Menysystem med grafiska komponenter VCS OS

Conny Tegström

19 februari 2008
Examensarbete i teknisk datavetenskap, 20 poäng
Handledare på CS-UmU: Jürgen Börstler
Handledare på BAE Systems Hägglunds AB: Jimmy Westerlund
Examinator: Per Lindström

UMEÅ UNIVERSITET
INSTITUTIONEN FÖR TEKNISK DATAVETENSKAP
901 87 UMEÅ

SAMMANFATTNING

Rapporten presenterar ett examensarbete på BAE System Hägglunds AB hösten 2007. Syftet med examensarbetet är att utveckla ett menysystem med grafiska komponenter till VCS OS det underliggande operativsystem i Hägglunds vagnar. Verktyget ska förenkla utveckling och konfigurering av användargränssnitt till företagets applikationer. Hägglunds problem idag är att VCS OS funktionalitet för grafiska displayer är primitiv. VCS OS hanterar enbart de primitiva grafiska operationerna rita linjer, cirklar, pixlar, texter etc. Problemet gör att projekten använder en betydande mängd arbete för att skapa användargränssnitt. Och att alla delar av användargränssnittet måste implementeras förhand. Problemet skapar också olikheter mellan de skapade användargränssnitten och att implementationstiden ökar. Andra problem som skapas är underhållsproblem, t.ex. samma fel måste rättas till i likartade versioner och att det introduceras ett flertal nya fel när arbetet utförs manuellt. Efter en diskussion med externa handledaren Jimmy om ytterligare förenklingar av utvecklingen för användargränssnitt, bestämdes en litteratur studie för att undersöka på vilket sätt examensarbetet kan utökas med en grafisk komponenteditor. Examensarbetets uppgift är därför att utveckla ett menysystem med ett bibliotek av grafiska komponenter som enkelt kan användas till att skapa användargränssnitt. Uppgiften löstes genom att sätta upp en planering, välja en arbetsmetod och utföra en studie av det underliggande systemet. Resultatet av arbetet är ett menysystem med ett bibliotek av grafiska komponenter som kan konfigureras med VCS konfigurationsverktyg VCSCfg.

ABSTRACT

This report describes a master thesis performed during the fall of 2007 at BAE System Hägglunds AB. The purpose of the master thesis was to develop a menu system with graphical components for VCS OS, which is the operating system in Hägglunds' vehicles. The utility should help Hägglund´s development as well as configuration of user interfaces for their applications. The issue now is that Hägglund´s operating system VCS OS has very primitive graphics functionality for the displays. It can only execute basic operations as lines, circles, pixels, labels etc. This problem affects the production that obtains a large workload when developing an user interface. Since all parts of the user interface is made manually. This problem resulted in variations among the user interfaces as well as a waste of time. Another problem occurred at maintenance when it resulted in errors in similarly versions. It also introduced new errors because the work was done manually. After a discussion with my instructor, about how to further ease the development of user interfaces a literature study was decided. The study was to investigate the possibility of creating an user interface builder. Consequently the goal of the master thesis is hence to develop a menu system with a toolbox of widgets that will ease the creation of user interfaces. The thesis was solved by planning, deciding a development model and a study of the underlying system. The solution was thus a menu system with a toolbox, that were able to be configured with VCS configuration utility VCSCfg.

FÖRORD

Jag vill tacka personalen på BAE Systems Hägglunds SPA-avdelning för sällskapet och hjälpen. Jag vill också tacka interna handledaren Jürgen Börstler och examinator Per Lindström på Umeå Universitet. Och ett ännu större tack till externa handledaren Jimmy Westerlund och hans kollega Magnus Berglund på BAE Systems Hägglunds, för det intressanta och lärorika arbetet jag fick möjlighet att utföra och för hjälpen jag erhållit. Jag vill även tacka min familj och flickvän för allt stöd under utbildningen. Och slutligen vill jag tacka Daniel Skoglund för sällskapet under examensarbetets utförande och opponenten Freddie Albertsman för kommentarerna av examensarbetet.

INNEHÅLLSFÖRTECKNING

FIGURFÖRTECKNING

Kapitel 1. INTRODUKTION

Rapporten presenterar ett 20p examensarbete på D-nivå vid Umeå universitet. Arbetstiden för examensarbetet skall motsvara 20 veckor. I det här kapitlet presenteras bakgrunden till arbetet, syftet med arbetet, en kortare information om företaget BAE Systems Hägglunds AB och en disposition av kapitlen.

1.1. BAKGRUND

VCS OS är ett underliggande operativsystem i Hägglunds inbyggda mjukvaruplattform VCS (Vehicle Control System). VCS OS primära uppgifter ligger i styrning av fordonens funktions- och säkerhetskritiska funktioner, VCS OS har ändå ett enklare grafiskt MMI (Man Machine Interface). Displayerna har idag en monokrom elektroluminansdisplay med två färger och en upplösning på 320x240 pixlar. I framtiden ser Hägglunds ett behov av att nyttja VCS för funktioner av instrumentpanelstyp, d.v.s. visa hastigheter, temperaturer och liknande. Därför kommer antagligen skärmen att uppdateras till en flerfärgselektroluminansdisplay med 8 färger och högre upplösning, men det är inte bestämt ännu att uppdatera displayerna.

1.2. SYFTE

Syftet med examensarbetet är att utreda möjligheten att skapa ett menysystem med ett bibliotek innehållande grafiska komponenter till VCS OS. Efter utredningen ska strukturen för systemet med ett urval av komponenterna implementeras. Det löser problemet att det krävs en betydande mängd arbete för att skapa användargränssnitt. T.ex. implementeras alla delar av gränssnitten förhand vilket skapar olikheter mellan de skapade användargränssnitten och att tiden försvinner till implementationen. Andra problem är underhållsproblem, t.ex. liknande fel ska rättas till i likartade versioner och att nya fel introduceras vid manuellt arbete. Lösningen skapas därför tid till att testa designkoncept och tänkbara lösningar vilket resulterar i en robustare produkt.

1.3. UNDERLIGGANDE SYSTEM

Det underliggande systemet kommer att beskrivas detaljrikare i kapitel 4, därför hänvisas läsaren för mer information till analysen i det kapitlet. Här presenteras en kortare version av kapitel 4 för att underlätta för läsaren i de kommande kapitlen.

VCS uppgift är att kontrollera och övervaka funktionerna i fordonet. Plattformen har ett par grundläggande arkitekturprinciper: statisk minneshantering, statisk exekvering, monolitisk och äkta skrivskydd. Begreppen förklaras detaljrikare i kapitel 4. VCS är också uppdelat i ett antal lager: ett applikationslager, operativsystemet VCS OS, hårdvaruabstraktionslagret HAL (Hardware Abstraction Layer) och konfigurationslagret. Av lagren kommer endast VCS OS och konfigurationslagret att studeras för examensarbetet. VCS OS

implementerar funktionerna i alla Hägglunds system som Input/Output, loggning, övervakning och CAN (Controller Area Network) [24] protokollet. VCS OS skapar också standardiserade gränssnitt mot andra plattformar som DIS (Diagnostic Information System) och VIS (Vehicle Information System). VCS OS är uppbyggt av ett flertal moduler som hanterar olika delar av systemet, t.ex. exekveringskärnan Rubus hanterar de statiskt skapade trådarna eller som MMI vilket hanterar grafisk utritning till display. I konfigurationslagret nyttjas ett verktyg VCSCfg (VCS Configuration) som konfigurerar VCS Bas för varje VCS-nod i systemet. VCSCfg konfigurerar också vilka moduler och funktioner som ska användas och hur. Konfigurationsverktyget är ett visuellt verktyg med ett grafiskt användargränssnitt vilket ger användaren ett enkelt sätt att konfigurera VCS-noder.

1.4. BAE SYSTEMS HÄGGLUNDS AB

BAE Systems är ett globalt företag som tillverkar avancerat materiel till en internationell marknad. Företaget är det fjärde största inom försvarsindustrin med 86000 anställda och omsätter 28 miljarder dollar. Svenska verksamheten innefattar kända varumärken som Bofors, CATS och Hägglunds.

BAE Systems Hägglunds är ett svenskt företag som ägs av BAE Systems. Företaget ligger i Örnsköldsvik, har 1211 anställda (år 2007) och omsätter 306 miljoner dollar. Hägglunds utvecklar, tillverkar och underhåller militära fordon inom följande områden [1]:

- Infanteri krigsföringsfordon (CV90).
- Armerade terrängfordon (Bv206S/BvS10).
- Mellanvikts hjul och bandvagnsfordon (SEP).
- Kanontornsystem.

1.5. RAPPORTDISPOSITION

Kapitel 2 - PROBLEMBESKRIVNING
Kapitlet presenterar detlajerat problemet som ska lösas med en beskrivning av problemet, målet och tillvägagångssättet.

Kapitel 3 - GRAFISKA KOMPONENTEDITORER
Kapitlet presenterar examensarbetets fördjupningsuppgift om grafiska komponenteditorer.

Kapitel 4 - ANALYS
Kapitlet presenterar analysen av det underliggande systemet med operativsystemet VCS OS, konfigurationsverktyget VCSCfg och VCS programmeringsspråk.

Kapitel 5 - SYSTEMETS ARKITEKTUR
Kapitlet presenterar lösningens arkitektur. Systemets delar är: menysystemet, grafikmotorn, knapphanteringen, hanteringen av de grafiska komponenterna och konfigurationen.

Kapitel 6 - SYSTEMTEST
Kapitlet presenterar ett konfigurationstest av menysystemet.

Kapitel 7 - SLUTSATSER
Kapitlet presenterar utvecklarens slutsatser av examensarbetet.

Kapitel 2. PROBLEMBESKRIVNING

Kapitlet presenteras problemframställningen, målen med examensarbetet, tillvägagångssätten och designmetoden för arbetet.

2.1. PROBLEMFRAMSTÄLLNING OCH MÅL

Problemet Hägglunds har idag är att den färdiga funktionaliteten för VCS OS hantering av grafiska displayer är primitiv. VCS OS klarar enbart av att hantera primitiva operationer som linjer, cirklar, pixlar, texter etc. Detta gör att projekten får lägga ner en betydande mängd arbete för att bygga upp hantering av menyer och grafiska objekt. För utvecklarna är det därför intressant med ett färdigt menysystem med tillhörande komponenter som enkelt kan konfigureras. Det Hägglunds vinner av att lösa problemet, är att större del av tiden kan användas till design, konfiguration och testning av användargränssnitt. Och vilket i sin tur leder till en robustare och säkrare produkt. Lösningen skulle också ge större möjligheter till vidareutveckling och utökning av menysystemet med t.ex. fler grafiska komponenter i biblioteket.

Målet är därför att det skapade menysystemet ska gå att konfigurera med hjälp VCS konfigurationsverktyg VCSCfg. Verktyget genererar källkoder som inkluderas vid byggningen av VCS. Och via konfigurationsverktyget ska det vara möjligt att skapa menysidor och grafiska komponenter. Sedan med ett konfigurerat menysystem kan då källkoden genereras till applikationen. Och för hantering av den statiskt genererade källkoden kommer en menyhantering, grafikmotor, knapphantering och grafiska komponenter implementeras.

I framtiden är det intressant att utveckla en komponenteditor. Komponenteditorn ska utnyttja menysystemet och användas för att designa och implementera färdiga applikationer. Examensarbetets fördjupningsuppgift kommer därför att utforska det området. Med komponenteditorer skapas möjligheten till att interaktivt placera ut grafiska komponenter i menysidor och på det sättet skapa färdiga användargränssnitt. Målet är att komponenteditorn ska generera källkoden för användargränssnittet till applikationen och att den genererade källkoden ska fungera med det utvecklade menysystemet.

Delmålen med arbetet:

Steg1: Utveckla ett menysystem.

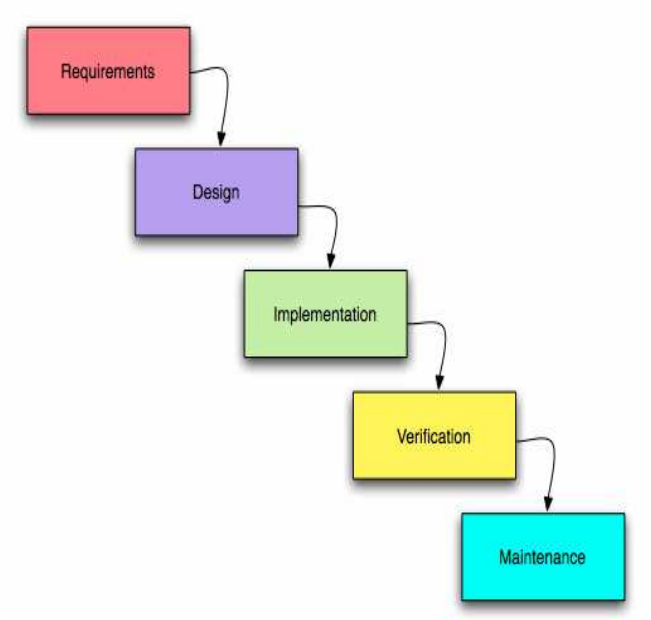
Steg2: Utöka menysystemet med grafiska komponenter.

Steg3: Utöka med möjligheter till att konfigurera menysystemet med hjälp av konfigurationsverktyget VCSCfg.

Steg4: Studiearbete om grafiska komponenteditorer och utveckling av ett designkoncept för en komponenteditor.

2.2. METODER

Arbetsmetoden för arbetet är vattenfallsmodellen, i modellen tas en fas i taget och användaren fortsätter till nästa fas på ett sekventiellt vis. En viss mån av iteration av faserna kommer att utföras för att revidera designen [2].



Figur 1 Vattenfallsmodellen

(Bilden är kopierad med tillåtelse enligt Creative Commons Attribution ShareAlike License v. 2.5 (http://creativecommons.org/licenses/by-sa/2.5/) från http:en.wikipedia.orgwikiWaterfall_model (2007-12-04)).

Arbetsplan

För att följa vattenfallsmodellen kommer först kraven att studeras och specificeras. Sedan i *Requirements* fasen sätts en tidsplan, problemformulering och kravlista för arbetet upp. I *Design* fasen sker en analys av det underliggande systemet, för att veta möjliga strukturer, designkrav och systemets begränsningar. Analysen används sedan för att hitta möjliga designlösningar för menysystemet, grafikmotorn, knapphanteringen och de grafiska komponenterna. I *Implementation* fasen implementeras designlösningarna, för att sedan verifieras och möjligtvis revideras av utvecklaren. Vid problem sker en återgång till *Design* fasen, för att säkerställa en genomarbetad lösning. Efter det verifieras implementationen i *Verification* fasen av externa handledaren, och vid problem sker en återgång till *Design* fasen för att lösa dessa. Efter examensarbetet verifieras arbetet ytterligare och därefter när produkten används måste det underhållas, vilket leder till *Maintenance* fasen.

Tidsplan för arbetet

- Specificering (tidsplan, problemformulering, etc.) (1 vecka)
- Litteraturstudier (strukturer, designkrav, underliggande problem, fördjupningsuppgift, etc.) (3 veckor)
- Design (Menysystem, Grafikmotor, grafisk komponenthantering, knapphantering, etc.) (4 veckor)
- Implementation (6 veckor)
- Verifiering (1 vecka)
- Rapportskrivning (5 veckor)

Beskrivning av situationen innan arbetet påbörjas

För att läsaren ska förstå situationen innan arbetet påbörjas, presenteras den grafiska och interaktiva funktionaliteten att tillgå för arbetet.

Grafisk utritning sker av en klass gfxdisplay. Klassen har ett antal primitiva funktioner för enklare grafik t.ex. `GFX_DrawLine`, `GFX_DrawPolygon`, `GFX_DrawBox` och `GFX_WriteText` för att rita linjer, polygoner, fyrkanter och skriva enklare text. Utritning till skärm sker med en funktion `GFX_Update` och analogt för att rensa skärmen `GFX_ClearDisplay`. Hantering av IO (Input/Output) sker via en klass som används för analoga och digitala signaler, klassen kommer att användas för att ta emot kommandon till menysystemet. Slutligen kommer arbetet att utvecklas under en simuleringsmiljö som finns att tillgå till VCS. Och Resterande delar av systemet kommer att byggas från grunden då inga datatyper som träd och grafer finns att tillgå.

Kapitel 3. GRAFISKA KOMPONENTEDITORER

Kapitlet presenteras studien om grafiska komponenteditorer med en beskrivning av verktyget, fördelarna och problemen med att använda verktyget, jämförelse av tre valda verktyg och en design av en komponenteditor inom examensarbetets område.

3.1. BESKRIVNING AV VERKTYGEN

Ett viktigt bidrag till användargränssnittsforskningen är skapelsen av vad som kommit att kallas *Interface Builders. A*ndra populära termer för verktygen är *Rapid Prototyper, Rapid Applikation Developer, User Interface Builder, Graphical User Interface Builder och User Interface Development Enviroment.* Grafiska komponenteditorer är interaktiva verktyg som tillåter interaktiva grafiska komponenter kallade Gadget och Widget [17] att bli placerad med en mus för att skapa fönster och dialogboxar. Ett viktigt skäl till verktygens framgång är att de använder grafiska medel för att uttrycka grafiska koncept som användargränssnittlayout, genom att man förflyttar aspekterna av användargränssnittets implementation från konventionell kod till ett interaktivt specifikationssystem. På det sättet har aspekter av användargränssnittets implementation blivit tillgängliga för de som inte är konventionella programmerare. Detta har lett till en förenklad utveckling av användargränssnitt, och på det sättet tillåtit experter inom olika områden att skapa prototyper och implementationer, som är välskapta för deras arbetsområde. Verktygen har också tillåtit visuella designers att bli mer involverade i användargränssnittets utseende. Programmerare har även gynnats p.g.a. att implementationstiden kraftigt reducerats när det gäller att skapa användargränssnitt. Förmågan av verktygen kan ses som att de har låg tröskel att använda och att de undviker en brant inlärnings kurva [4]. Verktyget tillhandahåller ett bibliotek med mjukvara och grafiska komponenter som byggnadsblock, men verktygen kräver mycket programmering för att koppla ihop komponenterna till varandra och till delen av systemet som inte är det grafiska användargränssnittet [5, 21].

En komponenteditor är m.a.o ett interaktivt verktyg för att hjälpa utvecklaren att skapa och placera grafiska komponenter i gränssnitt. Verktyget tillgodoser användaren med komponenter tillgängliga i olika verktygslådor som menyer, textetiketter och knappar, för att med komponenterna enkelt skapa användargränssnitt. Med verktyget kan då utvecklaren enkelt uppnå WYSIWYG(What You See Is What You Get) utan att vara en erfaren programmerare [3]. Komponenteditorer är tyvärr begränsade till att endast placera ut de statiska delarna av gränssnittet och kan inte hantera grafiska objekt som förflyttas i användargränssnittet [6].

3.2. FÖRDELARNA MED KOMPONENTEDITORER

I föregående avsnitt beskrevs vissa av fördelarna med komponenteditorer. I det här avsnittet presenteras ännu fler av fördelarna med verktyget.

Fördelarna är klassificerade i två kategorier [9, 25]:

3.2.1. Användargränssnittetskvalitet

Användargränssnittetskvalitet ökas för att användaren kan snabbt och enkelt skapa designmodeller och prototyper. Gränssnittet kan t.o.m. implementeras innan applikationen är färdig. Det blir också lättare att föra in ändringar i gränssnitten upptäckta via kundtester. Kvaliteten kan också ökas för att utvecklaren kan utveckla flera gränssnitt till en och samma applikation. Verktygen tillåter också att en stor del av tiden kan användas till att utöka verktyget istället för gränssnitten. Verktygen löser också underhållsproblem, för att användargränssnitt skapta av samma verktyg får bibehållna konstanta likheter. Med verktyget blir det också lättare att använda olika typer av specialister i skapandet, istället för att endast använda programmerare till utvecklingen. Det kan vara specialister inom områden som grafiker, kognitiva psykologer och MDI (Människa Dator Interaktion) specialister som styr utvecklingen av designen.

3.2.2. Källkodenskvalitet

Kvalitet i det här fallet betyder enklare och billigare källkod att skapa och underhålla för användargränssnittet. Källkodens kvalitets förbättras för att verktyget minskar mängden kod som utvecklaren måste skapa, vilket också resulterar i att tiden reduceras för att skapa användargränssnittet. En annan fördel med verktygen är att om implementations tiden av ett användargränssnitt minskas, kan fler iterationer utföras av en iterativ-design, vilket leder till att högkvalitativ källkod uppnås [4]. Med verktyget kan också specifikationen av användargränssnittet representeras, valideras och evalueras lättare. Verktyget ger en bättre modularisering genom att det separerar användargränssnittet från applikationen, vilket tillåter förändring av användargränssnittet utan påverkan av applikationen. Med separeringen kommer möjligheten med stora ändringar av applikationens interna algoritmer utan att påverka användargränssnittet. Med verktyget krävs lägre kunskap kring programmering och implementering av användargränssnitt, eftersom verktyget döljer komplexiteten av det underliggande systemet. Pålitligheten av användargränssnittet kommer att bli högre för att källkoden kommer från en högre nivå. Det blir också lättare att föra över en applikation från en hårdvara till en annan mjukvarumiljö för att användargränssnittet är endast isolerat till verktyget.

3.2.3. Slutsatser av fördelarna

Slutsatsen är att med verktyget kommer Hägglunds uppnå en högre kvalitet, vad gäller källkod och användargränssnitt. Vilket i sin tur leder till att man löser Hägglunds problem.

3.3. PROBLEMEN MED KOMPONENTEDITORER

I föregående avsnitt diskuterades fördelarna med komponenteditorer, men det finns också problem med verktygen, även om det handlar om felanvändning av verktygen [25].

3.3.1. Problemen

Ett allmänt problem är att det resulterande användargränssnittet kommer att få en större mängd kod jämfört med om det implementerats förhand utan ett verktyg. Det kan ses som en stor nackdel i mindre bärbara system som har ett begränsat lagringsutrymme. Även om verktygen förenklar sättet att skapa användargränssnitt, är det bara ett av problemen vid skapandet, verktygen kan ge för lite vägledning för att skapa bra användargränssnitt. Att använda komponenteditorer betyder ofta att det måste vara en logisk och fysisk separation av komponenter i applikationen som hanterar beräkningar, vilket kan vara svårt att uppnå. Ett annat problem gällande användarvänligheten är att vissa av verktygen kan vara svåra att använda och ha begränsad funktionalitet.

3.3.2. Slutsatser av problemen

Problemen kommer att undvikas då verktyget kommer att skapas inom företaget och därmed kan verktyget skräddarsys efter företagets behov. Ett annat problem vilket måste ses över är att verktyget skapar mer källkod än vad som krävs, eftersom VCS är ett väldigt utrymmesbegränsat system.

3.4. DET VERKTYGET BÖR INNEHÅLLA

Tidigare beskrevs komponenteditorer, med verktygets fördelar och nackdelar. Men om man ska utveckla en komponenteditor är det intressant att veta vad de bör innehålla? För att svara på den frågan har det utförts en forskning kring vad verktygen brukar innehålla. Och för att resonera kring om innehållet är korrekt utnyttjas MDI designkoncepten. MDI designkoncept beskriver hur människor interagerar med datorer och då finns det designprinciper som ska följas [26, 27].

I det här avsnittet diskuteras kring vad grafiska komponenteditorer bör innehålla ur ett MDI-perspektiv vad gäller moduler [8], grafiska komponenter och gränssnittsegenskaper [5].

Designprinciperna kortfattat är:

Simplicity
Utvecklaren ska inte kompromissa med användarvänligheten för bättre funktionalitet.

Support
Utvecklaren ska placera användaren i kontroll och försörja användaren med förebyggande assistans.

Familiarity
Det ska vara byggt på användarens tidigare kunskap.

Obviousness
Det ska vara objektorienterat och dessa objekt ska vara synliga och intuitiva.

Encouragement
Handlingar ska vara förutsebara och återställningsbara.

Satisfaction
Systemet ska ha en känsla av att användaren når framgång och reslutat.

Availability
Alla delar av systemet ska alltid vara tillgänglig.

Safety
Utvecklaren ska hålla användaren ifrån problem.

Versatility
Man ska tillhandahålla alternativa interaktion tekniker.

Personalization
Handlar om att tillåta användaren att göra egna inställningar.

Affinity
Föra objekten till liv via visuella designprinciper som ger klarhet och enkelhet i gränssnittet.

3.4.1. Moduler

De moduler verktyget bör tillhandhålla en användare är en huvudmeny, verktygslåda, hierarki, alternativredigerare och rityta.

Huvudmeny
Modulen ger användaren tillgång till kommandon och möjligheten att administrera projekt och filer. Det bör finnas möjlighet att administrera projekten så en användare kan skapa nya projekt, spara skapade projekt och tillsist öppna sparade projekt för att verktyget ska vara användarvänligt. Utan denna funktionalitet är verktyget hopplöst att använda i en utveckling av ett användargränssnitt. Modulen följer därmed *Safety* och *Support* designprinciperna.

Verktygslåda
Visar tillgängliga grafiska komponenter, det bör vara möjligt att välja en grafisk komponent och placera ut den i ritytan. De olika grafiska komponenterna ska visas som ikoner och bör ha reddigerbara standardinställningar. Anledning till detta är att en användare ska ha enkel tillgång till de grafiska komponenterna, vid utvecklingen av ett användargränssnitt. Utan en verktygslåda för representering av de grafiska komponenterna kommer tiden att lära sig verktyget, för en

användare att öka. Modulen utnyttjar *Obviousness*, *Availability* och *Familiarity* designprinciperna.

Hierarki
Visar alla grafiska komponenter i ritytan med en separat träd vy. Det bör vara möjligt att expandera noderna i trädet och kunna redigera strukturen av användargränssnittet och kunna välja en av de grafiska komponenterna. Att använda sig av en hierarkimodul för att representera strukturen är ett bra sätt för användaren att få en uppfattning över det skapade systemet, annars är det nästan omöjligt att visualisera det skapade användargränssnittets struktur. Det följer av en visuell designprincip som heter *Visual hierarchy,* andra principer som följs är *Availability*, *Simplicity* och *Satisfaction.*

Alternativredigerare
Visar alla valalternativ för en grafisk komponent. Användaren bör bli presenterad med de möjliga alternativen och de alternativ som inte är möjliga bör maskeras. För att enkelt kunna redigera komponenter bör en användare ha tillgång till en alternativredigerare. Det följer av designprincipen *Support.*

Rityta
Där användargränssnittet skapas, bör visa de grafiska komponenterna som de är under exekvering. Storlek och position bör kunna ändras interaktivt och det bör vara möjligt att kunna välja och redigera flera grafiska komponenter samtidigt. Att ha ett interaktivt utvecklingsverktyg för användargränssnitt där man inte har en rityta för att placera ut och förflytta de grafiska komponenterna, är logiskt omöjligt. Det bryter då mot *Availability*. Annars följer modulen *Versatility* designprincipen via modulensinteraktions teknik.

3.4.2. Grafiska komponenter

De grafiska komponenter som en komponenteditor behöver varierar beroende på vilken miljö verktyget utvecklar användargränssnittet. Kommer t.ex. det resulterande användargränssnittet att användas i en stationär pc eller i en mobiltelefon? På vilket sätt kommer användaren att interagera med gränssnittet? Det är viktiga frågor att ställa sig när man ska välja vilka grafiska komponenterna som stöds. Andra frågor som bör funderas på vid val av grafiska komponenter är:

Finns det tillgång till en mus?
Använder systemet endast knappar?
Har apparaten en touchscreen?
Finns det ett tangentbord?
Är det ett informationssystem som endast ska visa information?

Nedan beskrivs de vanligaste grafiska komponenterna:

Windows and dialog-boxes
Fönstren och dialogboxarna, de finns i alla WIMP (window, icon, menu, pointing device) system. Om man utvecklar ett system till ett vanligt WIMP-system är fönster och dialogboxar ett måste för användarvänligheten. Dialogboxarna används för att få interaktionsrespons av en användare. WIMP gränssnitten har regerat länge och än fast det diskuteras om andra mer interaktiva system kommer de att finnas kvar länge till för stationära datorer. Komponenten följer *Familiarity* designprincipen. Andra designprinciper komponenten har är *Obviousness* och *Affinity* [28].

Pull-down or Pop-up menus
Rullgardinsmenyer är en komponent för att representera olika menyalternativ i ett användargränssnitt. Den ökar användarvänligheten för fönsterhantering i WIMP-system och därför bör systemen ha tillgång till rullgardinsmenyer som kan öppnas och stängas.

Buttons
De vanliga knapparna som hittas i alla WIMP-system och de är också användbara i mindre system som mobiltelefoner, pda etc. för att representera telefonknappar. Knappar är ett intuitivt sätt att använda för att slå av eller på olika attribut i ett användargränssnitt.

Radio buttons and switches
Radioknappar används för att kunna välja ett enda alternativ t.ex. i en Internetundersökning när en användare endast ska kunna välja mellan alternativen kvinna eller man.

Scroll bars (horizontal and vertical)
Det är en komponent som används för att kunna välja vilken del av ett dokument som ska vara synligt. Dessa komponenter är väldigt intuitiva att använda och höjer användarvänligheten.

Data-entry field
Det är en grafisk komponent där man skriver in text, dessa utnyttjas ofta vid Internetundersökningar. De är viktiga för att kunna ta emot indata från en användare.

Field labels
Det är en enkel textetikett som oftast används för att förklara en del av användargränssnittet. Dessa är bra att använda för att beskriva olika delar av systemet. Om man t.ex. har en mätare som representerar temperatur kan det vara bra med textetikett som beskriver att det är temperatur.

Compound objects for lists, tables, trees, calendars
Verktygen bör ha grafiska komponenter för att representera listor, tabeller, träd och kalendrar, för att enkelt representera komplexa datastrukturer.

Boxes and separator lines
Enklare typ av komponent används för att strukturera utseendet av användargränssnittet. Används för det visuella intrycket av ett gränssnitt t.ex. för att användaren ska få en uppfattning om samhörighet grupperas delar av gränssnittet ihop. Vilket följs av *Affinity* designprincipen.

Sliders, gauges and meters
Dessa används för att kunna specificera värden och presentera värden. *Sliders* används för att kunna ställa in värden inom ett visst intervall och mätare för att representera värden inom intervall.

User created widgets
Det bör finnas möjlighet för användaren att själv skapa egna grafiska komponenter. Möjligheten är viktig så att verktyget kan utökas om det saknar komponenter, vilket följer *Personalization* designprincipen.

3.4.3. Slutsatser av vad gäller innehåll
Verktyget för examensarbetet behöver alla nämnda moduler och en extra för att skapa möjligheten att använda samma komponent i flera menyer. Vad gäller grafiska komponenter har de flesta implementerats. De komponenter som inte har implementerats är utanför examensarbetets, för att VCS endast använder knappar för interaktion från användaren. Ett intressant ämne är också att utforska möjligheten att använda användarskapade komponenter men det är för tillfället utanför examensarbetets ramar.

3.5. KRITERIER FÖR EVALUERING
Tidigare diskuterades komponenteditorers fördelar, problem och innehåll. Men för att evaluera om ett verktyg är bra eller dåligt och uppfyller alla fördelar, behövs en del kriterier. Kriteriernas relevans med examensarbetet är en viktig del i det här avsnittet och kommer att diskuteras efteråt. Här nedan följer ett antal kriterier med en kortare beskrivning som utnyttjas för att bestämma om ett verktyg är användbart i en tänkbar miljö. Med kriterierna skapas en uppfattning om verktyget uppfyller fördelarna [9]:

Djupet
Hur mycket av användargränssnittet som verktyget stöder? Hjälper det t.ex. med dialogboxar men inte med att skapa interaktiv grafik.

Bredden
Hur många olika användargränssnitts typer som stöds? Verktyget kanske inte stöder alla grafiska komponenter i det underliggande biblioteket och. kan nya komponenter isåfall läggas till?

Portabilitet
När det gäller portabilitet får evalueraren fråga sig om det resulterande användargränssnittet kan användas på olika plattformar som *Windows* och *Macintosh*.?

Verktygets enkelhet
Hur svårt är verktyget som används? Krävs det t.ex. en hög utbildad programmerare.

Effektiviteten för en designer
Hur lång tid tar det för en designer att skapa ett användargränssnitt med verktyget? Det är ofta relaterat till kvaliteten av verktyget.

Kvaliteten av det resulterande användargränssnittet
Genererar verktyget hög kvalitativa användargränssnitt? Hjälper verktyget designern att evaluera och förbättra kvaliteten? Många verktyg låter användaren att skapa för många typer av användargränssnitt, så de säkerställer inte förbättrad kvalitet.

Prestandan av det resulterande användargränssnittet
Hur snabbt är det resulterande användargränssnittet? Vissa verktyg läser in konfigurationen under exekvering vilket kan göra att det kan bli långsamt på vissa plattformar. En annan sak är att vissa verktyg kräver flera typer av stora bibliotek att vara i minnet under exekvering.

Priset
Vissa verktyg är gratis för forskningsorganisationer men för kommersiella verktyg kan det kosta från $200 till $50 000 beroende på deras funktionalitet.

Robustheten och Supporten
I en studie klagade användare av flera kommersiella verktyg på buggar i verktygens officiella releaser. Därför är *robustheten* viktig att kolla p.g.a. att verktygen oftast är svåra att använda och kräver träning. Därför är det också viktigt med support av utvecklaren.

Slutsatser av evalueringskriterierna
Intressanta kriterier för att evaluera verktyget i examensarbetet är t.ex. *Verktygets enkelhet*, då kan dessa frågor ställas: Hur enkelt blir det att skapa ett användargränssnitt? Kan Hägglunds designavdelning göra designmodeller och prototyper för framtida gränssnitt?. Andra viktiga kriterier är *Effektiviteten för en designer* och *Kvaliteten av det resulterande användargränssnittet*, för att: hur lång tid tar det att skapa ett gränssnitt? Hur blir kvaliteten av det resulterande gränssnittet? Kommer det uppnås en högre kvalitet än tidigare? Kan gränssnitten utvecklas effektivare än tidigare?. Två andra kriterier av intresse är *Bredden* och *Djupet*. Där ifrågasätts ifall verktyget stödjer alla de implementerade komponenterna i menysystemets bibliotek?. Tillsist är kriteriet *Prestandan* intressant för VCS är ett hårt tidsstyrt system, då undrar man ifall gränssnitten uppfyller VCS hårda tidskrav?.

Kriterier av mindre intresse är *Portabilitet* för att verktyget kommer att skapas inom Hägglunds, och det kommer aldrig att flyttas till en annan miljö. *Robustheten* och *Support* blir av samma anledning onödigt när

verktyget är skapat inom företaget. *Priset* kan vara viktigt för Hägglunds men det är isåfall den interna kostnaden att skapa verktyget som är intressant. Nästa avsnitt kommer nu att handla om en jämförelse av komponenteditorer.

3.6. JÄMFÖRELSE

Efter presentationen av evalueringskriterierna och innehållet av editorerna bör det utföras en jämförelse. Det här avsnittet beskriver därför en jämförelse av olika grafiska komponenteditorer. Jämförelsen utförs för att få en uppfattning om hur verktygen fungerar och vad som är viktigt att tänka på, när man ska skapa en komponenteditor. För jämförelsen bestämdes ett antal urvalskriterier och jämförelsekriterier för att välja editorer inom ett avgränsat område nära examensarbetet och för att utforska de intressantaste delarna.

3.6.1. Urvalskriterier

Intressant med examensarbetets jämförelse, är vad verktygen tillhandahåller för en utvecklare av ett användargränssnitt. En viktig frågeställning är: hur svårt är det att skapa ett användargränssnitt med verktyget? Andra saker av intresse är vad de erbjuder till en användare och hur de valt att presentera funktionaliteten, för att det ska bli enkelt och intuitivt. De komponenteditorer som valts i jämförelsen är de som endast hanterar interaktiv placering av grafiska komponenter och kan generera konfigurationskod av ett användargränssnitt. Anledningen till det är att examensarbetets verktyg endast ska interaktivt placera komponenterna och ge möjlighet till att konfigurera menysidornas komponenter. Och sedan med ett färdigt gränssnitt ska verktyget generera källkoden till VCS.

3.6.2. Kriterier för jämförelse

Valda kriterier för jämförelsen av komponenteditorer är:

Vilka moduler är de uppbyggda av

För att bestämma modulerna som ska användas i examensarbetets verktyg, har komponenteditorerna t.ex. en verktygslåda, ett fönster för konfiguration av grafiska komponenter etc. Ett annat intressant ämne är att kolla hur editorerna valt att presentera modulerna. Och om ett verktyg saknar en modul är det intressesant hur verktyget ersatt modulen.

Tillhandhållna grafiska komponenter

För att det knyter ihop med det utvecklade menysystemet och kan ge idéer till utökning. Men även för att kolla vilka komponenter som erbjuds och om det finns avancerade komponenter. En annan intressant sak är om verktygen tillhandahåller användarskapade komponenter.

Valmöjligheter

Intressant här är vilka valmöjligheter komponenteditorerna ger användaren och omnågot verktyg har speciella ”features”. Det är intressesant för att få funktionalitets ideér till det planerade verktyget.

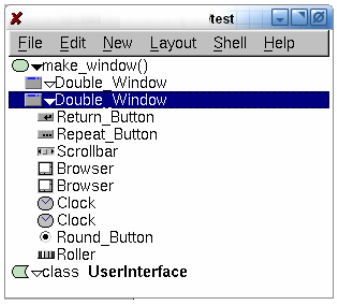
Gränssnittsegenskaper
Det är intressant vilka egenskaper de utvecklade användargränssnitten får.

3.6.3. FLUID (Fast Light User Interface Designer)

Här är en beskrivning varför *FLUID* valdes. *FLUID* är en del av FLTK (Fast Light ToolKit) och uttalas ”fulltick” det är ett *C++ GUI* toolkit för *UNIX/Linux*, *Microsoft Windows* och *MacOS X*. Den ger stöd för *3D* grafik via *OpenGL* och har en inbyggd *GLUT* emulering. Till *FLTK* finns ett grafiskt användargränssnitts verktyg *FLUID* som producerar FLTK källkod i *C++* som kommer att beskrivas och jämföras i det här avsnittet. *FLUID* valdes för att det är ett väldigt avskalat verktyg som främst riktar in sig på enkelheten att placera ut grafiska komponenter interaktivt och sedan utifrån det genererar Källkod i *C++*. Andra personliga anledningar till att verktyget valdes är att verktyget användes i utbildningen för att skapa gränssnitt [10, 11].

Moduler
FLUID består av fyra olika moduler, ett huvudfönster *FLUID Window*, en verktygslåda *FLUID Widget Bin*, en alternativredigerare *FLUID Widget Attribute panel* och för att skapa klasser och funktioner *FLUID Function/Method Dialog.*

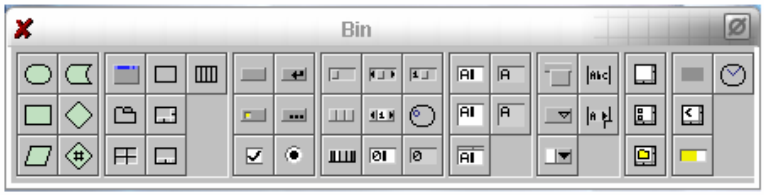
FLUID Window
Huvudfönstret med hierarkin där användargränssnitten skapas. I hierarkin finns de skapade fönstren med komponenternas inbördes placering. Huvudfönstret har fem menyer *File*, *Edit*, *New*, *Layout*, *Shell* och *Help*. *File*-menyn har funktionerna för att skapa, spara och öppna användargränssnitt. *File*-menyn har också ett *Write Code* menyval för att producera en source .cxx och en header .h fil i C++ för det skapade gränssnittet. I header filen finns biblioteket från *FLTK* som behövs för det skapade gränssnittet, och i source filen finns källkoden för gränssnittet, med de grafiska komponenternas placering och konfigurering. I *Edit* menyn finns klipp och klistra funktionaliteten och möjligheten till att öppna och konfigurera de komponenterna. I *New* menyn finns möjligheten att skapa alla komponenter i verktygslådan som grupper, knappar och menyer. I *Layout*-menyn finns en del placerings möjligheter och storleksalternativ för komponenterna. Slutligen i *Shell* kan kommandon exekveras och det används vanligt vis för att göra make skript, för att kompilera den genererade *FLTK*-källkoden [12].



Figur 2 FLUID Window

FLUID Widget Bin

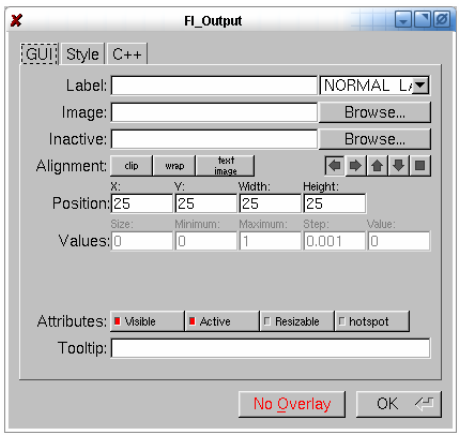
Verktyglådan, den har åtta typer av komponenter *code*, *group*, *buttons*, *valuators*, *text*, *menus*, *browsers* och *others*. *Code* kan skapa kodstycken för funktioner, deklarationer och klasser etc. Till *group* hör fönster och grupper, etc. Till *buttons* finns sex varianter av knappar och till *valuators* finns *sliders*, *scrollbars* etc. Det finns fem varianter av komponenter som tillhör texttypen och fem varianter av menus typen. Till *browsers* hör textfälten som kan läggas till som hjälp för användaren. Tillsist i *others* finns flera olika typer t.ex. en analog klocka.



Figur 3 FLUID Widget Bin

FLUID Widget Attribute Panel

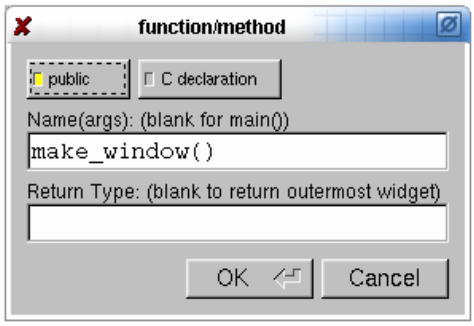
Alternativredigeraren, den har tre menyalternativ *GUI*, *Style* och *C++*. I *GUI* delen kan man ställa in attribut för varje komponent som placering och storlek i användargränssnittet och ifall den ska ha en textetikett eller bild. Attributen i *GUI* menyn är i huvudsak för att ställa in utseendet på komponenten och vilka som finns kan variera beroende på komponent. *Style-menyn* används för att ställa in text färger och fonter men även ramar av olika varianter för komponenten. I *C++* delen skrivs källkoden för vad som ska hända när en komponent aktiveras och för att bestämma vilka typer av händelser den ska reagera på. Det kan t.ex. vara om en knapp ska reagera på när den blir ner tryckt eller släppt.



Figur 4 FLUID Widget Attribute Panel

FLUID Funtion/Method Dialog

Modulen används för att skapa klasser, funktioner och variabeldefinitioner som inte syns i användargränssnittet. Metoderna kan sättas till att vara publika eller privata och det finns en möjlighet att välja metodens returdatatyp.



Figur 5 FLUID Function/Method Dialog

Slutsatser FLUID

Verktyget erbjuder det mesta vad gäller grafiska komponenter och valmöjligheter till en användare. Det tog inte lång tid att skapa ett enklare användargränssnitt och generera koden för gränssnittet. Det var inga svårigheter att skapa och placera grafiska komponenter från verktygslådan. Verktyget innehöll en variant av alla fem moduler som nämndes tidigare ritytan, hierarkin, alternativredigeraren, huvudmenyn och verktygslådan. Verktyget klarade också att skapa egna funktioner och metoder vilket låg utanför examensarbetets verktyg.

3.6.4. wxGlade

Här är en beskrivning varför *wxGlade* valdes. *WxGlade* är skrivet i *Phyton* och använder *wxPhyton* verktygslådan. Verktyget gör det enklare att skapa *wxWidget/wxPhyton* användargränssnitt. Och för tillfället kan verktyget generera källkoden i *Phyton*, *C++*, *Perl* och *XRC* (wxWidget XML resources) kod. *WxGlade* försöker efterlikna *Glade* som är en känd *GTK+/GNOME* grafisk användargränssnittseditor. Verktyget är tänkt att vara enkelt och hanterar därför endast skapandet och genereringen av användargränssnittet. Verktyget skapar därför endast koden för utritning och utelämnar implementeringen av komponenterna. Verktyget distribueras under *GNU* (General Public License) och är en gratis mjukvara. Verktyget valdes på grund av det som skrivs på deras sida om att verktyget inte är och kommer aldrig att bli en *IDE* (Integrated Development Environment). Utan ska endast vara ett enkelt verktyg för att sätta upp användargränssnitt. Andra anledningar till att verktyget valdes, är för att *wxGlade* är utvecklat i *Phyton* vilket varit ett av programmeringsspråken under examensarbetet [13,14].

Moduler

Verktyget består av fyra moduler: hierarkimodulen *wxGlade Tree*, alternativredigeraren *Properties*, den har också en verktygslåda och upp till flera fönster med ritytor beroende på antalet användaren skapar.

wxGlade

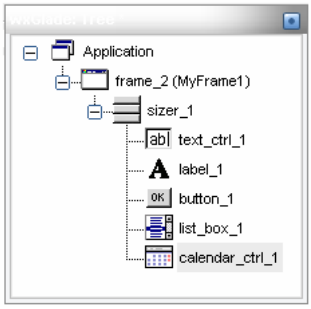
Verktygets huvudfönster och verktygslåda, modulen har tre menyer *File*, *View* och *Help*. I *File* finns möjligheten att spara, skapa nya och öppna projekt men också möjligheten att generera koden för det skapade gränssnittet. I *View* hittas t.ex. saker som att öppna hierarkifönstret och

attributredigeraren. I *View* finns också möjligheten att sätta egenskaper för verktyget som automatisk sparning och andra funktioner. Komponenterna som erbjuds är fönster, knappar, textetiketter, kalendrar, träd, listor etc. Verktyget ger också möjligheten till att skapa egna grafiska komponenter. För att placera ut komponenter med verktyget inuti ett fönster måste man använda komponenten *sizer*, den skapar fält där komponenterna kan placeras, m.a.o verktyget ger inte användaren möjligheten att interaktivt förflytta komponenterna inuti ritytorna.

Figur 6 wxGlade

wxGlade Tree

Hierarkimodulen där sätter användaren placeringen av komponenterna och bestämmer vilken komponent som ska redigeras. I bilden nedan kan man se en *sizer_1* med fem komponenter. Det är samma *sizer* komponent som nämndes i ett tidigare avsnitt för att skapa fält där komponenter kan placeras.

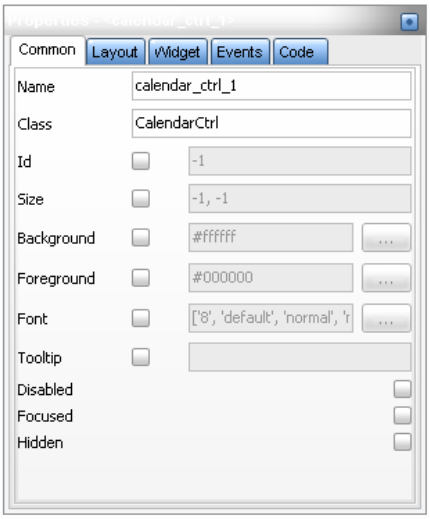


Figur 7 wxGlade Tree

Properties

Alternativredigeraren, där sätts attributen för komponenterna, attributen kan variera beroende på vilken komponent som är aktiverad i hierarkifönstret. Är det t.ex. *Application* som alla projekt har kan användaren bestämma till vilket programmeringsspråk källkoden ska genereras. Men om det är en grafisk komponent finns upp till fem menyalternativ beroende på komponent. Menyalternativen är *Common*, *Layout*, *Widget*, *Event* och *Code*. I *Common*-menyn kan användaren sätta

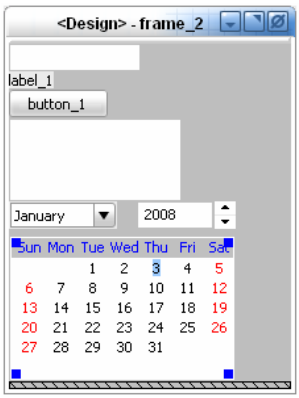
storlek, bakgrundsfärger, fonter etc. I *Layout*–menyn kan placering och kantlinjer sättas. I *Widget*–menyn sätts saker som är specifik för den komponenten. I *Events*–menyn kan man sätta vilka händelser komponenten ska reagera på och tillsist i *Code*-menyn kan extra kod skrivas för komponenten.



Figur 8 wxGlade Properties

wxGlade Rityta

Ritytan som inte ger möjlighet till att interaktivt placera ut de grafiska komponenterna, utan användaren måste skapa en sizer som nämndes tidigare. Med sizern skapas ett antal olika fält där komponenterna kan placeras. Verktyget ger dock möjligheten till en förhandsvisning av användargränssnittet, där man kan se och testa hur användargränssnittet fungerar.



Figur 9 wxGlade Rityta

Slutsatser wxGlade

Verktyget är väldigt lätt att lära sig och har ett bra utbud av grafiska komponenter, även en del avancerade som kalendern. Verktyget hade även möjligheten att skapa egna grafiska komponenter. Nackdelen med verktyget var att man inte kunde positionera de grafiska komponenterna men det vägs upp av verktygets enkelhet. Verktyget bestod av liknande

moduler som *FLUID*. Verktyget har också stora valmöjligheter med tanke på att den kan generera kod till fem programmeringsspråk och att verktyget har möjligheten att skapa egna komponenter. Det finns också stora valmöjligheter för varje komponent t.ex. hur de ska hantera händelser och att användaren kan skriva egen kod för varje komponent.

3.6.5. Glade

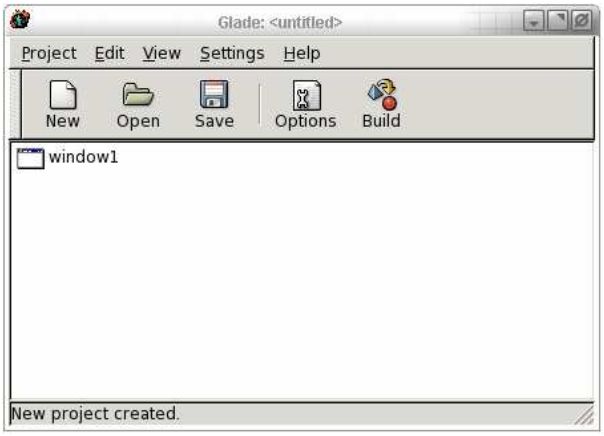
Här kommer en kort beskrivning av Glade och varför verktyget valdes. Verktyget används för att skapa användargränssnitt för *GTK+* verktygslådan och *GNOME* skrivbords miljön som är publicerad under *GNU GPL* Licensen. Användargränssnittet som skapas med *Glade* är sparat i *XML* format och genom att använda libglade biblioteket kan dessa applikationer laddas dynamiskt. Genom att använda libglade och *Glade XML* filer kan verktyget användas i ett flertal programmeringsspråk som *C*, *C++*, *Java*, *Perl*, *Python*, *C#*, *Pike*, *Ruby* och *Haskell*. Det är även enkelt att lägga till support för andra programmeringsspråk. Verktyget distribueras under *GNU* och är en gratis mjukvara. Anledningen till att verktyget valdes förutom att det är ett verktyg som enkelt placerar ut grafiska komponenter, är p.g.a att det är en föregångare till wxGlade och för att verktyget stödjer ett flertal programmeringsspråk. Den sista anledningen till att det valdes är att gränssnitten sparas i XML vilket är en intressant funktion som planeras till examensarbetet [15, 16].

Moduler

Verktyget består av fem moduler huvudmenyn *Glade*, hierarkin *Widget Tree*, attributredigeraren *Properties*, Verktygslådan *Pallete* och ett flertal ritytor som namnges av användaren.

Glade

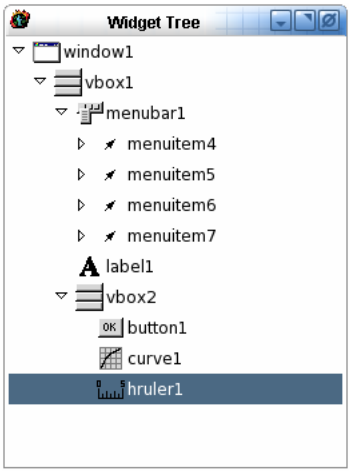
Huvudmenyn, den används för att hantera projekten, här ser användaren också de fönster som skapats. Modulen har fem menyer för att hantera projekten *Project*, *Edit*, *View*, *Setting* och *Help*. I *Project* finns funktionaliteten för att spara, öppna och skapa ett nytt projekt. Det är också här användaren kan generera koden för det skapade gränssnittet. I *Edit* menyn kan man kopiera, klippa ut, klistra in och ta bort de skapade fönstren. I *View* hanteras vilka moduler som är synliga och i *Setting* kan man t.ex. ställa in om gridden ska synas eller inte.



Figur 10 Glade main window

Widget Tree

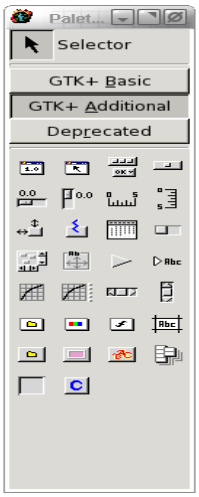
Hierarkimodulen, där ser användaren användargränssnittets uppbyggnad. Modulen är väldigt lik *wxGlade* och *FLUID*:s hierarkimodul. *Glade* använder precis som *wxGlade* en komponent som skapar fält där komponenter får placeras men här heter de *vBox* istället för *spacer*. Att det finns likheter mellan dessa två verktyg är inte konstigt när *wxGlade* försöker efterlikna *Glade*.



Figur 11 Glade Widget Tree

Palette

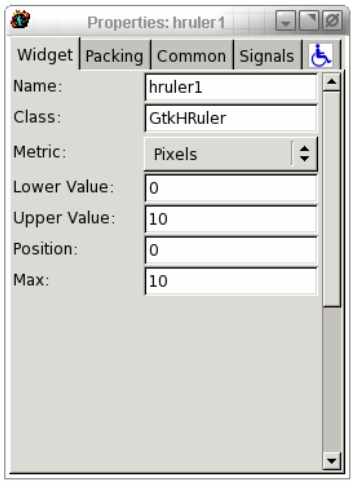
Glade verktygslådan, innehållet varierar beroende på typen av projekt. Om man skapar ett *GNOME*-projekt finns det en *GNOME*-flik med fler komponenter, annars finns *GTK+ Basic*, *GTK+ Additional* och *Deprecated*. I *GTK+* Basic finns komponenterna som fönster, knappar, textetiketter, träd, listor etc. I *GTK+ Additional* finns ett antal avancerade komponenter som kalendrar, grafiska kurvor, pop up fönster mm, i *Deprecated* finns de man ska undvika, för att det är föråldrade och utbytta komponenter.



Figur 12 Glade Palette

Properties

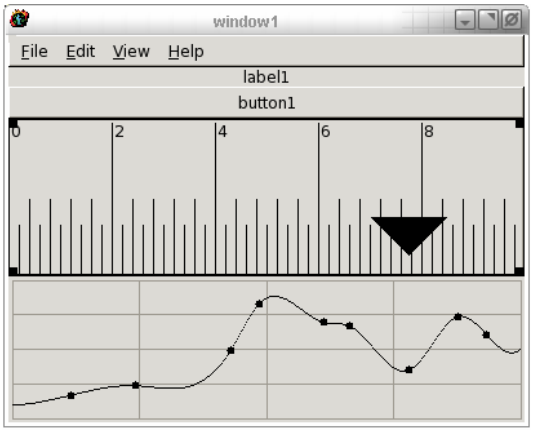
Attributredigeraren, den har fyra menyflikar för att sätta komponentattributen. Menyfliken *Widget* innehåller specifika attribut för komponenterna. I *Packing*–menyfliken sätts antalet celler som komponenten kan placeras i och i vilken cell komponenten ska finnas. *Common*–menyfliken är attributen som alla komponenter har t.ex. storlek, synlighet etc. Tillsist i *Signals*–menyfliken sätts de händelser komponenten ska reagera på och isåfall vad den ska göra vid en händelse.



Figur 13 Glade Properties

Ritytan

Ritytan fungerar liknande *wxGlade* att användaren skapar celler där de sedan placerar ut komponenterna.



Figur 14 Glade Rityta

Slutsatser Glade

Glade har ett stort utbud av grafiska komponenter, verktyget är enkelt att lära sig, och det tar inte lång tid att skapa det första gränssnittet. Verktyget har en egen version av modulerna som diskuterades i kapitlet om innehåll. Verktyget har stora valmöjligheter för en användare då de kan skapa egna grafiska komponenter. Det finns väldigt stora likheter mellan *Glade* och *wxGlade* med samma nackdel att man inte positionerar ut de grafiska komponenterna utan man skapar celler och via cellerna placerar ut komponenterna.

3.6.6. Slutsatser av jämförelsen

Slutsatserna av jämförelsen är att verktygen är väldigt enkla att använda för att skapa användargränssnitt. Verktygen skapar många möjligheter för gränssnitt och de kan vara utformade på rätt olika sätt, men verktygen har ändå liknande funktionalitet vad gäller interaktiv placering än om inte *Glade* och *wxGlade* tillhandahöll interaktiv positionering. Därför kommer examensarbetets verktyg att efterlikna *FLUID* vad gäller placering och positionering av komponenter och Glade när det gäller att spara gränssnittet med *XML*. Alla tre verktyg gav ideér för funktionalitet som bör finnas med i examensarbetets verktyg. Problemet med alla dessa tre är att de används för att skapa gränssnitt i *WIMP*-system. Vilket inte examensarbetets verktyg kommer att göra, för där tillhandahålls endast knappar för interaktion. Därför går flera av komponenterna i jämförelsen bort när det gäller att utöka menysystemets bibliotek. Det är t.ex. komponenter som fönster och dialogboxar som inte kan utnyttjas. Kunskapen från jämförelsen kommer nu att användas när kravlistan ska sättas upp.

3.7. KRAVEN FÖR EN EDITOR TILL VCS BAS

Efter studier kring vad komponenteditorer innehåller och jämförelsen av olika editorer, upptäcktes vad de behöver för funktionalitet, komponenter och moduler. Efter studien skapades därför en kravlista för examensarbetets komponenteditor. Det som har upptäckts är att en användare behöver möjligheten att administrera projekt och generera en konfiguration av ett gränssnitt. Det finns också krav som upptäckts i och

med jämförelsen och innehållskapitlet. Nedan presenteras kraven, det kan dock finnas ytterligare krav för komponenteditorn.

Möjligheten att skapa användargränssnitt
Betyder att en användare ska kunna skapa ett nytt projekt och där kunna utveckla användargränssnitt.

Möjligheten att spara användargränssnitt
Projekt som skapats ska gå att spara för att öppnas vid ett senare tillfälle för vidareutveckling.

Generering av source och headerfiler för VCS OS
Den grafiska komponenteditorn måste klara av att generera konfigurations filer till VCS OS. Den genererade källkoden ska också fungera ihop med det utvecklade menysystemet.

Möjlighet till att skapa menysidor
Användare måste ha möjligheten att skapa nya menysidor. Menysidorna ska kunna sättas till att vara undermeny till en av de andra skapade menysidorna.

Hierarkimodul för menyerna och de grafiska komponenterna
Modul där användaren kan se vilka menyer som är undermenyer och kunna se vilka grafiska komponenter som finns i varje meny.

Konfiguration av menysidorna
Användare ska kunna konfigurera menyernas placering i menyträdet, menyknapparnas funktioner och sätta vilken meny som är huvudmeny (ROOT).

Verktygslåda med grafiska komponenter
Verktygslåda med grafiska komponenter där en användare ska interaktivt kunna välja en komponent och placera ut den i en valfri skapad meny. Ett krav med verktygslådan är komponenterna får endast vara de som implementerats i VCS OS bibliotek.

Referenslista över de skapade grafiska komponenterna
Referenslista på de skapade komponenterna, för att användaren ska kunna återanvända samma komponent i flera menyer.

Interaktivkonfigurering av grafiska komponenter
Det ska vara möjligt för användaren att interaktivt sätta storlek och position på komponenterna. Men det får inte vara möjligt att överlappa komponenterna med varandra.

Parameterbaserad konfigurering av grafiska komponenter
Det ska vara möjligt att sätta storlek och position för komponenterna i textfält för att kunna specificera t.ex. position på pixelnivå. Textfälten ska uppdateras när en grafisk komponent förflyttas interaktivt. Man ska

också kunna konfigurera värden som namn, buffrar mm. Attributen storlek och position för en grafisk komponent ska dock tillhöra menyn.

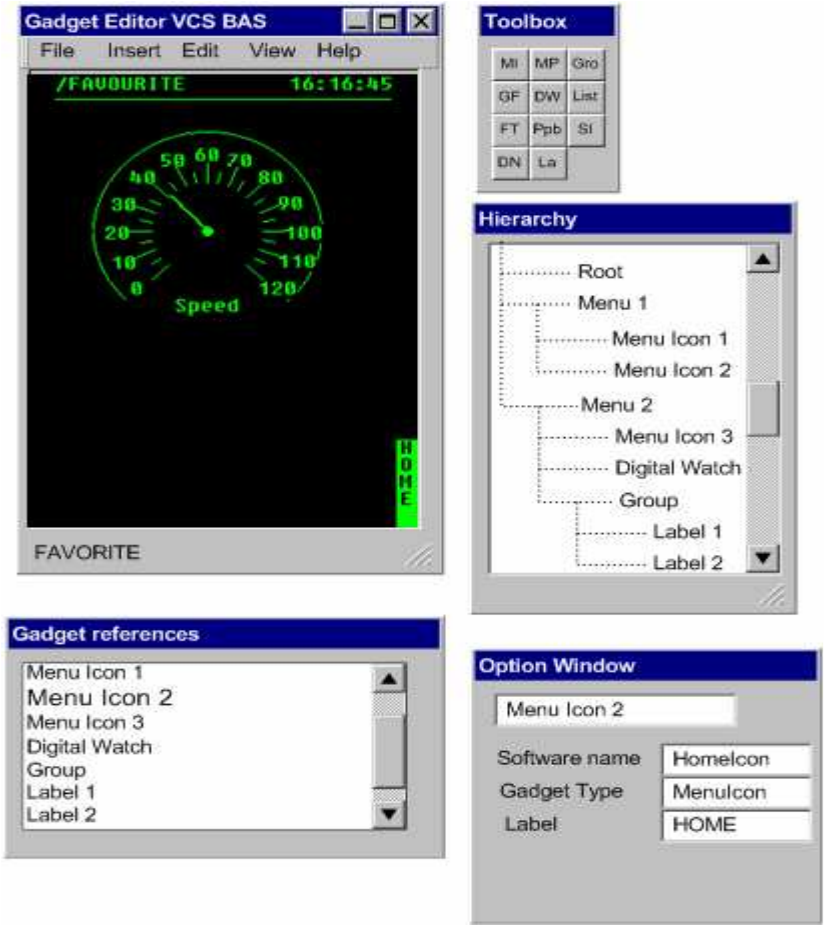
Möjlighet till att växla mellan menysidor
Användare ska kunna växla interaktiv mellan de skapade menysidorna.

Möjlighet att specificera storleken av ritytan
Det är för att en användare ska kunna skapa användargränssnitt för flera typer av displayer. Vilket är bra när Hägglunds planerar en utökning.

Reflektioner kring kraven
Vissa av kraven är relativt logiska och allmänna t.ex. kravet med möjlighet att kunna skapa menysidor. Andra krav är valda efter det underliggande systemet t.ex. kravet med en referenslista över skapade grafiska komponenter. Det kravet är för att det underliggande systemet har möjligheten att placera samma komponent i flera menyer. Som nämndes tidigare är vissa av kraven från jämförelsen och innehållslistan t.ex. kravet med en Verktygslåda med grafiska komponenter. Det garanteras dock inte att alla krav för komponenteditorn finns med. Nästa avsnitt kommer nu att beskriva lösningen på en komponenteditor till VCS där kravlistan utnyttjas för designen.

3.8. GADGET EDITOR VCS BAS

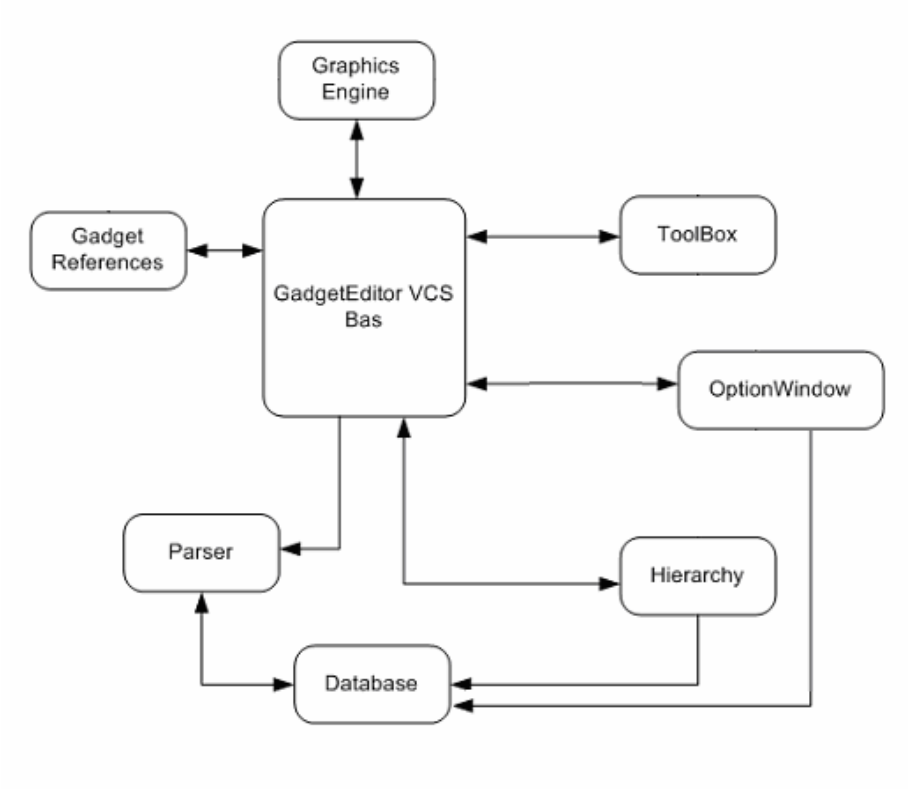
Här presenteras lösningen för en grafisk komponenteditor till VCS Bas. De moduler som systemet är uppbyggt av är: En huvudmeny *Gadget Editor VCS BAS* som visar ritytan, den har en del menyer som *File* där användargränssnitt skapas och sparas. *Insert* där nya menyer och komponenter skapas. I *Edit* ska menyer redigeras och tillsist i *View* ska det gå att välja vilka moduler som är aktiva. Verktyglådan *ToolBox* ska innehålla de grafiska komponenterna från biblioteket, och det ska gå att välja en komponent för att interaktivt placera ut komponenten i ritytan. Hierarkimodulen *Hierarchy* ska visa menyerna med deras grafiska komponenter, där ska användaren välja vilken komponent som ska visas i *Option Window,* för att i *Option Window* sätta komponentens attribut. I Gadget references modulen ska användaren kunna välja en av de skapade komponenterna för att sedan kunna flytta ut samma komponent i flera menyer.



Figur 15 Gadget Editor VCS BAS

3.8.1. Systemets arkitektur

GadgetEditor VCS Bas är huvudmodulen som hanterar kommunikationen mellan modulerna, modulen kontrollerar också när andra moduler är synliga för användaren. När en användare väljer en grafisk komponent antingen i *Toolbox*, *Hierarchy* eller *Gadget References* meddelar de huvudmodulen om den grafiska komponenten som ska skapas eller uppdateras. Huvudmodulen meddelar efter det *Graphics Engine*, *Hierarchy* och *OptionWindow* om den aktiva grafiska komponenten. *OptionWindow* och *Hierarchy* uppdaterar databasen kontinuerligt vid ändringar av det skapade användargränssnittets struktur och inbördes attribut. När huvudmodulen ska generera koden meddelas parsern, som kontaktar databasen för att ta reda på inställningarna för de skapade menyerna och komponenterna.



Figur 16 Gadget Editor VCS Bas systemstruktur

3.8.2. Gadget Editor VCS Bas

Systemets huvudmodul och rityta, hanterar vilka moduler som är synliga för användaren och trafiken mellan modulerna. Huvudmodulen sätter också vilken meny som är aktiv i ritytan. Designbeslutet som är taget för huvudmodulen är att den ska dirigera trafiken mellan modulerna för att få en modulär struktur av systemet.

3.8.3. Database

Databasens huvuduppgift är att lagra det konfigurerade menysystemet, det innefattar konfigurationen för varje grafisk komponent och där med placering, storlek, identitet och ifall de är valbara i en meny. Databasen lagrar också konfigurationen för knapparna i varje meny, vilka undermenyer som finns i varje meny och informationen för komponenterna vilket varierar från komponent till komponent. Ifall användaren vill spara det skapta användargränssnittet ska informationen i databasen utnyttjas till att skapa en enkel XML fil över det konfigurerade systemet.

3.8.4. GraphicsEngine

Grafikmotorns huvuduppgift är utritning och förflyttning av de grafiska komponenterna inuti ritytan. När storlek och position ändras för en komponent, meddelar modulen huvudmodulen som då uppdaterar *OptionWindow* om värdet.

3.8.5. Parser

Huvuduppgiften för parsern är att ta det konfigurerade datat från databasen och generera en source och header fil som kan användas i VCS OS.

3.8.6. Hierarchy

Hierarkimodulen ska visa menyträdet med menyernas komponenter. I hierarkin ska användaren få möjligheten att välja vilken komponent som ska konfigureras. Komponentens attribut ska då visas i *Option Window,* för t.ex. en meny ska användaren se attributen för knapparna, komponenterna och undermenyerna. Om användaren valt en komponent i någon av menyerna ska det gå att förflytta och placera komponenten i ritytan.

3.8.7. Option Window

Hanterar konfigurationen för menyerna och de olika grafiska komponenterna. Användaren ska kunna sätta attributen som storlek och position för en komponent. Modulen meddelar efter det huvudmodulen om det nya värdet så att ritytan kan uppdateras. För de grafiska komponenterna, ska det finnas storlek och position för varje meny komponenten tillhör.

3.8.8. Toolbox

Modulen skapar nya grafiska komponenter som ritas ut i huvudmodulen för den nuvarande aktiva menyn. När huvudmodulen får en uppdatering av en ny komponent ska den läggas till i *Hierarchy* och *OptionWindow*.

3.8.9. Gadget references

Modulen innehåller en lista på alla skapta komponenter, för att en användare ska kunna använda samma komponent i flera menyer.

Kapitel 4. ANALYS

Kapitlet presenterar en analys av det underliggandesystemet. Vilket innefattar allmänt om VCS, lagren och konfigurationen.

4.1. VCS ALLMÄNT

VCS Står för Vehicle Control System och plattformens huvuduppgift är att kontrollera och övervaka funktioner i ett fordon. VCS kontrollerar t.ex.[18]:

- Enklare funktioner som bromsljus.
- Komplexa funktioner som att öppna och stänga rampen, där flera villkor måste noga övervägas innan funktionen utförs.
- Funktioner som inte är synliga för användaren t.ex. att en bränsle pump kollar nivån i den interna bränsle tanken, och när nivån blir för låg pumpas interna tanken automatisk från den externa tanken.

4.2. VCS ARKITEKTURPRINCIPER

VCS har ett par grundläggande arkitekturprinciper:

Statisk minneshantering
VCS förbjuder dynamisk minnesallokering. Allt minne allokeras i kod vid kompilering. Det eliminerar tänkbara problem som minnesfragmentering, att minnet tar slut etc. Då vet Hägglunds efter länkningen att det finns tillräckligt med minne och kan säga exakt hur mycket minne som är ledigt för framtida förbättringar. Hägglunds vet också var i minnet kod och data har placerats vilket underlättar felsökningen.

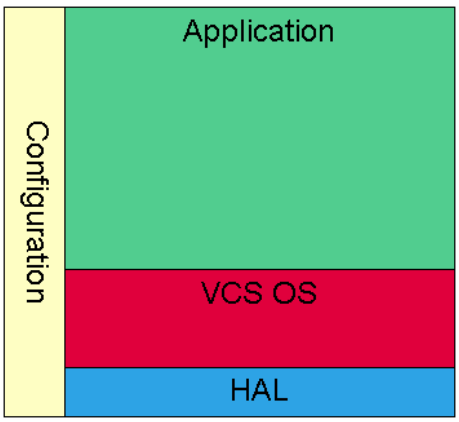
Statisk exekvering
Trådar definieras *offline* och inte vid exekvering. Den stora mängden trådar är hårt tidsstyrda, dvs. Det är bestämt på förhand när de ska exekvera.

Monolitisk
Mjukvaran i en elektronikenhet är monolit, dvs. i den ingår operativsystem, drivrutiner och applikationer. Det finns inga beroenden internt i noden när mjukvaran har blivit laddad.

”Äkta” skrivskydd
Data som inte är tänkt att ändras, t.ex. kod och konstant data, ligger i skrivskyddade minnen (Flashminnen). Det datat kan inte skrivas sönder av misstag under körning och därför är VCS noga med att separera dynamisk och konstant data VCS blandar t.ex. inte ihop dessa i samma struktur.

4.3. VCS LAGREN

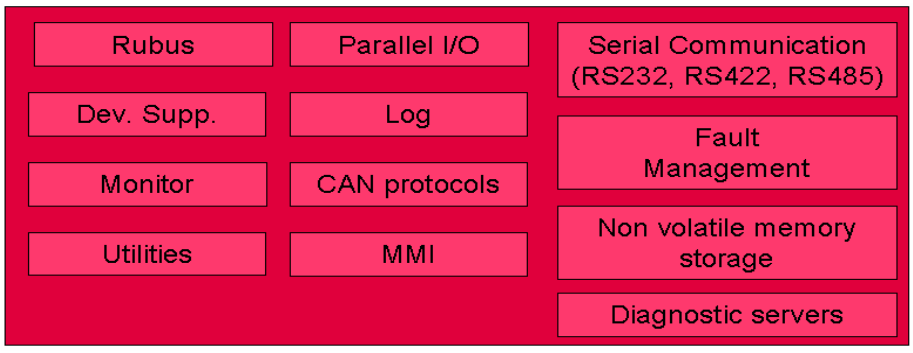
Inuti noderna är mjukvaran uppdelad i lager. Lagren har definierade skyldigheter, gränssnitt och krav. Lagren kan skapas självständigt, bytas ut och återanvändas mellan noderna. I den här rapporten kommer endast operativsystemet VCS OS och konfigurationslagret att presenteras, för det är de intressanta lagren för examensarbetet.



Figur 17 VCS Lagren

4.4. VCS OS

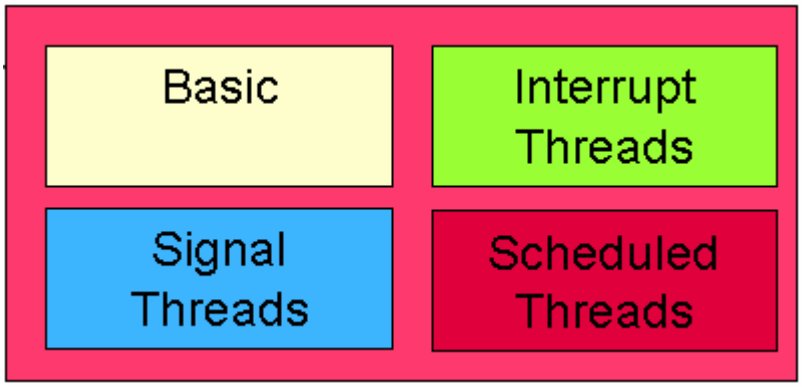
VCS OS implementerar funktionerna som finns i alla Hägglunds system Input/Output, loggning, övervakning och CAN protokoll. Den skapar även standardiserade gränssnitt mot andra plattformar, som DIS (Diagnostic Information System) och VIS (Vehicle Information System). VCS OS är uppbyggt av ett flertal moduler som hanterar olika delar av systemet t.ex. Rubus som är en exekverings kärna som hanterar de statiskt skapade trådarna eller MMI som hanterar grafisk utritning till display.



Figur 18 VCS OS Moduler

4.5. VCS OS RUBUS

Rubus är ett realtidsoperativsystem som är uppdelat i fyra kärnor Basic, Blå, Röd och Grön, Operativsystemet har även tre olika trådtyper Röda, Blå och Gröna [19].



Figur 19 Rubus kärnor

4.6. RUBUS KÄRNOR

Presentation av Rubus kärnor Basic, Blå, Röd och Grön.

Basic

Basic består av generella delar som köer/buffrar och brevlådor. Basic tjänster kan utnyttjas av alla trådar. Basic-köerna innehåller minnesblock av olika datatyper som har en fördefinierad storlek, storleken på köerna är även fördefinierad, och kärnan mäter nuvarande max användning av köerna. En brevlåda kan innehålla godtyckligt antal brev av godtycklig storlek, mottagaren får antingen läsa endast specifika brev eller alla.

Blå (Händelsestyrd)

Blåkärnan består av fem delar signaler, trådar, mutex, semaphores och meddelandeköer. Varje tråd har 16 privata signaler där fyra är reserverade till Rubus, vilket ger 12 lediga signaler för att använda till applikationerna. Signaler kan skickas från alla typer av trådar, vilket tillgodoser möjligheten för röda och gröna trådar att signalera händelser till blåtrådar. Meddelandeköerna fungerar som vanliga köer med skillnaden att inlägg till kön kan vara mindre än max storleken specificerad för meddelandekön. Blåtrådar måste vänta på inlägg till meddelandekön och alla typer av trådar har tillstånd att skicka meddelanden till kön.

Röd (Tidsstyrd)

Den röda kärnan består av två delar en för trådar och en för schemaläggaren. Alla röda trådar är schemalagda till en rödschemaläggare. En nod kan bestå av flera schemaläggare där en är alltid aktiv. Schemaläggarna används typiskt till att implementera olika exekverings tillstånd.

Grön (Interrupt)

Den gröna kärnan består endast av trådar.

4.7. RUBUS TRÅDTYPER

Presentation av Rubus tre trådtyper Röda, Blåa och Gröna.

Röda (Tidsstyrda)
Röda trådar exekveras efter schemaläggning och har en känd exekverings tid. De exekverar efter fördefinierad periodicitet och har en deadline när de måste ha exekverat klart, annars kommer ett deadline-overrun fel att inträffa. En rödtråd får om den behöver ersätta en annan rödtråd.

Blåa (Händelsestyrd)
Blåtrådar väntar på ett event och exekveras på en intern prioritet. De är tillsatta en prioritet mellan 1 och 14 där 14 är den högsta prioriteten. Trådarna är definierade offline, men kan avslutas och aktiveras dynamiskt. Två typer av trådar avslutas alltid blueIdle prioritet 0 och blueKernel prioritet 15.

Gröna (Interrupt)
Gröna trådar exekveras efter avbrottsevent och har en känd exekverings tid. De är tillsatta en viss prioritet. Den röda kärnan är ansedd att vara en gröntråd som tillsätter gröna trådar att ha en lägre prioritet än röda trådar.

4.8. VCS OS MMI

MMI ger stöd för att rita linjer, skriva text och rita grafik via GfxDisplay och TestDisplay. GfxDisplay ger stöd för grafisk funktionalitet som att rita linjer och cirklar [22]. TextDisplay ger stöd för Alphanumeriska text displayer.

4.9. VCS OS PARALLELL IO

IO stöder funktionalitet att konfigurera, läsa och skriva digitala och analoga signaler. Den hanterar filtrering som *rolling average filter* för analoga signaler och hanterar överskrivning både med hjälp av DIS och med konsolmonitorn.

4.10. VCS OS MONITOR

Monitormodulen ger stöd för menyer och kommandon för att kunna examinera variabler och interna tillstånd och för att exekvera kommandon.

4.11. VCS KONFIGURATION

För att konfigurera VCS funktionalitet används ett verktyg som heter VCSCfg (VCS Configuration). Och programmeringsspråket som utnyttjas för att skriva koden för genereringen av källkoden är Phyton [23].

VCSCfg

År 2004 skapades ett utvecklingsprojekt som tog fram VCSCfg. Verktyget konfigurerar VCS Bas för varje VCS-nod och genererar utifrån det källkoder som inkluderas vid byggning. VCSCfg tillåter användaren

att konfigurera vilka moduler och funktioner som ska användas och hur. Verktyget är visuellt med ett grafiskt användargränssnitt som ger användaren ett enkelt sätt att konfigurera VCS noder. VCSCfg kan läsa in och synkronisera med information från fordonsdatabasen SCDR (System Configuration Data Repository) som är en databasapplikation där alla noder och signaler i systemet är listade. Noderna och signalerna från SCDR är importerade till VCSCfg det är t.ex. loggar och CAN signaler.

Anledningen till att skapa ett konfigurations verktyg
VCS är statiskt detta gör att det är svårt att skriva generella moduler när man inte får använda dynamisk minnesallokering. Det går inte att vid exekvering skapa den konfiguration som krävs. VCS har därför separerat funktion och konfiguration till olika källkoder. konfigurationskällkoderna är oftast data och sällan kod. Tidigare handknackades konfigurationskällkoden vilket gav ett antal stora nackdelar:

- Klipp och klistra fel när man ska skapa tabeller.
- Fel i konfigurationen blir väldigt svårt att hitta.
- Man var tvungen att strukturera konfigurationskällkoden så att den var lätt att skriva och inte för att ge en bra struktur.
- Stora mängder kod med syftet att kontrollera att konfigurationen var riktig.

Phyton
Phyton är ett programmeringsspråk som används för att skriva genererings källkoden till VCS. Den genererade källkoden är .c och .h filer som innehåller konfigurationen för olika delar av systemet. Phython är ett modernt programmeringsspråk, skapat av Guido van Rossum 1991 och det är tillgängligt gratis på den officiella hemsidan phyton.org [20].

4.12. VCS PROGRAMMERINGSPRÅK

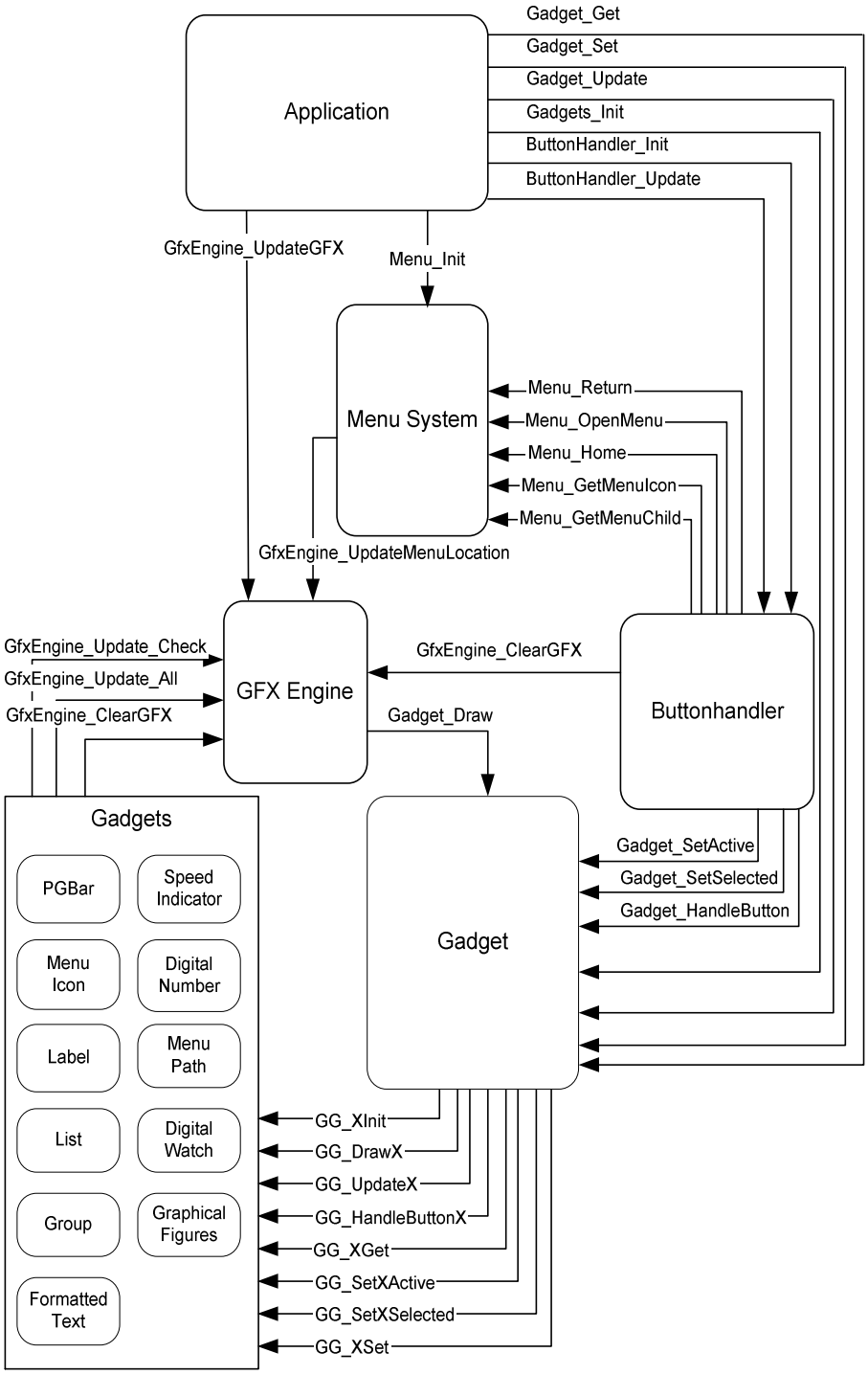
VCS programmeringsspråk är C vilket är ett dåligt val av programmeringsspråk p.g.a. fallgroparna som finns, därför har ett antal regler definierat på hur man ska använda programspråket, för att kunna motivera dess användning i säkerhetsrelaterade system. VCS Bas kodningsregler är baserade på MISRA-C 2004 som består av 141 regler varav 121 är obligatoriska där VCS definierar på vilket sätt reglerna ska följas m.a.o. om man ska vara mindre eller mer restriktiv än MISRA. Kodningsreglerna innehåller också en coding style guide som ska följas för VCS källkoder. Detta eftersom det är viktigare att man snabbt och enkelt kan läsa annans kod, än att snabbt och enkelt kunna skriva den. Kodningsreglerna gäller för alla VCS källkoder, oavsett projekt och de innehåller dessutom regler för hur man ska hantera ospecificerade, odefinierade och implementations definierade beteenden för C enligt ISO 9899/90 mer känd som ANSI-C. Ett par verktyg används även för att kontrollera att MISRA-C följes bland andra Polyspace och Lint.

Kapitel 5. SYSTEMETS ARKITEKTUR

Kapitlet presenterar det utvecklade systemets arkitektur. Systemets arkitektur består av en meny, grafikmotor, grafiska komponenter och knapphantering. I slutet av kapitlet diskuteras menysystemets konfiguration.

5.1. ARKITEKTUR

Systemets färdiga arkitektur kan ses i bilden nedan. Systemet består av fem huvudkomponenter *Application*, *Menu System*, *GFXEngine*, *Buttonhandler* och *Gadget*. Sedan finns det ett bibliotek med 11 st grafiska komponenter t.ex. *label*, *list* och *progress bar*. *Application* är en applikations klass som initierar systemet och uppdaterar *Buttonhandler* klassen. *MenuSystem* är en klass för hantering av menybyten den uppdaterar även *GFXEngine* vid ett menybyte. *Buttonhandler* är en klass som hanterar knapphändelser och beroende på typ av knapphändelse så antingen öppnar och stänger den menyer, eller aktiverar grafiska komponenter för att reagera på knappar. *Gadget* är en föräldraklass till de grafiska komponenterna som finns i systemet och anropar en grafisk komponents metod, beroende på vilken typ av grafisk komponent och metod det var som anropades. *GFXEngine* är en klass som hanterar uppdateringen och utritning av de grafiska komponenterna.



Figur 20 systemets arkitektur

5.2. MENY

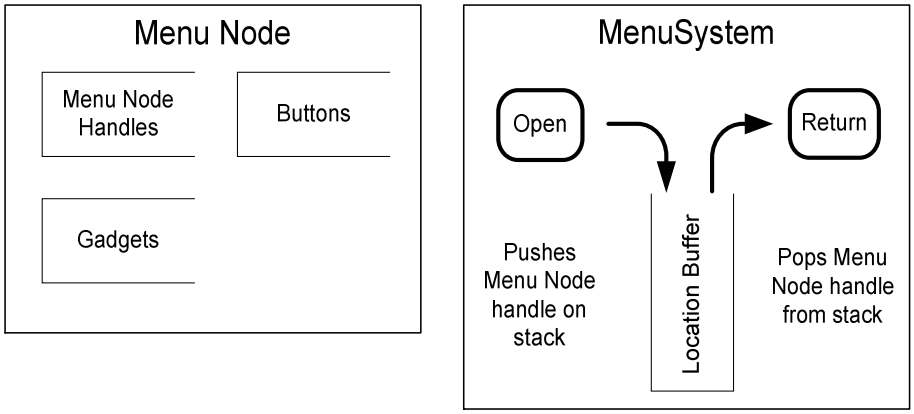
Avsnittet beskriver menyns funktionalitet, design och designbeslut.

5.2.1. Funktionalitet

Menyns funktion är att veta aktiv meny i systemet, skapade grafiska komponenter i systemet, konfigurerade knappar i aktiv meny och att uppdatera grafikmotorn vid ett menybyte.

5.2.2. Design

Menyn består av två delar: systemdel och menynodsdel. Systemdelen har en stack Location Buffer som lagrar Menu Node handles. Stacken skapar möjligheten att navigera tillbaka till föregående meny. En menynod har också information om grafiska komponenter, knappar och undermenyer.



Figur 21 MenySystem

Vid ett menybyte meddelar menyn grafikmotorn om den nya aktiva menyn.

5.2.3. Designbeslut

Designbesluten för menydelen är att den ska vara uppbyggd med en trädstruktur, vilket är det mest logiska och enkla sättet för att representera en menystruktur. För att tillhandhålla navigering för användaren lagras information om menyerna som besökts, informationen lagras i en stack för att det är ett enkelt sätt att representera navigering. Stacken är också tillräcklig, för en användare av systemet har inte tillgång till framåt navigering i menystrukturen. Den statiska lagringen är uppdelad i två datatyper menysystem och menynod, anledningen till det är att noderna inte ska behöva navigeringsinformation och det känns mer logiskt korrekt att det finns en enskild enhet för att veta aktiv menynod. Menynoden ska innehålla informationen för de grafiska komponenterna, knapparna, och undermenyerna som finns för varje menysida vilket är logiskt när en menynod representerar en menysida.

5.2.4. Konfiguration

Koden beskriver en konfiguration av en menynod och den är genererad via VCSCfg.

```
    _T("POWER"),   /* Menyns namn */
    1u,            /* Antal undermenyer */
    2u,            /* Antal gadgets */
    /* En Array av referenser till undermenyer */
    { 11u, NoMenu, NoMenu, NoMenu,
      NoMenu, NoMenu, NoMenu, NoMenu,
      NoMenu, NoMenu },
    /* En Array av Gadgets */
    {
      { {0,Group}, {15,0}, {244,20}, FALSE },
      { {9,Group}, {25,20}, {135,190}, TRUE },
    },
    /* En Array av Knappar */
    {
      { Menu, Menu_One, 0xff },
      { NoButtonType, NoMenuChild, 0xff },
      { NoButtonType, NoMenuChild, 0xff },
      { NoButtonType, NoMenuChild, 0xff },
      { NoButtonType, NoMenuChild, 0xff },
      { Back, NoMenuChild, 2 },
      { Prev, NoMenuChild, 26 },
      { Next, NoMenuChild, 4 },
      { Open, NoMenuChild, 5 },
      { Home, NoMenuChild, 10 }
    }
```

Källkoden visar att menyns namn är POWER och att menyn har en undermeny och två grafiska komponenter. De två grafiska komponenterna är gruppkomponenter som kan innehålla flera grafiska komponenter. Den första har identifikationsnummer 0 och är placerade på x-position 15 och y-position 0 med bredd 244 och höjd 20 och komponenten kan inte bli vald i knapphanteraren. Sedan är ett antal knappar definierade först en menyknapp som öppnar första menyn sedan fyra odefinierade knappar och sedan ett antal navigerings knappar som refererar till menyikoner 2, 26, 4, 5 och 10.

5.3. GRAFIKMOTOR

Avsnittet presenterar grafikmotorns funktionalitet, design och designbeslut.

5.3.1. Funktionalitet

Grafikmotorns funktion är att hantera utritning av grafiska komponenter och att det sker när det får ske t.ex. när en komponent i aktiv meny ber om uppdatering. Vid ett menybyte sker utritning av alla komponenter i den menyn.

5.3.2. Design

Grafikmotorn lagrar aktiv meny för att veta vilka grafiska komponenter som får ritas ut och uppdateras. Vid uppdatering kollar grafikmotorn

igenom aktiv meny om komponenten finns med och ifall den gör det ritas komponenten ut.

5.3.3. Designbeslut

Designbeslut för grafikmotorn är, referenser ska inte lagras för grafiska komponenter, som är aktiva utan istället lagras en referens till aktiv meny. Det beslutades för att det är en enkel lösning som kräver mindre lagring och för det är logiskt för grafikmotorn att kolla aktiv meny. Ett annat beslut är att när en komponent ber om uppdatering måste grafikmotorn kolla att komponenten finns i aktiv meny. Det är för att en grafisk komponent kan uppdateras när som helst utan att finnas med i aktiv meny och då får komponenten inte ritas ut. Vid uppdatering av aktiv meny ritas alla komponenter i den menyn ut. Skälet till detta är att alla komponenter ska ritas ut utan att behöva uppdatera alla komponenter i menyn var för sig. Uppdatering till display sker avskilt från utritning av komponenter, för att minska antalet uppdateringar till display, för annars sänks hastigheten. Tanken är att grafikmotorn ska vara den logiska kopplingen mellan menydelen och varje grafisk komponent. Skälet är att utvecklaren ville uppnå en modulär design, vilket leder till enklare utökning av fler grafiska komponenter i biblioteket, och en enklare design av menysystemet.

5.3.4. Ändrade beslut

Ett ändrat beslut för grafikmotorn är att ta emot knapphändelser och meddela komponenterna. Skälet till att det ändrades är att systemet blev invecklat och grafikmotorn komplex.

5.4. GADGET

Avsnittet beskriver föräldraklassen *Gadget* med klassens funktionalitet, design, designbeslut och de komponenterna som implementerats i systemet.

5.4.1. Funktionalitet

Gadget klassens funktion är att koppla lös de grafiska komponenterna från systemet på ett objektorienterat vis. klassen tillhandahåller därför metoder och funktioner för initialisering, uppdatering, utritning, etc. Klassen initialiserar också alla grafiska komponenterna i ett konfigurerat system.

5.4.2. Design

Klassen definierar de datatyper och funktionsarrayer som beskriver en komponent. Funktionsarrayerna lagrar alla funktioner som en komponent ska innehålla. Funktionerna är:

GG_XInit()

Funktion för initialisering av en komponent och beroende på komponentens typ kan det t.ex. vara att sätta stackpekare och initialisera variabler.

GG_DrawX()
Funktion som ska hantera den grafiska komponentens utritning.

GG_UpdateX()
Funktion som ska uppdatera den grafiska komponenten.

GG_HandleButtonX()
Funktion som inte används av alla komponenter. Den används ifall den givna komponenten ska aktiveras för att hantera knapphändelser.

GG_XGet()
Standard *Get()* metod, används för att hämta värden från en grafisk komponent, värdet kan t.ex. vara aktivt element i en lista.

GG_XSet()
Standard *Set()* metod, används till att sätta attribut för en grafisk komponent, attributet kan t.ex. vara att sätta positionen av en slider.

GG_SetXSelected()
Funktion för att sätta grafiska komponentens selected attribut.

GG_SetXActive()
Funktion för att särra grafiska komponentens active attribut.

En grafisk komponent beskrivs av dessa tre attribut:

- Ett id i två delar, typ och nummer t.ex. pgbar 2.
- Storlek med en x komponent och y komponent.
- Position med en x komponent och y komponent.

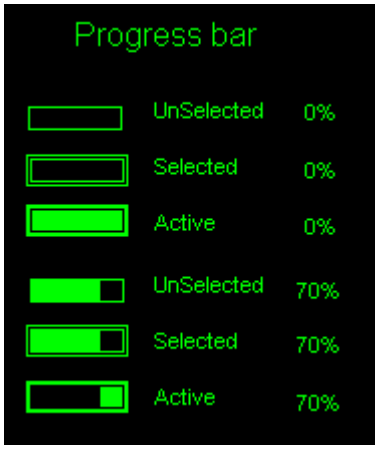
5.4.3. Designbeslut

Designbeslut för grafiska komponenter är att de inte ska ha information om storlek och position utan informationen ska lagras i menynoderna, det beslutades för att en grafisk komponent ska kunna finnas i flera menynoder. Alla anrop till en grafisk komponent ska vara ett av de publika anropen beskrivna i föregående avsnitt. En grafisk komponent måste be om uppdatering från grafikmotorn innan den kan ritas ut, och det ska endast vara grafikmotorn som anropar en grafisk komponents ritmetod. En grafisk komponent ska konfigureras ifall den ska sättas till att vara aktiv och vald. Det beslutades för interaktiva delen av systemet, för att kunna navigera mellan komponenterna och aktivera dem. De grafiska komponenterna ska hantera knapphändelser, det utnyttjas av samma interaktiva anledning som tidigare t.ex. ifall en grafisk komponent har blivit vald ska den därefter reagera på knapphändelser. Alla grafiska komponenter som finns konfigurerade för ett menysystem ska bli initialiserade när föräldraklassen blir initialiserad, det är för att säkerställa att komponenterna blivit initialiserade innan användning. För att identifiera specifika komponenter har komponenterna specifika nummer och typer. Vid uppdatering av en komponent som tillhör en grupp ritas endast den specifika komponenten och inte hela gruppen. Det är för att slippa onödiga utritning av komponenter. Men det finns en komponent som kräver utritning av alla komponenter, då utnyttjas en metod för att

rita alla komponenter. Komponenten Label kan ha referenser till andra grafiska komponenter. Det används för att navigera från en Label till en annan komponent. Sista beslutet är föräldraklassen för att koppla lös komponenterna från systemet, det bestämdes för att systemet ska vara modulärt och enkelt för att kunna lägga till nya grafiska komponenter i systemet.

5.4.4. Progress bar

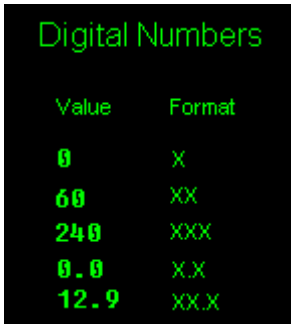
Komponenten fungerar som en *progress bar* och kan ha tre tillstånd unselected, selected och active. Tillstånden utnyttjas, ifall komponenten ska reagera på knapparna för att sätta värdet på komponenten.



Figur 22 Progress bar

5.4.5. Digital Numbers

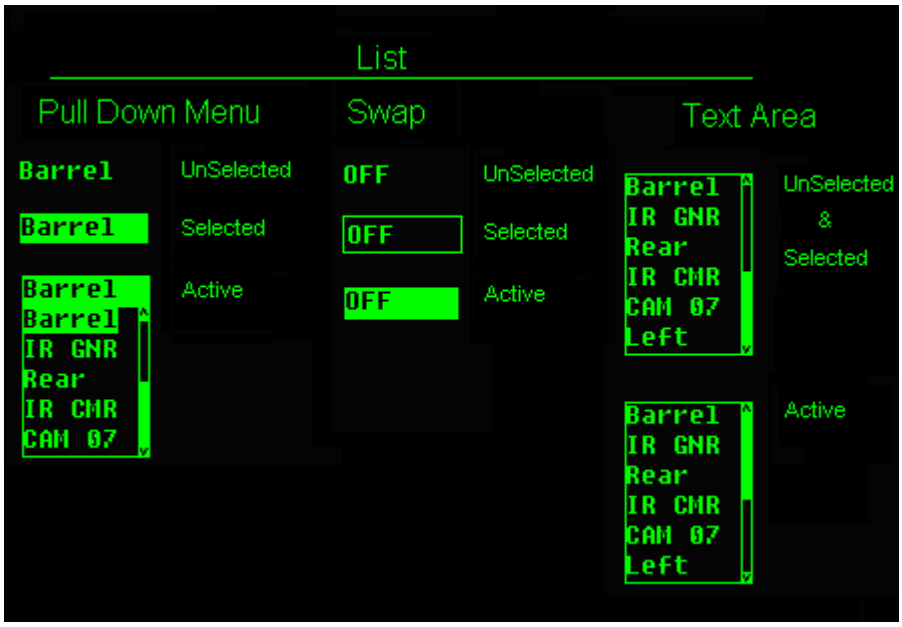
Grafisk komponent för digitalsiffror där användaren kan tillsätta formatet dvs. antalet siffror på heltalsdel och antalet siffror på decimaldelen.



Figur 23 Digital Numbers

5.4.6. List

Innehåller en lista av element, listan kan hanteras på tre sätt *Pull Down Menu*, *Swap* och *Text Area*. *Pull Down Menu* kan öppnas som en rullgardinsmeny och därefter välja ett av elementen i listan att vara aktivt, *Swap* växlar mellan elementen för att bestämma aktivt element och *Text Area* representerar endast informationen i listan.



Figur 24 List

5.4.7. Menu icons

Grafisk komponent som representerar en menyikon och för att visa händelsen av en knapptryckning kan menyikonen antingen vara vald eller inte vald.

Figur 25 Menu Icon

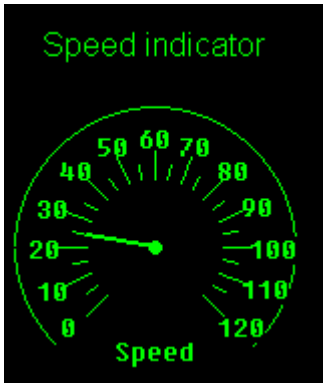
5.4.8. Digital Watch

Digital klocka som visar tiden i formatet:
{timmar : minuter : sekunder}.

Figur 26 Digital Watch

5.4.9. Speed indicator

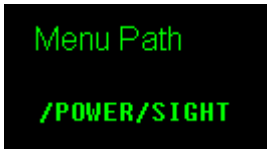
Hastighetsmätare där användaren kan sätta start och slut värdet på hastighetsmätaren t.ex. 0 till 120 som visas på bilden.



Figur 27 Speed Indicator

5.4.10. Menu path

Grafisk komponent som visar sökvägen till aktiv meny.



Figur 28 Menu Path

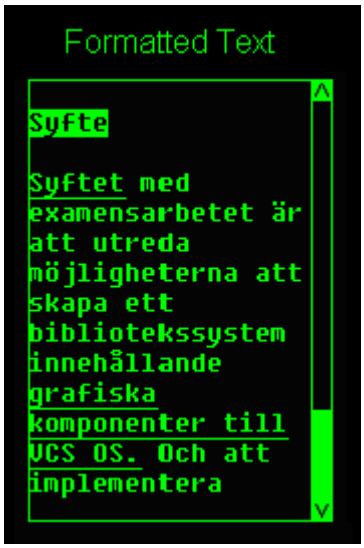
5.4.11. Label

Textetikett som har en navigeringsfunktion där man kan lägga till en referens till andra komponenter, t.ex. en referens till listan och användaren kan då navigera sig från textetiketten till listan.

Figur 29 Label

5.4.12. Formatted Text

Grafisk komponent med styrkoder för att ange saker som (huvudrubrik, understruket, punktlista etc.), m.a.o en komponent med minimerad html. Komponenten har egenskaper som multiline och wrapping av text och en slider för att kunna navigera upp och ned i texten. Tanken är att använda den för att skriva löptext eller liknande.



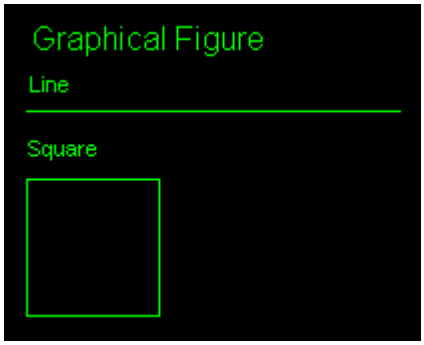
Figur 30 Formatted Text

5.4.13. Group

Grafisk komponent för att gruppera ihop komponenter med varandra, vilket gör det enklare att skapa ett menysystem. T.ex. om man vill att menyikoner för navigering ska finnas i samtliga menyer, då kan användaren skapa en grupp med fem menyikoner och lägga till gruppen i varje meny, istället för att lägga in dem enskilt.

5.4.14. Graphical Figure

Grafisk komponent som är enklare grafiska primitiver t.ex. linjer, cirklar och fyrkanter.



Figur 31 Graphical Figure

5.5. KNAPPHANTERING

Avsnittet presenterar knapphanteringen med klassens funktionalitet, design och designbeslut.

5.5.1. Funktionalitet

Knapphanterarens funktion är att lyssna på knapphändelser och uppdatera menysystemet eller komponenterna beroende på händelse.

5.5.2. Designbeslut

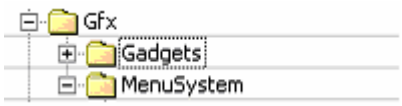
Designbeslut för knapphanteringen är att den behöver en buffer för lagring av valbara komponenter, m.a.o när man öppnar en meny skapas en lista på alla komponenter i aktiv meny, för att kunna växla mellan komponenterna i den menyn. Knapphanteraren behöver också en stack för aktiva komponenter, för att få en historik över aktiverade komponenter, när komponenter aktiverar andra komponenter. Dock krävs ett förbud mot att samma komponent förekommer flera gånger för att förhindra rekursion. Alla knapphändelser ska skickas vidare till den översta aktiva komponenten på stacken, förutom ”Back” händelsen, då ska översta på stacken ”poppas” för att återgå till föregående komponent. Ett annat beslut är vid byte av menysida, rensas skärmen för att allting ska ritas om. Knapphanteraren sätter när en komponent är aktiv eller vald. Sista designbeslut är att knapphanteraren ska vara lös kopplad från grafikmotorn för att ha erhålla ett modulärt system.

5.5.3. Ändrade beslut

Ett ändrat beslut är att skapa en funktion som söker igenom menyn efter nästa valbara komponent, istället för att skapa en lista över alla komponenter, anledning till att det ändrades är att det blir för komplext att kolla vilken som är nästa komponent.

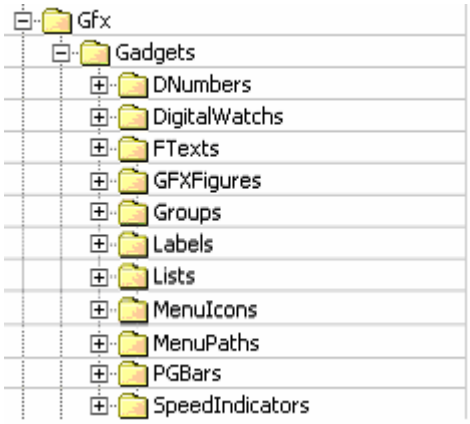
5.6. KONFIGURERING

Konfigureringen sker i VCSCfg:s Gfx katalog som har två underkataloger, en för komponenterna *Gadgets* och en för menysystemet *MenuSystem*.



Figur 32 VCSCfg Gfx

I *Gadgets* finns komponenterna som skapas och konfigureras i ett system. Alla skapade komponenter har en unik typ, och får ett unikt nummer för att identifiera enskilda grafiska komponenter vid genereringen av källkoden till applikationen. Komponenterna har olika attribut beroende på komponentens typ t.ex. *Labels* har ett attribut för vad det ska stå i textetiketten.



Figur 33 VCSCfg Gadgets

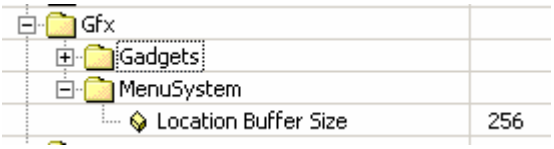
I *MenuSystem* skapas och konfigureras menyerna. Varje meny har tio attribut för att bestämma en menys undermenyer. I varje meny finns en katalog för konfigurering av menyns knappar, och en katalog för att skapa referenser till grafiska komponenter. För varje referens till en komponent kan man sätta position och storlek, och ifall komponenten ska väljas interaktivt i systemet. I nästa kapitel beskrivs utförligare en konfiguration av systemet.

Kapitel 6. SYSTEMTEST

Kapitlet presenterar ett test av systemet, där ett nytt menysystem skapas med ett antal grafiska komponenter och testas i en simuleringsmiljö.

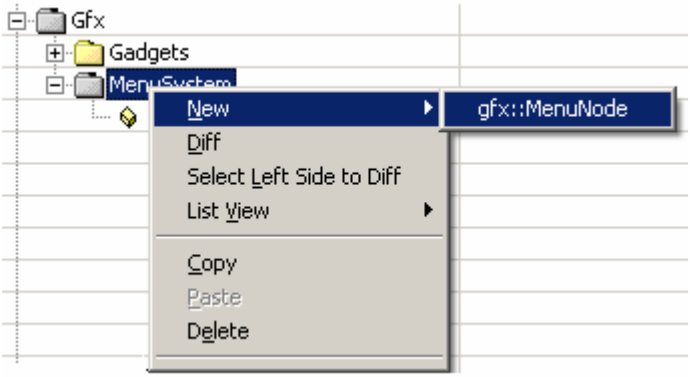
6.1. KONFIGURERING

Testet visar först en konfiguration av ett menysystem där två menyer skapas. Menyerna tillsätts ett antal referenser till förkonfigurerade grafiska komponenter. För att skapa ett nytt menysystem öppnas *Gfx* två kataloger *Gadgets* och *MenuSystem*. I *Gadgets* är de förkonfigurerade grafiska komponenterna och i *MenuSystem* finns de konfigurerbara menyerna.



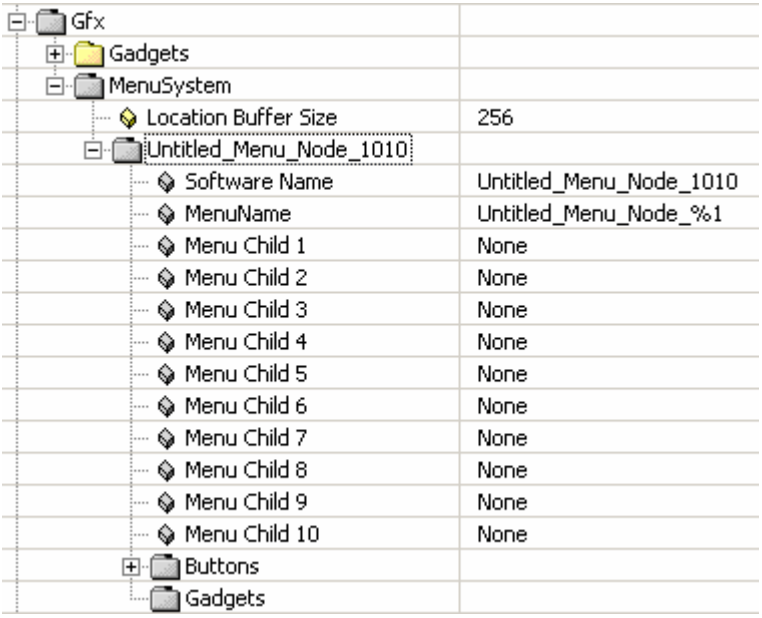
Figur 34 Systemkatalogerna

För att skapa två menyer höger klickar man på *MenuSystem* mappen i VCSCfg och väljer New -> gfx::MenuNode.



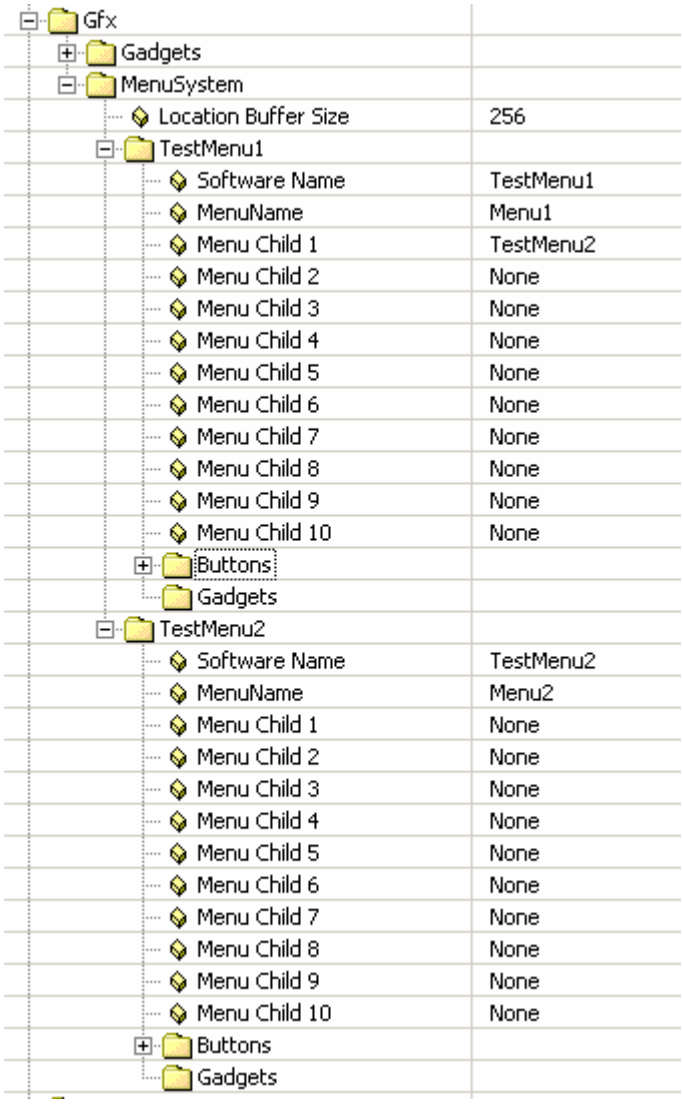
Figur 35 Att skapa en meny

När en meny har skapats är det tid för konfiguration av menyn. Dess parametrar är initialt inte satt till några värden, t.ex. *Software Name* och *MenuName*. En meny har av två undermappar *Buttons* och *Gadgets* och tio referenser för undermenyer. *Buttons* mappen beskriver de tio knapparna i en meny. *Gadgets* mappen innehåller referenser för de grafiska komponenterna i menyn.



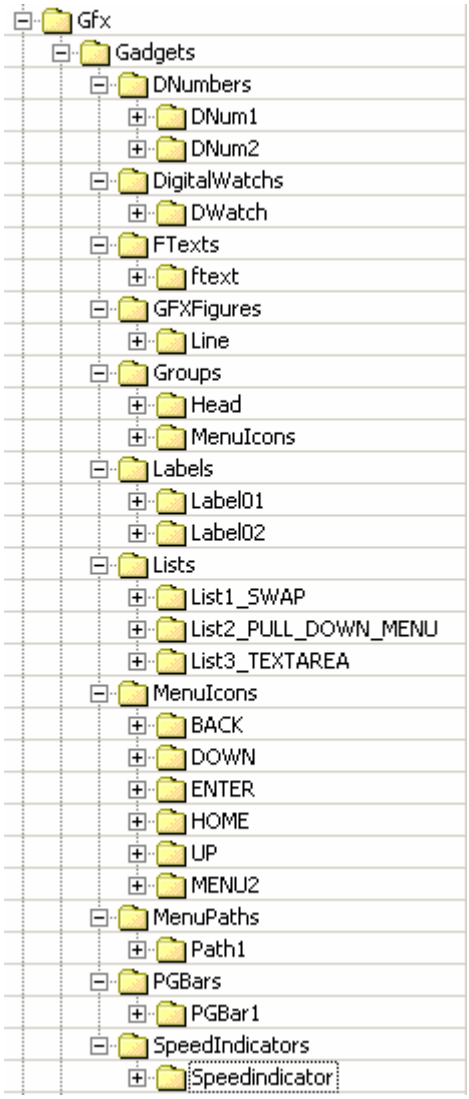
Figur 36 En skapad meny

Först sattes parametrarna för menyerna. En meny fick *Software Name* *TestMenu1* och dess *MenuName* blev *Menu1* den andra menyn fick *TestMenu2* och *Menu2* analogt. *TestMenu1* tillsattes *TestMenu2* som undermeny. Sedan måste referenser tillsättas för komponenterna och en konfigurering av knapparna i menyn utföras. Nedan visas en bild över konfigurationen av de två menyerna.



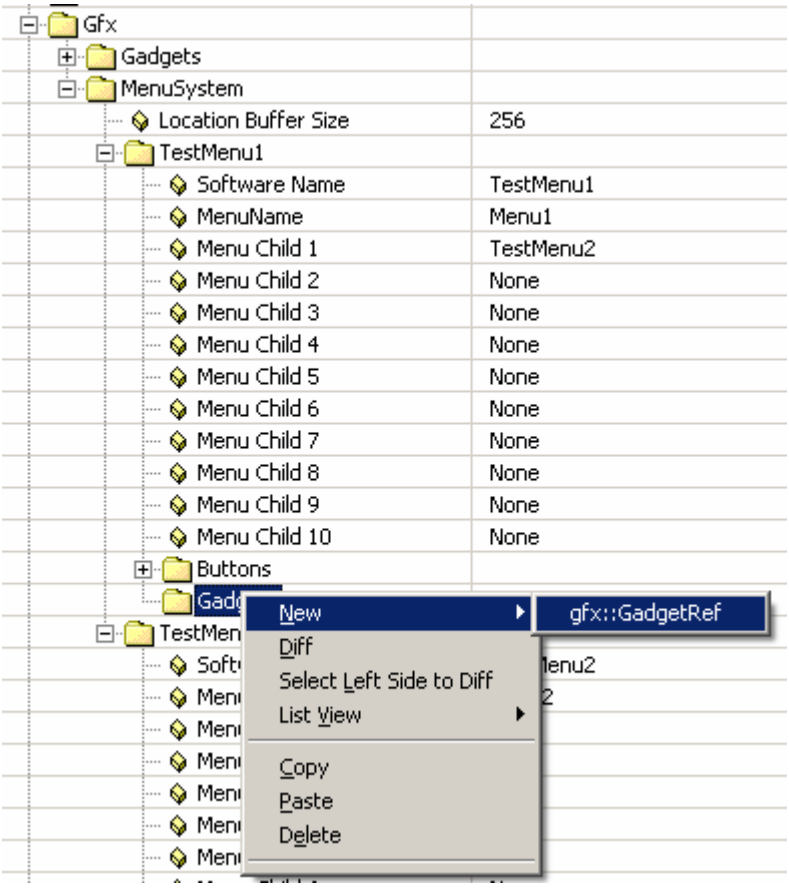
Figur 37 Två menyer

För testet hade ett antal grafiska komponenter förkonfigurerats som bilden visar nedan. De grafiska komponenter som kommer att användas för testet är *DigitalWatch*, *Group*, *Label*, *MenuIcon*, *Progressbar*, *SpeedIndicator*, *GFXFigures* och *MenuPath*.



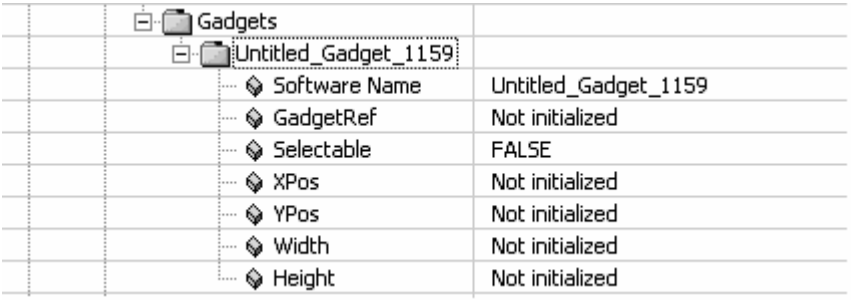
Figur 38 Ett antal olika grafiska komponenter

För att skapa referenser till grafiska komponenter, måste referenserna skapas i menyns *Gadgets* mapp. På mappen höger klickar man och väljer New -> gfx::GadgetRef.



Figur 39 Att lägga till grafiska komponenter i en meny

När en ny referens är skapad, sätter man parametrarna för den referensen. T.ex. placeraing i menyn, storlek och om den kan väljas interkativt via knapparna när applikationen exekveras. Parametern *GadgetRef* sätter man till någon av de förkonfigurerade grafiska komponenterna.



Figur 40 En ny referens till en grafisk komponent

För *Menu1* har fyra referenser skapats. *Head* den här referensen innehåller en klocka, linje och menysökväg. *MenuIcons* refererar till navigerings menyikonerna *BACK*, *UP*, *DOWN*, *ENTER* och *HOME*. *MenuIcon_Menu2* refererar till en menyikon för meny två och den har etiketten *MENU2*. *SpeedIndicator* är den sista referensen och det är en hastighetsmätare

Software Name	GadgetRef	Selectable	XPos	YPos	Width	Height
Head	Head	FALSE	15	0	244	20
MenuIcon_MENU2	MENU2	FALSE	0	0	15	64
MenuIcons	MenuIcons	FALSE	225	0	15	320
SpeedIndicator	Speedindicator	FALSE	40	40	140	140

Figur 41 Konfigurerade grafiska komponenter meny 1

Sex knappar är konfigurerade för *Menu1*. En av typen *Menu* som refererar till undermeny 1 och menyikon *MENU2*. Sedan är fem knappar konfigurerade för navigering *BACK*, *UP*, *DOWN*, *ENTER* och *HOME*.

Type	Menu Child	MenuIconRef
Menu	Menu_One	MENU2
NoButtonType	NoMenuChild	None
NoButtonType	NoMenuChild	None
NoButtonType	NoMenuChild	None
NoButtonType	NoMenuChild	None
Back	NoMenuChild	BACK
Prev	NoMenuChild	UP
Next	NoMenuChild	DOWN
Open	NoMenuChild	ENTER
Home	NoMenuChild	HOME

Figur 42 Konfigurerade knappar meny 1

För *Menu2* har fyra referenser skapats, *Head* refererar till samma *Head* som refererades i *Menu1*. *Label* refererar till en komponent av typen *Label*. *MenuIcons* refererar till samma *MenuIcons* som refererades i *Menu1* och *Progressbar* refererar till en komponent av typen *PGBar*.

Software Name	GadgetRef	Selectable	XPos	YPos	Width	Height
Head	Head	FALSE	15	0	244	20
Label	Label01	TRUE	40	40	100	15
MenuIcons	MenuIcons	FALSE	225	0	15	320
Progressbar	PGBar1	FALSE	140	40	50	15

Figur 43 Konfigurerade grafiska komponenter meny 2

De olika knapparna som är konfigurerade är navigeringsknapparna *BACK*, *UP*, *DOWN*, *ENTER* och *HOME*.

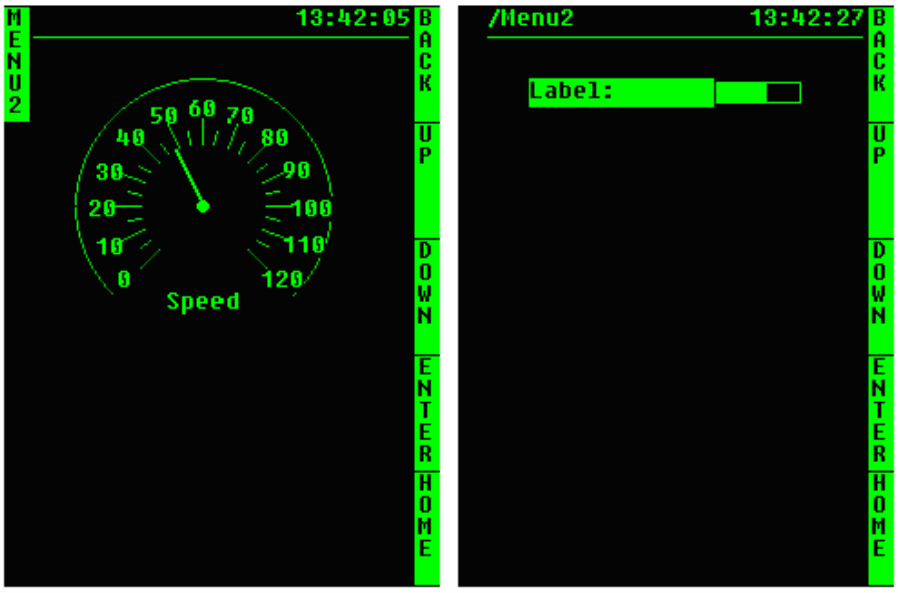
Type	Menu Child	MenuIconRef
NoButtonType	NoMenuChild	None
NoButtonType	NoMenuChild	None
NoButtonType	NoMenuChild	None
NoButtonType	NoMenuChild	None
NoButtonType	NoMenuChild	None
Back	NoMenuChild	BACK
Prev	NoMenuChild	UP
Next	NoMenuChild	DOWN
Open	NoMenuChild	ENTER
Home	NoMenuChild	HOME

Figur 44 Konfigurerade knappar meny 2

Nu när menysystemet är konfigurerat och de två menyerna har fått ett namn, referenser till grafiska komponenter och menyernas knappar är konfigurerade ska koden genereras, kompileras och byggas till applikationsprogrammet.

6.2. RESULTAT

Testet var lyckat. Nedan visas två bilder av det konfigurerade systemet en för *Menu1* och en för *Menu2*. Första menyn fungerade, den fick navigerings ikoner, klockan och hastighetsmätaren mm. Genom att trycka på knappen för *MENU2* ikonen kunde man navigera till *Menu2*. Som man ser fick *Menu2* en *Label*, en *ProgressBar*, navigeringsikoner och klockan. I *Menu2* har sedan *Label* blivit vald och värdet på *PGBar* ställts om till ca 60%.



Figur 45 Menu1 och Menu2

Kapitel 7. SLUTSATSER

Kapitlet presenteras slutsatserna av arbetet de begränsningar, framtida arbeten och reflektioner som utvecklaren tagit.

7.1. MÅL & BEGRÄNSNINGAR

I kravspecifikation som utfärdades av BAE Systems Hägglunds AB har samtliga mål utförts. Begränsningar som finns i systemet är mindre detaljer som att införa möjlighet att konfigurera arrayer och buffrar. Problemet jag löst är tiden det tar för Hägglunds att utveckla användargränssnitt till deras fordon. Istället finns då en möjlighet att lägga denna tid till att utöka detta system och därmed nå en robustare produkt i framtiden. Verktyget skulle utvecklas för att innan fanns ett väldigt primitivt grafiskt utbud, m.a.o endast enklare primitiver för grafisk utritning t.ex. linjer, cirklar och enklare textsträngar. Med verktyget har arbetet av menysystem satts på en högre nivå.

7.2. FRAMTIDA ARBETEN

Det utvecklade systemet kan utökas på ett flertal sätt. Det går t.ex. att utöka med fler grafiska komponenter i biblioteket och utöka med ett användargränssnittsverktyg som är beskrivet i kapitel 3. Vissa delar av systemet främst vissa av de grafiska komponenterna skulle kunna förfinas t.ex. formaterade-textkomponenten. Vad gäller systemet som helhet är det inga större förändringar som krävs.

7.3. REFLEKTIONER ÖVER ARBETET

Det som var bra med arbetet var möjligheten att få testa sina kunskaper för att utveckla ett system och att få vara självgående med handledning när problem dök upp. Mindre bra var att av säkerhetsskäl fanns ingen tillgång till Internet på arbetsdatorerna.

KÄLLFÖRTECKNING

[1] BAE Systems AB*, Presentation of BAE Systems in Sweden*
BAE Systems in Sweden.PPT.

[2] Wikipedia, *Waterfall Model*
http://en.wikipedia.org/wiki/Waterfall_model
(Besökt 2007-11-05)

[3] Wikipedia, *WYSIWIG*
http://sv.wikipedia.org/wiki/WYSIWYG
(Besökt 2007-11-23)

[4] Myers, B.A., Hudson, S.E., Pausch, R., (2000) Past, Present, and Future of user Interface Software Tools*, ACM Transaction on Computer-Human Interaction,* **7**:3-28.

[5] Shneiderman, B, Plaisant, C (2005) *Designing the User Interface,* Pearson Education Inc.

[6] Myers, B.A, Rosson, M.A., *Survey On User Interface Programming,* Carnegie Mellon University*, IBM T.J.* Watson Research Center. Maj 3 - 7 1992. sid 195-202

[7] Brad A. Myers. "A Brief History of Human Computer Interaction Technology." ACM interactions. Vol. 5, no. 2, March, 1998. pp. 44-54.

[8] Molin, P, Ström, F, *A GUI Builder for Erlang/GS,* Uppsala Universitet. December 4 1996.

[9] Brad A. Myers. "User Interface Software Tools" August 1994
CMU-CS-94-182. This report supersedes CMU-CS-92-114 from February, 1992, published as: Brad A. Myers. "State of the Art in User Interface Software Tools," Advances in Human - Computer Interaction, Volume 4. Edited by H. Rex Hartson and Deborah Hix. Norwood, NJ: Ablex Publishing, 1993. pp. 110-150.

[10] Wikipedia, *FLTK*
http://en.wikipedia.org/wiki/FLTK
(Besökt 2007-12-15)

[11] FLTK hemsida, *FLTK Fast Light Toolkit*
http://www.fltk.org/
(Besökt 2007-12-15)

[12] FLTK dokumentation, *FLTK2.0 Documentation*
http://www.fltk.org/doc-2.0/html/index.html
(Besökt, 2007-12-15)

[13] wxGlade hemsida, *wxGlade a GUI builder for wxWidgets*
http://wxglade.sourceforge.net/
(Besökt, 2007-12-15)

[14] Wikipedia, *WxGlade*
http://en.wikipedia.org/wiki/WxGlade
(Besökt, 2007-12-15)

[15] Glade hemsida,*Glade - a User Interface Designer*
http://glade.gnome.org/
(Besökt, 2007-12-15)

[16] Wikipedia, *Glade Interface Designer*
http://en.wikipedia.org/wiki/Glade_Interface_Designer
(Besökt, 2007-12-15)

[17] Wikipedia, *Widgets*
http://sv.wikipedia.org/wiki/Widgets
(Besökt, 2007-11-23)

[18] BAE Systems AB,*VCS Bas Teknisk generell*
VCS Bas Teknisk generell.ppt

[19] BAE Systems AB, *VCS arkitektur*
VEHICLE-#244588-v3B-VCS_Basic_Architecture.PPT

[20] Phyton homepage, *Python Programming Language -- Official Website*
http://www.python.org/
(Besökt, 2007-11-02)

[21] Wikipedia, *Graphical user interface builder*
http://en.wikipedia.org/wiki/Graphical_user_interface_builder
(Besökt, 2007-12-20)

[22] BAE Systems AB, *GFXDisplay Dokumentation*
30_1513-API_Description_GfxDisplay.doc

[23] BAE Systems AB, *VCSCfg Manual*
VCSCfg UserManual.doc

[24] Wikipedia, *Controller Area Network*
http://en.wikipedia.org/wiki/Controller_Area_Network
(Besökt, 2008-01-14)

[25] Zongming Fei, *GUI Builder Tools*
http://www.cc.gatech.edu/classes/cs6751_97_winter/Topics/gui-builder/
(Besökt, 2008-01-15)

[26] IBM, *Design basics*
http://www-03.ibm.com/easy/page/6
(Besökt, 2008-01-15)

[27] Wikipedia, *Människa–datorinteraktion*
http://sv.wikipedia.org/wiki/MDI
(Besökt, 2008-01-15)

[28] Wikipedia, *Dialog box*
http://en.wikipedia.org/wiki/Dialog_box
(Besökt, 2008-01-15)