miljö.

Prototypstödet i Interface Builder är begränsat till gränssnittstestet. Trots detta blir stödet förhållandevis gott eftersom de fördefinierade interaktionsobjekten fungerar i testen. Dessutom ger objektlänkarna möjligheter att definiera en del av beteendet i programmet.

### 9.6.1 Tidrapporten

Tidrapporten realiserades med hjälp av interaktionsobjekt som knappar, textfält, inmatningsfält och en ordbehandlare (se bild 36). Funktionerna var begränsade till att textinmatningsfälten och ordbehandlaren fungerade. Indatakontroll saknades.

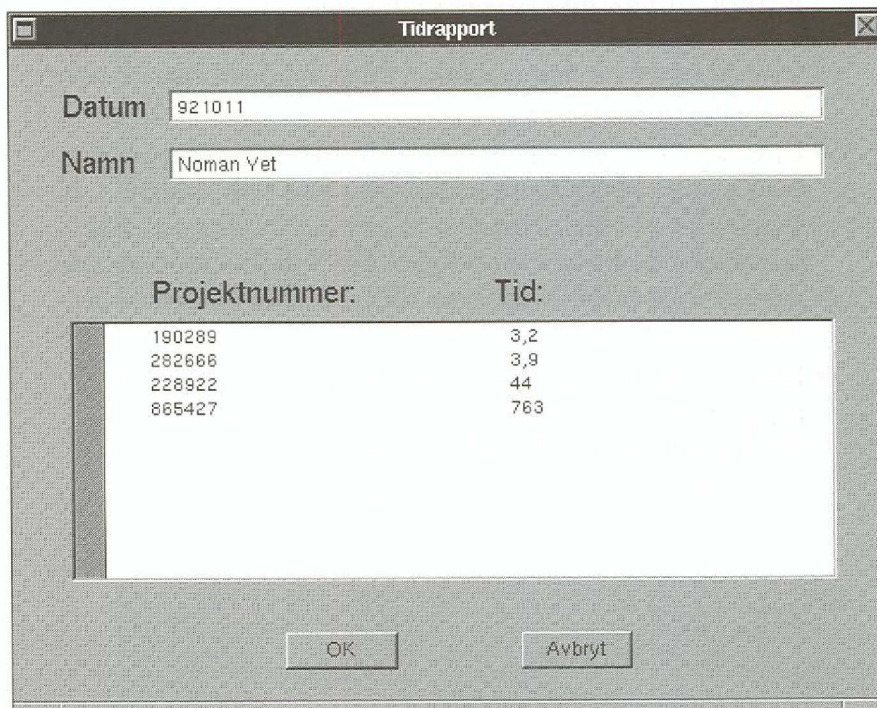


Bild 36. Tidrapporten under gränssnittstest i Interface Builder

## 9.6.2 Ritverktyget

Ritverktyget var inte möjligt att realisera i Interface Builder utan att realisera programmet med klasser och metoder i Objective C-kod. Vi valde därför att inte realisera prototypen utan istället enbart konstatera att Interface Builder inte framför allt är avsett för att utveckla prototyper.

## 9.7 Slutsatser

- Interface Builder är tillsammans med övriga delar av programutvecklingsmiljön i Nextstep ett kraftfullt verktyg, tack vare att miljön är objektorienterad.
- Några av problemen med den annars så önskvärda modulariseringen i objektorienterade utvecklingsmiljöer finns även i Nextsteps utvecklingsmiljö. För det första finns det ett stort antal klasser att hålla reda på och välja mellan. För det andra har alla klasser flera metoder. Detta medför att inlärningströskeln blir hög för att utnyttja utvecklingsmiljön på ett effektivt sätt. Kommer programutvecklaren över denna tröskel finns det dock stora vinster att hämta.
- Interface Builder i sig är en kraftfull layout-redigerare med många användbara funktioner. Speciellt kraftfullt är skapandet av objektlänkar som gör det möjligt att enkelt återanvända objekt som tidigare blivit definierade. Att det dessutom är möjligt att lägga till egna eller inköpta interaktionsobjekt är en stor fördel.
- Interface Builder är inte huvudsakligen avsett för att fungera som ett prototypverktyg. Trots detta är det möjligt att realisera förhållandevis komplicerade prototyper utan att skriva programkod.



# 10. Open Interface

**Klassificering:** Layout-redigerare och Flerplattformsvrtyg

**Miljö:** Macintosh, Windows, OS/2, Motif, Open Look, Decwindows

**Version:** 1.03

**Tillverkare:** Neuron Data

**Leverantör:** Duke Systems

Open Interface är ett av de nyare verktygen för flera plattformar som kommit ut på marknaden. Det understöder idag Macintosh, Motif, Open Look, Windows, Presentation Manager och DEC Windows. Även utvecklingsmiljön finns att tillgå på samtliga plattformar. Open Interface är utvecklat av Neuron Data i USA som tidigare är mest kända för sitt expertsystemskal Nexpert Object.

Open Interface består huvudsakligen av två delar:

- En *layout-redigerare*, Open Editor
- Ett *kodbibliotek*

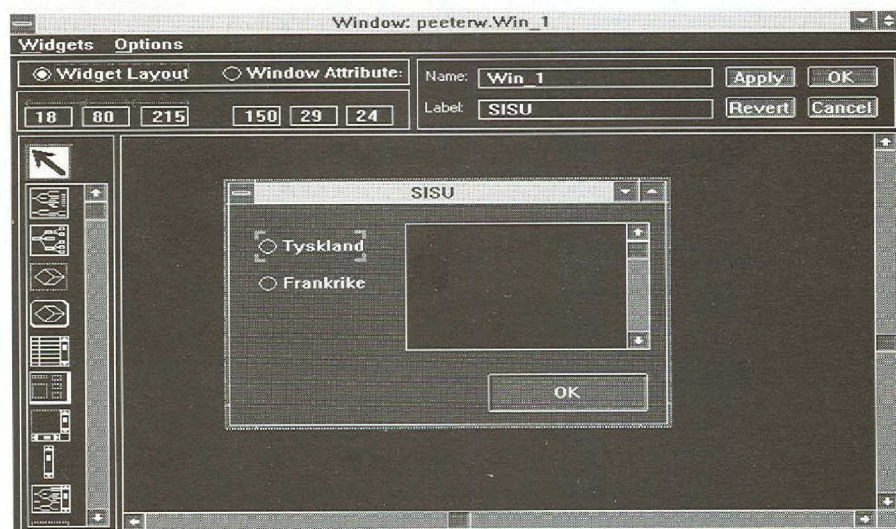


Bild 37. Open Editor, layout-redigeraren som följer med Open Interface



## hårdvarunyckel

Open Interface för Windows levereras med en hårdvarunyckel som måste sättas i skrivarpporten på utvecklingsmaskinen för att verktyget ska gå att köra.

## största gemensamma nämnaren

Ett stort generellt problem när man skapar verktyg som ska stöda flera standarder för gränssnitt är den så kallade "Största Gemensamma Nämnaren", SGN. Enkelt uttryckt kan SGN-problemet sägas vara: Hur ska man skapa ett gemensamt gränssnitt för programmering (API) där man även kan utnyttja de funktioner som inte är gemensamma för samtliga understödda standarder?

Open Interface försöker lösa SGN-problemet genom att realisera en union av alla understödda standarder i sitt eget kodbibliotek. Resultatet blir att utvecklaren kan använda interaktionsobjekt (knappar, rullningslistor, et c) oberoende av vilken standard de definieras i. Denna lösning är idag inte helt genomförd då det saknas några interaktionsobjekt, t ex uppnålningsbara menyer som definieras i Open Look.

Open Interface använder inte de färdiga kodbibliotek som finns för värdfönstersystemen utan istället fönstersystemet på en låg nivå. Exempelvis har man i realiseringen av Motif X inte använt sig av Motifs egna kodbibliotek utan istället Xlib direkt. Det medför att man egentligen har skapat sin egen realisering av gränssnittsstandarderna. Och inte nog med det. I och med att Open Interface realiserar en union av alla understödda standarder, måste Open Interface även se till att realisera och visualisera de interaktionsobjekt som saknas i värdstandarderna.

## 10.1 Layout

### layout-redigerare

Open Editor är den layout-redigerare som följer med Open Interface. I Open Editor definieras de fönster som programmet ska utnyttja och i fönstren placeras sedan interaktionsobjekten. Interaktionsobjekten finns tillgängliga på en palett (se bild 37).

### fördefinierade objekt

De fördefinierade interaktionsobjekt som följer med Open Editor är knappar, textfält, listboxar o s v. Däremot saknas skjutreglage och mätare. Utöver de vanliga interaktionsobjekten finns det några speciella objekt, som t ex redigerare för data som representeras i ett nätverk. - Dessa interaktionsobjekt behövs framför allt när man bygger expertsystem.

Förutom att definiera interaktionsobjekt kan utvecklaren även definiera resurser av typen strängar, mönster, färger o s v. Dessa resurser blir sedan åtkomliga i programmet.

Open Editor innehåller funktioner för att justera interaktionsobjekt i förhållande till varandra. Dessutom kan utvecklaren även definiera om

och hur interaktionsobjekten ska flyttas eller om storleken ska ändras om fönsterstorleken ändras. Utvecklaren kan t ex definiera att ett textfält förstoras i takt med fönstret.

Det finns möjligheter att lägga till interaktionsobjekt som man definierat själv och som sedan kan utnyttjas. Dessa objekt placeras också på den palett som finns.

Det finns ett enkelt provkörningsläge där utvecklaren kan prova sitt gränssnitt statiskt, d v s att knappar trycks in, att det går att skriva text i textfält, o s v. Dessutom är det möjligt att välja vilken gränssnittsstandard som ska användas vid testet. Det är fullt möjligt att få ett Motif-utseende i Open Editor trots att man använder Windows.

**definiera  
egna objekt**

**provkörningsläge**

## 10.2 Beteende och koppling till kod

Kod i Open Interface skrivs i C. Open Editor skapar en resursfil och mallar till ett C-program. Mallarna innehåller prototyper för de funktioner som hanterar händelser i gränssnittsobjekten.

För att definiera beteendet för ett interaktionsobjekt specialiseras C-mallarna på de platser som avser händelser för objektet. I funktionsprototyperna som skapats av Open Editor är det klart markerat var man ska lägga in egen kod.

Kodbibliotek i Open Interface tillhandahåller alla funktioner som krävs för att hantera ett användargränssnitt. Det innehåller helt enkelt de rutiner som vanligen finns i värddplattformens kodbibliotek (API). Med andra ord finns det ingen ytterligare funktion i Open Interface för att underlätta skapandet av program. Att använda verktyget kan jämföras med att skriva ett program som utnyttjar fönstersystemens kodbibliotek.

Open Interface tillhandahåller varken någon kod-redigerare eller kompilator. Istället ska utvecklaren använda någon av de vanliga miljöerna för utveckling av program som t ex Microsoft C.

**C-kod**

**kodbibliotek**

## 10.3 Arbetsätt

För att skapa ett program börjar utvecklaren med att, i Open Editor, definiera de fönster med interaktionsobjekt som behövs. Vid definierandet av fönstren är det också möjligt att prova användargränssnittet och dessutom se hur det ter sig i olika gränssnittsstandarder.



## C-mallar

När detta är klart skapar Open Editor en resursfil samt mallar till ett C-program. C-mallarna är förhållandevis kompletta i den bemärkelsen att de är möjliga att kompilera omedelbart. De funktioner programmet kommer att ha är identiska med de funktioner programmet har när gränssnittstestet utförs i Open Editor.

C-mallen används sedan för att definiera beteendet för interaktionsobjekten. Beroende på vilken miljö för utveckling av program som används, finns det tillgång till olika kod-redigerare och kompilatorer.

## C-kod översätts och länkas

När beteendet är definierat måste C-koden kompileras och länkas tillsammans med kodbibliotek i Open Interface.

Open Editor kan uppdatera förändrade C-mallar. Läger utvecklaren till ett nytt interaktionsobjekt i sitt användargränssnitt, är det inte nödvändigt att skapa helt nya C-mallar. Open Editor ändrar de befintliga även om de är förändrade.

När ett program ska distribueras måste den körbara (exekverbara) filen som fås efter kompileringen följa med. Dessutom krävs ett kodbibliotek för att kunna köra programmet. För att användaren ska få utnyttja detta kodbibliotek måste licensavgift betalas.

## 10.4 Hjälp

### direkt hjälp

I Open Editor finns det direkt hjälp. Däremot finns det ingen direkt hjälp för kod-bibliotek, vilket gör att det till en början blir mycket letande i handböcker. Att dessa är mycket tjocka och inte precis lättgenomträngliga underlättar inte.

I Open Interface finns det inte någon funktion för att realisera hjälpfunktioner i ett program. Det tillkommer systemutvecklaren att realisera på egen hand.

## 10.5 Flyttbarhet och prestanda

### välja gränssnittsstandard

Som tidigare nämnts försöker Open Interface lösa SGN-problemet genom att realisera en union av alla understödda standarder i sitt eget kodbibliotek. En bieffekt av detta är att utvecklaren kan bestämma vilken gränssnittsstandard han vill använda. Det är möjligt att använda t ex Motif på en Macintosh. Det finns dock ett avsteg. Macintoshs gränssnitt kan endast fås på en Macintosh. Detta beror på att Neuron Data inte vill bli stämda för att kopiera Macintoshs användargränssnitt på andra plattformar.



Ett problem som uppstår då man utvecklar ett program med hjälp av Open Interface är att välja interaktionsobjekt. Det är inte önskvärt att introducera främmande interaktionsobjekt för användaren. Kort och gott kan man säga att Open Interface tyvärr inte följer någon stilguide.

**följer ej stilguide**

Open Interface saknar metod att hantera de mer avancerade funktioner som kan finnas i fönstersystemet. Som exempel på detta kan nämnas att Open Interface inte kan ta hand om DDE (Dynamic Data Exchange) och OLE (Object Linking and Embedding) i Windows. Utvecklaren kan visserligen utnyttja dessa funktioner genom att använda värdsystemets kodbibliotek, men då blir resultatet icke flyttbar kod.

**klarar ej  
DDE och OLE**

För att flytta sitt program mellan olika målmiljöer måste utvecklaren flytta resursfilen och C-källkoden till målmiljön. Sedan måste både resursfilen och C-programmet i värdsystemet åter kompileras. Open Interface innehåller stöd för att dölja skillnader mellan olika realiseringar av C-kompilator.

**flytta program**

Prestandaproblem med program som är realiserade med Open Interface liknar problemen med de program som realiserats i vanliga miljöer för utveckling av program av typen Microsoft C.

## 10.6 Prototypstöd

Open Interface innehåller ett begränsat prototypstöd. Det stöd som finns är begränsat till Open Editor. Det går att göra fönster-layout och att testa sitt gränssnitt statiskt, d v s att knappar trycks in, att det går att skriva text i textfält, o s v. Dessutom är det möjligt att välja vilken gränssnittsstandard som ska användas vid testning.

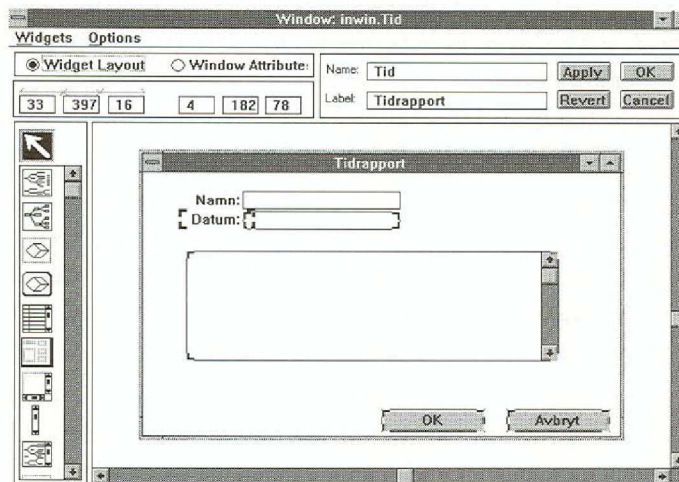
**begränsat  
prototypstöd**

Utöver detta saknar Open Interface stöd. Open Interface är helt enkelt inte i huvudsak avsett för att vara ett verktyg för att utveckla prototyper. Istället ska det ses som ett hjälpmedel för att få flyttbara program.

### 10.6.1 Tidrapporten

Tidrapporten gick att realisera i Open Editor som en något statisk prototyp. Vi byggde upp den med hjälp av knappar, textfält och statiska ledtexter. Trots att den blev förhållandevis statisk fyllde den ändå kraven för ett färdigt program eftersom det trots allt var möjligt att fylla i textfälten samt att trycka på knapparna.

**statisk prototyp**



*Bild 38. Tidrapporten under uppbyggnad i Open Editor*

## 10.6.2 Ritverktyget

Ritverktyget var inte möjligt att realisera i Open Interface utan att skriva en mängd C-kod. Vi valde därför att inte realisera prototypen utan istället enbart konstatera att Open Interface inte är ett verktyg för prototyper som innehåller dra-och-släpp.

## 10.7 Slutsatser

- Open Interface kan vara ett intressant alternativ om det finns ett behov av att realisera ett program som ska vara flyttbart mellan olika plattformar. Annars har detta verktyg egentligen inga andra fördelar framför andra verktyg.
- Möjligtvis kan en fördel vara layout-redigeraren Open Editor. Den har en hel del användbara funktioner som t ex test av gränssnitt och möjligheten att definiera hur interaktionsobjekt ska bete sig då fönstret ändrar storlek.
- En nackdel är att Open Interface realiserar fönstersystemen själv. Detta gör verktyget känsligt för uppdateringar av värdfönstrets system. Dessutom realiserar inte Open Interface program till programkommunikation, t ex DDE och OLE. Detta innebär att det ändå kan vara svårt att realisera program som verkligen blir lätt flyttbara.
- Open Interface är definitivt inte ett verktyg som kan användas för att snabbt ta fram prototyper som innehåller någon form av funktioner.



# 11.Plus

---

**Klassificering:** Lekmannaverktyg och Totalverktyg

**Gränssnittsstandard:** Macintosh

**Miljö:** Macintosh, Windows

**Version:** 2.5.1

**Tillverkare:** Spinnaker Software

**Leverantör:** Idea AB

Plus är ett verktyg för att utveckla program för Macintosh och Windows. Det är utvecklat av Spinnaker Software. Verktöget är en sk Hypercard-klon, och är, liksom Hypercard, ett kortbaserat verktyg. Sådana verktyg karaktäriseras av att de bygger på en kortmetafor, man talar om buntar, kort, bakgrund o s v. Största skillnaden mellan Plus och Hypercard är att Plus innehåller fler interaktionsobjekt.

## Hypercard-klon

Det finns en något modifierad variant av Plus som säljs under namnet Oracle Card. Den är speciellt avsedd för att användas tillsammans med Oracles databashanterare.

Plus består huvudsakligen av två delar:

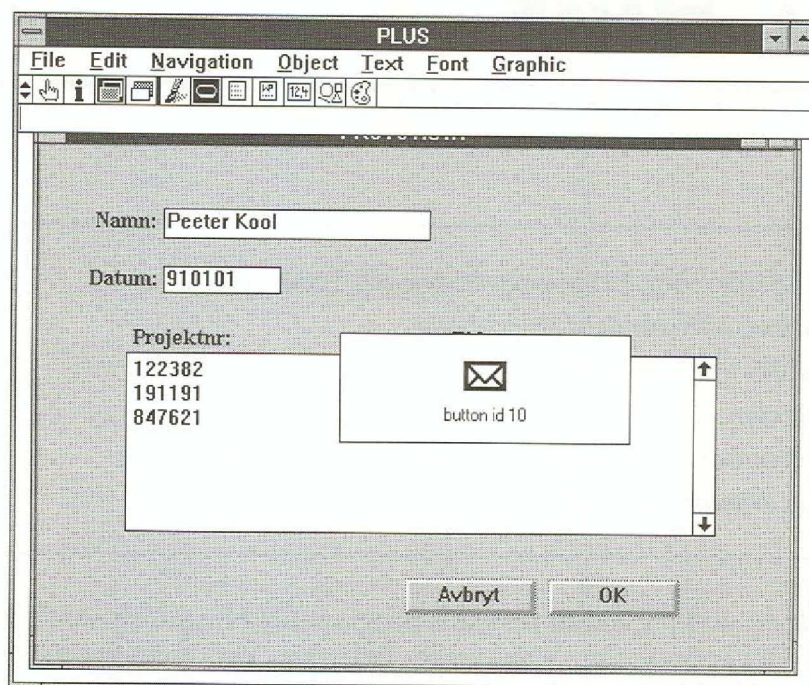
- Ett *programmeringsspråk*, PPL (Plus Programming Language) som är en utvidgning av Hypertalk som ingår i Hypercard.
- En *slutanvändarmiljö* som också är *utvecklingsmiljön*. Plus skiljer alltså inte på utveckling och körning..

## 11.1 Layout

Ett användargränssnitt byggs upp genom att utvecklaren organiserar kort i buntar. Det är enbart möjligt att se ett kort i taget från en bunt. På kortet placeras interaktionsobjekt och grafik.

Korten är i sin tur uppdelade i två skikt lagda på varandra, förgrunds- och bakgrundsskikt. Bakgrundsskiktet kan vara gemensamt för flera kort och det utnyttjas för de interaktionsobjekt och den grafik som ska användas på flera kort.





*Bild 39. Plus utvecklingsmiljö med verktygsbalk*

**sex typer av interaktionsobjekt**

Interaktionsobjekten i Plus skapas antingen genom ett menyval eller genom att dubbelklicka i verktygsbalken (se bild 39). Plus innehåller sex typer av interaktionsobjekt: knappar, textfält, ordbehandlare, datafält, grafiska objekt och bildobjekt. Dessa sex typer är i sin tur uppdelade: en knapp kan t e x vara en knapp, en radioknapp eller en checkbox. När ett nytt interaktionsobjekt skapats får det ett standardutseende. Till interaktionsobjekten hör ett antal egenskaper, t e x utseende, namn, typsnitt. Förutom de fördefinierade interaktionsobjekten är det möjligt att skapa egna genom att använda bildobjektet.

Det finns inga funktioner för att justera interaktionsobjekt inbördes. Utvecklaren är hänvisad till att justera på fri hand.

**ritverktyg**

Verktygsbalken innehåller, förutom interaktionsobjekt, även ritverktyg. Ritverktygen används bl a för att skapa statisk grafik i användargränssnittet. Handverktyget i balken används för att övergå till körningsläge.

## 11.2 Beteende och koppling till kod

**liknar engelskt talspråk**

Plus innehåller ett eget programmeringsspråk, PPL. PPL är en utvidgning av språket Hypertalk i Hypercard. Liksom Hypertalk försöker PPL efterlikna det engelska talspråket. Om t e x variabeln x ska tilldelas värdet tio, uttrycks det i PPL-syntax som "put 10 into x". Detta försök till likhet har gjort PPL rörigt och pratigt, vilket medför att det krävs tid

för inlärning för att kunna utnyttja PPL effektivt. En fördel är dock att PPL är ett otypat språk, vilket gör det enkelt att realisera generella funktioner.

**otypat**

Beteendet hos interaktionsobjekt bestäms genom att utvecklaren skriver PPL-kod som knyts till interaktionsobjektet. För interaktionsobjekten finns det ett antal meddelanden (händelser) som kan specialiseras, t ex mouse up, mouse down och enter key.

**objektens beteende**

Naturligtvis finns det funktioner i PPL för att förflytta sig mellan olika kort, bakgrunder och buntar eftersom Plus bygger på en kort-metafor.

Förutom att knyta kod till interaktionsobjekt är det möjligt att även knyta kod till kort, bakgrunder och buntar. I Plus finns en specificerad väg som meddelanden går. När t ex musknappen släpps upp på ett interaktionsobjekt skickas meddelandet "mouse up" först till objektet. Om inte objektet tar hand om meddelandet fortsätter det till kortet för att därifrån gå vidare till bakgrunden och sist till buntan.

Ett problem som uppstår på grund av att kod kan knytas till olika objekt är att överblicken förloras. Plus innehåller ingen hjälp för att hitta var en specifik funktion finns realiserad. I större Plus-program är det svårt att hitta var ett visst givet kodavsnitt finns, vilket medför att programmet blir svårunderhållet.

**dålig överblick**

För att skriva kod i Plus finns det tillgång till en kod-redigerare. Kod-redigeraren är av enkel typ och innehåller funktioner av typen sök. Det finns dessutom möjligheter att få mallar för de händelser och anrop som finns inlagd i koden i PPL. Det görs även en automatisk inskjutning (indentering) av kod.

**kod-redigerare**

Det saknas möjlighet att bygga upp en menybalk med menyer i Plus. Däremot är det möjligt att bygga upp något som liknar en menybalk med hjälp av knappar och en inbyggd funktion för dyk-upp-menyer.

Att dynamiskt skapa nya interaktionsobjekt i Plus är enkelt. För att t ex skapa en ny knapp anropar utvecklaren menykommandot "skapa ny - knapp". Alla interaktionsobjekts egenskaper, inklusive kod, kan förändras under körning från PPL-kod. Dessutom är det möjligt att utnyttja alla ritverktyg i PPL-koden.

**skapa nya objekt dynamiskt**

Det finns också möjligheter att bygga ut PPL med egna funktioner. För Plus på Macintosh är det möjligt att utnyttja samma typ av funktions-tillägg som för Hypercard, s k XCMDS (External Commands). XCMDS är program som skrivits i t ex C och sedan kompilerats. När väl ett XCMD är installerat i Plus så fungerar det som ett vanligt funktionsanrop. För PLUS under Windows finns motsvarande möjligheter tillgängliga genom att utnyttja DLL-bibliotek (Dynamic Link Libraries).

**XCMD**



**svårt rätta till  
misstag**

Plus sparar kontinuerligt programmet både under utveckling och körning. Detta kan leda till att misstag kan bli svåra att rätta till. Om man t ex raderar en knapp av misstag och inte direkt väljer menyalternativet ångra, finns det ingen möjlighet att rätta knappen och dess kod.

En annan effekt av att Plus sparar kontinuerligt är att innehållet i textfält inte försvinner då programmet avslutas utan finns kvar nästa gång programmet startas. Detta gäller dock inte variabelvärden.

## 11.3 Arbetsätt

Ett program i Plus motsvaras av en eller flera buntar. Därför måste buntan eller buntarna skapas först. Sedan gäller det att bygga upp de kort och bakgrunder som bildar användargränssnittet. Interaktionsobjekten placeras på korten och bakgrunderna och objektens utseende ställs in. Den statiska grafiken byggs upp med hjälp av ritverktygen.

**kod knyts till  
olika nivåer**

Interaktionsobjektens beteende skapar utvecklaren genom att skriva kod för objektet med hjälp av kod-redigeraren. Mer generell kod som ska gå att nå från flera olika ställen knyts antingen till kortet, bakgrundsskiktet eller buntan. De funktioner som definierats på ett kort kan endast anropas direkt från detta kort. De funktioner som definierats på bakgrundsskiktet kan anropas från de kort som använder denna bakgrund och de funktioner som knyts till en bunt kan nås från alla kort i buntan.

**kan provköras  
i alla lägen**

Det är möjligt att i alla lägen provköra programmet i och med att utvecklingsmiljön och slutanvändarmiljön är desamma. När handverktyget används provkörs programmet.

I Plus finns inga funktioner för att avlusa programmet.

**stegvis utveckling**

Plus understöder stegvis (inkrementell) utveckling tack vare den integrerade utvecklings- och slutanvändarmiljön.

## 11.4 Hjälp

Med Plus följer två hjälpbuntar. En för själva Plus och en för programmeringsspråket PPL. Dessa buntar är uppbyggda med funktioner för Hypertext, d v s det går att följa referenser mellan ord i texten o s v. Hjälpbuntarna är användbara särskilt då de medföljande handböckerna är förhållandevis röriga.

För de program som utvecklas i Plus finns dock inga färdiga funktioner för att skapa hjälp. Det är tillkommer programutvecklarna att realisera dem.

## 11.5 Flyttbarhet och prestanda

I och med att PPL tolkas (interpreteras) vid körning av Plus-program kan det bli problem med prestanda. Redan i våra små testprototyper märktes detta. Speciellt i ritverktyget, där dra-och-släpp ingick, blev det eftersläpning mellan markörens position och objektets. Överlag får större program realiserade i Plus problem med prestanda.

Ett sätt att undvika dessa problem är att flytta ut delar av programmet i så kallade XCMD eller DLL-funktioner. Dessa kan anropas inifrån PPL-kod men är realiserade med hjälp av någon vanlig programmeringsmiljö, till exempel C.

För att distribuera ett Plus-program behövs de buntar som realiserar programmet och Plus i sig om inte användarna redan har tillgång till dessa. Det finns för övrigt en speciell run-time-version av Plus.

Plus innehåller funktioner för att flytta program mellan Plus i Windows-miljö och Plus i Macintosh-miljö. Det är enkelt att flytta program som enbart är realiserade med hjälp av kod i PPL. Det uppstår däremot problem om programmet utnyttjar XCMD- eller DLL-funktioner. Dessa går inte att flytta utan måste skrivas om för den aktuella plattformen.

**dåliga prestanda**

**flytta ut delar  
av programmet**

**flyttbart**

## 11.6 Prototypstöd

Plus passar utmärkt för att utveckla prototyper av flera orsaker. Slut användarmiljön är densamma som utvecklingsmiljön, vilket ger ett gott stöd för ett stegvist arbete. Dessutom är det möjligt att i programkod utnyttja alla funktioner som finns i utvecklingsmiljön, till exempel ritverktygen, skapa knappar och ta bort knappar. Ytterligare en egenskap som är värdefull vid arbete med prototyper är att innehållet i textfälten inte försvinner när programmet avslutas. Detta medför att det är enkelt att lägga upp och förändra testdata.

Plus har också en fördel i och med att programmeringsspråket PPL är otypat. Detta sparar en hel del tid vid prototyparbete eftersom det då inte behövs datakonverteringar.

**passar för  
prototyper**

### 11.6.1 Tidrapporten

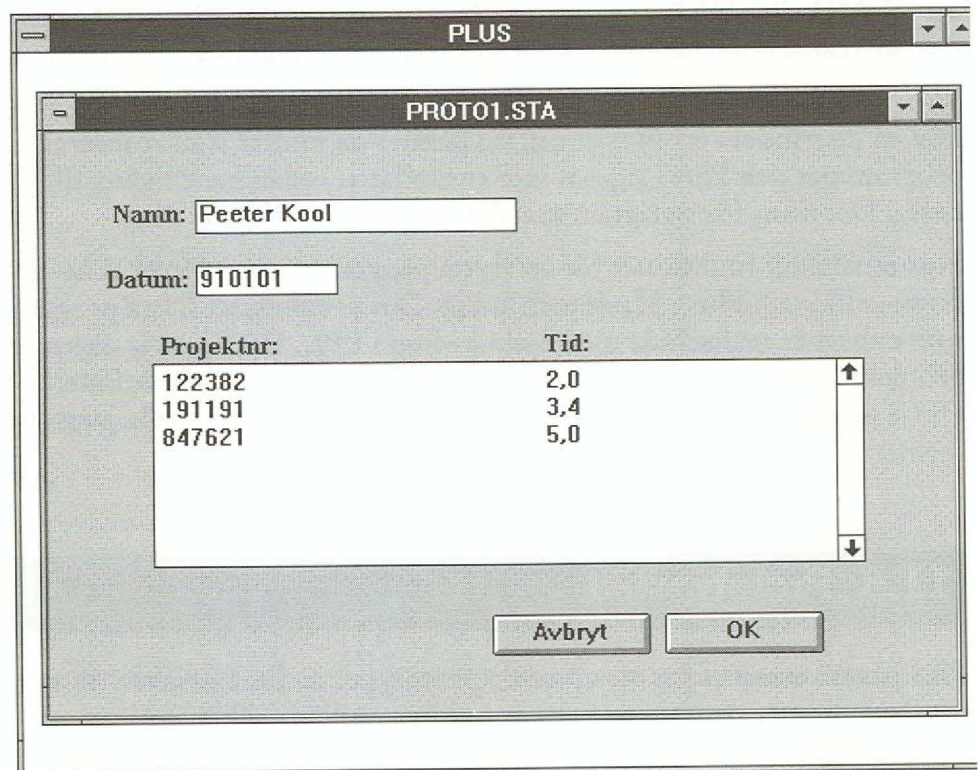
Tidrapporten byggde vi upp med hjälp av ett kort med knappar och fält. Ledtexterna byggdes upp av textfält där vi först skriver in ledtexten för



att sedan låsa den. Indatafälten byggdes upp av textfält som inte var låsta. Indatakontrollen i textfälten görs när användaren trycker på OK-knappen.

I och med att innehållet i textfält inte försvinner mellan körningar av Plus-program, innehåller ritverktyget en funktion som "radera textfältens innehåll" när programmet startas.

Sammanfattningsvis var det enkelt att realisera tidrapporten i Plus.

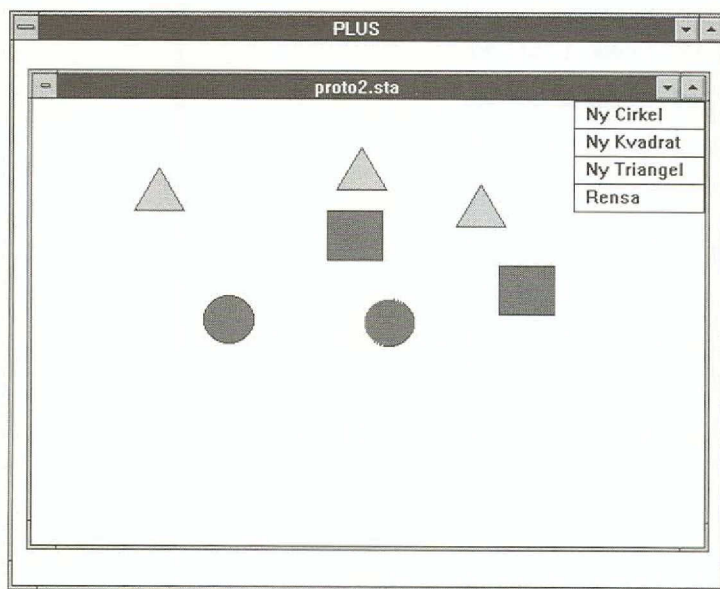


*Bild 40. Tidrapporten under körning*

### 11.6.2 Ritverktyget

Ritverktyget var betydligt besvärligare att realisera. Detta berodde till stor del på att Plus inte har färdiga funktioner för att utföra direktstyrning av typen dra-och-släpp. Koden för att utföra dra-och-släpp i ritverktyget (se kod nedan) blev förhållandevis omfattande och gav upphov till prestandaproblem.

**ej dra-och-släpp**



*Bild 41. Ritverktyget under körning*

Cirkeln, kvadraten och triangeln realiserades med hjälp av knappar som fick olika ikoner beroende på vilken typ av objekt de var.

Som tidigare nämnts är det besvärligt att skapa menyer i Plus. Istället realiserades menyn som knappar.

Sammanfattningsvis var ritverktyget svårt att realisera och arbetet krävde ingående kunskaper om Plus. Dessutom var prestanda dåliga. Det blev en ordentlig eftersläpning när vi drog ett objekt.

Nedan ges ett kodexempel på hur det är möjligt att realisera direktstyrning i Plus:

```

on MouseDown
  global gSelected, gDX, gDY
  set the highlight of me to true
  -- Spara positionerna för musklicket
  put item 1 of loc of button short name of me - item 1 of the
clickloc into gDX
  put item 2 of loc of button short name of me - item 2 of the
clickloc into gDY
end MouseDown

-- ...and drag object
on MouseStillDown
-- Dra objektet
  global gDX, gDY

```



```

    put the mouseLoc into ml
-- Normalisera positionen
    add gDX to item 1 of ml
    add gDY to item 2 of ml
-- se till att objektet inte dras utanför kortet
    put max(item 2 of ml,59) into item 2 of ml
    put min(item 2 of ml, 316) into item 2 of ml
    put max(item 1 of ml,32) into item 1 of ml
    put min(item 1 of ml, 470) into item 1 of ml
-- Flytta objektet
    set loc of button short name of me to ml
end MouseStillDown
on mouseUp
    set the hilight of me to false
end mouseup

```

## 11.7 Slutsatser

- Plus är ett verktyg som passar bra för att realisera prototyper tack vare att utvecklingsmiljön och slutanvändarmiljön är integrerade. Dessutom ger kort-metaforen möjligheter att snabbt bygga upp olika alternativa lösningar för gränssnitten.
- Plus kan vara ett intressant verktyg för program som ska realiseras för både Windows- och Macintosh-miljö.
- Till nackdelarna med Plus hör programmeringsspråket PPL. Det är så pass speciellt att det tar tid att lära sig det ordentligt.
- Plus sparar kontinuerligt programmet under utveckling och körning. Detta kan leda till att misstag inte går att ångra i efterhand.
- Att använda Plus för annat än prototyper och små program är förenat med vissa svårigheter. Dels uppstår det problem med underhållet eftersom det är svårt att överblicka en större mängd PPL-kod, dels är det svårt att få godtagbara prestanda i större program.

# 12.TeleUse

---

**Klassificering:** Layout-redigerare och UIMS

**Gränssnittsstandard:** OSF/Motif

**Miljö:** Plattform: DEC RISC, DEC VAX, HP Apollo X500, HP9000, IBM PS/2, IBM RS/6000, MS-DOS PC, Sun 3/4 Teleuse skapar C- och UIL-kod.

**Version:** 2.1

**Tillverkare:** Telesoft, USA

**Leverantör:** Alsys/Telesoft

Teleuse är något så ovanligt som ett svenskt system för hantering av gränssnitt (UIMS) som har haft stora framgångar internationellt. Det har blivit ett känt namn när det gäller utveckling av Motif-baserade program i Unix-världen.

Teleuse är ett klassiskt UIMS som skiljer sig från vanliga verktyg för gränssnitt genom att det inte bara används för att konstruera programmens gränssnitt utan också är aktivt under körning. Hela eller delar av programmets dialog hanteras av systemet, vilket gör det lättare att konstruera och förändra gränssnittet.

Teleuse består av fyra delar:

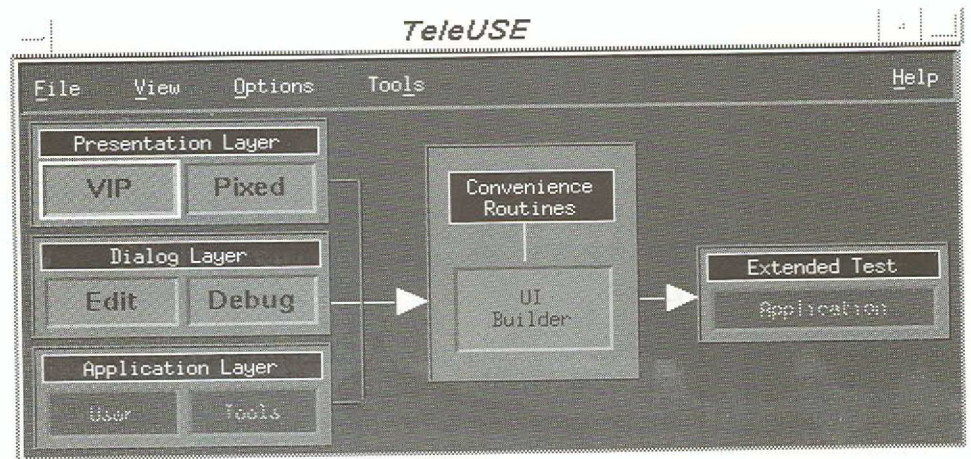
- *VIP* (Visual Interaction Programming), som är ett en layout-redigerare där användargränssnittets statiska delar specificeras.
- *Dialoghanteraren*, där händelser i gränssnittet behandlas med hjälp av det egna programmeringsspråket D. Ett händelsestyrt och regelbaserat språk som är framtaget för att beskriva interaktionen mellan programkoden och det grafiska gränssnittet och för att specificera programmets dialog med användaren. Till språket finns en översättare (kompilator), en tolkare (interpretator) och en avlusare (debugger).
- *Ett funktionsbibliotek*, som består av över 300 C-funktioner som kan anropas från D-kod och C-kod.

**svenskt system**

**klassiskt UIMS**



- *UI Buildern*, som är en funktion som med hjälp av Unix-verktyget "make" bygger samman programkoden och de olika delarna i Teleuse till ett körbart program.



*Bild 42. Introduktionsfönster med Teleuses struktur*

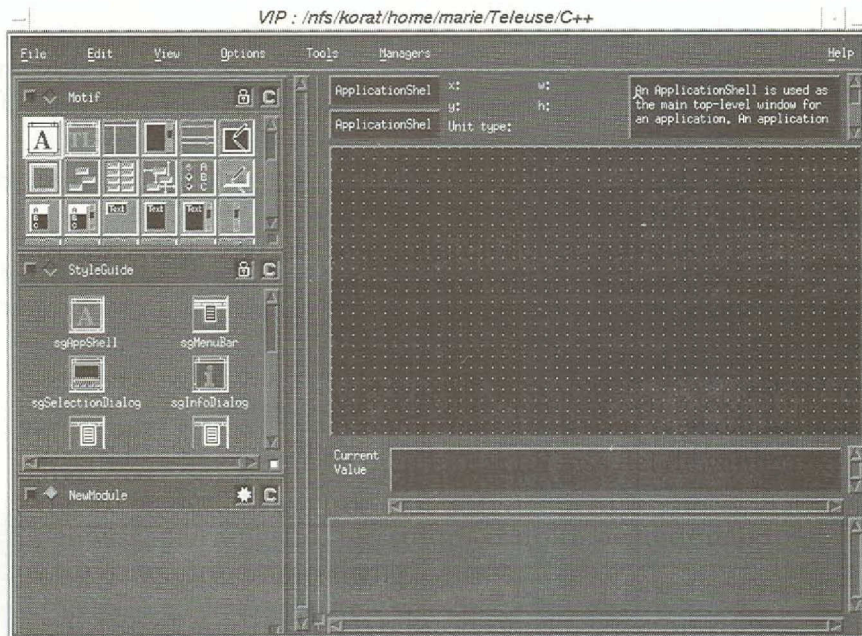
#### programkod

Utvecklingen av programkoden är skild från Teleuse. Utvecklingsmiljön för programkoden kan dock kopplas till Teleuse. Till exempel kan Centerlines Codecenter-miljö för C-programmering kopplas till Teleuse. Codecenter ger då stöd under programmeringsfasen och Teleuse stöder uppbyggandet av modulerna. Båda avlusarna kan användas tillsammans. Programkoden kan vara skriven i C, Ada eller C++.

## 12.1 Layout

#### fristående interaktionsobjekt

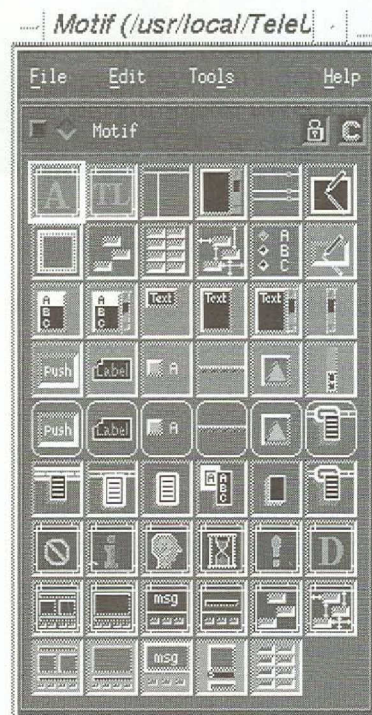
Vid utvecklingen av programmets presentationsdel används det interaktiva utvecklingsverktyget VIP (se bild 43). Interaktionsobjekten som används som byggklossar är fristående och finns lagrade i moduler, d v s paletter, som laddas i VIP. En grundmodul laddas automatiskt då man startar VIP (se bild 44). Modulen innehåller alla interaktionsobjekt enligt standardspecifikationen OSF/Motif samt några interaktionsobjekt som är specifika för Teleuse.



*Bild 43. Bild på VIP*

Interaktionsobjekten (widgets) presenteras som ikoner. De placeras på arbetsytan med direktstyrning. På arbetsytan visas gränssnittet med det utseende det kommer att få vid en körning av programmet.

**direktstyrning**



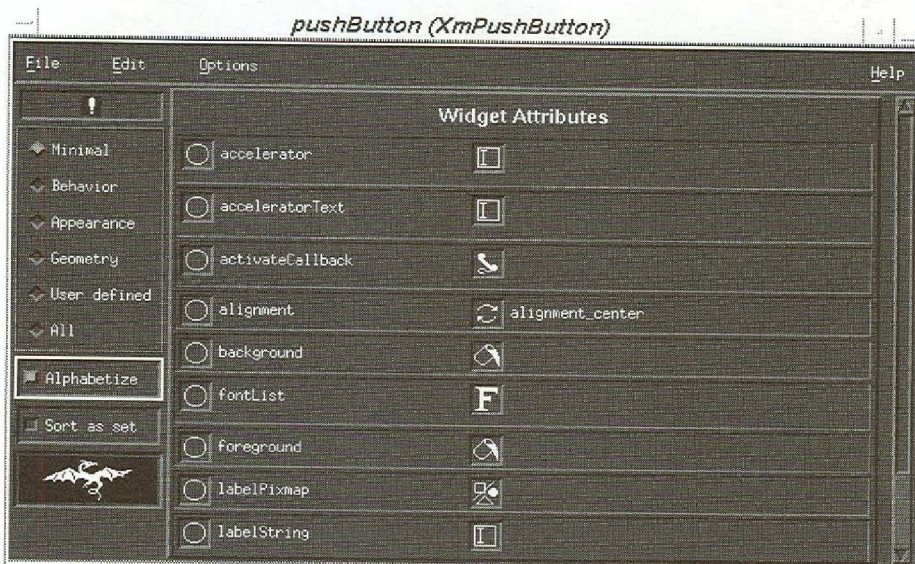
*Bild 44. Grundmodul med interaktionsobjekt enligt OSF/Motif*



## objektens egenskaper

För varje interaktionsobjekt i gränssnittet finns ett fönster där alla objektets egenskaper presenteras (se bild 45). Dessa egenskaper kan vara allt ifrån höjd (height) och namn (labelString) till olika händelser som objektet kan reagera på, t ex activateCallback. Fönstret används även för att modifiera egenskapernas värden. De möjliga värdena presenteras genom dyk-upp-menyer. Förutom de existerande egenskaperna för ett objekt kan man även definiera egna egenskaper i detta fönster.

Det krävs en viss träning för att veta vilka egenskaper som ska ändras för att gränssnittets layout ska bli den önskade. Det är till exempel inte självklart vad som menas med "mnemonic" för Motifs interaktionsobjekt XmLabel.



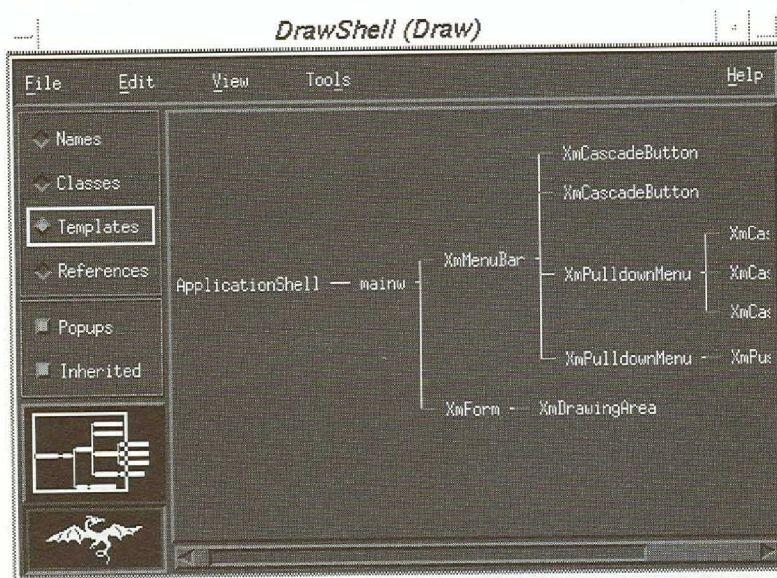
*Bild 45. Egenskapsfönstret*

## träddredigerare

Att utveckla ett grafiskt gränssnitt med Teleuse påminner mycket om Motifs sätt att arbeta med en hierarki av interaktionsobjekt. VIP erbjuder därför inte enbart arbetsytan med direktstyrning för att specificera gränssnittet utan också en träddredigerare, där gränssnittets hierarkiska struktur visas och redigeras i form av en grafisk trädstruktur över gränssnittets olika delar (se bild 46).

Träddredigeraren och arbetsytan kompletterar varandra och används parallellt för att redigera utseende och struktur.





*Bild 46. Bild på trädredigeraren*

I trädredigeraren ges en mycket god översikt över hur gränssnittets delar är kopplade. Redigeraren har en liten översikt som visar vilken del av den totala strukturen som användaren ser för tillfället.

**bra översikt**

Trädredigeraren gör det lättare att komma åt de delar i det statiska gränssnittet som inte är direkt åtkomliga på VIPs arbetsyta, t ex dyk-  
upp-menyer. Förutom namnet på gränssnittets delar kan trädredige-  
raren visa klasstillhörighet och arvsstruktur.

Teleuse har även en villkorsredigerare (constraints editor). I den finns det möjlighet att ange hur objektens placering och storlek ska ändras om fönstret de tillhör ändrar storlek under körning.

**villkorsredigerare**

Det går inte att få en utskrift av trädet eller gränssnittslayouten i Teleuse. För att få en bild av dessa måste man använda operativ-  
systemets funktion för skärmbildsutskrift.

I Teleuse finns möjligheten att dekorera gränssnittet med bilder. Det finns en pixelredigerare för att rita bilder. Till exempel kan man skapa egna ikoner till knappar eller rita en bild på företagets logo som sedan kan användas som dekor. Bilderna sparas i XPM-filer.

**dekor**

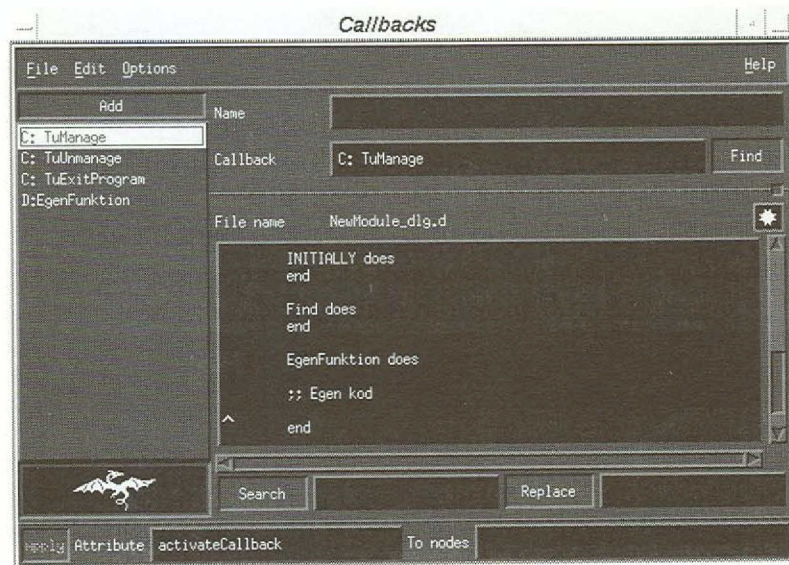
## 12.2 Beteende och koppling till kod

I egenskapsfönstret för interaktionsobjekten definieras, förutom utseendet hos objekten, även deras beteende, d v s vad som händer då objekten aktiveras och vilken funktion i koden som ska anropas. Här kan utvecklaren på traditionellt vis göra ett anrop (callback) till en funktion skriven

**objektens beteende**



i C-kod. I Teleuse kan utvecklaren även, liksom i andra UIMS, anropa dialogdelen.



*Bild 47. Kod-redigeraren för anrop i C-kod och D-kod*

#### dialogdel

Teleuses dialogdel kan skrivas i en kodredigerare kopplad till VIP (se bild 47) eller i en fristående kodredigerare. Där kan funktioner i både C- och D-kod skrivas.

I dialogdelen beskrivs programmets dialog genom programspråket D. Detta språk är framtaget för att underlätta hanteringen av händelser i programmet.

Den information som finns tillgänglig om en händelse vid ett traditionellt anrop till en C-funktion, d v s från vilket interaktionsobjekt händelsen blev aktiverad och vad som utlöste den, finns även tillgänglig då man anropar en D-funktion. Det är även möjligt att skicka med egna variabler i anropet.

#### enkel syntax

D-koden har en enkel syntax som gör det lätt att komma åt alla attribut till alla objekt i gränssnittet.

Om t ex programfönstret refereras av variabeln "pf", kan attributet "value" för ett objekt vid namn "Title" av klassen "XmText" refereras genom uttrycket:

```
val := pf->Title.value.
```

Som jämförelse är standardmetoden för att ta fram attributvärden för Motif-objekt i C-kod:

```
n = 0;
XtSetArg(args[n], XtNvalue, &val); n++;
XtGetValues(Title, args, n);
```

När ett interaktionsobjekt och dess egenskaper refereras i D-kod behöver inte hela arvskedjan anges. Det enda kravet är att objektnamnet, t ex Title, är unikt och exakt stavat som det från början definierats i VIP. Att så är fallet får utvecklaren själv kontrollera för Teleuse ger ingen sådan hjälp.

Trots sin enkla syntax kan D användas för att realisera tämligen stora delar av programdynamiken. Enkla funktioner som att addera tal i två inmatningsfält och presentera resultatet i ett utmatningsfält kan programmeras i D-kod.

Från D-koden kan nya interaktionsobjekt skapas under körning av programmet. Dessutom kan de existerande objektens attribut manipuleras. Man kan t ex få rörliga objekt i gränssnittet genom att ändra x- och y-koordinaterna.

I D-språket måste man deklarerera till vilken typ en viss variabel hör. Typerna kan t ex vara widget eller integer. Det finns dock möjlighet att anpassa språket genom att utöka uppsättningen typer. De egna typerna skrivs i C.

De funktioner som inte lämpar sig att programmera i D-kod, programmeras i C-kod. C-koden anropas på samma sätt som D-koden, d v s direkt från attributfönstren i Vip, eller från D-koden.

Teleuse innehåller även ett funktionsbibliotek skrivet i C. Där finns över 300 olika funktioner för att initiera och manipulera delar av programmet under körning. Funktionerna i biblioteket anropas framförallt från D-koden, men vissa anropas även från C-koden.

**nya objekt  
under körning**

**funktionsbibliotek**

## 12.3 Arbetsätt

Vid utvecklingen av ett gränssnitt i VIP öppnas först en ny tom modul. Till denna modul kopieras sedan från någon annan modul de interaktionsobjekt som behövs för det program som ska utvecklas. De kopierade interaktionsobjekten blir egna fristående objekt som kan modifieras för att passa det tänkta programmet. De nya objekten kallas för mallar (templates).

Mallarna kan även vara komplexa interaktionsobjekt, uppbyggda av flera grundläggande interaktionsobjekt, t ex en menybalk med några fördefinierade standardmenyer.

Vid layout av ett gränssnitt kan en kopia eller en instans av mallen fogas in i gränssnittet. En kopia blir ett fristående objekt. En förändring av mallen påverkar inte kopian.

**mallar**

**kopia**



### **instans**

En instans däremot kan påverkas av förändringar i mallen. Det avgörande är om instansens olika egenskaper har blivit förändrade i förhållande till mallens egenskaper. De attributvärden som inte blivit förändrade följer mallens värden. De värden som är förändrade följer inte mallen även om mallens motsvarande värden förändras. Ta t ex utseendet på en knapp. Mallens knapp är röd men instansens blå. Däremot är båda knapparna lika stora. Om man ändrar knappens färg i mallen behåller instansens knapp sin blå färg. Om man däremot ändrar knappens storlek i mallen följer instansens knapp detta nya värde. Resultatet beror på att färgens attributvärde men inte storlekens är ändrat i instansen.

Det finns även möjlighet att i mallen specificera vilka värden som måste följa mallens och vilka som får ändras.

### **förenklar ändringar**

Möjligheten att ärva attributvärden är mycket användbar eftersom det förenklar ändringar i gränssnittet. Till exempel kan man ändra färg och beteende på alla knappar i ett program enbart genom att ändra mallknappen.

### **rörig arvsstruktur**

Arvsstrukturen blir dock lätt oklar och rörig. Det finns inte något riktigt bra sätt att visa vilka objekt som är fristående kopior och vilka som är instanser.

Utseendet hos gränssnittet styrs av interaktionsobjekten, mallarna och deras attributvärden. Objektens placering bestäms med direktstyrning. För finjusteringen ändras x- och y-koordinaternas värden.

Sedan gränssnittet är uppbyggt, dialoghanteringen i D-kod är klar och eventuell programkod i C-kod framtagen, ska delarna byggas samman till en fungerande enhet. Detta sker med hjälp av informationen i en inställningsfil, uxb-filen. Filen innehåller t ex information om placeringen av viktiga kodbibliotek och i vilka filer programkoden ligger.

### **UI-buildern**

Programmet byggs samman av de olika komponenter som är specificerade i inställningsfilen med hjälp av UI-builder. UI-builder skapar de ytterligare filer som Teleuse kräver samt kompilerar och länkar delarna till ett program.

Teleuse har möjlighet att skapa programskelett till D-filen, C-filen och till uxb-filen utifrån den statiska delen som är gjord i VIP. Denna funktion skriver dock över den existerande filen utan att ta hänsyn till om kod har skrivits till de redan genererade reglerna.

## **12.4 Hjälp**

### **utförlig hjälp**

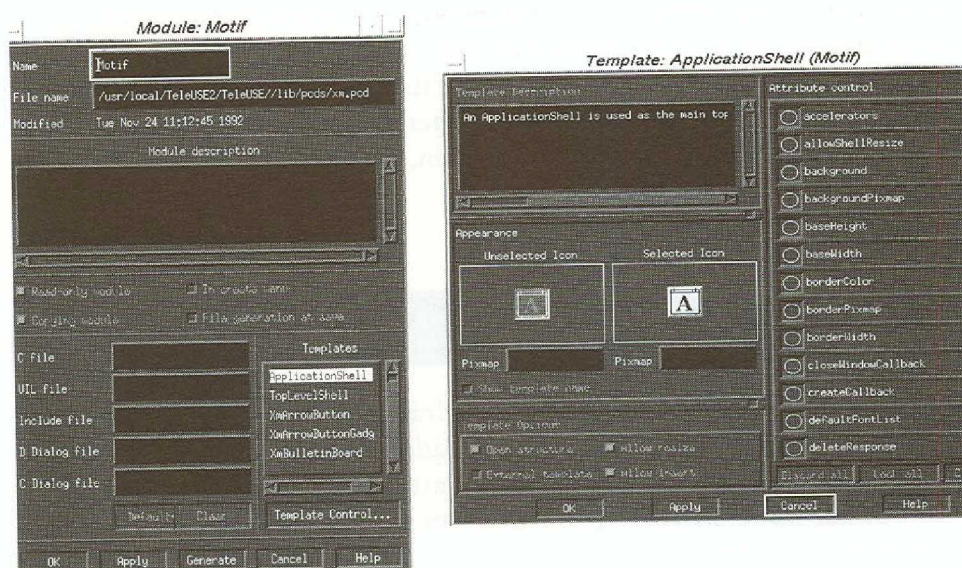
Hjälpfunktionen i Teleuse är utförlig och täcker de flesta områden. Det finns lättåtkomlig och utförlig hjälp om de delar som har att göra med funktionen hos själva verktyget. Till exempel finns hjälp för funktionen

hos de olika fönstren i VIP och en mer utförlig förklaring av de felmeddelanden som visas då användaren gör något otillåtet.

Teleuse har förutom funktionell hjälp även hjälp för interaktionsobjekten och deras attribut.

För varje modul av interaktionsobjekt finns en hjälpruta där modulen beskrivs och de filer, t ex C- och D-filer, som är kopplade till modulen är specificerade. Vidare finns en beskrivning för varje interaktionsobjekt i modulen. Med denna hjälpfunktion kan man även lägga in hjälp för egna mallar. Användaren kan skapa en egen ikon och ge en beskrivning av funktionen hos mallen. Möjligheten att lägga till egen hjälp är mycket användbar eftersom mallobjekten ska kunna återanvändas i andra program och av andra användare.

**lägga till egen hjälp**



**Bild 48.** *Hjälpfunktion för modulerna och deras interaktionsobjekt*

Användaren har förmodligen svårt att komma ihåg namn och parametrar för de över 300 funktionerna som finns inbyggda i Teleuse och som används i D-koden. Tidigare har det enda minnesstödet och den enda orienteringshjälpen varit manualen. I senaste versionen av Teleuse finns det dock datorstött hjälp i form av Unix-man-blad.

För utvecklaren är det även lätt att lägga in hjälptexter i det framtagna programmet. Programutvecklaren kan till exempel använda någon av de fördefinierade dialogobjekten från Motif, XmMessageBox och XmMessageDialog.

**Unix-man-blad**



## 12.5 Flyttbarhet och prestanda

### alla Unix-plattformar

Teleuse fungerar på alla vanliga Unix-plattformar.

Det färdiga programmet beskrivs i sin helhet av en UIL-fil och C-kod. UIL-formatet är standard för specifikationer av gränssnitt inom Motif. Detta innebär att det är möjligt att flytta ett program som är utvecklat med hjälp av Teleuse mellan olika Unix-plattformar. Dessutom medför det att det är möjligt att utföra underhåll med andra verktyg än Teleuse, t ex en ordbehandlare.

### kan läsa in UIL-filer

Enligt tillverkaren klarar Teleuse även att läsa in UIL-filer. Det innebär att Teleuse kan läsa in och modifiera statiska gränssnitt som är skapade med andra verktyg.

### stora program

De körbara programmen blir tämligen stora, minst 3 Mbyte, även i de fall då de bara innehåller ett fönster. Det beror på att ganska omfattande bibliotek från både Motif och Teleuse finns med. Storleken ökar dock inte nämnvärt när programmet har ett mer komplicerat gränssnitt. Storleken påverkar inte heller snabbheten, som hos de skapade prototypprogrammen var helt normal.

## 12.6 Prototypstöd

### kan ändras och provas direkt

Eftersom D-koden och specifikationen från VIP tolkas (intempreteras) vid utvecklingen av ett gränssnitt, kan både D-koden och VIP-specifikationen ändras och provas direkt, utan att programmet behöver byggas om. Förslag till ändringar i gränssnittet och dialogen kan på så sätt lätt provas av användaren.

När prototypen är klar, översätts den till ett färdigt program av UI-builder.

### logg-fönster

I VIP finns dessutom en testfunktion, med vars hjälp det är lätt att simulera alla viktiga funktioner hos gränssnittet som inte beror på D-koden eller programkoden. Till denna testfunktion finns det kopplat ett loggfönster. I detta visas vilka händelser som skulle ha sänts till D-koden respektive C-koden, om programmet körts på riktigt.

### 12.6.1 Tidsrapporten

De grundläggande objekten, såsom textfält och knappar, innebar inga problem. Här fanns redan färdiga objekt att tillgå. Att bygga en mer komplicerad struktur, såsom tabellen med rullningslist, var svårare. Det

stora problemet var att veta vilka interaktionsobjekt som ska användas och hur de ska kombineras.

Det bästa angreppssättet var att försöka bryta ner problemet i mindre delar. Raderna i tabellen kunde brytas ut. En ny mall som fick namnet Rad skapades och den innehöll två textfält i en XmRowColumn. Mellan textfälten förflyttade man sig med ett tabuleringsslag.

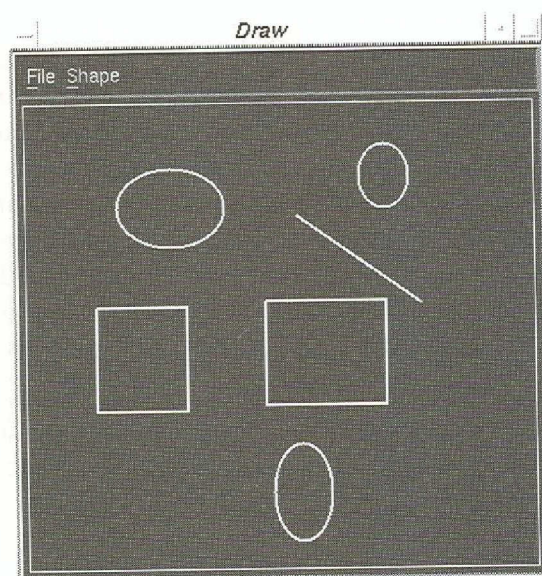
Tabellen byggdes sedan upp genom att stapla flera instanser av mallobjektet Rad i en ny XmRowColumn.

För att få tabellen och rullningslistan att interagera var det nödvändigt att skriva D-kod. Vid behov skapar D-koden nya instanser av Rad som läggs sist i tabellen.

**D-kod nödvändig**

## 12.6.2 Ritverktyget

Menyn, ritytan och dialogboxen innebar inga problem. Ritytan och dialogboxen fanns redan i Motifs grundmodul och menyn kunde byggas upp från de existerande grundprimitiverna i grundmodulen. Dokumentation och exempelbiblioteket i Teleuse innehåller exempel som visar hur menyer byggs upp.



*Bild 49. Bild på ritverktyget*

I ritytans attributfönster kan ett klick kopplas till aktiviteten att rita ett grafikobjekt genom händelsen activateCallback. Koden som hanterar markering, utritning och förflyttning av ett grafiskt objekt i ritytan måste skrivas i C.

**C-kod nödvändig**



## 12.7 Slutsatser

- För användare med stor erfarenhet av Motif är Teleuse lätthanterlig redan från början. En Motif-programmerare finner här alla interaktionsobjekt från Intrinsics och OSF/Motif, plus ytterligare några interaktionsobjekt i Teleuses egen objektmängd, Tuw.
- För en användare som saknar erfarenhet av Motif kan de olika interaktionsobjekten med namn som shell och widget och deras olika kombinationer verka förvirrande.
- I Teleuse måste användaren förstå vilka Motif-objekt som ska användas i en given situation och hur objekten fungerar och interagerar med varandra. Teleuse presenterar vilka möjligheter som finns, inte minst genom den utförliga hjälpen, men tar inte bort komplexiteten i Motif.
- Utvecklaren kan dock dölja Motifs komplexitet genom att bygga egna paletter med interaktionsobjekt som är specificerade på en högre nivå. Detta är också rekommenderat arbetssätt, framför allt i större projekt och när man vill skapa en standard för en organisation.
- Teleuse lämpar sig inte särskilt väl för ett objektorienterat språk, som t ex C++, i och med att programkoden är fristående från gränssnittet. Uppdelningen i programkod och gränssnitt står i motsats till det objektorienterade synsättet, som innebär att kod och gränssnitt är invävda i varandra. Många av fördelarna med ett objektorienterat språk går därför förlorade om det används tillsammans med Teleuse. Fördelarna med en uppdelning av kod och gränssnitt, inte minst vad gäller överskådligheten, överväger till viss del fördelarna med objektorienterad utvecklingsmiljö, framför allt i större utvecklingsprojekt.
- Teleuse är ett kraftfullt verktyg för att bygga program med Motif-gränssnitt och blir förmodligen mer användbart ju större programmeringsprojektet är.

# 13. Visual Basic

**Klassificering:** Totalverktyg och Lekmannaverktyg

**Gränssnittsstandard:** Windows

**Miljö:** DOS och Windows

**Version:** 2.0

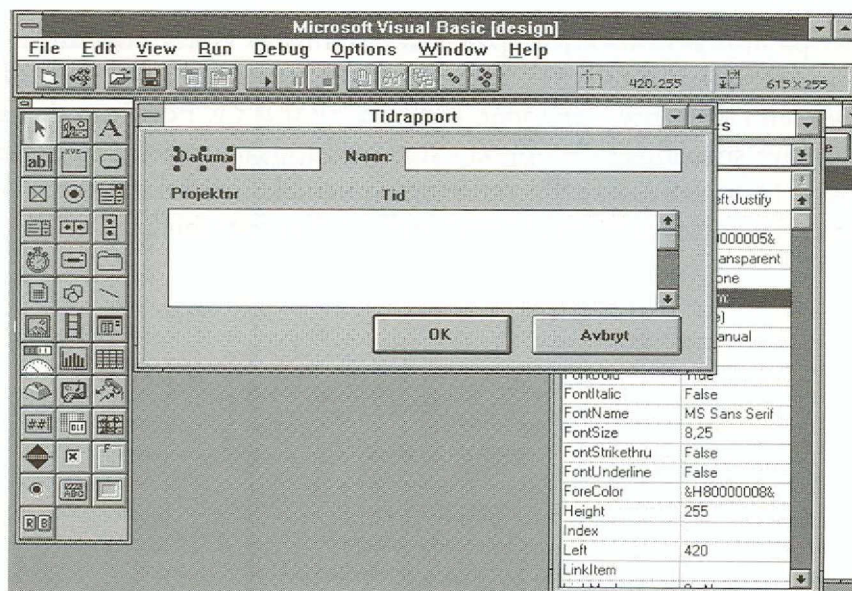
**Tillverkare:** Microsoft

**Leverantör:** Microsoft

Visual Basic är ett av de populärare utvecklingsverktygen för Windows. Det är utvecklat av Microsoft som har lanserat Visual Basic som ett verktyg för att skapa Windows-program.

Visual Basics utvecklingsmiljö består huvudsakligen av två delar:

- En *layout-redigerare* där utvecklaren kan välja mellan de fördefinierade interaktionsobjekten från en palett.
- Ett *programmeringsspråk*, Basic, som gör det möjligt att definiera interaktionsobjektens beteende.



*Bild 50. Programutvecklingsmiljön i Visual Basic*



**stöd för DLL, DDE,  
OLE och ODBC**

**olika versioner**

Det är tydligt att målgruppen för Visual Basic är utvecklare av program eftersom verktyget inte använder sig av någon enkel bruksmodell som t ex Hypercard med sin kortmetafor. Tröskeln för att använda verktyget är förhållandevis hög. Det krävs helt enkelt programmeringskunskaper för att utnyttja Visual Basic.

Visual Basic hör enligt Microsoft till en av deras mest strategiska produkter på Windowsmarknaden. Därför är understödet och integrationen av Visual Basic i Windows väl utbyggt. Visual Basic innehåller stöd för DLL (Dynamic Link Libraries), DDE (Dynamic Data Exchange), OLE (Object Linking and Embedding) och ODBC (Open Database Connectivity).

Visual Basic säljs i olika versioner beroende på hur mycket tillbehör som finns med. Den mest avancerade versionen, "Professional", innehåller bland annat en kompilator för att skapa hjälp enligt Windows-standard samt extra dokumentation. Dessutom medföljer fler interaktionsobjekt än i standardutgåvan.

Ett Visual Basic-program består av ett antal formulär (forms) med interaktionsobjekt samt moduler av generella funktioner.

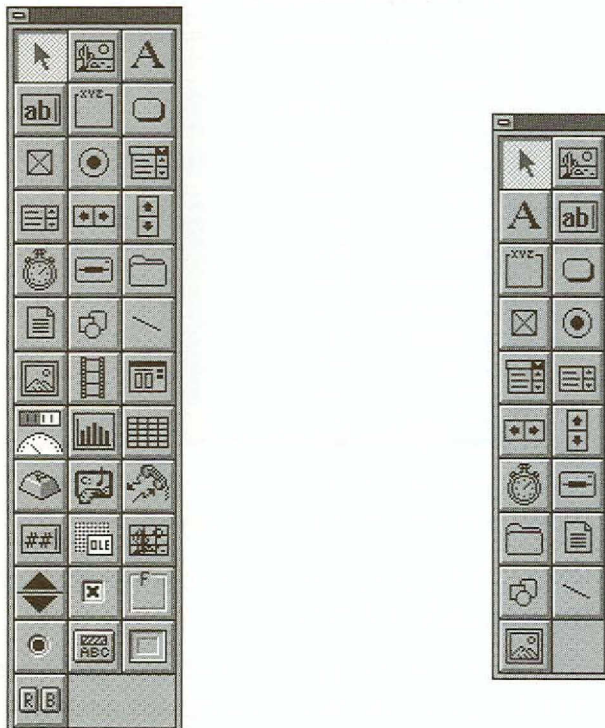
## 13.1 Layout

**standardversion**

**professionell version**

Interaktionsobjekten placeras på formulär. Formulär i Visual Basic är helt enkelt de fönster som finns i programmet. Ett program kan innehålla flera olika formulär.

Innehållet på interaktionsobjektspaletten bestäms av vilket utförande man använder sig av (se bild 51). I standarduppsättningen ingår de vanligaste objekten i Windows, knappar, radioknappar o s v. Förutom dessa ingår även mer specialiserade objekt i standarduppsättningen t ex timer och fillistor. Den professionella versionen innehåller en hel del användbara objekt, bl a ett tabellobjekt (grid) och ett diagramobjekt.



*Bild 51. Paletter med interaktionsobjekt som hör till den professionella versionen (vänster) och standardversionen (höger).*

Tillvägagångssättet för att placera ut interaktionsobjekt på ett formulär är det samma som för de flesta ritprogram. Först väljer man interaktionsobjekt för att sedan med hjälp av musen dra det till ett formulär.

För att justera positionen på ett objekt har utvecklaren tillgång till ett sk stödraster (grid) som den kan justeras efter. Stödrastrets storlek är inställningsbar. Däremot saknas det funktioner för att justera objekten gentemot varandra, vilket gör att det är arbetsamt att få ett symmetriskt utseende på ett formulär.

Alla objekt har ett antal grundegenskaper som kan förändras. Som exempel kan nämnas storlek, position, bakgrundsfärg, titel och typsnitt. Dessa egenskaper visas och definieras i en egenskapsredigerare (se bild 52). De egenskaper som har ett ändligt värdeförråd, t ex "Enabled" som kan vara sant eller falskt, kan man välja direkt ur en lista istället för att fylla i värden.

Menyer och menybalk skapas i särskild menyredigerare. Det går att definiera kortkommandon som ska höra till ett menyalternativ.

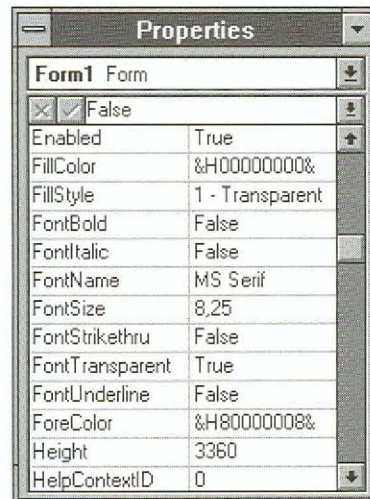
Det går att skapa nya interaktionsobjekt i Visual Basic men utvecklaren måste själv programmera objektens utseende och beteende. Det är t ex

**stödraster**

**objektens grundegenskaper**

**skapa nya objekt**





*Bild 52. Egenskapsredigeraren i Visual Basic används för att definiera objektets grundegenskaper.*

möjligt att skapa en speciell knapp genom att utnyttja bildobjektet och i programkod definiera utseende och beteende.

Det finns ingen möjlighet att på ett enkelt sätt återanvända dessa objekt eftersom man inte kan bygga ut paletten med egna objekt definierade i Visual Basic. Enda möjligheten, förutom att klippa och klistra, är att importera ett formulär som innehåller dessa objekt.

#### **köpa objekt**

Förutom de tillägg som finns i den professionella versionen kan man köpa andra interaktionsobjekt från en oberoende leverantör. Det finns bland annat speciella bildobjekt som kan hantera bildformat som inte annars kan användas i Visual Basic.

#### **skapa egna objekt**

För den mer avancerade användaren av Visual Basic finns det även möjlighet att bygga ut paletten med egna objekt genom att utnyttja en extern kompilator, t ex Borland C++. En beskrivning av hur detta går till medföljer den professionella versionen av Visual Basic.

## **13.2 Beteende och koppling till kod**

#### **dialekt av Basic**

Programmeringsspråket i Visual Basic är en dialekt av Basic. Det är möjligt att definiera egna typer, procedurer och funktioner. Men det går inte att skriva funktioner som returnerar något annat än grunddatatyper.

Programkod i Visual Basic kan knytas till en modul, ett formulär eller till ett interaktionsobjekt. Men den kod som knyts till enskilda objekt kan inte vara av godtycklig art utan den måste vara en specialisering av de händelser som finns fördefinierade för objektet. Med händelser avses

här klick, dubbelklick, tangentnedtryckning o s v. Den kod som är nuten till moduler och formulär kan däremot vara av godtycklig art.

Beteendet i ett Visual Basic-program bestäms av den kod som knyts till objektens händelser. Det finns också ett antal beteenden som är fördefinierade, t ex att en knapp trycks in om man klickar på den utan programmering.

En egenskap som skiljer Visual Basic från de flesta verktygen på marknaden idag är att den understöder direktstyrning. Det finns färdiga händelser för att hantera dra-och-släpp. Tyvärr är det inte möjligt att själv lägga till metoder och instansvariabler till objekten. Detta medför att utvecklaren får en hel del extraarbete med att definiera och utnyttja datastrukturer och procedurer som egentligen skulle kunna höra till objekten. Med andra ord saknas möjligheter i Visual Basic som vanligen finns i en fullt objektorienterad utvecklingsmiljö.

Interaktionsobjekt kan skapas dynamiskt i programkod under körning. Dessa objekt måste dock vara av sk "Control Array"-typ vilket innebär att de utgår från ett mallobjekt. Ett indexnummer är det enda som skiljer objekten åt. Detta indexnummer används då man vill referera till ett specifikt objekt i programkoden.

För att redigera kod finns en kodredigerare (se bild 53). Programkoden har en färgkodning så att det är lätt att se vad som är nyckelord och kommentarer o s v. Däremot saknas tyvärr en funktion för indrag, vilket medför att man blir tvungen att själv hålla reda på indragen.

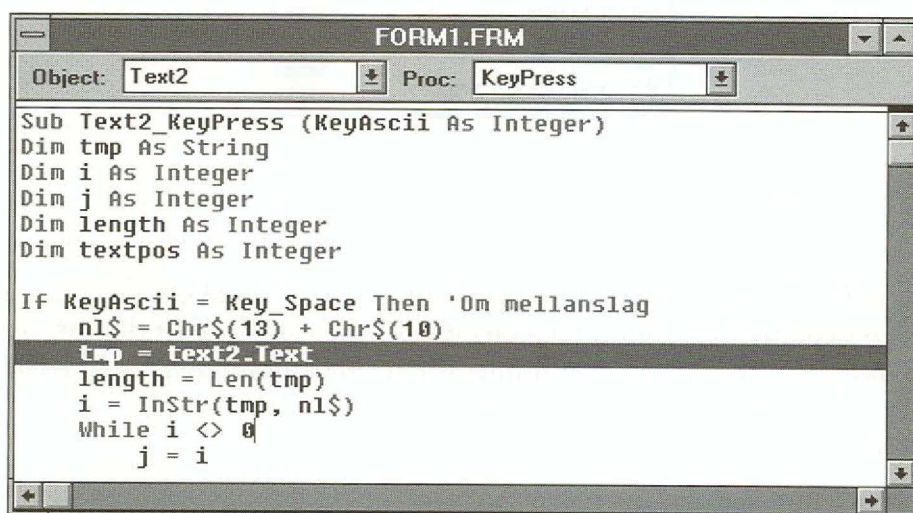
Ett problem med att ha kod knuten på detta sätt till interaktionsobjekt och formulär är att det blir svårt att få en bra överblick. Det är inte alla

**direktstyrning**

**Control Array**

**kodredigerare**

**dålig överblick**



```
FORM1.FRM
Object: Text2 Proc: KeyPress
Sub Text2_KeyPress (KeyAscii As Integer)
Dim tmp As String
Dim i As Integer
Dim j As Integer
Dim length As Integer
Dim textpos As Integer

If KeyAscii = Key_Space Then 'Om mellanslag
  n1$ = Chr$(13) + Chr$(10)
  tmp = text2.Text
  length = Len(tmp)
  i = InStr(tmp, n1$)
  While i <> 0
    j = i
```

*Bild 53. Kodredigeraren*



fritext-söka

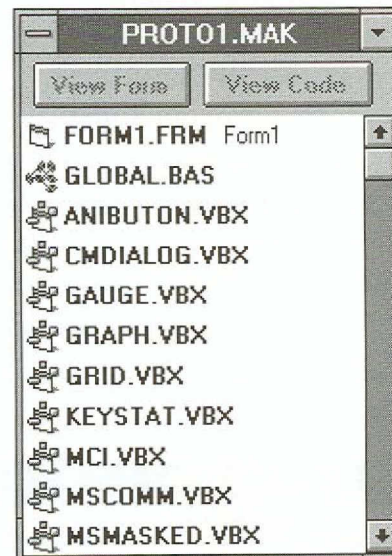
gångar lätt att veta var en specifik rutin finns. Den hjälp som finns i Visual Basic är att man kan fritext-söka i koden.

DLL-bibliotek

I och med att programmeringsspråket i Visual Basic kan byggas ut med hjälp av DLL-bibliotek är det möjligt att utvidga språket med nya operationer. Via DLL- gränssnittet kommer man även åt Windows eget gränssnitt för programmering, vilket gör det möjligt att komma åt de Windows-funktioner som inte finns definierade i Visual Basic.

## 13.3 Arbetsätt

För att skapa ett program med Visual Basic, skapar utvecklaren först ett s k projekt. Ett projekt består av formulär, moduler och tillägg (se bild 54). I formulären placeras de interaktionsobjekt som ska utnyttjas.



*Bild 54. Projektfönstret i Visual Basic visar de moduler, formulär och tillägg (VBX) som ingår i ett projekt.*

syntaxkontroll

Programkod knyts till objekten. När denna kod skrivs i kodredigeraren utförs omedelbart en enkel syntaxkontroll, vilket medför att det inte är nödvändigt att provköra programmet för att hitta syntaxfel. Visual Basic kräver en variabeltyp-deklaration med hjälp av en inställning. Det är dock förvalt att variabler får en underförstådd typ.

provköra

Det är möjligt att i alla lägen provköra programmet. Det finns ett speciellt provkörningsläge i utvecklingsmiljön. I detta läge finns det möjligheter att kontrollera och ändra variabelvärden. Spårningsfunktioner finns också tillgängliga, både för variabler och proceduranrop. Bryt-

punkter understöds liksom stegvis körning. Allt detta sammantaget gör att Visual Basic passar utmärkt för stegvis (inkrementell) utveckling.

Däremot understöder inte Visual Basic att flera utvecklare arbetar parallellt på samma program. Den möjlighet som finns för att hantera detta är import och export av formulärdefinitioner.

När programmet är färdigt kan Visual Basic skapa en körbar (exekverbar) fil. Men det krävs trots allt ett kodbibliotek (run time library) för att köra programmet. Vid distributionen av ett programmet krävs med andra ord att man, förutom den körbara filen, också skickar med kodbiblioteket. Dock är detta bibliotek fritt för spridning utan licensavgift.

Om programmet har tilläggsobjekt måste även ett kodbibliotek för dessa distribueras. De tillägg som medföljer den professionella versionen är fria för distribution. Dock finns det tillägg från oberoende leverantörer som kräver licensbetalning för varje installation.

## 13.4 Hjälp

Visual Basic har en utförlig hjälpfunktion som följer Windows-standard. Hjälpfunktionen innehåller i stort sett samma information som handböckerna och i vissa fall till och med mer. Det finns även kodexempel som visar hur man realiserar olika funktioner.

Det finns stöd för att realisera hjälpfunktioner för programmet. Utvecklaren kan koppla hjälp till såväl enskilda interaktionsobjekt som till hela programmet. Det kräver dock att Windows standardprogram för hjälp används. I den professionella versionen av Visual Basic medföljer en kompilator som producerar hjälpfiler för Windows standardhjälp.

## 13.5 Flyttbarhet och prestanda

Prestanda i Visual Basic är fullt tillräcklig även för förhållandevis stora program. Programkoden kompileras i viss utsträckning, vilket borgar för acceptabla prestanda. Dessutom finns det möjlighet att utnyttja DLL-bibliotek för att göra delar av ett program snabbare.

Visual Basic finns idag både för DOS och Windows. Flyttbarheten mellan dessa två miljöer har vi dock inte provat.

**stegvis utveckling**

**ej parallellt arbete**

**kodbibliotek**

**utförlig hjälp**

**tillräckliga prestanda**

**DOS och Windows**



## 13.6 Prototypstöd

Visual Basic passar bra för enklare prototyputveckling eftersom den väl understöder stegvis utveckling på ett bra sätt. Mer komplexa prototyper med invecklade grafiska beteenden, som exempelvis animeringar, kan däremot vara besvärligare att utveckla.

Vi bör understryka att prototyputveckling i Visual Basic måste göras av en programmeringskunnig person. Visual Basic är inte ett verktyg som medger att en normal slutanvändare kan realisera en prototyp.

De testprototyper vi realiserade går snabbt att utveckla i Visual Basic. Det krävdes omkring en halv dags arbete för vardera prototypen.

**kunskap krävs om programmering**

### 13.6.1 Tidrapporten

Inmatningsfälten realiserades med hjälp av interaktionsobjektet inmatningsfält. Indatakontrollen är av en enkel typ. Datumfältet accepterar endast numeriska värden genom att den vid inmatning kontrollerar att tecknen är siffror.

Projektnr	Tid
7434632	6,8
44465	8,3
098533	9,0
907645	0,5

*Bild 55. Tidrapporten under körning*

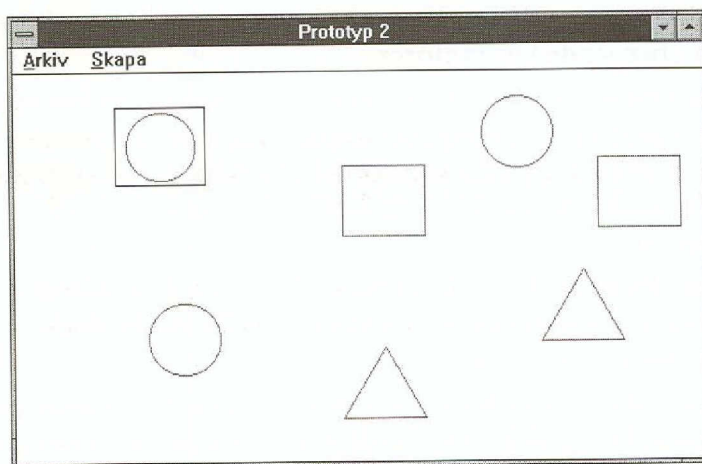
Det besvärligaste var att få det rullningsbara fältet att formatera indata på ett korrekt sätt i kolumner. För detta gick det åt ungefär 15 rader Basic-kod. När väl denna kod var skriven var det enkelt att få fram en fungerande prototyp.

**Basic-kod krävs**

## 13.6.2 Ritverktyget

Menybalken med menyalternativ skapades av menyredigeraren. Cirkeln, triangeln och kvadraten var av olika "Control Array"-typer och bestod i grunden av interaktionsobjektet bild.

När man väljer menyalternativ Ny Cirkel, skapas en ny instans av cirkel dynamiskt. Uppritningen av cirkeln gjordes med hjälp av Visual Basics egna ritkommandon.



*Bild 56. Ritverktyget under körning*

Att realisera direktstyrningen i Ritverktyget var enkelt eftersom det finns färdiga händelser som sköter direktstyrning. Den kod som behövs var därför endast (kommentarer efter '-tecken) :

```
Sub theSq_Click (Index As Integer)
' Kod som exekveras då man klickar på någon kvadrat
deselect 'Procedur som selekterar alla objekt
theSq(Index).BorderStyle = Line 'Visa ram kring detta
objekt
theSq(Index).Dragmode = Automatic 'Tillåt dra-och-släpp
End Sub
```

```
Sub Form_DragDrop (Source As Control, X As Single, Y As
Single)
' Kod som exekveras då något släpps i fönstret
Source.Move X, Y 'Flytta det objekt som släpptes till den
nya
' positionen
End Sub
```



Hade Visual Basic varit fullt objektorienterat skulle realiseringen av ritverktyget varit betydligt enklare. Nu var vi tvungna att bygga upp en bakomliggande datastruktur som höll reda på objektens tillstånd, t ex om de är valda för att realisera prototypen. Sammanfattningsvis var det ändå förhållandevis enkelt att realisera prototypen.

## 13.7 Slutsatser

- Visual Basic är ett av de intressantaste verktygen för Windows på marknaden idag. Det är lättanvänt och välintegrerat i Windows-miljön. Dessutom finns det tillägg att köpa från oberoende leverantörer.
- Visual Basic understöder stegvis utveckling på ett bra sätt vilket medför att verktyget lämpar sig för snabb utveckling av prototyper. En stor fördel är också att Visual Basic understöder hela utvecklingskedjan från prototyp till program.
- Visual Basic är ett effektivt verktyg för snabb systemutveckling. Ett mer objektorienterat synsätt skulle dock göra det än mer effektivt.
- Visual Basic understöder inte att flera personer arbetar på ett projekt parallellt. Det medför att det kan vara svårt att använda Visual Basic i större systemutvecklingsprojekt som involverar flera systemutvecklare.

# 14. VUIT

---

**Klassificering:** Layout-redigerare

**Gränssnittsstandard:** OSF/Motif

**Miljö:** VMS 5.4, Ultrix 4.0 eller SunOS 4.1.1.

Verktyget och skapade program kan även användas mot andra realiseringar av X-Windows.

**Version:** 2.0

**Tillverkare:** Digital Equipment

**Leverantör:** Digital Equipment AB

Vuit tillåter att utvecklaren interaktivt bygger ett programs gränssnitt. Verktyget kan skapa ett skelett för den programkod som behövs för att få ett fungerande skal. Funktioner i programmet som aktiveras av gränssnittet måste programmeras i C.

Verktyget består av ett konstruktionsprogram som utvecklaren använder för att bygga och delvis prova gränssnittet. Sedan sparas en specifikation av gränssnittet i en datafil där specifikationen är uttryckt i ett speciellt språk, UIL (User Interface Language).

UIL-specifikationen måste sedan översättas till en binär form som kan läsas av programmet. Detta utförs med ett fristående program, "uil". Resultatet blir en UID-fil. När man sedan kör programmet läser detta automatiskt in UID-filen och bygger upp gränssnittet.

En fördel med denna uppdelning är att den ger stor flexibilitet vid utformningen av gränssnittets innehåll. Om utvecklaren t ex gör små förändringar i gränssnittets utseende behöver programmet inte kompileras och länkas på nytt. Det enda som krävs är att gränssnittets UIL-specifikation omkompileras.

**flexibilitet**

## 14.1 Layout

Då Vuit startas presenteras ett huvudfönster med ett antal funktioner som är åtkomliga via menyer, samt en arbetsyta där gränssnittet skapas (se bild 57). Till vänster finns en arbetsmeny som innehåller de kon-



## trädstruktur

struktionselement utvecklaren har tillgång till. Menyn har en trädstruktur och varje gren kan expanderas eller krympas så att innehållet visas eller döljs.

## dra-och-släpp

Nya objekt skapas interaktivt enligt principen dra-och-släpp, d v s man tar tag i ett original på paletten och drar iväg med en kopia av detta till önskad plats.

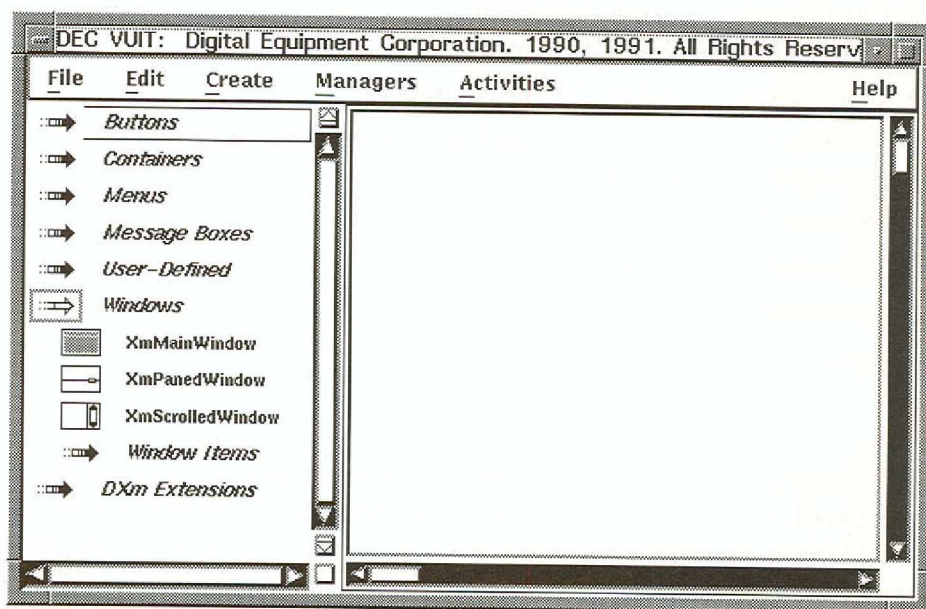


Bild 57. Vuits huvudfönster

## OSF/Motif Toolkit

Vuit tillhandahåller alla standardkomponenterna i OSF/Motif Toolkit. Vuit grupperar komponenterna ungefär efter användningssätt. Gruppen Windows innehåller följande baskomponenter:

- *XmMainWindow* som är den typiska grundkomponenten för de flesta program. Den innehåller vanligen en funktionsmeny överst, rullningslister nedtill och till höger samt en arbetsyta i mitten.
- *XmPanedWindow* som är ett fönster med en avgränsare som kan flyttas vertikalt.
- *XmScrolledWindow* som är ett fönster med rullningslister.

De komponenter utvecklaren kan placera i ett basfönster återfinns i gruppen Window Items som bl a innehåller XmMenuBar (som också återfinns i gruppen Menus, XmForm och Containers) och XmScrollBar.

På arbetsytan placerar man vanligen en behållare (container) från gruppen Containers, vilken bl a innehåller:

- *XmBulletinBoard* som tillåter fri placering av underordnade objekt, ungefär som på en anslagstavla.

## anslagstavla

- *XmForm* som tillåter att underordnade objekt placeras med givna inbördes förhållanden, som i ett formulär.
- *XmRowColumn* som används för att placera underordnade objekt i jämna rader eller kolumner, som i menyer och tabeller.

formulär

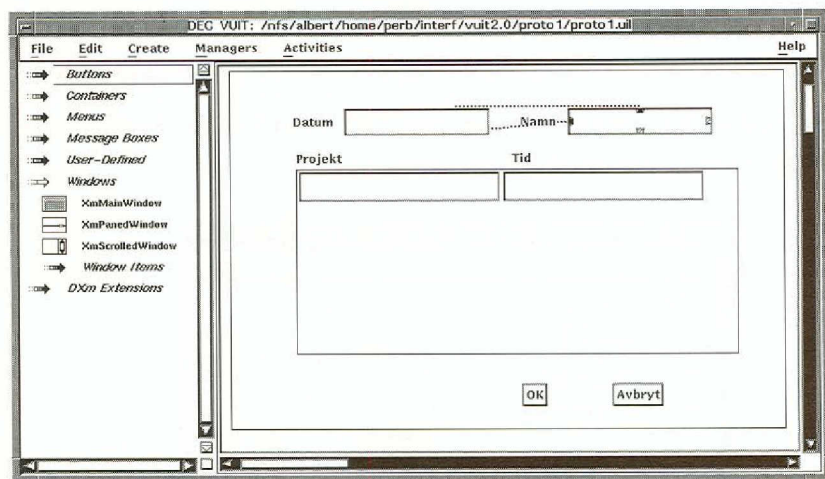
meny

Dessa behållare kan fyllas med andra behållare, knappar av olika former från gruppen Buttons eller med primitiver från Container Items, såsom inmatningsfält, ledtexter, ramar och skiljelinjer.

Menyer byggs av objekt ur gruppen Menus och fylls med objekt ur undergruppen Menu Items.

Slutligen finns ett speciellt objekt, "user", som gör det möjligt att referera till objekttyper som inte finns inbyggda i Vuit. För ett sådant objekt måste utvecklaren ange namnet på en procedur i programmet som skapar objektet.

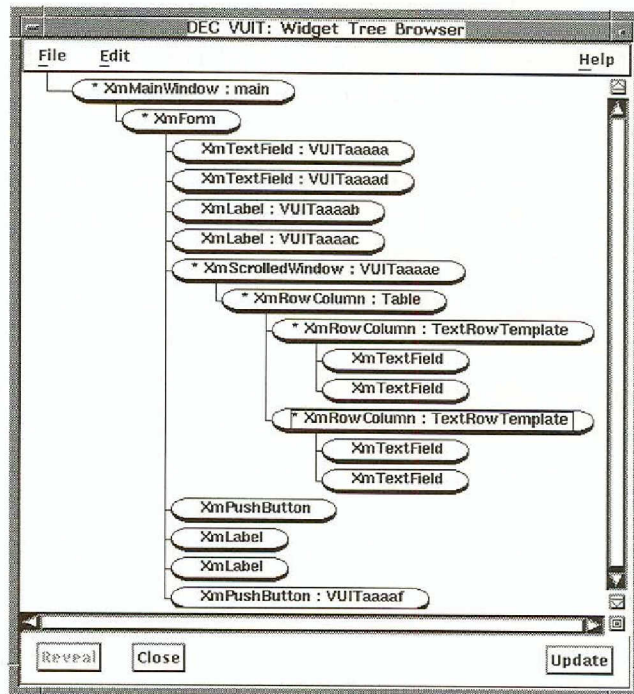
I tidrapporten utnyttjades bl a *XmForm*, *XmTextField* och *XmPushButton*.



*Bild 58. Tidrapporten under uppbyggnad med ett inmatningsfält utvalt för modifiering.*

De gränssnittsobjekt som används kan visas i en hierarkisk trädstruktur i ett speciellt fönster (se bild 59).





*Bild 59. Vy av en hierarkisk trädstruktur av gränssnittsobjekt*

Varje objekttyp har ett formulär där utvecklaren kan ange värden på alla egenskaper som är användbara på denna typ.

Resource	Binding	Value
XmNwidth	imm	153
XmNheight	imm	33
XmNcolumns	imm	1

*Bild 60. Egenskapsformulär för inmatningsfält*

Vuit tillhandahåller alla grundprimitiver i OSF/Motif. Detta möjliggör bra kontroll av gränssnittets innehåll och beteende. OSF/Motif Toolkit innehåller ett antal primitiver som tillåter strukturering. Därför går det också att utvidga mängden primitiver genom att kombinera de redan existerande.

**egna primitiver**

Det går även att definiera egna primitiver. Utvecklaren skapar då t ex en XmTable och infogar denna genom att dels lägga in primitiven i Vuits beskrivning av arbetsmenyn, dels länka en ny version av Vuit som innehåller realiseringen av den nya primitiven.

Konsekvensen blir att det skapade programmet följer gränssnittets innehåll i OSF/Motif så länge utvecklaren inte bygger nya primitiver. Verktøget kan naturligtvis inte ge någon information om huruvida en viss kombination av primitiver eller en icke-standard-primitiv följer OSF/Motif.

Gränssnittets utseende styrs i viss mån av vilka objekt utvecklaren väljer och hur objektens resurser anges. För att få ett tilltalande utseende kan man välja att samla vissa objekt i behållare av typen XmForm eftersom där går att specificera hur objekten ska placeras i förhållande till varandra. Därigenom anpassar sig formuläret i viss mån till sitt innehåll. Det kräver dock viss träning för att lyckas med komplicerade formulär.

**anpassar sig  
efter innehållet**

## 14.2 Beteende och koppling till kod

Händelser i gränssnittet överförs till programmet i form av anrop till s k callback-procedurer. Dessa anrop specificeras i formuläret för respektive objekt. Man kan endast ange procedurer för de händelser som faktiskt kan inträffa i det objekt man för tillfället betraktar. Dessa händelser presenteras i en meny.

**anrop**

Emellertid måste utvecklaren alltså väl inse hur objektet ifråga fungerar innan han kan specificera korrekta kopplingar. Exempelvis kan man för en knapp specificera MrmNcreateCallback, XmNactivateCallback, XmNdestroyCallback, XmNarmCallback, XmNdisarmCallback och XmNhelpCallback. Det blir naturligtvis inte omedelbart uppenbart vilken av dessa händelser som motsvarar att användaren klickat på knappen ifråga.

Den procedur som ska anropas anges med ett symboliskt namn. Om namnet inte redan existerar måste man definiera proceduren i Vuits procedur-redigerare. Man kan då ange vissa standardaktiviteter som Vuit förstår och kan skapa kod för, t ex visa fönster, avsluta programmet eller anropa programmet. Det går då även att koppla det symboliska namnet till en existerande subrutin i programkoden.



Det går att definiera nya typer av kopplingar genom att specificera en sk "translation table" och binda den till ett objekt (se kapitel Tidrapporten nedan) men detta tillvägagångssätt är inte beskrivet i dokumentationen.

## 14.3 Arbetsätt

Arbetsgången blir i huvuddrag:

1. Bygga gränssnittet interaktivt med hjälp av Vuit. Vissa funktioner kan provas direkt under konstruktionsfasen.
2. C-kod genereras, och ev "makefile", till programskelett.
3. UIL-specifikationen genereras.
4. Skapa binärkod, kopplad till programmet, samt UID-filen med hjälp av "make".
5. Prova gränssnittet med programmets funktioner, identifiera felaktigheter eller olämplig utformning hos gränssnittet, modifiera gränssnittet med Vuit. Börja om från 3.

### UIL-syntax

Gränssnittet i det skapade programmet beskrivs av den text i UIL-syntax som Vuit sparar och läser. Detta gör det möjligt att underhålla beskrivningen med andra verktyg än Vuit, t ex en ordbehandlare. Det är också teoretiskt möjligt att översätta beskrivningen till något annat verktygs specifikationspråk.

### väl lämpad för produktunderhåll

Vuit stöder några egenskaper hos UIL som är mycket väl lämpade för produktunderhåll. En sådan egenskap är möjligheten att definiera namn på värden (literals) av ett flertal olika typer. Detta gör det möjligt att på en plats ange ett värde som använts på många olika ställen i ett program. Det kan t ex finnas ett antal dialoger i ett program som var och en består av ett formulär med diverse olika primitiver. Varje formulär innehåller vanligen en knapp för att avbryta dialogen och texten är "Cancel". Genom att sätta ett namn på denna text, t ex "cancel\_button\_text" och referera till namnet varje gång man skapar en sådan knapp, får alla sådana knappar samma text. Man kan sedan omdefiniera "cancel\_button\_text" till texten "Avbryt" och därigenom påverka alla sådana knappar med ett enda handgrepp. Dessa namn kan exporteras från specifikationen av gränssnittet till programkoden, vilket innebär att utvecklaren med hjälp av Vuit kan definiera en mängd resurser som programmet hämtar vid behov.

Ett problem med Vuit är att det inte finns något bra sätt att kombinera egen programkod med det programskelett som Vuit skapar. Om programmet måste göra någon annan initiering än den som finns inbyggd i programskelettet måste detta förändras. Sådana förändringar går förlo-  
rade när skelettet återskapas.

**går ej att kombinera  
kod och skelett**

## 14.4 Hjälp

I Vuit har man direkt tillgång till hjälptexter genom att begära hjälp och sedan peka på det objekt man vill ha mer information om.

**direkt hjälp**

Hjälptexter kan även knytas till olika delar av det skapade programmet med hjälp av callback-procedurer. Det finns inget enkelt sätt att koppla en hjälptext till ett objekt i gränssnittet. Det är visserligen inte svårt att definiera en procedur som visar en text i ett fönster men det borde finnas några färdigförpackade hjälpprocedurer direkt tillgängliga

## 14.5 Flyttbarhet och prestanda

Det skapade gränssnittet måste naturligtvis kompileras och länkas på en maskin med bibliotek som innehåller stöd för OSF/Motif Toolkit. Programmet kan köras mot en annan X-realisering.

**kompilering  
och länkning**

Vuit fungerar också mot andra X-realiseringar men såväl Vuit som de skapade programmen har naturligtvis OSF/Motif-utseende, vilket kan verka främmande i en annan miljö.

Snabbheten hos Vuit och de skapade prototypprogrammen föreföll normal. Eftersom prototyperna är ganska små är det svårt att säga vad som händer i ett verkligt fall.

**normala prestanda**

## 14.6 Prototypstöd

Vuit kan i det färdiga gränssnittet simulera alla viktiga egenskaper som inte beror på aktiviteter i programmet. Utvecklaren kan enkelt starta provkörning av gränssnittet från en meny i verktyget. I tidrapporten

**provkörning**



kunde vi således prova inmatningsfälten och knapparna men inte projektlistan, eftersom det inte finns något egentligt tabellbegrepp med lämpliga fördefinierade operationer.

Projekt	Tid
23190	7
22020	4

*Bild 61. Tidsrapporten under provning*

Utskrifter om vissa aktiviteter, t ex anrop av programprocedurer, visas i ett speciellt fönster under provkörningen.

### 14.6.1 Tidsrapporten

Alla komponenter utom tabellen kunde direkt realiserats och provas med konstruktionselementen i Vuit.

#### tabellen problem

Tabellen erbjöd ett visst problem då någon sådan struktur inte finns direkt tillgänglig. Emellertid har man tillgång till alla existerande primitiver. Dessutom finns ett visst stöd att hämta i Motifs resurshanterare. Vi definierade tabellen som ett fönster med en rullningslist (realiserad med `XmScrolledWindow`). Rullningslistan skulle innehålla en kolumn (`XmRowColumn`) med rader (`XmRowColumn`) som var och en bestod av två inmatningsfält (`XmTextField`). Kruxet här var att skapa och ta bort rader dynamiskt utan att behöva realisera all nödvändig programkod. Att ta bort en tabellrad var inget problem. Vi anropade bara `XtDestroyWidget`. Att skapa en tabellrad var något svårare. Lösningen blev att definiera en tabellrad som en mall med ett känt namn. Vi kunde sedan anropa funktionen `MrmFetchWidgetOverride` för att skapa en ny tabellrad enligt mallen.

Vi reducerade således problemet till att anropa två existerande rutiner vid rätt tillfällen och med lämpliga parametrar. Detta gjordes med följande två procedurer som vi kodade själva:

```

void add_text_row(w, e, p, n)
Widget    w;
XEvent    *e;
String    *p;
Cardinal  *n;
{
    Widget    new;
    MrmType   class;
    Cardinal  c;

    c = MrmFetchWidgetOverride(s_MrmHierarchy, *p,
                               XtParent(XtParent(w)),
                               "proto1_table_row",
                               NULL, 0, &new, &class);
    if (c != MrmSUCCESS)
        s_error("can't add table row");
    XtManageChild(new);
}

```

I anropet till MrmFetchWindowOverride representerar:

- s\_MrmHierarchy ett handtag till resursfilen som erhållits vid initieringen
- \*p en parameter från tangentbindningen, i detta fall namnet på tabellradsmallen
- XtParent(XtParent(w)) föräldern till det inmatningsfält där tangenttryckningen äger rum, d v s uttrycket ger den yttre av XmRowColumn-objekten, vilket är tabellen
- "proto1\_table\_row" namnet på det nya tabellradsobjektet

```

void del_text_row(w, e, p, n)
Widget    w;
XEvent    *e;
String    *p;
Cardinal  *n;
{
    XtDestroyWidget(XtParent(w));
}

```

I detta fall var det enklast att binda funktionerna till tangenttryckningar i inmatningsfälten i tabellen, exempelvis <Return> för att skapa en ny rad sist i tabellen och <Shift><Delete> för att ta bort en aktuell rad. De båda bindningarna kunde skrivas in i en "translation table" i ett formulär i verktyget. De måste dock även installeras i inmatningsfälten genom att skicka tabellen till en rutin som anropas då ett inmatningsfält skapas, en s k "MrmCreateCallback". Önskad koppling specificerades i formuläret för inmatningsfälten. Själva rutinen, som vi kodade själva, visas nedan:



```

void text_create(w, tbl, reason)
Widget          w;
XtTranslations  tbl;
unsigned long   *reason;
{
    XtOverrideTranslations(w, tbl);
}

```

Nu råkar `text_create` och `XtOverrideTranslations` ha samma ordningsföljd på de första parametrarna. Därför kunde vi lika gärna definierat `text_create` med hjälp av Vuits procedur-redigerare. Man anger då att namnet "text\_create" i själva verket är en existerande rutin som heter `XtOverrideTranslations`. Så i detta fall hade Vuit även kunnat skapa proceduren ovan.

Förutom deklarerationer och namnregistrering var detta alltså all kod som behövdes för att realisera dynamiken. Som väl framgick av diskussionen krävde detta mer kunskap än arbete.

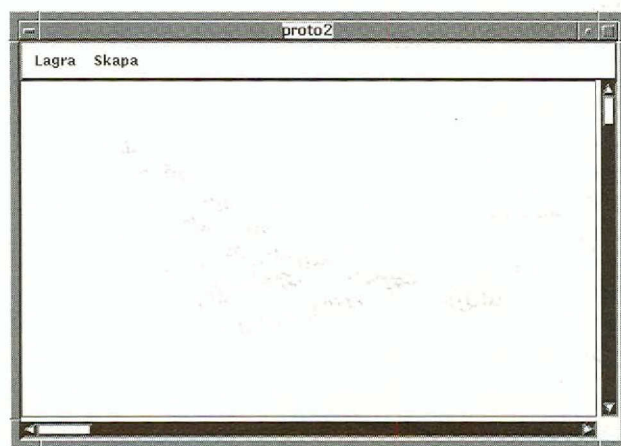
**krävs mer  
kunskap än arbete**

## 14.6.2 Ritverktyget

Menyn, ritytan och dialogboxen innebar inga problem. Vi kunde enkelt definiera menyn och koppla menyningångarna till funktioner i programmet och till att visa dialogboxen. Vidare kunde vi koppla ett klick i ritytan till aktiviteten för att dölja dialogboxen.

All grafikhantering måste realiseras i programmet, alltså: skapa symbol, rita symbol på begäran, välja symbol, flytta symbol, radera symbol. När dessa funktioner väl fanns kunde vi dock lätt koppla dem till olika händelser.

**grafikhantering  
i programmet**



*Bild 62. Ritverktyget under körning*

## 14.7 Slutsatser

- Eftersom Vuit använder dra-och-släpp är interaktionen med verktyget mycket enkel under uppbyggnadsfasen.
- Möjligheterna att navigera i en specifikation av ett gränssnitt är i stort sett väl utformade. Via menyer och formulär kan utvecklaren inspektera objekt av olika kategorier, söka efter objekt enligt olika kriterier. Det finns även möjlighet att se den hierarkiska strukturen hos de olika komponenterna i ett fönster. Tråkigt nog hålls detta fönster inte aktuellt av Vuit utan utvecklaren måste begära uppdatering i stort sett varje gång det behövs.
- Det krävs en viss listighet för att bygga program som kräver dynamik i sitt användargränssnitt.
- Verktyget saknar möjligheter att direkt skriva ut eller på annat sätt spara en bild av det gränssnitt som skapats. Man måste gå omvägen via ett skärmdumpprogram för att erhålla en sådan.
- Det tar viss tid innan man blir förtrogen med verktyget eftersom det finns ett mycket stort antal formulär.
- Man behöver ibland skriva små procedurer som fäster programmet vid befintliga funktioner. Det beror på att formen hos en s k callback ges av verktyget i en form som inte alltid stämmer med omgivningen. Tyvärr kan dessa procedurer inte skrivas in och underhållas direkt i verktyget.
- Ibland tvingas utvecklaren in i dialoger vid olämpliga tillfällen. När man skapar objekt av vissa typer kommer omedelbart ett namngivningsformulär upp på skärmen. Man är då tvungen att hitta på ett namn. Detta innebär att tankeprocessen störs eftersom man som utvecklare i detta moment fokuserar på gränssnittets struktur snarare än på objektens namn.
- Verktyget är inte helt stabilt. I vissa lägen blir det felavbrott. Då skapar verktyget automatiskt en temporär fil och från denna går det att återskapa specifikationen.





# 15.XVT

---

**Klassificering:** Layout-redigerare och Flerplattformsverktyg

**Gränssnittsstandard:**

**Miljö:** Windows, Motif, Macintosh, OS/2, Open Look

**Version:** Kodbibliotek 2.0, XVT-Design 1.0

**Tillverkare:** XVT

**Leverantör:** Programvaruhuset Edvina

XVT, Extensible Virtual Toolkit, är ett verktyg för flera plattformar som understöder ett stort antal standarder: Macintosh, Motif, Open Look, Windows och Presentation Manager. XVT är utvecklat av ett företag i USA som också heter XVT.

**stöder flera  
standarder**

XVT består av två delar:

- Ett *kodbibliotek*
- En *layout-redigerare*, XVT-design

XVT är ett av de första försöken med att skapa ett verktyg för flera plattformar som understöder flyttning av program mellan olika plattformar. Verktöget har funnits på marknaden sedan 1988.

Ett stort generellt problem när man skapar verktyg som ska stöda flera standarder för gränssnitt är den så kallade "Största Gemensamma Nämnaren", SGN. Enkelt uttryckt kan SGN-problemet sägas vara: Hur ska man skapa ett gemensamt gränssnitt för programmering (API) där man även kan utnyttja de funktioner som inte är gemensamma för samtliga understödda standarder?

**största gemensamma  
nämnanen**

XVT försöker lösa SGN-problemet genom att definiera ett kodbibliotek som bildar ett extra lager mellan värdsystemets kodbibliotek och programet. Anropen till kodbiblioteket i XVT översätts till värdsystemets. Enkla operationer, som t ex ritning, avbildas direkt på värdsystemets kodbibliotek. Mer komplexa operationer, som att visa dialogbox, avbildas inte direkt utan bildar istället en mängd anrop mot värdsystemets kodbibliotek.

**kodbibliotek**



Ett program som är utvecklat med XVT kommer att bestå av ett antal filer med C-kod samt en resursfil som beskriver fönstrens utseende m m.

## 15.1 Layout

### layout-redigerare

I XVT ingår en layout-redigerare, XVT-design (se bild 63). Den används för att definiera fönstrens utseende. Fönster skapas genom ett menyalternativ och samma sak gäller de övriga interaktionsobjekten. Objekten dras sedan ut i fönstret.

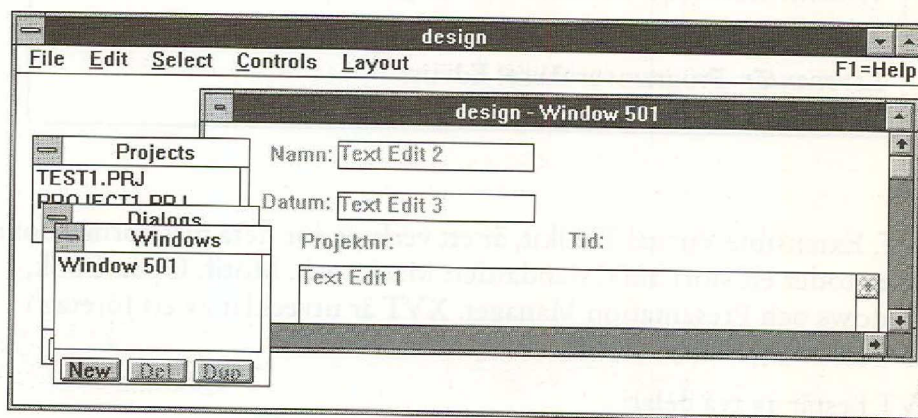


Bild 63. Layout-redigeraren XVT-design som följer med XVT

### interaktionsobjekt

De interaktionsobjekt som finns med i XVT-design är knappar, statisk text, inmatningsfält, listboxar och rullningslistor. Till dessa interaktionsobjekt hör ett antal egenskaper som kan förändras, t ex innehållet i statiska texter.

### symmetri

XVT-design innehåller funktioner för att justera interaktionsobjekt inbördes, vilket gör det enkelt att få ett symmetriskt användargränssnitt. Däremot saknas det funktioner för att definiera vad som ska ske med interaktionsobjekten då fönstret blir större eller mindre.

### ej egna objekt

Det finns ingen möjlighet att i XVT-design använda eller skapa egna interaktionsobjekt. Det finns inte heller någon möjlighet att provköra gränssnittet. Detta gör det svårt att visualisera ett tänkt användargränssnitt på ett tidigt stadium.

### ej provkörning

Förutom interaktionsobjekt kan utvecklaren definiera strängresurser. Dessa strängresurser finns sedan tillgängliga i programmet.

## 15.2 Beteende och koppling till kod

Kod i XVT skrivs i C. XVT-design skapar en resursfil och mallar till ett C-program. Mallarna innehåller prototyper för de funktioner som hanterar händelser i gränssnittet.

För att definiera beteendet för ett interaktionsobjekt specialiseras C-mallarna på de platser som avser händelser för objektet.

Kodbiblioteket i XVT tillhandahåller alla funktioner som krävs för att hantera ett användargränssnitt. Det innehåller helt enkelt de rutiner som vanligen finns i värddplattformens kodbibliotek (API). Med andra ord finns det inga ytterligare funktioner i XVT för att underlätta skapandet av program. Att använda verktyget kan jämföras med att skriva ett program som utnyttjar fönstersystemens kodbibliotek.

XVT tillhandahåller varken någon kod-redigerare eller kompilator. Istället ska utvecklaren använda någon av de vanliga miljöerna för programutveckling, som t ex Microsoft C.

C-kod

kodbibliotek

ej kod-redigerare  
eller kompilator

## 15.3 Arbetssätt

För att skapa ett program med XVT börjar utvecklaren med att, i XVT-design, definiera de fönster med interaktionsobjekt som behövs i programmet. När detta är klart skapar XVT-design en resursfil samt mallar till ett C-program.

C-mallarna används sedan för att definiera beteendet för interaktionsobjekten. Beroende på vilken miljö för utveckling av program som används finns det tillgång till olika kod-redigerare och kompilatorer.

När beteendet är definierat måste C-koden kompileras och länkas tillsammans med kodbiblioteket i XVT. Sedan är det möjligt att prova programmet.

En stor nackdel i XVT-design är att det saknas möjlighet att uppdatera de befintliga C-mallarna. Om utvecklaren vill ändra i ett gränssnitt som skapats tidigare, t ex lägga till en knapp, så klarar inte XVT-design att uppdatera de C-programfiler som redan finns. Istället skapas nya C-mallar. Detta medför att den nya mallen måste jämkas ihop manuellt med det program som redan finns.

Det finns dock en annan omständlig metod för att hantera tillägg av interaktionsobjekt. I och med att resursfilen som XVT-design skapar är läsbar och syntaxen finns definierad, är det möjligt att utföra ändringar direkt i resursfilen.

c-mallar

uppdaterar  
ej mallar



När ett XvT-program ska distribueras får enbart den körbara (exekverbara) filen från översättningen följa med. Eftersom XVT utnyttjar värdsystemets kodbibliotek krävs ingen koddel.

## 15.4 Hjälp

I XVT-design finns det direkt hjälp om själva verktyget. Däremot saknas direktansluten hjälp för kodbiblioteket och dess anropssyntaxer. Dock är medföljande handböcker lättlästa och enkla att hitta i.

Det finns inte något stöd i XVT för att realisera hjälp i det program som utvecklas. Det tillkommer utvecklaren att realisera denna hjälp.

## 15.5 Flyttbarhet och prestanda

Som tidigare nämnts realiserar XVT sitt flyttbara kodbibliotek genom att översätta XVT-anropen till anrop i värdsystemet. Detta har gjort det möjligt att infoga förhållandevis avancerade flyttbara funktioner i XVT. Så t ex finns det funktioner för utskrift.

Däremot saknar kodbiblioteket funktioner för att hantera de mer avancerade funktionerna i fönstersystemen som DDE (Dynamic Data Exchange) och OLE (Object Linking and Embedding) i Windows. För att kunna utnyttja dessa är utvecklaren hänvisad till värdsystemets kodbibliotek. Detta ger naturligtvis upphov till kod som bara kan flyttas om den skrivs om för den aktuella plattformen.

För att flytta ett program mellan två värdsystem krävs det att källkoden och resursdefinitionen åter kompileras. Dessutom krävs det att XVTs kodbiblioteket för den aktuella målmiljön används.

XVT innehåller stöd för att dölja skillnader mellan olika realiseringar av C-kompilatorer.

Prestandaproblemen för XVT-program liknar problemen med de program som realiserats i vanliga miljöer och programutvecklingsmiljöer av typen Microsoft C. Förvisso kan XVT introducera lite extra tidsåtgång då XVT-anrop kan översättas till en mängd anrop mot värdsystemet. Men skillnaden blir i de flesta fall helt försumbar.

**lättlästa  
handböcker**

**avancerade  
flyttbara funktioner**

**ej DDE och OLE**

## 15.6 Prototypstöd

XVT innehåller egentligen inget prototypstöd. Det stöd som finns är begränsat till XVT-design. Det går som mest att göra en statisk layout av fönster, men eftersom XVT-design inte innehåller någon form av provkörning av gränssnittet är det svårt att se hur det färdiga användargränssnittet kommer att se ut.

Verktyget är helt enkelt inte avsett som ett verktyg för att utveckla prototyper utan ska istället ses som ett hjälpmedel för att få flyttbara program.

inget prototypstöd

### 15.6.1 Tidsrapporten

Tidsrapporten gick att realisera i som en statisk layout utan några som helst inbyggda funktioner (se bild 64). Vi byggde upp tidsrapporten med hjälp av knappar och textfält. I och med att gränssnittets form inte kan provköras i XVT-design är det svårt att visualisera det tänkta gränssnittet.

statisk layout

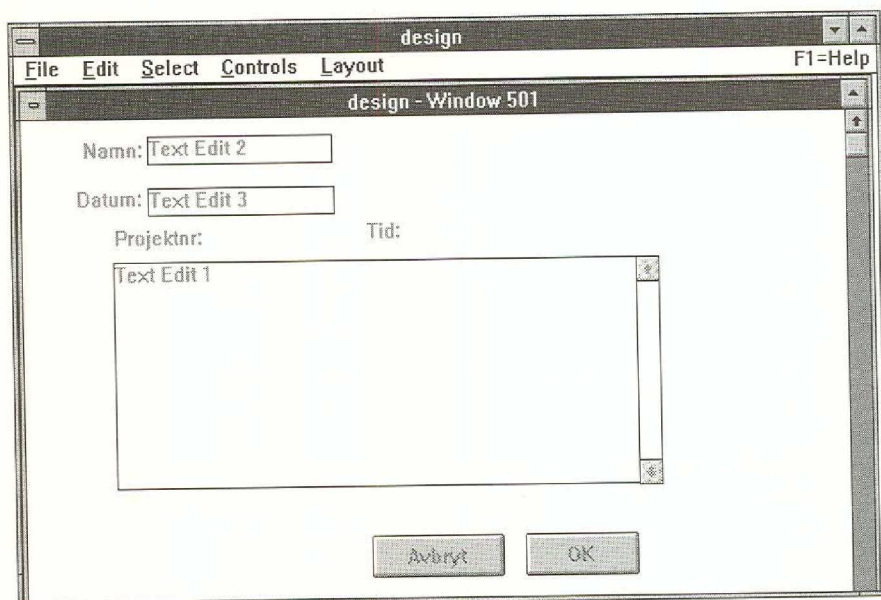


Bild 64. Tidsrapporten i XVT-design



en mängd C-kod

## 15.6.2 Ritverktyget

Ritverktyget var inte möjligt att realisera i XVT utan att realisera programmet fullt ut med hjälp av en mängd C-kod. Vi valde därför att inte realisera prototypen utan istället att enbart konstatera att XVT inte är ett verktyg för att utveckla prototyper.

## 15.7 Slutsatser

- XVT kan vara ett intressant alternativ om det finns ett behov att realisera ett program som ska vara flyttbart mellan olika plattformar. XVT har egentligen inga andra fördelar jämfört med andra verktyg.
- XVT realiserar inte program till programkommunikation, t ex DDE och OLE, vilket gör att det ändå kan vara svårt att realisera program som verkligen blir lätt flyttbara.
- Layout-redigeraren, XVT-design, är inte speciellt användbar eftersom den har mycket begränsade funktioner. Att inte på ett enkelt sätt kunna lägga till en knapp i ett användargränssnitt är en stor brist i en layout-redigerare.
- XVT är inte ett verktyg för att utveckla prototyper.

# 16. Verktygs- leverantörer

---

Nedan följer en informell lista med namn på ett antal verktyg och adresser till deras leverantörer. Om det finns en svensk leverantör har vi angett dennes adress, annars tillverkarens. Vi har angett fönstermiljö samt grupperat dem enligt tidigare beskriven klassificering i den mån information om verktygen varit känd. Listan innehåller de verktyg som finns beskrivna i denna rapport men också andra.

Vi gör inga anspråk på att listan är komplett. Dessutom, med tanke på den snabba förändringstakten på marknaden, kommer den förmodligen att snabbt bli inaktuell.

Tabell 3: Verktygsleverantörer

Verktygsnamn	Typ	Fönstermiljö & Programmeringsspråk	Leverantör
Actor	Layout-redigerare	Windows	The Whitewater Group 1800 Ridge Ave. Evanston, IL 60201 USA Tel: +1 800 869 1144
Agora UIMS	UIMS	Motif Open Look	ASET 2957 Clairmont Road Suite 200 Atlanta, GA 30329 USA
Application Manager	Totalverktyg	Presentation Manager	Mandat Consult Kungsbroplan 1 112 27 Stockholm Tel: 0 652 58 80 Fax: 654 77 39
Aspect	Multiplattformsvetkyg	Motif Open Look Presentation Manager Windows Macintosh	Open 655 Southpointe Court Suite 100 Colorado Springs, CO 80906 USA



Tabell 3: Verktögsleverantörer

Verktögsnamn	Typ	Fönstermiljö & Programmeringsspråk	Leverantör
AT&T Display Construction Set	UIMS	Open Look	AT&T Network Systems USA
Builder Xcessory	Layout-redigerare	Motif C	ICS Inc. 201 Broadway Cambridge, MA 02139 USA Tel: + 1 617 621 0060
CASE:W CASE:PM	Layout-redigerare	Windows Presentation Manager  C, C++, Cobol	Software Selection Johannelundsv. 3 194 81 Upplands Väsby Tel: 08 59 07 73 90 Fax: 08 59 07 73 91
Contessa	Layout-redigerare	Motif Open Look  Unix, VMS	Contexture Systems One Exeter Plaza Boston, MA 02116 USA
Craftman	Totalverktyg	Nextstep  Craftscript	Xanthus AB Box 20161 161 02 Bromma Tel: 08-635 30 00 Fax: 08-98 70 67 E-mail: xanthus@xanthus.se
Dataviews	UIMS	Motif  C	Konsulthuset Kaserntorget 6 411 18 Göteborg Tel: 031 17 02 80 Fax: 031 17 02 84
DevGuide	Layout-redigerare	Open Look  C	Sun Microsystems Svenska AB Box 51 164 94 Kista Tel: 08 623 90 00 Fax: 08 623 90 05
DialogBuilder	UIMS	Motif	Siemens Nixdorf Informationssystem Box 1329 171 26 Solna Tel: 08 705 20 00

Tabell 3: Verktygsleverantörer

Verktygsnamn	Typ	Fönstermiljö & Programmerings-språk	Leverantör
Dynagraph X	Layout-redigerare	Motif C	Konsulthuset Kaserntorget 6 411 18 Göteborg Tel: 031 17 02 80 Fax: 031 17 02 84
Easel	Totalverktyg	Windows Presentation Manager DOS	WM-Data Software AB Box 27030 102 51 Stockholm Tel: 08 670 20 00 Fax: 08 670 20 60
Enfin/2	Totalverktyg	Presentation Manager Windows	Software Selection Johannelundsv. 3 194 81 Upplands Väsby Tel: 08 59 07 73 90 Fax: 08 59 07 73 91
EXOCode	Layout-redigerare	Motif Open Look C	Expert Object 7250 Cicero Ave., Suite 201, Lincolnwood, IL 60646 USA Tel: + 1 708 676 5555
Eyescream	Layout-redigerare	Presentation Manager Windows DOS C	Realtidsgrafik AB Box 14035 720 14 Västerås Tel: 021 - 30 38 60 Fax: 021 - 30 38 76
ezX	Layout-redigerare	Motif C, Ada	Sunrise Software Int. 170 Enterprise Center Middletown, RI 02840 USA Tel: + 1 401 847 7868
GINA	Layout-redigerare	Motif C, C++, Common Lisp	GMD (German National Research Center for Computer Science) Box 1240 D-5205 Sankt Augustin 1 Tyskland email: berlage@gmdzi.gmd (Erbjuds gratis)



Tabell 3: Verktögsleverantörer

Verktögsnamn	Typ	Fönstermiljö & Programmeringsspråk	Leverantör
HyperCard	Lekmannaverktyg Totalverktyg	Macintosh	Apple Computer Box 31 164 93 Kista Tel: 08 703 30 00 Fax: 08 703 30 01
Interface Builder	Layoutredigerare	Nextstep Objective C	Next Sverige AB Kanalvägen 10A 194 61 Upplands Väsby Tel: 08 590 889 30 Fax: 08 592 505 90
ISA Dialog Manager	Layoutredigerare	Motif Windows Presentation Manager C, Cobol, Fortran, Pascal	ISA Azenbergstrasse 35 D-7000 Stuttgart 1 Tyskland
MOB	Layoutredigerare	Motif C, C++	KOVI Design Automation 2350 Mission College Suite 390 Santa Clara, CA 95054 USA Tel: + 1 408 982 3840
ObjectBuilder	Layoutredigerare	Motif Open Look C++	Memory Data AB Box 1467 171 28 Solna Tel: 08 705 76 00 Fax: 08 705 76 01
Objectviews C++	Multiplattformverktögs	Motif Open Look C++	Quest Systems Corp. 2700 Augustine Drive M/S 242 Santa Clara, CA 95054 USA Tel: +1 408 496 1900
Open Dialogue	Layoutredigerare	Motif C, C++, Fortran, Pascal, Ada, Unix	Hewlett-Packard Sverige Box 19 164 93 Kista Tel: 08 750 20 00 Fax: 08 752 77 81

Tabell 3: Verktygsleverantörer

Verktygsnamn	Typ	Fönstermiljö & Programmerings-språk	Leverantör
Open Interface	Multiplatt- formsverktyg  Layout- redigerare	Motif Open Look Macintosh Windows Presentation Manager C	Duke Systems AB Kottbygatan 7 164 75 Kista Tel: 08 - 703 27 80 Fax: 08 - 703 94 10
Open Look Express	Layout- redigerare	Open Look	Quest Systems Corp. 2700 Augustine Drive M/S 242 Santa Clara, CA 95054 USA Tel: +1 408 496 1900
Phantom	Renoverings- verktyg	Presentation Manager	Entra Phantom Gustavslundsvägen 151G 161 36 Bromma Tel: 08 80 97 00 Fax: 08 80 97 33
Plus	Lekmanna- verktyg  Totalverktyg	Windows  Hypertalk	Idea AB Box 35 760 40 Vaddö Tel: 0176 524 00 Fax: 0176 523 90
Prototyper	Layout- redigerare	Macintosh  C, Pascal	SmethersBarnes P.O. Box 639 Portland, OR 97207 USA Tel: +1 503 274 7179
ScreenMachine	Layout- redigerare	Motif  Ada	Objective Interface Syst. 1875 Campus Common Dr. Suite 310 Reston, Virginia 22091 USA Tel: + 1 703 264 1900
Serpent/Agora	UIMS	Motif Open Look  C, Ada, SLANG	ASET 2957 Clairmont Road Suite 200 Atlanta, GA 30329 USA



Tabell 3: Verktygsleverantörer

Verktygsnamn	Typ	Fönstermiljö & Programmerings-språk	Leverantör
Skylight	UIMS	Motif	Interactive Technology Park Plaza West #416 10700 S.W. Beaverton Hillsdale Highway Beaverton, OR 97005 USA
SL-GMS	UIMS	Motif Open Look	SL Corp. 240 Tamal Vista Blvd. Suite 100, Hunt Plaza, Corte Madera, CA 94925 USA
Snapix	Layout-redigerare	Motif	ADNT 4 rue Louise Massote 78430 BUC Frankrike Tel: + 33 1 3956 5333
SoftOption	Layout-redigerare	Motif Windows	Cambridge Connectivity 197 High Street Cottenham, Cambridge CB4 4RX Storbritannien
Supercard	Lekmanna- verktyg  Totalverktyg	Macintosh  HyperTalk	Aldus Sverige Box 47 164 93 Kista Tel: 08 752 30 00 Fax: 08 751 49 55
TAE Plus	UIMS	Motif  C, C++, Ada Unix, VMS	TAE Support Office Code 522 NASA/GSFC Greenbelt, MD 20771 USA Tel: +1 301 286 6034
TeleUse	UIMS  Layout-redigerare	Motif  C, C++ D Unix, VMS	Alsys/Telesoft Box 2014 149 02 Nynäshamn Tel: 08 520 690 10 Fax: 08 520 209 65

Tabell 3: Verktygsleverantörer

Verktygsnamn	Typ	Fönstermiljö & Programmerings-språk	Leverantör
Tigre Interface Designer	Totalverktyg	Motif Open Look  Smalltalk	Tigre Object Systems 3004 Mission St. Suite D Santa Cruz, CA 95060 USA
UIM/X	Layout-redigerare	Motif  C Unix, VMS	Visual Edge Software 3870 Côte Vertu Montreal, Québec H4R 1V4 Canada Tel + 1 514 332 6430
Visual Basic	Totalverktyg  Lekmannaverktyg	Windows DOS  Basic	Microsoft Box 27 164 93 Kista Tel: 08 752 56 00
VisualWorks	Totalverktyg Multiplattform-verktyg	Motif Open Look Macintosh Windows Smalltalk	Memory Data AB Box 1467 171 28 Solna Tel: 08 705 76 00 Fax: 08 705 76 01
VUIT	Layout-redigerare	Motif  C Ultrix, VMS	Digital Equipment Allén 6 172 89 Sundbyberg Tel: 08 629 80 00 Fax: 08 28 85 36
VZ Programmer	Totalverktyg	Windows Presentation Manager  C++	Objektutveckling AB Box 143 234 23 Lomma Tel: 010 201 37 93 Fax: 040 41 27 76
Wintran	Layout-redigerare	Windows	Guideware 2483 Old Middlefield Way Suite 224 Mountain View, CA 94043 Tel: + 1 415 969 6851
XDesigner	Layout-redigerare	Motif  C	Imperial Software Technology 95 London Street Reading RG1 4QA Storbritannien Tel: + 44 734 587 055



Tabell 3: Verktögsleverantörer

Verktögsnamn	Typ	Fönstermiljö & Programmerings-språk	Leverantör
XFacemaker	UIMS Layout-redigerare	Motif C	Advanced Imaging Parmmätarg 22 112 24 Stockholm Tel: 010 223 16 68
X-Pressions User Interface Systems	UIMS	Motif C	Jonathan 150 Boush St. Town Point Center Norfolk, VA 23510 USA
XSculptor	Layout-redigerare	Motif Open Look	KOVI Design Automation 2350 Mission College Suite 390 Santa Clara, CA 95054 USA Tel: + 1 408 982 3840
Xtra XWidgets	UIMS	Motif Open Look	Graphical Software Technology 1559 E. Pacific Coast Highw. Suite 300 Hermosa BEach, CA 90254 USA
XVT	Multiplattform-verktyg Layout-redigerare	Motif Open Look Macintosh Windows Presentation Manager C Unix, VMS	Programvarubolaget Edvina Box 6036 191 06 Sollentuna  Tel: 08 626 20 50 Fax: 08 626 20 51







# SNABBFAKTA OM SVENSKA INSTITUTET FÖR SYSTEMUTVECKLING

---

Svenska Institutet för Systemutveckling, SISU, är en forskningsorganisation som utvecklar och sprider kunskap om informationsteknologins användningsområden.

SISU instiftades av regeringen 1984.  
1992 omsatte institutet, som finansieras av svenskt näringsliv genom medlemskap och NUTEK, ca 35 Mkr. SISU har 45 anställda.  
Verkställande direktör är professor Thomas Falk.

Institutet är indelat i tre kunskapsområden:

- Verksamhetsanalys
- Informationssystem
- Interaktion & Kommunikation

SISU har tagit initiativ till fem nya forskningsområden av stor ekonomisk dignitet för svenskt näringsliv och offentlig förvaltning:

- Systemarvet
- Systemutvecklingens ledtider & kvalitet
- Affärskommunikation & EDI
- Informationsteknologins ekonomi & management
- Informationsrationalisering

Institutet har ett stort internationellt kontaktnät och samarbetar med ett flertal universitet och företag, framför allt i Europa.

Kunskaper överförs genom rapporter, utredningar, kurser, seminarier, kompetensnät och, framför allt, genom medlemmarnas deltagande i projekt. Den långsiktiga forskningsinriktningen läggs fast i treåriga ramprogram, i samverkan med medlemmarna.

---



Electrum 212, 164 40 Kista  
Isafjordsgatan 26  
Telefon 08-752 16 00 Telefax 08-752 68 00