Organisationsenhet
Division Communications

Tjänsteställe, Utfärdad av, Telefon
KFF, Johan Nyvaller
KFF, Hans Ström
Mottagare

Datum
2000-09-08

Dokumentbeteckning
GN/SDF:333

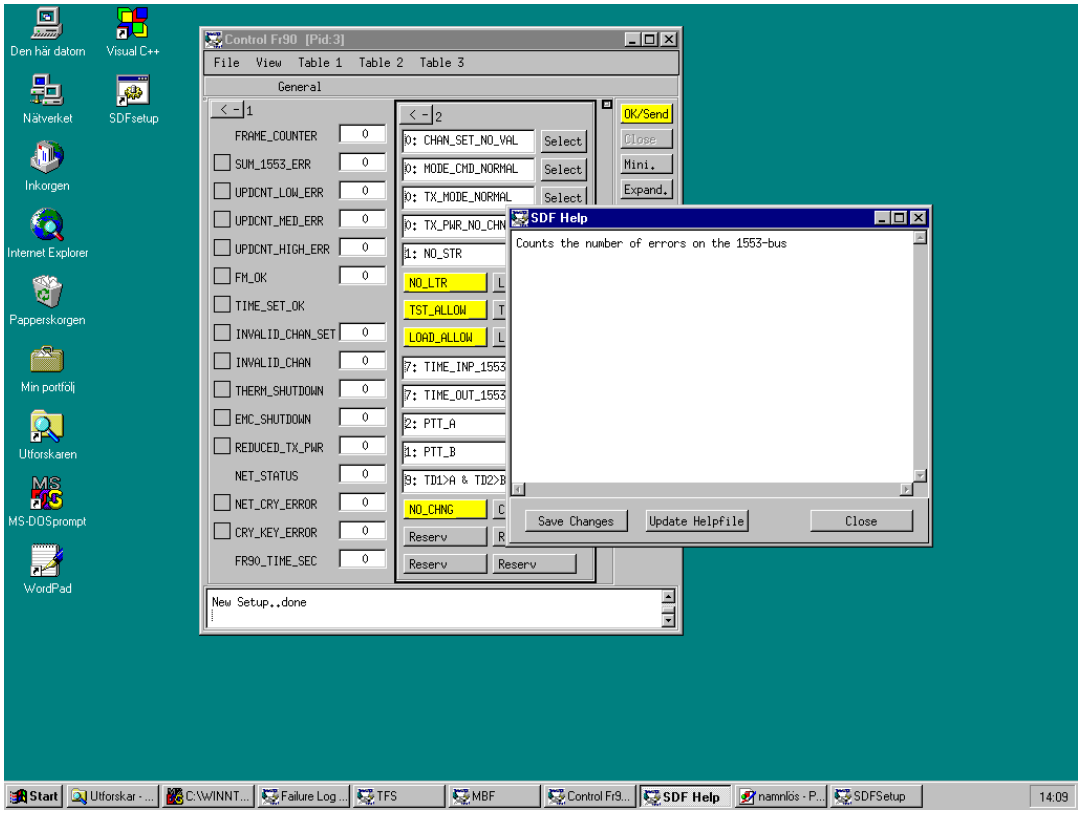
Informationsklass
Öppen

Hjälptexthantering med realtidskrav

För

AerotechTelub's

SystemDatorFunktion (SDF)



Utfört av:
Johan Nyvaller och Hans Ström

Handledare:
Rikard Lindell

Arbetet utfört vid
Institutionen för Datateknik
Mälardalens Högskola
VÄSTERÅS 2000-09-25

Sammanfattning

Detta examensarbete har till uppgift att utreda behovet av en hjälptextfunktion till System Dator Funktionen (SDF) som ingår i Tactical Radio System (TARAS) testrigg. Syftet är att finna en lösning på problemet och realisera det. Arbetet har delats in i fem delmoment.

- Utredning av behov.
- Utredning av genomförande och teknik.
- Analys och design.
- Genomförande.
- Testning

Varje moment utgör ett delmål i arbetet där det handlar om att få fram hur hjälptexterna ska fungera, integreras i systemet och testas.

En stor del av arbetet har varit att sätta sig in i hur systemet fungerar. Mycket tid har gått åt till att läsa dokumentation, samt att lära sig verktyget TeleUSE som har använts till att skapa gränssnitten, och språket D som beskriver en del av systemets beteende.

Realtidskraven gör att alla sökningar måste ha en maximal tid för att hitta rätt text vilket utesluter all form av fritextsökning. Problemet löstes med en multidimensionell matris som innehåller adresser till textsträngar. Indexet till matrisen får man genom att använda sig av varje MMI-komponent unika ID. Adressen till textsträngens start läggs på motsvarande objekts plats i matrisen. Den lösningen ger en konstant accesstid av hjälptexten till ett objekt.

Det bästa sättet att infoga funktionen i systemet är att låta den bilda en egen process. De ändringar som måste göras i det redan befintliga systemet blir då minimala. Det som behöver läggas till är kommunikation mellan de existerande processerna och hjälpprocessen samt att varje MMI-komponent måste ges ett unikt ID.

Användargränssnittet har designats för att vara så enkelt som möjligt. Ett fönster som kan visa text samt ett antal knappar för att manipulera databasen ger den funktionalitet som behövs.

Det ofta svårt att konstruera bra testfall. Speciellt svårt är det att på ett effektivt sätt testa vissa ytterligheter, t.ex. vad som händer när en stor buffert som aldrig är ämnad att bli full blir det. Händelsen måste dock hanteras om den mot all förmodan skulle inträffa. Under utvecklingen kan denna testas genom att man begränsar buffertens storlek, men vid testning av det slutgiltiga systemet är detta ej möjligt.

Den här problematiken innebär att ett test aldrig kan bli helt fullständigt hur många fall man än konstruerar. Hjälpfunktionen har dock testats så långt det är möjligt inom ramen för examensarbetet.

FÖRORD

Vi vill tacka vår handledare på AerotechTelub, Marléne Westman och alla andra på avdelning KF som har stått ut med oss. Vi riktar ett speciellt tack till de som har bjudit på tårta.

Vi vill särskilt tacka Anders Hage och Henry Sirviö för deras hjälp med att utrota en synnerligen elak bugg ur programmet.

Nästa tack går till vår handledare Rikard Lindell på Mälardalens högskola som har tagit sig an oss.

Ett stort tack till Jan Gustafsson, vår examinator för hans snabba sätt att handlägga vår rapport.

Vi vill även tacka alla som har tagit sig tid att läsa vår rapport och ge synpunkter på dess innehåll.

Vi måste prisa den goda maten på mässen, matstället på området, för deras utsökta mat. Våra sprängfulla magar och förstorade fettvalkar finner detta faktum ytterst beklagligt.

INNEHÅLLSFÖRTECKNING

1 INLEDNING

Denna rapport behandlar det examensarbete som gjordes på AerotechTelub AB i Arboga under perioden 2000-06-13 – 2000-09-08. Uppgiften som löstes var av storleken, tio poäng c nivå.

Rapporten vänder sig till personer med utbildning inom datateknik, men är även avsedd för de som i framtiden skall underhålla SDF systemet. Innehållet kräver att läsaren har grundläggande kunskaper i programmering, datatstrukturer och algoritmer, samt viss kunskap om realtidssystem.

1.1 Bakgrund

AerotechTelub har utvecklat ett system som simulerar radiofunktionen i systemdatorn i JAS39 och i centraldatorn i JA37 samt kommunikationen med flygradiosystemen Fr90 och SRa80. Det grafiska gränssnittet mellan dessa funktioner och användaren ger väldigt lite information om vad alla knappar och indikatorer som visas har för uppgift. Det finns korta namn som dessutom ofta är svårbegripliga förkortningar. Av den anledningen så finns det ett behov att kunna få fram mer information om enskilda funktioner i systemet i form av hjälptexter. Eftersom programmet skall simulera styrning av radiotrafiken mellan två jaktplan så finns det realtidskrav som måste hållas, vilket utgör en begränsning.

1.2 Syfte

Syftet med examensarbetet är att komma fram till en lämplig metod att hantera dessa hjälptexter. Metoden ska sedan realiseras genom att skapa en hjälptextfunktion och införa den i systemet. Det gäller att komma fram till ett lämpligt GUI, samt att finna en metod att hanterar hjälptexterna så att realtidskraven i systemet inte bryts.

1.3 Läsanvisningar

Simulatorn går under namnet ”systemdatorfunktionen” och kommer att förkortas SDF i texten.

När ett begrepp introduceras för första gången i texten skrivs detta med kursiv stil.

I den ursprungliga SDF dokumentationen används ordet ”objekt” för att beskriva både delar av användargränssnittet och data i systemet. Vi har dock valt att separera dessa genom att kalla de synliga delarna i gränssnitten för *komponenter* och de data och funktioner som knyts till dem för *objekt*.

2 PROBLEMBESKRIVNING

Examensarbetet går ut på att designa och implementera ett användbart hjälpsystem till systemdatorfunktionen.

För att definiera funktionaliteten hos hjälpsystemet måste en behovsutredning genomföras som kan ligga till grund för den vidare designen och problemanalysen. Dessutom kommer det att krävas en förståelse för hur systemdatorfunktionen är uppbyggd och fungerar, för att på bästa sätt kunna integrera hjälpsystemet.

3 TILLVÄGAGÅNGSSÄTT

Det här avsnittet beskriver det tillvägagångssätt som har använts för att genomföra arbetet.

Ett delmoment i arbetet har varit att definiera vilka funktioner som hjälpsystemet skall tillhandahålla. För att ta reda på detta genomfördes en behovsutredning som ledde fram till ett antal alternativa utseenden på hjälptextfunktionen. Alternativen samt det slutgiltiga valet av lösning redovisas i avsnitt 5.

För att få ett bättre underlag för en problemanlys av den valda lösningen genomfördes först en undersökning av designen och funktionen hos det existerande systemet. De delar av SDF som är relevanta för det här arbetet beskrivs i avsnitt 4.
Med detta som grund analyserades den valda lösningen för att identifiera de problem som den gav upphov till. Ett antal alternativa lösningar till dessa finns beskrivna med för och nackdelar i avsnitt 6.

Efter att ha vägt för och nackdelar mot varandra togs ett beslut om vilka lösningar som skulle användas i implementationen.

Med designen fastslagen specificerades de olika detaljerna i lösningarna och hjälpsystemet implementerades. Denna del av arbetet beskrivs i avsnitt 7.

Slutligen har systemet testats, både i utvecklingsmiljön och på plats i TARAS testrigg, testningen beskrivs i avsnitt 8.

4 BESKRIVNING AV SDF

Det här avsnittet syftar till att ge en översiktlig bild av hur systemdatorfunktionen är uppbyggd, samt belysa några detaljer i implementationen som har varit av intresse för det här arbetet.

För en mer komplett beskrivning av SDF hänvisas till Ref[1].

4.1 Vad är SDF?

Systemdatorfunktionen (SDF) är ett datorprogram som ingår som en delkomponent i en testmiljö för *TARAS* (TActical RAdio System) vid AerotechTelub AB i Arboga.
Syftet med SDF är att simulera systemdatorn i flygplanet JAS39 Gripen och centraldatorn i JA37 Viggen. SDF kommunicerar via en militär databuss med beteckningen 1553B med flygradiosystemen Fr90 (JAS39) och SRa80 (JA37).

4.2 SDF:s uppbyggnad

Systemdatorfunktionen är ett komplext system som består av flera olika processer som samverkar på olika sätt. En del processer är gränssnitt mot användaren medan andra hanterar lågnivå funktioner som t.ex. busskommunikation.

4.2.1 Processer

Här följer en lista på de processer som ingår i SDF systemet, samt en kort beskrivning av dess funktion.
Eftersom syftet med det här examensarbetet är att implementera ett hjälpsystem för SDF, så är det främst de processer som innehåller gränssnitt mot användaren som är intressanta. Dessa processer har markerats med en stjärna i tabellen

Vissa av processerna kan finnas i flera instanser, en för Fr90 och en för SRa80. För att skilja processerna åt så tilldelas varje process ett ID-nummer vid uppstart. I tabellen så anges processernas ID-nummer för respektive system inom parentes.

Processnamn	Funktion
SDF-starter (10)	Uppstart av SDF systemet, samt initiering av 1553 bussen
SDF-setup* (1)	Inläsning av setupfiler
Framehandler (2)	Hantering av 1553 bussen
Manöverbord* (Fr90 3) (SRa80 11)	Styrning av radiosystem Fr90 och SRa80 samt presentation av driftstatus
Indikatorbord* (12 & 13)	Presentation av testdata för SRa80
Timehandler (9)	Tidshantering mot resp. Radiosystem
Timesetup* (8)	Gränssnitt för tidshantering
RT-Reciever* (Fr90 5) (SRa80 15)	Styrning av datasändning till radiosystem
RT-Transmitter* (Fr90 4) (SRa80 14)	Styrning av datamottagning från radiosystem
Saver (Fr90 6) (Sra80 16)	Hantering av mottaget data
Loader (Fr90 7) (SRa80 17)	Hantering av datautsändning
PGM-Loader (21)	Hantering av programladdning av Fr90
PGM-Loadersetup* (20)	Gränssnitt för programladdning av Fr90
Medium-Handler (18)	Hantering av dataöverföring mellan SRa80 och Fr90 vid databasinitiering
Low-Handler (19)	Hantering av dataöverföring av manövermeddelanden mellan SRa80 och Fr90
Error-Log (22)	Hantering av fel-loggar från Fr90 och SRa80

4.2.2 Gränssnitten

SDF är designat för att vara ett flexibelt system, därför är flera av gränssnitten gjorda för att konfigureras dynamiskt. Detta görs via s.k. *setupfiler.* En setupfil innehåller bl.a. information om vilka objekt som skall finnas i ett gränssnitt och vilka funktioner som skall knytas till dessa.

De gränssnitt som kan konfigureras är:

- Manöverbord
- Indikatorbord
- RT-Reciever
- RT-Transmitter

I ett gränssnitt kan det finnas 2 typer av s.k. MMI-komponenter, *Indikatorkomponenter* och *manöverkomponenter*.
Indikatorkomponenter har till uppgift att presentera data för användaren och består av förutom ett namn, även en färgad ”lampa” och/eller en textruta för t.ex. en räknare.

Manöverkomponenter används för att välja data som skall skickas till radiosystemen. Manöverkomponenter kan vara av två olika typer, antingen en knapp (typ 1) eller en lista med olika val (typ 2). Typ1 förekommer alltid i par om två ömsesidigt uteslutande knappar.

Endast manöverbordet kan innehålla manöverkomponenter, övriga gränssnitt har endast indikatorkomponenter

Varje gränssnitt är uppdelat i s.k. *subsystem*. Varje subsystem kan innehålla ett visst maximalt antal komponenter av olika typ. T.ex. så kan subsystem 1 i manöverbordet maximalt innehålla 16 st indikatorkomponenter, och inga manöverkomponenter, medan subsystem 2 kan innehålla maximalt 6 st. manöverkomponenter av typ 1 och 10 st. av typ 2. I bilaga 4 finns bilder som visar exempel på några konfigurerade gränssnitt.

4.3 Konfigurering av SDF

Setupfilen är den fil som anger hur hela SDF skall konfigureras. En fullständig setupfil innehåller:

- Trafiklista för 1553 bussen.
- Texter till de olika processernas MMI-komponenter, samt kopplingar mellan komponenter och funktioner
- Styrning av datagenerering/datamottagning

De delar av setupfilen som är intressanta i det här arbetet är de som hanterar MMI-objekt, för mer detaljerad information om innehållet i setupfiler och vilka parametrar som kan anges för olika objekt hänvisas till Ref[2].

Varje process beskrivs i setupfilen som en hierarki av objekt, där varje objekt ges ett ID nummer. Detta skapar en kedja av ID nummer som kan användas för att referera till enskilda objekt.

Överst i hierarkin anges processens ***Procid*** (procedur id), dess namn och ev. andra parametrar.
På nästa nivå i hierarkin beskrivs processens ***Subid*** (subsystem), dessa används för att gruppera underliggande objekt.
Under ett subid kan det finnas två typer av objekt, ***Msgid*** (message) och ***Aobjid*** (abstrakt objekt).

Aobjid används för att skapa kopplingar mellan Mobjid (se nedan) och funktioner i SDF. För ett Aobjid anger man bl.a. vilken funktion i SDF som objektet refererar till, och varifrån eventuellt indata skall hämtas.

Msgid används för att definiera meddelanden på 1553-bussen, eller för att gruppera Mobjid.
När ett Msgid används för att definiera 1553 meddelanden så grupperar det ett antal ***Bobjid*** (bussobjekt) som beskriver kommunikation på 1553-bussen.

När ett Msgid anges i ett gränssnitt så grupperar det ett antal ***Mobjid*** (MMI objekt).

Ett Mobjid beskriver en MMI komponent. Här anges bl.a. komponentens namn (som syns i gränssnittet), vilken typ av komponent det handlar om, och en referens till antingen det Aobjid som beskriver komponentens funktion eller, för vissa manöverkomponenter, ett Bobjid.
För manöverkomponenter av typ 2 så beskriver ett Mobjid ett enskilt val i listan.

Vid inläsning av setupfilen lagras alla objekt och dess egenskaper i en datastruktur kallad *OBJDB* (**OBJ**ect **D**ata**B**ase). OBJDB är global över hela SDF systemet, och det är därifrån som t.ex. gränssnitten hämtar information om hur de skall konfigurera sig.

OBJDB lagrar även information om alla processer i systemet. Varje process har två stycken statusflaggor som indikerar om den är aktiv (exekverande) eller inte samt om exekveringen är tillfälligt stoppad eller inte. Dessutom så finns det några flaggor som anger status för hela SDF, bl.a. om en ny setupfil skall laddas, och om systemet skall avslutas. De olika processerna i SDF kontrollerar regelbundet dessa flaggor.

4.3.1 Exempel på setupfil

I det här exemplet initieras en indikatorkomponent och en manöverkomponent i manöverbordet för Fr90. Indikator 1 i subsystem 1 räknar antalet *frames* som har skickats på 1553 bussen och i subsystem 2 har två Mobjid lagts in i en lista. Resultatet av det här exemplet kan ses i bilaga 4.

```
Procid:
        Namn ”Framehandler”
        Id 2
        Frek 60
        Frame 7

        Subid:
                Namn ”Normal”
                Id 1
                Aktiv True

                Aobjid:
                        Namn ”Framecounter”
                        Id 1
                        Funk Framehandle          #Anger vilken funktion objektet refererar till


Procid:
        Namn    "Control Fr90"
        Id      3
        Frek    2
        Aktiv   Fr90on

        Subid:
                Namn    "General"
                Id      1

                Msgid:
                        Id      1

                        Mobjid:     Id 1    Namn "FRAME COUNTER"
                                    Ref 0 Procid 2 Subid 1 Aobjid 1          # Anger varifrån
data skall hämtas
                                    Typ Valonly              # Anger att ingen indikator lampa
skall visas
                                    Indata Counter           # Anger typ av indata
        Subid:
                Namn     ”General”
                Id      2

                Msgid:
                        Id 1 Namn ”Manöver Test”
                        #Ref 0 Procid ?? Subid ?? Msgid ?? Bobjid ??        I det här exemplet
finns inga bussobjekt
                        Typ Output
                        Indata Procid 3 Subid 2 Msgid 1 Mobjid 2            #default data
                        Kod 1

                        Mobjid:     Id 1 Namn ”Listval nr 1”
                                    Typ Lista Kod 0

                        Mobjid:     Id 1 Namn ”Listval nr 2”
                                    Typ Lista Kod 1
```

4.4 Driftmoder i SDF

SDF kan användas i 3 olika driftmoder, *Normalmod*, *Initieringsmod* och *Programladdningsmod*.

I normalmod sker all normal hantering av radiosystemen Fr90 och SRa80.
I initieringsmod sker endast databasinitiering av Fr90 och SRra80.
I programladdningsmod sker endast programladdning av Fr90.

Vilken mod som skall användas styrs dels genom innehållet i setupfilen och dels genom ett val som görs av användaren vid inläsning av setupfilen. Det som främst skiljer de olika moderna åt är vilka processer som startas automatiskt vid systemstart samt vilka som kan startas av användaren.

4.5 Implementationsdetaljer

4.5.1 Allmänt

Alla lågnivå funktioner i SDF är implementerade i **ANSI-C,** medan gränssnitten följer X-Windows standard och är skapade med verktyget **TeleUSE**. Att X-Windows standard används innebär att det måste finnas en X-server tillgänglig under Windows NT för att systemet skall fungera.
Funktionen hos gränssnitten är implementerade i ett slags script-språk kallat **D**. D-kod har en högre abstraktionsnivå än C och gör att man inte behöver bekymra sig om detaljer i X-Windows. Det finns även möjlighet att anropa C-funktioner direkt från D-kod. Vid kompilering av D-kod översätts den först till C-kod.

Mer om D-kod finns att läsa i avsnitt 7.4.1

4.5.2 Källkodsstruktur

För de processer som har gränssnitt mot användaren, så består källkoden som regel av två delar. Dels en C-fil som innehåller lågnivå funktioner för att t.ex. skriva och läsa från buffrar, och dels en D-fil som innehåller kod för att hantera inmatningar från användaren.

Dessutom inkluderar alla processer en enhet som kallas *adi* (**A**pplication **D**evice **I**nterface) som innehåller alla de funktioner som är specifika för operativsystemet (Windows NT).

4.5.3 Processkommunikation

För kommunikation mellan processer används funktioner för delat minne som tillhandahålls av Windows NT. Närmare bestämt används filer på hårddisken som mappas mot processernas lokala minnesrymder. För mer information om denna metod, se Ref[4]. Filerna kallas *buffrar* i SDF, OBJDB är ett exempel på en sådan. Vilka buffrar som skall skapas i systemet bestäms av en fil kallad **devices.map**, som anger buffrarnas namn samt deras maximala storlek.

4.5.4 Start av processer

Vilka processer som ska startas i de olika moderna bestäms av 3 st filer , **normal.aut, init.aut** och **pgml.aut**. Dessa är vanliga textfiler där man anger filnamn och parametrar (process id) för de processer som skall startas i respektive mod.
Det är dock inte alla processer som startas via dessa filer, t.ex. så är start av ”framehandler” hårdkodad i systemet.

5 BEHOVSUTREDNING

För att ta reda hur hjälptexterna ska fungera och bäst ge den hjälp som behövs så gjordes en behovsutredning. Det är även intressant att veta i vilken omfattning som hjälptextfunktionen ska gälla.

5.1 Tillvägagångssätt

Genom att prata med några personer som använt SDF så insamlades information om vilka funktioner som är önskvärda.

5.2 Resultat

Undersökningen visar att man vill kunna få fram en kort hjälp snabbt, men man vill även kunna söka vidare på relaterat ämne via länkar eller via en fritext sökning. Önskemålen kan formuleras i några representativa förslag som ger olika förutsättningar.

- Fritextsökning. –Söka efter ett ord i en textfil och visa det avsnittet.
- Meny alternativ via högermusknapp. –Vid ett högerklick på en komponent visas en meny där man kan välja funktioner, läs hjälp/ändra hjälp m.m.
- Direkthjälp via höger musknapp. –Genom att högerklicka på ett objekt så ska ett fönster med relaterad text visas.

- Välja hjälp från lista. –Alla objekt som har en relaterad hjälptext ska läggas i en lista som kan nås från en meny.

5.3 Analys

Med utgång från hur systemet fungerar och dess krav så gjordes en analys. Syftet var att finna det förslag eller den kombination av förslag som kan uppfylla önskemålen bäst samt kunna genomföra.

- Fritextsökning. –Går ej att få ett värsta fall om man inte begränsar storleken på textfilen som får finnas. Linjär sökning av text belastar processorn hårt och är därför olämplig i ett system som har realtidskrav som SDF har. Dessutom kräver en sökfunktion mycket arbete för att bli användbar.
- Meny på höger musknapp. –Det verkar lite krångligt ur användarens synvinkel att behöva gå via en meny. Alternativet ger dock användaren flera valmöjligheter vad gäller hantering av hjälpfunktionen.
- Direkthjälp via höger musknapp. -Användaren får önskad hjälptext omedelbart, om den skulle saknas så får han/hon reda på det omgående.
- Välja hjälp från en lista. -Att behöva söka igenom en lista efter en hjälptext verkar lite krångligt. Speciellt om det finns många hjälptexter så blir hela proceduren ineffektiv. Det kan även vara svårt att få en bekräftelse på att det inte finns någon hjälptext tillgänglig.

5.4 Omfattning

SDF är ett system med flera olika funktioner och flera fönster. Behovet att se en hjälptext är kanske inte lika stort i alla fönster? Vissa funktioner används inte så ofta och vissa är intuitiva?

Vid en närmare granskning så ser man att de delar som är krångliga och ändras i setup-filen emellanåt är manöverborden och indikatorborden, medan fönstret för felloggen knappast behöver ändras eller någon närmare förklaring för sina funktioner. Vi har därför valt att endast implementera hjälpfunktionen för de processer vars gränssnitt kan konfigureras via setupfilen, d.v.s. Manöverbordet, Indikatorborden, RT-Reciever och RT-Transmitter

5.5 Slutsats

Att använda sig av funktionen att söka efter en text i en fil eller använda sig av någon sorts nyckelords sökning är inte effektivt och gör bara systemet långsamt. Därför är det inte lämpligt att använd sig av en sådan funktion till hjälpen i SDF.

Om man gör den omedelbara hjälptexten som kan fås i ett popup fönster tillräckligt stor, så kan den hjälp som behövs ges. Av den anledningen så erbjuder lösningen med ett popup fönster en bra och intuitiv lösning till problemet.

Hjälptextfunktionen blir tillräcklig om det finns ett fönster som visar en text åt gången, den senast valda. Fönstret skall kunna stängas och öppnas automatiskt när ett objekt högerklickas. Fönstret kan även ha några knappar som ger en viss funktionalitet som t.ex. att spara en inskriven text.

6 PROBLEMANALYS

Den föreslagna lösningen ger upphov till ett antal frågor som måste besvaras innan hjälpsystemet kan implementeras. Dessa kan delas upp i fem punkter.

1. Hur ska hjälpen integreras med resten av SDF systemet?
2. Hur ska de olika hjälptexterna lagras?
3. Hur ska en hjälptext knytas till ett specifikt objekt?
4. Hur ska man skilja på hjälp för olika setup-filer?
5. Var ska hjälpfunktionen förvara hjälptexterna, fil eller minnet?

Varje punkt har besvarats med två alternativ där alternativets för och nackdelar också tas upp. Denna analys har legat till grund för den slutliga lösningen på det problemet.

6.1 Integrering i SDF

Alternativ 1
Hjälpsystemet implementeras som en egen process som exekverar parallellt med övriga processer i SDF.
Fördelar:
Hjälpsystemet kan lätt kopplas bort om det inte behövs.
Flera processer kan utnyttja samma hjälpprocess.
Nackdelar:
Någon form av interprocess kommunikation måste implementeras

Alternativ 2
Hjälpen implementeras i varje gränssnitt där hjälp önskas.
Fördelar:
Man slipper implementera kommunikation mellan olika processer vilket, leder till kortare svarstider.
Nackdelar: Samma jobb måste göras i alla gränssnitt där hjälpen

skall finnas, vilket avsevärt försvårar framtida förändringar och underhåll.
En tidskritisk process kan bli upptagen av att hantera hjälpen istället för att sköta sina ordinarie funktioner, t.ex. trafik på 1553-bussen.

6.2 Lagring av hjälptexter

Alternativ 1

Hjälptexterna lagras som ren text i en separat fil som på något sätt knyts till motsvarande setup-fil.
Fördelar:
Existerande setupfiler behöver inte modifieras. Man slipper göra ingrepp i koden som läser setupfiler.
Det finns möjlighet att ha flera olika hjälpfiler till samma setupfil (ex. olika språk).
Nackdelar:
Det blir ingen tydlig koppling mellan hjälptexter och motsvarande funktioner.
Det blir ytterligare en fil i systemet att hålla reda på.

Alternativ 2

Hjälptexterna lagras i setupfilen tillsammans med det objekt de tillhör.
Fördelar:
Det blir en tydlig koppling mellan hjälptexten och motsvarande funktion.
Hjälpen kan skrivas in samtidigt som setupfilen skapas.
Nackdelar:
Setupfilen blir ännu mer komplicerad och svårläst, speciellt om man vill ha långa texter.
Koden som läser in setupfilen måste modifieras för att hantera hjälptexterna.

6.3 Koppling av hjälptext till objekt

I det fall då hjälptexterna inte skrivs direkt i setupfilen så behövs det någon mekanism som knyter en text till ett visst objekt.

Alternativ 1

Den befintliga hierarkin med *procid->subid->msgid->mobjid* används för att identifiera rätt hjälptext.
Fördelar:
Blir lätt att implementera eftersom inga tillägg behöver göras till setupfilen.
Nackdelar:
Man kopplar texten till ett specifik objekt i gränssnittet, istället för

till objektets funktion. Detta gör det svårt att återanvända hjälpfiler om man byter plats på funktioner.
De objekt som skall vara klickbara måste känna till ovanstående ID-nummer.
Uppslagning av texter måste ske i flera steg, vilket kan ta längre tid än i nästa alternativ.

Alternativ 2

Varje objekt i setupfilen som har en hjälptext får ett associerat indexnummer som knyter objektet till hjälptexten.
Fördelar:
En text kopplas direkt till den funktion den beskriver, istället för till den knapp som utlöser funktionen.
Man kan använda indexnumret för att direkt indexera texter i minnet, vilket innebär snabb uppslagning av texter.
Nackdelar:
Setupfilen och inläsningen av denna måste modifieras för att hantera indexnumren.
Objektdatabasen (OBJDB) måste modifieras för att kunna lagra indexnumren.

6.4 Koppling av hjälp till setupfil

I det fall då hjälptexterna skrivs i en separat fil så krävs det en mekanism som kopplar en hjälpfil till rätt setupfil.

Alternativ 1

I setupfilen specificeras det med något nytt nyckelord var den tillhörande hjälpfilen finns.
Fördelar:
En användare av systemet behöver inte känna till något om hjälpfilen och var den finns.
Nackdelar:
Inläsningen av setupfilen måste modifieras.

Alternativ 2

Setup programmet modifieras så att man förutom setupfil även anger vilken hjälpfil som skall användas.
Fördelar:
Setupfilen behöver inte ändras om man vill byta hjälpfil.
Det är lätt att byta ut hjälpfilen (personliga hjälpfiler, olika språk m.m)
Nackdelar:
Gränssnittet för setupprogrammet måste modifieras.
Kopplingen mellan setup och hjälpfil blir ganska lös.
Om en felaktig hjälpfil anges riskerar man att få fel hjälpinformation.

6.5 Uppslagning av hjälptexter

Alternativ 1

Hjälpfunktionen hanterar, läser och skriver, hjälptexterna direkt från fil.

Fördelar:

Det behövs inte skapas minnesutrymme för hjälptexterna. Den minnesrymd som processen använder behöver inte bli så stor.

Nackdelar:

Fil hantering tar tid samt att de olika posterna kan inte accessas direkt.

Alternativ 2

Alla hjälptexter lagras i en databas som finns allokerad i minnet.

Fördelar:

Texterna kan lagras på ett sådant sätt att det går snabbt att hitta dessa. Det går snabbare att accessa primär minnet än en fil.

Nackdelar:

Att lagra alla texter samt sökstrukturen i primärminnet kräver att det finns gott om utrymme där. Det kommer även att ta tid att bygga upp databasen samt att spara ner den igen.

6.6 Design av hjälpfunktionen

Vid beslutet om vilka lösningar som skall användas så har hänsyn tagits till de för och nackdelar som redovisas för varje alternativ. Dessutom har hänsyn tagits till de realtidskrav som finns i SDF genom att välja lösningar som stör SDF:s egentliga arbete så lite som möjligt.

Hjälpsystemet implementeras som en egen process som integreras med övriga processer i SDF. Detta beslut har tagits eftersom det innebär att inga stora ingrepp behöver göras i någon av de befintliga processern. Beslutet ger även den bästa lösning ur återanvändnings och underhållssynpunkt, eftersom alla ingrepp i hjälpprocessens kod i framtiden inte kommer att påverka de andra processerna direkt. Risken för att en tidskritisk process ska låsas upp minskar också, eftersom hjälpprocessen kan ges lägre prioritet och på det viset tvingas ge de kritiska processerna företräde.

Hjälptexterna sparas i en egen fil, eftersom setupfilerna är tillräckligt stora och svårlästa som de är. Att lägga in hjälptexter i dessa skulle bara göra dem ännu större och mera svårlästa. Dessutom blir det en mer komplicerad process att läsa dessa filer för setup programmet. Den utökade funktionen som krävs för att ha hjälptexter i setupfilen

kräver att programmets redan komplexa struktur ändras till en ännu mera komplex, eftersom det kommer att finnas flera saker att ta hänsyn till i setup filen. Operationer som att uppdatera och lägga in nya texter i en setupfil kommer även dessa att få hög komplexitet, då dessa texter ska infogas i en fil som innehåller massa annan information som saknar betydelse för hjälptexterna. Genom att ha hjälptexterna i en separat fil så kan dessa lagras tillsammans med den information som är intressant för hjälptextfunktionen. Funktionerna som då krävs för att handskas med den filen blir då mindre komplexa.

Eftersom systemet har realtidskrav så måste alla accesser vara snabba och ta en konstant tid. Det är därför direkt olämpligt att använda sig av filhantering. För att kunna garantera att alla hjälptexter kan hittas och levereras inom en viss tid, så måste de lagras i någon lämplig struktur eller på något annat sätt som gör att de kan hittas omgående. Det kan bara göras om dessa finns lagrade i primärminnet. Av den anledningen så ska en databas som handhar hjälptexterna skapas, för analys och design av denna se avsnitt 7.6.

Vi kommer att använda den befintliga hierarkin med ID-nummer för att knyta en text till en komponent, eftersom detta innebär det minsta ingreppet i den befintliga koden. Dessa ID-nummer gör det även lättare att identifiera de komponenter som behöver hjälp, samt att ett unikt ID-nummer kan användas till att indexera datastrukturer som en matris.

Vilken hjälpfil som skall användas specificeras direkt i setupfilen, eftersom det lägger ett mindre ansvar på slutanvändaren av systemet.

7 IMPLEMENTATION

Följande avsnitt beskriver detaljer i den implementation av hjälpfunktionen som har gjorts.

7.1 Utformning av användargränssnittet

Eftersom hjälpfunktionen är designad för att vara enkel att använda så krävs inget komplicerat gränssnitt.

Huvuddelen av gränssnittet, ett fönster som visar text, är redan given av designen. Vad som återstår är att definera ytterligare funktionalitet.

Eftersom en användare skall kunna skriva in nya texter och modifiera existerande under programkörning så görs textfönstret redigerbart, dessutom läggs en knapp till för att spara ändringar i databasen.

Om mycket ny text har skrivits in (t.ex. vid skapandet av en ny hjälpfil) så bör man kunna uppdatera hjälpfilen utan att avsluta hela SDF (se avsnittet om hantering av hjälpfiler). Även denna funktion utförs lämpligen via en knapp i gränssnittet.

Användaren bör kunna dölja hjälpfönstret när det inte behövs längre, ytterligare en knapp krävs för detta.

Slutligen kan det vara intressant att veta vilken hjälpfil som används. Istället för att visa detta i hjälpfönstret så utökas gränssnittet för *sdf-setup,* som redan visar aktuell setupfil, till att även visa aktuell hjälpfil.

Det färdiga gränssnittet får då följande utseende:

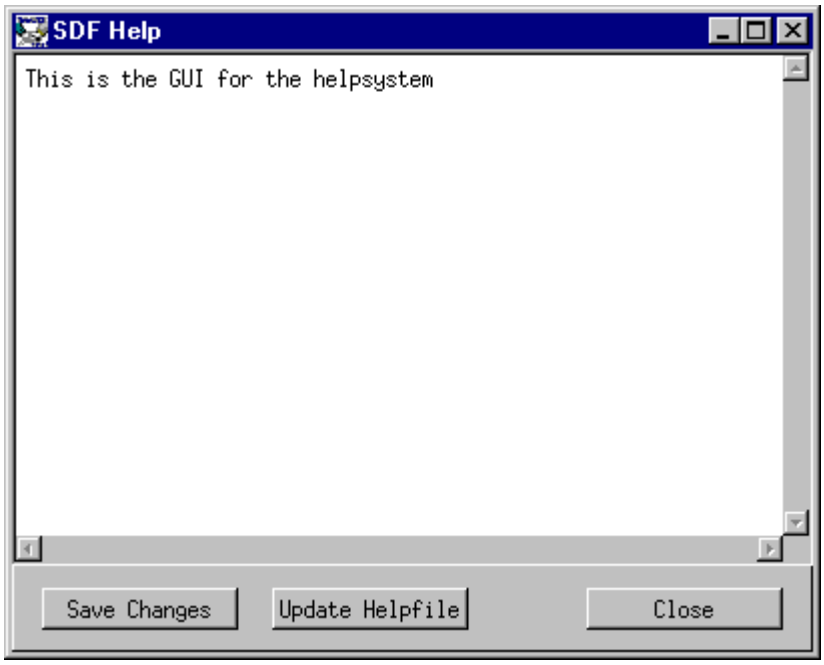


Bild 1: *Gränssnittet för hjälpfunktionen*

7.2 Hjälpfiler

Följande avsnitt specificerar detaljer om de filer som skall användas för att lagra hjälptexter.

7.2.1 Filformat

En hjälpfil byggs upp av poster, där varje post har följande utseende:

ID: <procid> <subid> <msgid> <mobjid>
Några rader med
lämplig text till ett objekt.

Där ”ID: xx xx xx xx” anger ID-numren för det objekt som efterföljande text tillhör.

7.2.2 Hantering av hjälpfiler

Vid inläsning av en ny setupfil i SDF så skickas namnet på den associerade hjälpfilen till hjälpprocessen, som utifrån den bygger upp textdatabasen i minnet. Alla förändringar som görs till databasen sker endast i minnet, hjälpfilen uppdateras först när hjälpprocessen avslutas, eller när användaren själv väljer att göra det. Syftet med detta är att minimera antalet diskaccesser eftersom det kan störa funktionen hos SDF.

Om en setupfil inte specificerar någon hjälpfil så används en ”default” fil med namnet **default.hlp**.

7.2.3 Begränsningar

Av programmeringstekniska skäl så har längden på en rad i hjälpfilen begränsats till 256 tecken.
Längden på den text som kan anges för ett enskilt objekt har begränsats, dels för att hålla nere storleken på textdatabasen, och dels för att kunna definera ett ”värsta fall” vid testningen av hjälpfunktionen. Gränsen är godtyckligt satt till 1Kb, vilket borde räcka för de hjälpbehov som finns i SDF. Om behov av mer text skulle uppstå i framtiden så kan dock storleken enkelt utökas genom en enda ändring i källkoden.

7.3 Interprocess kommunikation

För kommunikationen mellan hjälpprocessen och övriga processer i SDF så införs en ny global buffert kallad **HLPBUF**, som hanteras på samma sätt som övriga buffrar i systemet (se avsnitt 4.5.3). Valet att använda en global buffert för kommunikation gjordes eftersom det innebär att man kan utnyttja redan befintlig funktionalitet i SDF.

Buffertens utseende:
Filnamn: **HLPBUF**
Storlek: **Max 512 byte**
Innehåll:

```
{
        int procid, subid, msgid, mobjid;
        int service_request;
        int busy;
        char help_filename[256];
}
```

När en process begär hjälp om ett objekt så fylls först de fyra ID-fälten i med ID-numren för det aktuella objektet. Därefter sätts *service_request* flaggan som signalerar till hjälpprocessen att en hjälpförfrågan finns.

busy flaggan sätts av hjälpprocessen under tiden som databasen byggs upp från hjälpfilen vilket orsakar ett tillfälligt stop i starten av SDF. Detta görs eftersom långa filaccesser när SDF är aktiv kan störa tidskritiska funktioner.

help_filename innehåller namnet på den hjälpfil som skall användas, och ifylls av processen **sdf-setup** vid inläsning av aktuell setupfil.

7.4 Implementation av användargränssnittet

Användargränssnittet har precis som alla andra gränssnitt i SDF skapats med verktyget **TeleUSE**, och funktionaliteten är beskriven med D-kod samt med ett antal C-funktioner som är gränssnitt mot databasen och resten av SDF.

7.4.1 Allmänt om TeleUSE program

Ett program skapat med **TeleUSE** är *händelsestyrt*, d.v.s. allt som sker i programmet genererar händelser som kan aktivera olika kodavsnitt där programmets arbete utförs. Ett sådant kodavsnitt kallas i D-kod för en *Devent*. Exempel på sådant som kan orsaka händelser är t.ex. ett musklick på en knapp, eller att användaren avslutar programmet. Man kan även explicit anropa en Devent från en annan med funktionen **Send()**.

Gränssnittet i ett **TeleUSE** program byggs upp av s.k. *widgets*. En widget kan t.ex. vara en knapp i ett fönster (som också är en widget), en meny eller en textruta. En widget har olika egenskaper som bestämmer hur de ser ut och beter sig, t.ex. så kan man för en knapp definera dess storlek, position, färg m.m. samt vilken Devent som

skall aktiveras när man klickar på den.
Man kan också definera egna egenskaper för en widget, vilket är något som har utnyttjats vid implementationen av hjälpfunktionen.

7.4.2 Uppbyggnad

D-koden för användargränssnittet är uppbyggd av totalt 6 st Devents.

Setup är det Devent som anropas vid inläsning av en ny setupfil i SDF. Den sparar ned aktuell databas på fil, och läser in en ny hjälpfil.
SaveText anropas när en ny eller ändrad text skall sparas i databasen.
FileBackup anropas när användaren vill spara databasen på fil.
HideWindow anropas när användaren vill dölja hjälpfönstret.
Runner är det Devent som gör det största jobbet. Detta Devent som körs periodiskt kontrollerar om någon process har skickat en begäran om hjälp, och hämtar då rätt hjälptext från databasen. Här kontrolleras också vissa statusflaggor i OBJDB som indikerar om en ny setupfil skall laddas, eller om SDF skall avslutas.
Terminate anropas vi programavslut, och sparar aktuell databas samt frigör en del minne.

Förutom dessa så finns det i alla D-program ett Devent som heter **INITIALLY** som anropas när programmet startas. Här tolkas programmets inargument, och en del initieringar görs.

Förutom D-koden så ingår även vissa C-funktioner i gränssnittet. Dessa används bl.a. för att läsa vissa statusflaggor i HLPBUF och OBJDB, samt fungerar som gränssnitt till databasen.

Flödesdiagram för **Runner, Setup** och **Terminate** eventen finns i bilaga 5. Fullständig programkod (C & D) finns i bilaga 7.

7.5 Modifikationer av befintliga gränssnitt

För att hjälpfunktionen skall fungera måste naturligtvis de redan existerande gränssnitten i SDF stödja detta.

Gemensamt för alla gränssnitt är att ett nytt Devent, **GetHelp()**, har skapats som anropas vid ett högerklick på ett objekt. Vilken del av komponenten som reagerar på klicket skiljer sig mellan indikator och manöverkomponenter.

Indikator och manöverkomponenter motsvaras av widgets i TeleUSE, dessa widgets är i sin tur uppbyggda av en hierarki av andra widgets.

Ex. Hierarkin för en indikatorkomponent.

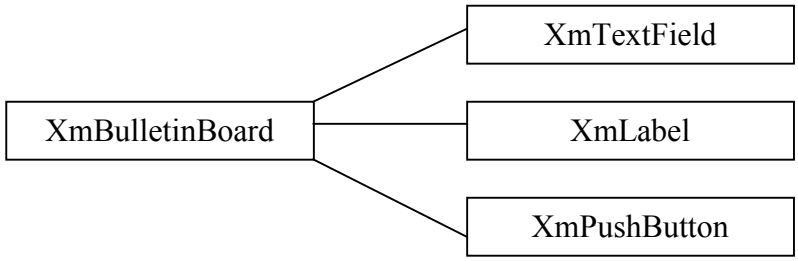


Bild 2. *Bilden visar hierarkin för en indikatorkomponent*

För den widget som representerar en indikatorkomponent så har de nya attributen *subid, msgid, mobjid* definerats för den text-widget som visar objektets namn. Dessa fylls i med objektets ID nummer när gränssnittet konfigureras från OBJDB. En process *procid* finns alltid tillgängligt som en global variabel, och behöver ej lagras i komponenten.

Dessutom anropas **GetHelp()** vid ett högerklick på texten. Valet att lägga hjälpfunktionen på objektets namn gjordes eftersom det är den enda delen av en indikatorkomponent som garanterat alltid är synlig.

För den widget som representerar en manöverkomponent av typ 1 defineras de nya attributen för knappen, som även reagerar på högerklick.

Manöverkomponenter av typ 2 är lite speciella eftersom de inte direkt motsvarar ett enskilt Mobjid, utan istället innehåller en lista av sådana.
Här har hjälpfunktionen av tekniska skäl placerats på hela listan istället för på enskilda element. Detta innebär dock inte någon nackdel eftersom de olika elementen ofta hör ihop, och kan beskrivas i samma text.
ID attributen har placerats i knappen som tar fram listan med val, och ett högerklick på denna aktiverar hjälpfunktionen.

7.6 Utformning av databasen

7.6.1 Analys

Varje komponent i SDF har ett unikt ID. Detta ID baseras på vilken process den tillhör, vilket subsystem den tillhör o.s.v. Det faktum att det finns en hierarki som gör att alla komponenter har ett unikt ID, kan användas till att skapa en sökväg till en komponents hjälptext. Denna sökväg blir lika lång för alla komponenter.

Komponenternas ID har utseendet: process ID . sub ID . msg ID . mobj ID

Antag att det finns en array som innehåller adresser till andra arrayer. Genom att indexera den arrayen med komponentens process ID fås en referens till en annan array, som också innehåller adresser till andra arrayer. Genom att använda komponentens sub ID på den nya arrayen så kommer man vidare till nästa array osv. Mönstret är att de olika delarna av komponentens ID, från vänster till höger, används till att ta sig igenom en samling nästlade arrayer. Hela kedjan leder till den sista arrayen som innehåller adressen till objektets hjälptext. Denna nås genom att indexera sista arrayen med sista delen av komponentens ID (mobj ID). Se bild nedan.

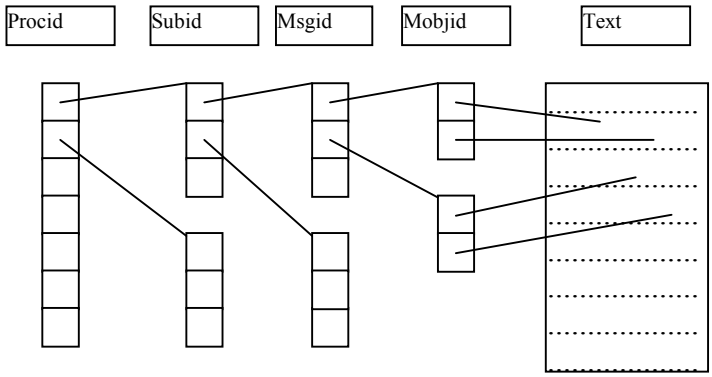


Bild 3. *Bilden visar hur sökvägen ser ut. Från första arrayen som indexeras med Procid fås en ny array, vilken beror på indexet, som har adresser till nästa hierarki av arrayer. Slutligen så fås en adress till en*

Genom att använda en struktur som har beskrivits kan en text läggas varhelst där det finns plats för den i minnet. Om det inte spelar någon roll var texten finns så länge man har en referens till den plats där den börjar, så kan man allokera ett minnesblock för ändamålet att spara texter. Genom att markera var texten slutar någonstans så kan man avsluta en text och fortsätta med att bygga på med nästa hjälptext. Alla texter kan då tillåtas ha olika längder. Ett block kan då fyllas med en massa texter av varierande längd och i en ordning som saknar betydelse för hur objektens ID förhåller sig till varandra i avseende på processtillhörighet eller någon annan del av ID betydelse.

Att ha ett stort block som kan innehålla alla texter är att föredra framför att allokera en ny minnesarea för varje text. Dels minskar den dynamiska minneshanteringen till att nästan vara obefintlig dels får man alla texter i samma minnesrymd. Dynamisk minneshantering innebär att man måste belasta operativsystemet som administrerar minnet. Detta gör att systemet blir långsammare och svårare att

förutsäga. Genom att samla alla texter i samma minnesrymd slipper man kanske att drabbas av sidfel. Sidfelen gör att operativsystemet måste mappa om minnet vilket gör att allt kommer att ta längre tid.

Eftersom det är okänt hur mycket minne som behövs vid programstart kan blocket inte skapas statiskt. En mätning av filens längd ger den minsta storleken som behövs. För att få plats och lägga in flera hjälptexter kan en offset läggas på. Då kommer blocket att få varierande längd mellan tillfällena, beroende på hur många hjälptexter det finns nedsparade. Eftersom textblocket kommer att allokeras dynamiskt behövs en kontrollstruktur som håller reda på var blocket är allokerat, hur stort det är osv.

Om det allokerade minnet för hjälptexterna tar slut måste mer minne allokeras på något sätt. Storleken på det allokerade blocket måste vara känt samt dess position i minnet. Det behövs av den anledningen skapas någon kontrollstruktur till det blocket också som kan hålla reda på minnesblocket. Det måste alltså gå att skapa flera minnesblock till hjälptexterna vid behov. Dessa behöver i sin tur ha någon kontrollstruktur som håller reda på dessa. Av den anledningen bör man nog ha kontrollstrukturerna i en länkad lista.

Då det är känt att när datorns minne hanteras dynamiskt blir systemet långsamt och risken för haverier är stort, bör sådan hantering av minnet undvikas. Även det faktum att länkade listor kan växa till långa kedjor som är tröga att traversera bidrar till att storleken på minnesblocken till hjälptexterna bör ges sådan storlek att endas en allokering av minnesblocket behöver göras. Det ska då räcka med ett ha ett kontrollblock. I värsta fall kan det behöva allokeras ytterligare ett minnesblock för hjälptexterna och ett kontrollblock till det minnesblocket, som resulterar i en länkad lista av storleken två.

7.6.2 Design

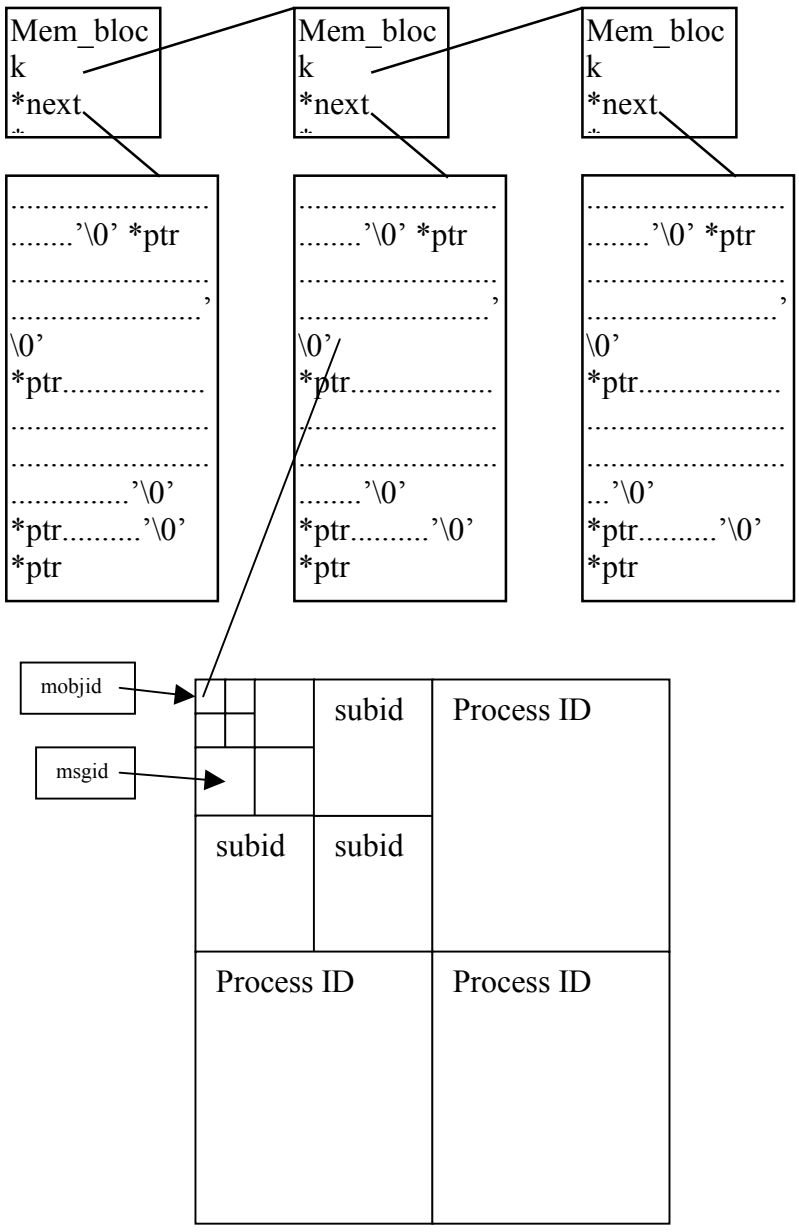


Bild 4. *Bilden försöker illustrera den slutliga designen med kontrollblock. Minnesbuffertar och en multimatris som pekar ut var de olika texterna börjar.*

Att skapa många lösa arrayer blir lite rörigt, men om man skapar en array med flera dimensioner, *en multimatris,* så kommer dessa att sitta ihop i en struktur och vara kopplade till varandra. Det blir också lätt att ändra storleken på strukturen om förutsättningarna skulle ändras. Det enda som behöver göras, är att ändra på värdet för berörd vektor. Jämfört med en samling lösa arrayer som måste allokeras eller

avallokeras i koden, vilket är ett mera komplicerat arbete än att ändra på några parametrar. Att ha många arrayer som är löst kopplade gör också att ändringar, som har med arrayerna att göra, lätt skapar fel som är svåra att hitta. Alla med erfarenhet av underhåll av program vet hur svårt det är att följa sådana strukturer.

Eftersom en komponents ID är uppbyggt av flera delar, ett process ID, ett sub id o.s.v. så passar det utmärkt att låta varje ID indexera varsin vektor i multimatrisen. Då ett objekts ID består av fyra olika delar ska dimensionen vara av storleken fyra. Allokeringen kommer att ske statiskt.

Char* multimatris[procid][subid][msgid][mobjid];

7.6.2.1 Format på textsträngar i minnesblocket

Början på en textsträng pekas ut av en adress som finns i multimatrisen. Slutet måste dock markeras på något sätt. Det görs enklast med ett nulltecken, ’\0’. En hjälptext ska även kunna ändras och sparas tillbaka på samma plats. Det är inga problem om den nya strängen är kortare än den gamla. Om den är längre så måste den resterande delen sparas någon annanstans och kunna hittas igen. Genom att låta null tecknet följas av en adress, så kan fortsättningen av texten länkas till föregående del. Dvs resten av hjälptexten sparas någon annan stans och adressen till den platsen sparas efter texten som fick plats. När hjälptexten saknar en fortsättning, så låter man den adressen vara en nulladress. Konstruktionen löser även problemet då textblocket tar slut när en text håller på att läggas in. Adressen till den plats där textsträngen fortsätter, pekar på en plats i det nya minnesblocket som allokeras.

Formatet blir då:

- Sträng som får plats: (”hjälptext som är inlagd i bufferten” ’\0’ NULL)
- Sträng som delas : (”första delen” ’\0’ *char) (”nästa del” ’\0’ NULL)

”...” : Är en textsträng.
’\0’ : Är nulltecknet.
*char : Är adressen till den plats där textsträngen fortsätter.
NULL : Nolladress som säger att textsträngen inte fortsätter.

7.6.2.2 Kontrollblockets utseende

Kontrollblocket har till uppgift att hålla reda på information om blocket med hjälptexter samt att peka ut nästa kontrollblock om det

finns något. Den information som måste finnas om textblocket är var det finns, hur stort det är samt hur stor del av blocket som är användt. Strukturen får följande utseende:

```
Struct mem_block{
        Char* textblock;
        Long size;
        Long used;
        Struct mem_block* next;
}
```

textblock : pekar ut var minnesbufferten börjar.
size : anger buffertens storlek.
used : håller reda på hur mycket av bufferten som är använt.
next : pekar ut nästa kontrollblock om det finns något.

7.6.2.3 Textblocket

Textblocket är en minnesbuffert som allokeras som typen char, eftersom det är mest text som ska in. Det underlättar även administrationen av bufferten, eftersom allt mäts i byte vilket är storleken för en char. Det som inte är text i bufferten är referensen eller saknandet av referens till fortsättning av text.

7.6.2.4 Hanteringen av hjälptextsträngarna

Hjälptexterna kopieras in i bufferten via en loop, eftersom det är viktigt att hålla reda på var i bufferten man är samt hur mycket som läggs in i den. När platsen är fylld avslutas textsträngen med ett nulltecken. Fick hela strängen plats så läggs NULL adressen in efter nulltecknet annars så läggs adressen till fortsättningen in. För att kunna lägga in adressen på rätt plats så måste indexet peka på platsen bakom nulltecknet.

När en hjälptext hämtas från bufferten måste indexet som används till att peka ut tecknen i hjälptexten hamna efter nulltecknet som avslutar textsträngen. Det är där som adressen till fortsättningen av hjälptexten ska ligga om det finns en fortsättning. Annars fås en obestämd adress som kan peka vart som helst.

För att kunna gå över från texthantering till adresshantering så behövs en dubbelpekare, adressen till en char pekare. Det gör man med följande operation.

```
Char **end;
```

End = (char **)&plats_i_bufferten[index_efter_nulltecken];

Nu finns det en pekare av adresstyp, som kan användas till att läsa den adress som finns på den platsen. Eller vid skrivning kan en adress sparas på den platsen. Vid skrivning av en sträng som får plats ska en NULL adress alltid tilldelas den platsen.

*end = NULL;

7.6.2.5 Relevanta operationer på databasen

Dessa operationer är de som är avgörande för hur databasen fungerar. Deras pseudokoder kan hittas i bilagorna. Övriga funktioner som inte är avgörande för hur databasen fungerar behövs, men är mera standardoperationer och kan antas vara underförstådda.

*int create_db(char *db_file);*
Bygger upp databasen från specificerad fil.

*int insert_in_database(char *Text, int prid, int subid, int msgid, int mobjid);*
Sätter in en text på rätt plats i databasen.

*int insert_oldplace(mem_block_t *mem, char *place_in_buff, char *text);*
Byter ut befintlig text i databasen mot ny.

*int insert_last(mem_block_t *mem, char *place_in_buffer, char *Text);*
Sätter in ny text i databasen efter alla andra texter.

*void get_helptext(char *hlptext, int procid, int subid, int msgid, int mobjid);*
Hämtar hjälptexten med hjälp av objekts ID och lägger den i refererad sträng.

7.7 Övrigt

7.7.1 Process ID

Process ID 1-22 är upptagna av andra processer i SDF. Standard värde för hjälpprocessen väljs därför till 23.

7.7.2 Inargument

Hjälpprogrammet kan ta två inargument. Om argumentet ”icon” ges så kommer programmet att starta som en ikon. Man kan också ange ett process ID som då ersätter standardvärdet.

8 TESTNING

Den testning som har utförts har endast syftat till att kontrollera funktionen hos hjälptextfunktionen. Eftersom en ny programenhet har skapats bör ett fullständigt test av SDF utföras enligt Ref[3]. Genomförandet av detta test är dock något som kräver kompetens som ligger utanför det här examensarbetet, och lämnas därför som en framtida uppgift.

Under utvecklingen så har de två delarna i hjälpsystemet, databasen och gränssnittet genomgått kontinuerlig testning för att hitta de uppenbara felen.
När de två delarna integrerades med varandra och med resten av SDF så genomfördes ett mer systematiskt test genom att ett antal testfall designades för att se att hjälpsystemet uppfyllde de ställda kraven. Testfallen gicks igenom både när SDF kördes i utvecklingsmiljön och på plats i TARAS testrigg.
Förutom det nyss nämnda testet genomfördes ett ”stresstest” där databasen fylldes med maximalt med text (drygt 40 Mb) varefter ett antal testfall utfördes.

Testspecifikationerna återfinns i bilaga 2, och resultaten i bilaga 3.

9 RESULTAT

Arbetet har resulterat i en egen process till ett redan befintligt system kallat SDF, som är uppbyggt av flera processer. Genom att ha en egen process så har ingrepp i de andra processerna minimerats. Dessutom har ytterligare en buffert för kommunikation tillförts systemet. Detta för att tillåta kommunikation till hjälpfunktionen som annars inte skulle kunna utföra sin uppgift. Det faktum att kommunikation måste användas, gör att hjälpen tar längre tid på sig. Om hjälpen låg inbakad i de andra processerna så kan den reagera snabbare. Detta skulle dock leda till ett komplexare system, samt de ändringar som måste göras blir fler.

Hjälpfunktionen erbjuder en begränsad hjälp eftersom den bara kan visa texter som är kopplade till en komponent. Hjälpfunktionen kan utökas till att kunna visa textavsnitt om ett speciellt ämne. Från dessa ämnen så ska man kunna gå vidare i sin sökning. För att det ska vara möjligt så behöver en metod som uppfyller realtidskraven tas fram.

Att kunna motivera ett grafiskt gränssnits utseende är svårt. Vad är naturligt för alla världens människor? Kan det motiveras på någon psykologisk grund eller finns det någon sorts mall? Det befintliga gränssnittet till hjälpfunktionen är litet och kan motiveras med att eftersom det går snabbt att se vilka knappar som finns, så finns det ingen större anledning till att gruppera dem på något speciellt sätt. Mycket av funktionen som finns kommer från Motif objekten som bygger upp fönstret. Alla funktioner som finns i textfönstret har bestämts genom inställningar som har gjorts på det objektet.

Testningen är en viktig del av programutvecklingen. För att kunna testa funktionen ordentligt så krävs det möjlighet att använda sig av funktionen när hela systemet används. Under arbetet fanns inte denna möjlighet vilket innebär att hjälpfunktionen endast är testad med resten av SDF passiv. Ett fulltständigt systemtest av SDF med hjälpfunktion är något som bör utföras i framtiden.

10 SUMMERING

Arbetet har omfattat flera områden. I och med att det handlar om ett redan befintligt system som ska tillföras en ny funktion så har arbetet delats in flera steg som kan betraktas som separata arbeten i sig själv.

Att få förståelse för hur det redan befintliga systemet fungerar och vilka lösningar som har använts i det har varit en del av arbetet. Det kräver att mycket av dokumentationen som finns gås igenom. Den är tungläst och saknar mycket av den information som skulle vara bra att

få för någon person som aldrig har haft med systemet att göra. Lite pseudokod, som är löstryckt ur något samanhang, ger inte så mycket. Ett kritiskt moment i arbetet har varit att förstå hur setup processen fungerar. Att ha ett flödesschema som beskriver hur en setup går till och hur setup-filen läses hade varit till stor hjälp. I stället så var det enbart kodläsning som gällde.

För att göra framtida förändringar i systemet lättare, så behöver det skapas någon översiktlig dokumentation. Den dokumentationen kan vara flödesscheman över hur processerna jobbar, tillståndsmaskiner osv. Syftet är att skapa en bild över hur allt fungerar, som är lätt att förstå. Tröskeln är nu hög när det gäller att förstå hur systemet fungerar för någon som aldrig har haft något att göra med det.

Hur ska hjälpen se ut? Vilka funktioner ska den ha? Hur ska gränssnittet se ut? Det är frågor som måste besvaras innan något egentligt arbete kan göras. Detta moment har bestått av att själv försöka skapa en bild av hur det ska se ut. Även intervjuer har gjorts för att få en bättre bild av det hela. Ett problem med att analysera intervjuerna är, att skilja vilda idéer som framförs bara för att få säga något från egentliga önskemål som tillgodoser ett behov. Resultatet från den utredningen gav ett gränssnitt samt vilka funktioner som ska kopplas till den.

Nästa del har handlat om att hitta ett sätt att införa hjälpfunktionen i systemet. Vilka delar finns redan som kan användas och vilka delar måste tillföras? Det enklaste sättet att inte hamna i någon konflikt med redan befintlig funktonalitet är att låta hjälpfunktionen vara en egen process. Detta beslut undviker att koppla hjälpfunktionen till övriga processer. Med det så ska framtida underhåll av hjälpfunktionerna underlättas samt att det blir lättare att införa hjälpfunktionen i systemet.

Testningen av hjälpfunktionen har pågått under utvecklingen av koden där olika funktioner har testats separat. När alla delar till hjälpfunktionen var klara sattes de ihop och testades tillsammans. Efter att den testen var godkänd så integrerades hjälpfunktionen med resten av systemet. Funktionen ansågs klar när den hade klarat integreringstestet.

11 REFERENSER

Ref[1] AerotechTelub AB
Programsystembeskrivning för SD-funktionen. GN/SDF:036 Utg. 3

Ref[2] AerotechTelub AB
Operatörsmanual till SD-funktionen i TARAS Testrigg. GN/SDF:167 Utg. 6

Ref[3] AerotechTelub AB
Programfunktionstestspecifikation till SD-funktionen. GN/SDF:010 Utg. 5

Ref[4] Microsoft Corporation.
MSDN online Library http://msdn.microsoft.com/library/

12 FÖRTECKNING ÖVER BILAGOR