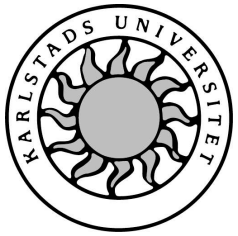


Datavetenskap

Velio Roumenov, Jan Igerud

Grafisk gränssnittsprogrammering för handdatorer

Examensarbete

2004:22

Grafisk gränssnittsprogrammering för handdatorer

Velio Roumenov, Jan Igerud

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Velio Roumenov

Jan Igerud

Godkänd, 040603

Handledare: Hannes Persson

Examinator: Donald F. Ross

Sammanfattning

Handdatorer är den kanske snabbast växande grenen av datorer idag. De ger dess användare en möjlighet att bearbeta information i stor sett närsomhelst, varsomhelst och i en för användaren välkänd miljö.
Det faktum att handdatorer blir bättre och snabbare ger oanade möjligheter för utvecklingen av applikationer för dessa datorer. Det är dock viktigt att känna till hur en applikation för en handdator skall utformas för att en användare skall kunna använda den. Handdatorer har ett flertal restriktioner såsom storlek, vikt, skärmyta, inmatning med mera. Tillsammans med det faktum att dess användare ofta inte har den datorvana som återfinns hos användare av stationära datorer. Dessa faktorer gör att applikationsutveckling till handdatorer skiljer sig markant från utvecklingen av applikationer för stationära datorer.
Uppsatsen inriktar sig mot att ge en övergripande bild av handdator, hur dess grafiska användargränssnitt bör designas och hur några av de största handdator operativsystemen hanterar grafikbearbetning.

Graphical Interface Programming for handheld computers

Handheld computers are probably the fastest growing area of computers today. They give its user an ability to handle information at anytime, anywhere and a for the user familiar environment. The fact that handheld computers become better and faster gives unforeseen capabilities for the development of applications for this kind of computers. It is, however, important to know how an application for a handheld computer should be designed so that the user feels comfortable using it. Handheld computers have several restrictions such as size, weight, screen space and input. Along with the fact that users of handheld computers often don't have the particular computer skills which is common among ordinary personal computer users. This are factors that make application development prominently different for handheld computers versus personal computers. This thesis whishes to give an overall view of handheld computers, how its graphical user interface should be designed and how several of the large handheld computer operatingsystems handle graphics.

Tack

Vi vill tacka Hannes Persson som har stött oss i mot- och medvind. Han har alltid kommit med konstruktiv kritik för att arbetet skall bli väl genomarbetat. Vi har alltid fått hjälp även då vi inte har haft möten bokade på listan. Tack.

Innehåll

Figurer

1 Introduktion

Vi tror att handdatorer kommer att spela en stor roll i människans vardag i framtiden[1]. Då dessa troligen kommer att användas av personer med varierande datorkunskaper så bör handdatorer ha ett lättanvänt och intuitivt användargränssnitt. För att nå detta mål är det viktigt att det finns riktlinjer för design av det grafiska användargränssnittet, vidare så behövs riktlinjer för val av ett grafiskt användargränssnitts API (eng. Application Programming Interface). Att göra ett bra val av API innan utvecklingsfasen kan underlätta i ett projekt och på så sätt tillföra att man får en välfungerande applikation som uppskattas av användare.

1.1 Problemformulering

En applikationsutvecklare behöver ha ett bra applikationsprogrammeringsgränssnitt som avskärmar utvecklaren från detaljer i utvecklandet av GUI (eng. Graphical User Interface). Detta för att dels höja produktiviteten, men även för att skapa ett standardutseende för användaren.

1.2 Syfte

Uppsatsens syfte är att utvärdera APIn för utveckling av GUI för handdatorer. Ett annat syfte med uppsatsen är att ta upp grafiska användagränsnitts riktlinjer. Detta för att framhäva de skillnader som finns mellan GUI design för stationära datorer och handdatorer.

1.3 Avgränsning

Uppsatsen syfte är inte att ge en detaljerad beskrivning om hur grafiska applikationer programmeras, utan att ge en överblick över de mekanismer som är involverade när ett

[1] Vi baserar detta främst på intryck som vi fått från PalmOS[23], Symbian[24] och Microsoft[25], baserat främst på deras företags presetationer men även på försäljningsstatestik från företagen.

grafiskt användargränssnitt skapas.

1.4 Mål

Huvudmålet är att ta fram utvärderingskriterier och utvärdera ett antal APIer enligt dessa kriterier. Ytterligare mål är att presentera viktiga aspekter inom designområdet för utveckling av grafiskt användargränssnitt. Vidare skall uppsatsen presenteras på ett sådant sätt så att ett företag, privatperson eller en kursansvarig skall kunna använda denna uppsats för att kunna fatta ett beslut om vilket API som passar dem bäst.

1.5 Disposition

Uppsatsen inleds med en kort introduktion till handdatorer i kapitel 2. Detta för att läsaren på så sätt skall kunna se de möjligheter och begränsningar som finns hos handdatorer. Då en bra design av det grafiska användargränssnittet är viktigt i handdatorer så presenteras därefter i kapitel 3 ett antal riktlinjer för design av grafiska användargränssnitt. Vidare fortsätter uppsatsen i kapitel 4 med en övergripande introduktion till de APIer som har valts att behandlas i uppsatsen. Uppsatsen fortsätter därefter i kapitel 5 med en metodbeskrivning av hur valet av APIer gjordes. Här presenteras även de utvärderingskriterier som använts för att utvärdera de APIer som beskrevs innan. Vidare så utvärderas även dessa APIer i detta avsnitt. I kapitel 6 redogör uppsatsen för våra erfarenheter av de APIer som presenterats i kapitel 4 samt kapitel 5. Här redogörs även för ett antal scenarier som kan uppstå beroende på vilken situation som man befinner sig i. Vidare så presenterar vi här vilket API som vi tycker är lämpligast att använda. Slutligen, i kapitel 7, presenteras våra slutsatser med avseende på projektet som helhet.

2 Bakgrund för handdatorer

I detta kapitel så ges en översikt av användningsområden och historik för handdatorer. Detta för att öka kunskapen om dess användare och de miljöer där de används. Dess kunskaper är essentiella då ett välfungerande grafiskt användargränssnitt skall utvecklas. Vidare behandlas vad handdatorer har använts till och vad de kommer att användas till. Kapitel 2 kommer att presentera några fysiska begränsningar för handdatorer såsom storlek, strömförbrukning med mera, som man måste beakta när man skapar applikationer för handdatorer. Detta för att ge en uppfattning om hur och vad som kan komma att utvecklas hos handdatorer.

2.1 Översikt

I uppsatsen är definitionen av handdator följande: En dator som är designad för att utföra ett fåtal specifika operationer (den här definition kommer av eng. "Informations appliances"[26]). Av specifikationen så framgår det att det handlar om allt från en digital klocka till en handdator. Här följer några punkter som redogör för skillnader mellan en handdator och en vanlig PC. Dessa är en sammanställning från. [26]

- Har en begränsad funktionalitet och ändamål.
- Behöver inte vara uppgraderingsbar eller påbyggnadsbar.
- Har en begränsad livslängd, användaren vill troligen byta ut hela apparaten efter några år, då hårdvaran ofta inte går att uppdatera.
- Kan uppfattas som billigare än en stationär dator.
- Lättare att använda och underhålla.
- Enkel att lära sig och använda.
- Ingen förväntning ställs på att användaren har datorvana.

2.2 Användningsområden för handdatorer

I detta avsnitt kommer de olika användningsområdena för handdatorer att presenteras. Detta för att ge en förståelse för vad de används till idag och vad vi tror att de kommer att användas till i framtiden.

2.2.1 Historia

Utvecklingen av handdatorer påbörjades av Alan Kay när han på 1970-talet jobbade för Xerox i Paulo Alto med en maskin som hette DynaBook [22]. Kay hade tidigt en vision av en maskin som skulle vara möjlig att bära med sig och skriva på med hjälp av en penna och som skulle kunna kommunicera med andra maskiner.
Företaget PSION, utvecklade från början program och spel till Sinclair datorer. PSION grundare, David Potter, kom på idén med bärbara dator på en Grekisk Taverna i London. Idén ledde till att PSION 1984 släppte sin första bärbara dator som då kallades för Organizer (se figur 2.1). 1986 släpptes Organizer II (se figur 2.1) som innehöll mera minne, större skärm och fler applikationer, bland annat en kalender, klocka och kalkylatorer. [15]

(a) PSION I

(b) PSION II

(c) GUI på en MC400

(d) GUI på en MC400

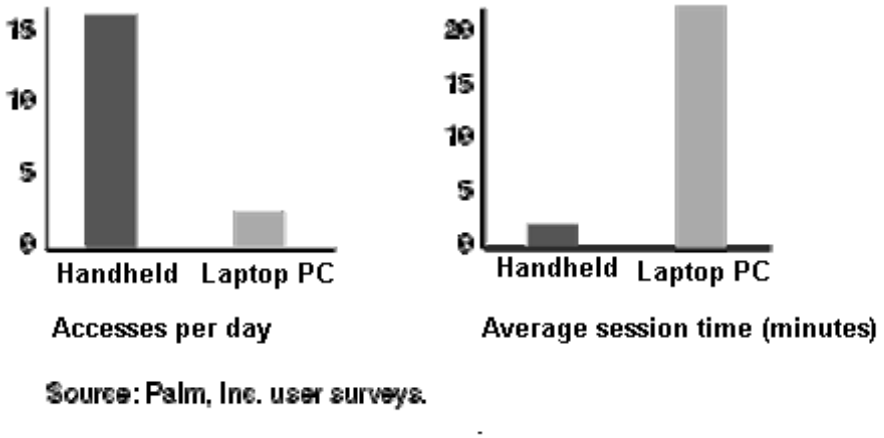
Figur 2.1: Organizer I, Organizer II och MC400 GUI

1989 släppte PSION MC 400, återigen en för sin tid mycket imponerande dator, vars storlek motsvarar ett A4 ark och den hade många intressanta funktioner (tex touchpad, möjlighet för att ansluta modem, fax, kortläsare mm) och många applikationer. Det mest revolutionerande var dock det nya operativsystemet EPOC (skapat av PSION själva).

Detta operativsystem gav stöd för användandet av GUI (se figur 2.1), detta i en tid då de flesta datorer fortfarande använde CLI (eng. Command Line Interface) inmatning i program. EPOC utvecklades vidare och används idag av Symbian. [21]
Den kanske mest kända handdatortillverkaren idag är förmodligen Palm. Palm startade 1996 och insåg tidigt att det fanns tre nyckelegenskaper för att göra en bra handdator:

Bärbar En handdator måste vara lätt att ta med sig överallt. Detta kriterie innebär i sin tur att handdatorn måste vara strömsnål, ha ett litet format och vara lätt att bära.

Användbarhet En handdator måste vara lätt att använda. Detta innebär att man bör utveckla dess applikationer och tjänster från ett användarperspektiv. GUIt måste vara förutsägbart för användare som ofta är en ovan datoranvändare. En handdator används ofta, men under korta sessioner detta jämfört med en vanlig dator där användare har långa sessioner, men inte använder den lika ofta något som figur 2.2 visar. [3]



Figur 2.2: Tid/åtkomst handdator vs PC

Flexibilitet En handdator måste vara flexibel, det vill säga att man på ett lätt sätt ska kunna anpassa både hårdvaran och mjukvaran efter de behov som användaren eller organ-

isationer har. Detta för att kunna tillgodose specialkrav från företag och organisationer som vill använda sig av en handdator.

Dessa kriterier har lett till att Palm var störst på handdator 2002 (enligt IDC, personal companions). PalmOS används idag på ca 30 miljoner sålda apparater och har ca 96.000 utvecklare. Dessa kriterier är numera ett mantra för alla handdatorutvecklare. [16]

Microsoft lanserade 1995 sin första version av Windows CE, ett operativsystem med samma Look-and-feel som många användare är vana vid från vanliga Windows. De har dock enligt uppsatsförfattarnas mening inte bidragit med något ytterligare för handdatorer. [30]

2.2.2 Nutid

I dagens läge så finns det generellt sett tre stora operativsystem för handdatorer på marknaden, SymbianOS, PalmOS och Windows CE. Microsoft har nyligen släppt en Windows XP version för sina handdatorer men i dagens läge så finns det inte implementerat på handdatorer. Det finns fler tillverkare men de är har en mycket liten del av marknaden varför de inte diskuteras här.
Alla dessa tillverkare har ur användarsynpunkt väldigt stora likheter, till exempel så sker inmatning ofta genom en tryckkänslig skärm. Få människor vill gå omkring med ett tangentbord under armen, något som innebär att man vill begränsa antalet inmatningar som krävs av en användare för att utföra det som personen önskar. Detta ställer i sin tur krav på att menyer och liknande inte får vara gjorda hur som helst (mer om detta i kapitel 3). Alla anammar dessutom de tre kriterier som togs upp tidigare (det finns dock vissa skillnader till exempel vid implementation av kalendern). De använder sig dessutom av det så kallade pappersparadigmet. Pappersparadigmet går ut på att när en användare lämnar en applikation så sparas all data som användare matat in. Användare kan inte stänga av applikationer som exekverar, detta för att användare inte skall behöva tänkta i banor

av att han eller hon exekverar en applikation. Något som kan verka avskräckande för en användare. När minne behöver frigöras så sköter operativsystemet om att stänga ned de applikationer som inte används.

På detta sätt så kan man se att en handdators begränsningar inte bara är av ondo utan att de faktiskt tvingar applikationsutvecklare att tänka på av vem och hur applikationen skall användas. Troligen kommer detta bidra till att en person som i vanliga fall har svårt för att hantera en stationär dator, hanterar en handdatorer med större entusiasm. Något som kommer att bidra till en större publik. Handdatorer gör det även lättare för människor att kommunicera med varandra, nästan överallt och på många olika sätt. Detta kan ses som en naturlig fortsättning på den trend som har påbörjats av mobiltelefonerna. Dagens handdatorer har dessutom en rad synkroniseringsmöjligheter med andra enheter genom till exempel Bluetooth[2], 3G[3] med mera, något som göra att det nästan bara är fantasin som sätter gränser för vad man vill använda den till för uppgifter.

2.2.3 Framtid

Handdatorer kommer att ha en mycket ljus framtid, inte minst på grund av mantrat att anpassa applikationerna för användaren och inte tvärtom, som så ofta sker på stationära datorer. Detta tillsammans med att användare ofta på ett ganska enkelt sätt kan få information vid ett speciellt ögonblick. Till exempel kan användaren få kartor över en stad eller filmutbudet på närmaste biograf och att den informationen kan presenteras på ett kort och koncist sätt. Handdatorer kommer troligen leda till ett ännu mera distribuerat system av information som man vill ha som användare. Till exempel så kan kylskåpet skapa en inköpslista av matvaror och skicka den till handdatorn och dessutom påminna användaren om detta när användaren närmar sig en affär med det utbudet. En användare som kör bil kan med samma handdator få information om väglag, trafikproblem med mera och få förslag på en alternativ väg. När användaren är på jobbet så kan handdatorn informera

[2]För mer information om bluetooth se www.bluetooth.org.
[3]för mer information om 3G se www.3Gpp.org.

om möten och åtaganden med mera. Det finns dock vissa nackdelar med handdatorer som kan kommunicera med andra enheter. Informationen på dem är inte säker, något som är en stor nackdel eftersom de ofta kan innehålla privat information som till exempel SMS, epost, anteckningar, möten och telefonnummer. Det kommer att vara av stor vikt att skydda den här informationen från obehöriga då handdatorer utbyter information med andra system.

2.3 Begränsningar

Handdatorer har av naturliga skäl vissa begränsningar för att passa dess användare. Dessa begränsningar såsom vikt och storlek (skärm, maskinstorlek, minne) skapar även en väl definierad miljö för att skapa applikationer. Applikationerna måste hela tiden använda apparatens prestanda maximalt, vilket gör att applikationsutveckling för handdatorer i vissa avseenden kan skilja sig markant från utveckling av applikationer för en PC.

2.3.1 Storlek

Skärmstorlek har en avgörande roll då många av en handdators meddelanden är visuella. Skärmstorleken varierar mellan olika handdatorer, men det finns vissa rekommenderade riktlinjer beroende för vilket operativsystem som applikationen utvecklas för.

- Symbian - 230,240x260 pixels
- PalmOS - 160x160, 320x320 pixels
- Windows CE (XP) - 240x320 pixels
- PSION - 640x420 pixels

Dessa tenderar att vara de vanligaste storlekarna hos handdatorer men även andra kan förekomma, varför det är viktigt att man undersöker vilka handdatorer som skall använda applikationen.

2.3.2 Strömförsörjning

Strömförsörjning är något som man måste beakta vid utveckling av applikationer för en handdator. Eftersom handdatorer har en begränsad strömförsörjning (använder sig ofta av batterier) bör applikationer använda så lite processorkraft och grafiska funktioner som möjligt, då dessa drar mycket ström. Palm designade bland annat sitt skrivalfabet efter att det var energisnålt att hantera.

2.3.3 Kapacitet

Kapaciteten hos en handdator vad det gäller minne (RAM, ROM), processorkapacitet varierar kraftigt och det sker en snabb ökning av kapacitet varför inga direkta riktlinjer finns. Klart är dock att det är markanta skillnader mellan en normal dator och en handdator och detta måste tas i beaktning när applikationen utvecklas.

3 Grundidéer bakom GUI design för handdatorer

I föregående kapitel behandlades handdatorer, detta för att ge läsaren en översiktlig bild av handdatorer. I nästa kapitel skall de APIer som har tagits med i uppsatsen beskrivas. Innan dessa presenteras så bör läsaren få grundläggande kunskaper inom design av grafiska användargränssnitt. I detta kapitel så beskrivs riktlinjer som designers bör ha till sitt förfogande då de skall designa ett grafiskt användargränssnitt. Innan man börjar med själva designen är det viktigt att gå igenom de riktlinjer som finns, då detta kan vara en god hjälp till en väldesignad applikation. Först förklaras innebörden av vad ett grafiskt användargränssnitt är. Kapitlet fortsätter sedan med att ge en kort introduktion till varför man har grafiska användargränssnitt. Därefter så presenteras riktlinjer för design av grafiska användargränssnitt, här tas både generella och handdatorspecifika riktlinjer upp.

3.1 Översikt

Ett grafiskt användargränssnitt är det som finns mellan användaren av ett program och programmet som används. Ett grafiskt användargränssnitt kan ses som en förmedlare som gör att programmet och användaren lätt kan förstå varandra.

3.2 Bakgrund

I början av datorns utveckling så fanns inga grafiska användargränssnitt där användare kunde trycka på en knapp så att ett kommando utfördes. Då fick man skriva in sina kommandon i en kommandotolk i till exempel DOS. Denna typ av användargränssnitt kallades för CLI.

Om man till exempel ville se vilka filer som fanns i en specifik mapp så fick man skriva kommandot **dir** i DOS. Det finns även kommandon som är mycket mer komplicerade, detta var dock inget problem eftersom datorn endast användes av personer som kunde hantera dem. På sikt så behövde datorn göras användarvänlig även för användare som inte kunde

hantera dessa kommandon. Man behövde helt enkelt skapa visuella knappar som skulle ersätta dessa kommandon, därav namnet **grafiskt** användargränssnitt. Ett bra grafiskt användargränssnitt behöver dock vara väldesignat. Knappar och andra grafiska objekt får inte ligga vart som helst. Knapparna bör ha bilder eller text som ska vara lätt att förstå.

Ett företag som tillverkar ett dåligt grafiskt användargränssnitt skulle förlora på det. Användare som inte förstår det grafiska användargränssnittet skulle lätt kunna byta till en annan applikation som har ett bättre grafiskt användargränssnitt tillverkat av ett annat företag.

3.3 Riktlinjer för GUI design

I detta delkapitel kommer riktlinjer för GUI design att behandlas. Först så förklaras vad GUI riktlinjer är för något, när ska dessa följas och när de inte ska följas. Vidare behandlas några generella riktlinjer som kan vara bra att ta till sig då man designar ett GUI.

3.3.1 GUI design

Vid design av GUI så använder man sig av riktlinjer som finns framtagna för att kunna underlätta designen av ett grafiskt användargränssnitt. I detta delkapitel så behandlas de generella riktlinjerna som finns för GUI design.

Vad är GUI riktlinjer GUI riktlinjer är en samling av rekommendationer som GUI designers följer då de skapar ett GUI. Enligt [10] kan dessa riktlinjer vara:

1. Generella riktlinjer som har tagits fram genom studier.
   Dessa kan ofta vara väldigt viktiga eftersom de har tagits fram genom studier där man har lagt ut mycket tid på att testa olika perspektiv. Studierna kan vara sådana där man har testat människans beteende i olika miljöer osv.

2. Designstandarder som har utvecklats inom företaget.

   Många företag har sina egna standarder som de följer vid design av GUI. Det kan var så att de inte följer de generella standarderna som finns helt och hållet, utan har modifierat dessa så att det passar dem på ett bättre sätt.

Dessa är inte de enda exemplen som finns men här är det meningen att bara ge en förståelse för vad GUI riktlinjer kan vara.

Varför skall man följa dessa riktlinjer? Många företag har tagit fram sina egna riktlinjer för design av GUI, detta på grund av att applikationer som utvecklas hos dem skall ha ett liknande grafiskt användargränssnitt. Genom att man håller sig till dessa riktlinjer vid designen av GUI till en viss applikation så kommer det medföra att alla applikationer utvecklade hos företaget kommer att ha ett liknande grafiskt användargränssnitt. Detta kommer i sin tur medföra att användarna av företagets applikationer inte kommer att ha problem med att lära sig hur applikationen fungerar, detta är ju syftet med grafiska användargränssnitt. Detta medför även att själva designen av GUIt inte kommer att ta stor del av utvecklingsfasen av en applikation, då man slipper tänka på hur den grafiska utformningen skall se ut. Riktlinjer är inte framtagna så att de ska följas helt strikt utan kan ibland brytas då man finner en bra grund för detta. [10] Sådant behandlas lite kort nedan.

När ska man inte följa riktlinjerna? Det finns fall då det kan vara så att dessa riktlinjer inte är lämpliga att följa. Riktlinjer är ofta för generella och därför kanske inte är lämpliga i vissa fall. Dessa riktlinjer kanske inte specificerar exakt hur du ska göra, det kan till exempel uppstå problem då du ska ge användaren en valmöjlighet och du har två alternativ att tillgå. Du som designer vet då inte exakt vilken riktlinje du ska tillämpa. Riktlinjer kan även bli för specifika och måste då brytas. Till exempel så kan någon riktlinje som är väldigt specifik säga att man inte får ha mer än sex stycken poster i en meny. Just

i detta fall så kan det ju vara så att man måste lägga till en extra post, därför så kommer man då att skapa en meny till för att kunna lägga till denna post. Mera menyer ser ju dock till att GUIt blir mera komplicerat. I sådana fall kan det därför vara lämpligt att strunta i regeln och lägga till denna post i menyn med sex poster. Det kan uppstå flera sådana situationer där man måste bryta dessa riktlinjer, men det är även därför de inte är någon lag som man måste följa. [10]

3.3.2 Generella Riktlinjer för GUI design

Här så ska det nu presenteras ett antal riktlinjer som man kan följa då man ska designa ett GUI till sin applikation. Dessa riktlinjer är inte specifika för stationära datorer eller handdatorer utan kan tillämpas på båda. De nedanstående riktlinjerna är en sammanställning av de riktlinjer som finns i [32].

- Kontroll av miljön

  En användare som inte kan kontrollera sin omgivning blir inte glad. Att användare har kontrollen över sin omgivning innebär att applikationen skall svara tillbaka på ett korrekt sätt då användaren har utfört till exempel någon knapptryckning. Med korrekt sätt så menas det sätt som användaren väntade sig att svaret skulle vara.

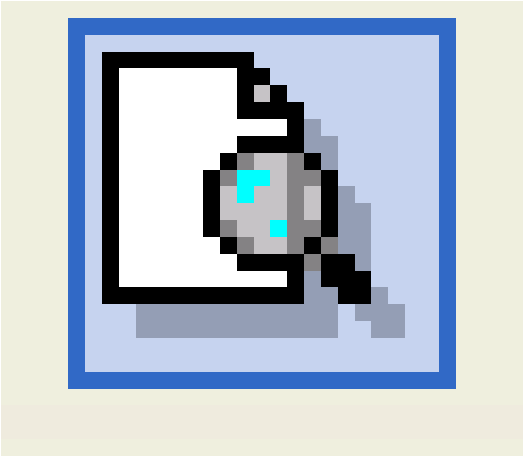
- Användarval i en applikation

  Alla användarval som behöver göras i en applikation skall hållas till ett minimum. Innan en applikation lanseras bör alla situationer som kan uppstå i applikatione vara såpass utredda (och omhändertagna av applikationen) att de få val som är kvar är absolut nödvändiga. Valen som behöver göras skall presenteras på ett informativt sätt för användaren. Detta för att användare inte skall känna sig överväldigad eller bortkommen av alla val som kan göras, vilket i sin tur leder till att de slutar använda applikationen. Denna riktlinje kommer av tanken bakom handdatorer; de skall var lätta att använda för alla. Applikationer utvecklas efter en noggrann förstudie av mål-

gruppen för applikationen. Applikationen skall täcker ett behov som finns samtidigt som den göra allt för hålla det så enkelt som möjligt för användaren.

- Metaforer

  Metafor kommer från grekiskan *metaphora* och innebär att man uttrycker sig på ett bildligt sätt. I GUI sammanhang görs detta genom avbildning av vardagliga funktioner. En bild på en papperskorg markerar att användare kan kasta något i den. Att använda så många metaforer som möjligt ger ett bra användargränssnitt. I ett program där man har ett förstoringsglas förstår man direkt hur man ska förstora dokumentet. Men det är även så att metaforer inte alltid fungerar. Detta kan bero på att man har valt någon form av dålig metafor. Istället för att ha en metafor som ger användaren fel intryck så ska man helst undvika denna metafor. En metafor som inte ger en verklighetsbild kommer endast att förvirra användaren, vilket kommer att medföra att användaren gör fel. Exempel på en metafor som kan orsaka förvirring kan vara förstoringsglaset i Windows applikationer se figur 3.1.



Figur 3.1: Förstoringsglas i Windows applikationer

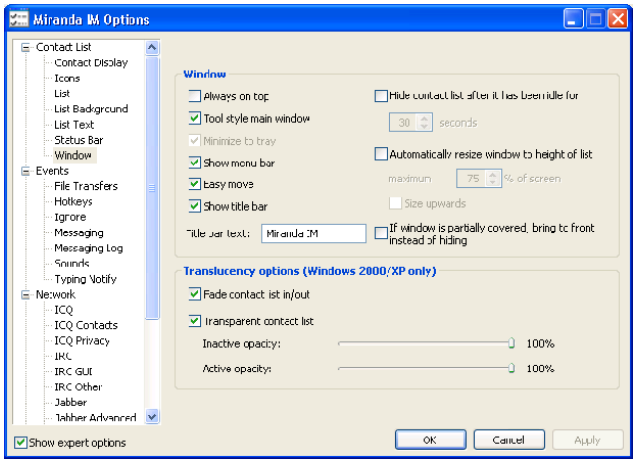
- Affordance

  Affordance innebär att den grafiska designen inbjuder användaren till att utför en

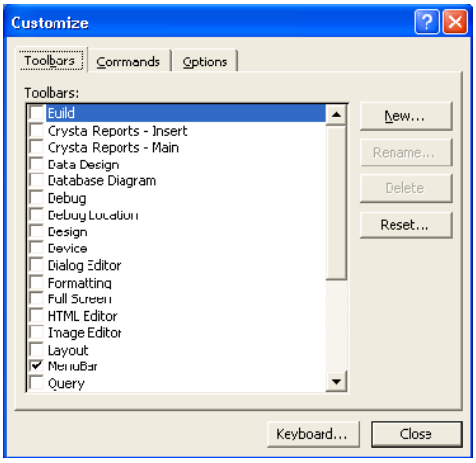
handling, till exempel trycka på en knapp. Detta kan göras genom att GUIt försöker avbilda mekanismer som finns i verkligheten, i fallet med en tryckbar knappar så görs det lättast genom att lägga skuggor runt knappen. Skuggorna gör att knappen ser tre dimensionell ut och därmed liknar en verklig tryckbar knapp.

- Tabbade Dialoger

  Ett problem som kan uppstå är då man har dynamiska dialoger, som i vissa fall så har man dialoger som en typ av lista bredvid fönstret där man gör sina val. En potentiell användare av din applikation kan vara sådan som inte har någon aning om att man faktiskt kan välja från listan bredvid och att på detta sätt så kommer det att dyka upp ett nytt fönster. Hade man istället haft tabbade dialoger så behöver inte detta problem uppstå, eftersom tabbarna längs upp i menyn är väl formade och man kan lätt se att dessa är valbara. Även om man har tabbar så kan användaren ändå bli förvirrad. Detta är i fallet då man har flera rader av tabbar. Problemet uppstår då man trycker på någon av radernas tabbar så kommer denna rad att automatiskt hamna på första raden det är just detta som kan förvirra användaren. Detta kan du som programmerare lösa genom att man har en scroll bar som visar att det finns fler tabbar att välja mellan, istället för att ha flera rader av tabbar. Se figurer 3.2, 3.3 och 3.4.



Figur 3.2: Dialogruta utan tabbar, användaren kan missförstå.



Figur 3.3: Dialogruta med tabbar.

Figur 3.4: Dialogruta med tabbar i en Symbian OS baserad handdator.

- Konsekvent design

  Med konsekvens i grafiska användargränssnitt menas att de designers som gör ett GUI använder sig av de vedertagna riktlinjer för grafisk design som finns. Detta medför att användare känner igen (från andra applikationer) de funktioner och ikoner som finns i och inlärningskurvan av applikationen blir lägre. [Palm, Symbian, Windows CE].

- Skapa design för extremfall

  Den grafiska designen av applikationen bör vara inriktad mot att fungera även under (för designen) ovanliga förutsättningar. Denna design riktlinje medför ofta att designen blir mycket lättanvänd under (för designen) normala förhållanden. Ett exempel kan vara att skapa design som möjliggör inmatning för personer som har begränsad finmotorik, till exempel på grund av skyddsutrustning eller sjukdom.

- Text i dialogrutor

  Det finns två viktiga aspekter vad det gäller utformningen av text dialogrutor. Den först aspekten är att de flesta användare har en tendens att ignorera (helt eller delvis)

dialogrutor som innehåller mycket information. Det är därför oftast rekommenderat att använda korta men koncisa meningar i dialogrutor. Den andra aspekten innebär att språket i dialogrutor inte skall uppfattas som skrämmande för användare. Detta leder till obehag för användare något som kan göra att hon byter till en annan produkt. Ett vanligt exempel på detta är frågan som ofta ställs innan ett program avslutas 'Är verkligen du säker på att du vill avsluta?' en bättre fras anser vi vara 'Avsluta nu?'.

3.3.3 Handdatorspecifika riktlinjer för GUI design

Som det tidigare nämnts i uppsatsen så har handdatorer stora begränsningar jämfört den traditionella PCn[4]. Dessa begränsningar ger upphov till att designers som skapar GUI till handdatorer inte har samma möjligheter som hon har vid skapandet av ett GUI för en traditionell PC. Detta främst med avseende på storleken av handdatorns skärm för visning av information. Andra begränsade aspekter finns såsom inmatnings möjligheter.
Handdatorer används av en stor grupp människor många vars kunskaper om datoranvändning varierar. Det är därför av största vikta att ett GUI utformas så att det passar alla användare oberoende av tidigare erfarenheter. Exempel på detta kan vara att avancerade inställningar inte visas för användaren då dessa skulle kunna förvirra användare.
Uppsatsen kommer nu att behandla de riktlinjer som inriktar sig mot design av grafiska användargränssnitt till handdatorer.

Visuella uppfattningsförmågan Att ta hänsyn till människans visuella uppfattningsförmågan är ett måste då designern skall designa ett bra GUI. En designer kan utnyttja uppfattningsförmågan genom att placera grafiska element i GUIt på platser där användaren lättare uppfattar dem (tex i hörnen av skärmen). Ett annat sätt är att animera informationen eller att visa ett grafiskt objekt större än de kringliggande. Detta görs för

[4]PC (eng, Personal Computer) används för att betekna den datorarkitektur som IBM introducerade 1981 och som nu är den typ av dator som många dator användar som hemdator.

att visa användaren den viktigaste informationen snabbt.
Områden på skärmen som ögonen har en tendens att dra sig till för att vila är också viktiga att tänka på, här ges några exempel på sådana områden:

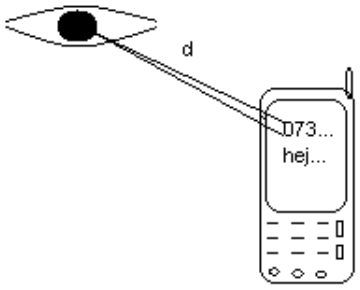
- Områden utan information
- Horisontella linjer
- Platser med mjuka färger

Detta kan vara bra att tänka på då designers ska skapa ett GUI. Viktig information kan exempelvis placeras i hörnen av skärmen dit ögonen har en tendens att uppmärksamma information snabbt. Ett annat exempel kan vara att ha områden som är tomma på information så att användaren får en möjlighet att vila ögonen. [28]

Riktlinjer vid GUI design till handdatorer I uppsatsen beskrivs nu de riktlinjer som rekommenderas av de företag som vi studerat (Symbian, Microsoft och PalmOS). Dessa företag använder sig av i princip samma riktlinjer.

- Text

  Texten på en handdator skärm skall vara stor nog att läsas av alla användare men tillräckligt liten så att all väsentlig information kan presenteras på skärmen samtidigt. För att beräkna storleken på texten kan formler såsom denna användas: rekommenderade textstorleken (h) för ett givet läsavstånd (d) skall vara h = 0.0052*d (avståndet givet i centimeter) se figur 3.5 för exempel. [28]



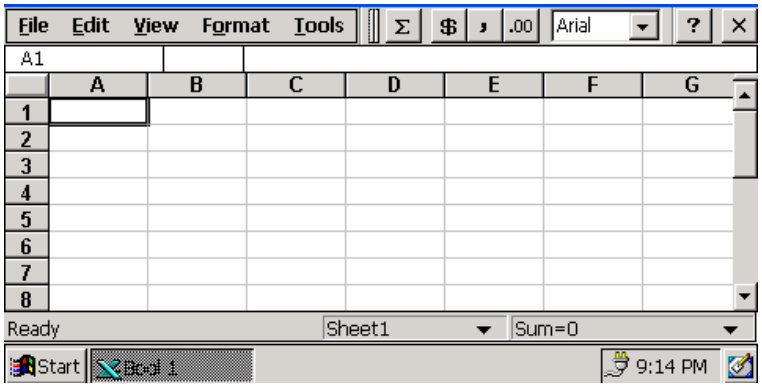
Figur 3.5: Storleken på texten beroende på avståndet

- Färg

  Alla handdatorer har inte stöd för visning av färg varför det kan vara olämpligt att använda färg för att visa information i GUIt. Designen bör oftast göras på ett sätt som att det endast finns en svart-vit skärm till förfogande. Då designen av GUIt är klart kan färger tillämpas. De flesta GUI APIer tillhandahåller funktioner för att testa om den givna enheten stödjer visning av färger. Att kontrollera om färger stöds är starkt rekommenderat. En viktig aspekt är att ungefär 8% av den manliga och 0.5% av den kvinnliga populationen i världen är färgblinda. [28]

- 80/20 regeln

  Denna regel säger att då ca 80% av alla funktioner i en PC applikation nästan aldrig används så skall dessa funktioner inte inkluderas när en liknande applikation skall skapas på en handddator, utan bara de ca 20% som används. Denna lösningen bidrar till att spara skärmyta men även göra applikationerna så lätt att använda som möjligt.I figur 3.6 visas hur Microsoft Excel ser ut i Windows CE. [3]



Figur 3.6: Excel som körs på en Windows CE emulator

- Snabb åtkomst till funktioner

  Desto oftare en funktion används desto färre inmatningar bör krävas för att komma åt funktionen. Ett exempel på detta är att åtkomsten till kalendern på en SonyEricsson P800 telefon kräver ca tre inmatningar. Detta jämfört med att ändra språk inställningarna på samma telefon, vilket kräver fem till sex inmatningar.[3]

- Text till knappar och menyer

  Grundregeln här är att använda så kort text som möjligt, använd till exempel & istället för 'and'.För menyer så kan separatorer användas för att ha liknande menykommandon tillsammans. Dessa förkortningar bidrar till att skapa ett överskådligt GUI på att en handdator vars skärmstorlek är starkt begränsad.[1]

- Design för korta sessioner

  Användningssessioner för en handdator är korta men frekvent återkommande. Detta jämfört med en vanlig PC där en användningssession är jämförelsevis lång och därför inte har samma frekvens per dag. PalmOS har gjort en undersökning om detta som redovisas i figur 2.2. Korta session innebär att en handdator applikation måste ge användaren den information som efterfrågas snabbt, då en väntan på 15 sekunder är

tydlig på en handdator men inte på en PC. Detta kan göras genom att applikationen försöka förutse vad en användare vill göra. [3]

- Avmarkera omöjliga val

  Att avmarkera omöjliga val är en teknik som går ut på att icke valbara menyval skrivs i grå text (istället för svart). Detta för att visa för användaren att valet finns men inte är valbart. Ett exempel på en situation där detta bör göras är när användaren försöker kopiera en text utan att först ha markerat någon text. Omöjliga menyval bör inte tas bort från menyn då detta lätt skapar förvirring för användaren då samma meny kan innehålla olika val vid olika tidpunkter i programmet. En annan fördel med att avmarkera menyval i användargränssnitt är att man i förväg vet att menyn inte blir för stor för skärmen. [1]

- Användaren skall inte känna sig dum

  Designen av ett GUI skall vara sådant att alla användare ska kunna använda de applikationerna som finns. Om en användare misslyckas med att utföra den uppgift som hon vill utföra, eller gör upprepade fel på grund av den information som hon får av det grafiska användagränssnittet, så kommer hon med största sannolikhet att sluta använda applikationen. [32]

- Förutsägbarthet

  Med förutsägbarhet i GUI design menas att användaren skall med information från GUIt kunna förutse vad som händer när ett val i applikatione utförs. Exempel på detta är att funktionen spara i program ofta visualiseras i form av en diskett. Om användaren inte kan förutsäga hur programmet reagerar så kan detta skapa irritation. [3]

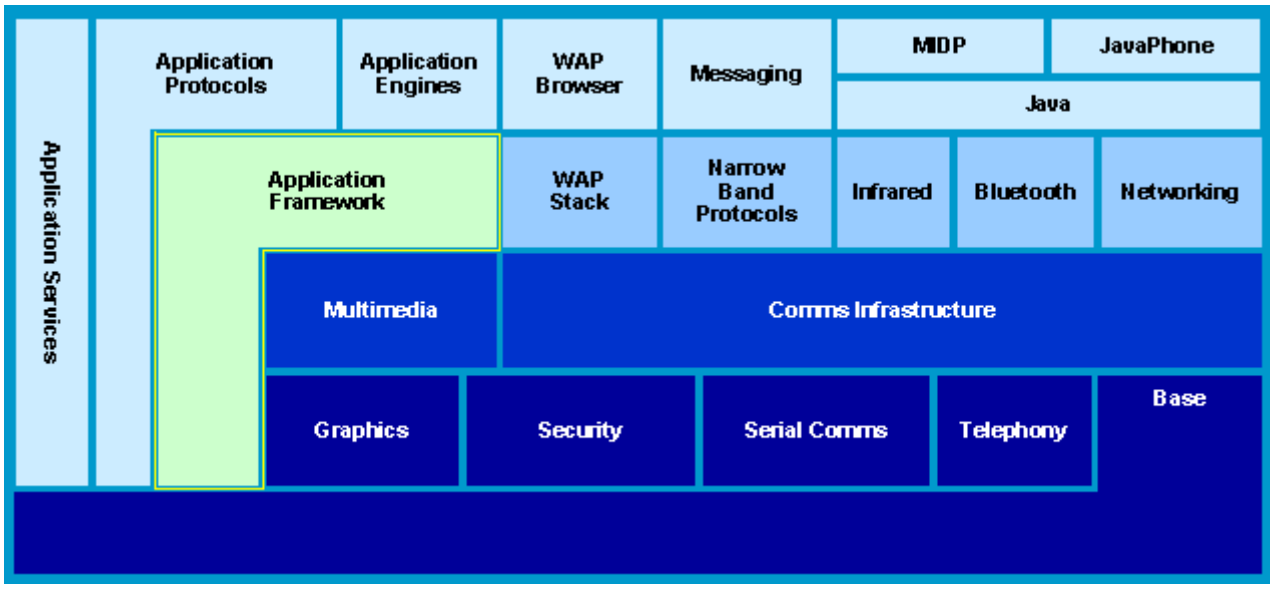
4 Introduktion till API

I föregående kapitel har uppsatsen behandlat handdatorer och GUI riktlinjer. Detta för att ge läsaren en överblick av dessa begrepp. I detta kapitel kommer uppsatsen att behandla de fyra APIer som vi uppsatsförfattarna funnit lämpliga att utvärdera. Varför dessa APIer valdes beskrivs i kapitel 5.1. Kapitlet ger en övergripande genomgång av de APIer som undersökts och hur deras olika delar interagerar med varandra och operativsystemet på respektive plattform.

4.1 Symbian OS

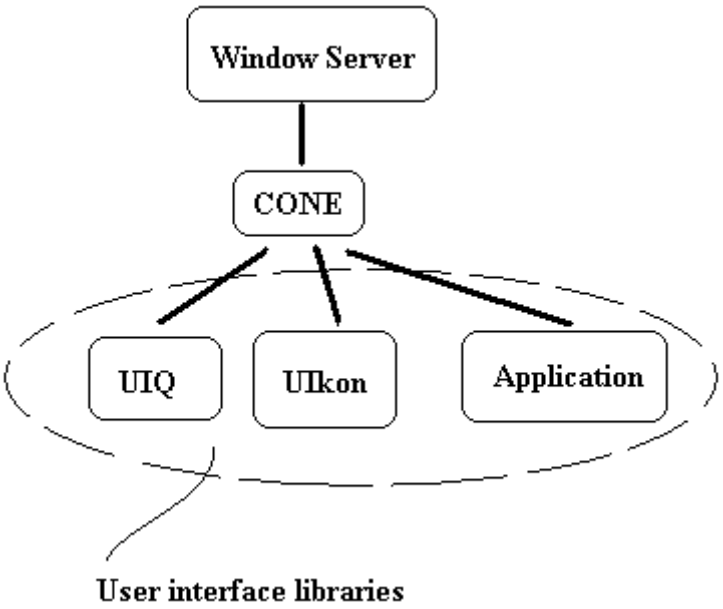
Översikt Symbian ger intryck av att vara ett genomarbetat operativsystem. Det implementerar och presenterar flera bra programmeringsparadigmer som till exempel MVC[5]. Stor vikt läggs på att programmering skall ske objektorienterat. Det som dock förefaller vara det viktigast vid programmering är deras vilja att spara och ta tillvara på de resurser som finns tillhands, även om händelsehantering och felhantering förefaller vara mycket viktiga designdelar.[27]

[5]Mer information om MVC, läsa på http://www.cs.kau.se/ hannes/gui/resource/Model-View-Controller.pdf.



Figur 4.1: Strukturen på var grafik APIt ligger i OSet

Symbians grafikhantering Likt de flesta andra operativsystem är Symbian OS uppdelat i moduler, som kan ses i figur 4.1. Under Application Framework hanterar Symbian en mängd grafiska funktioner såsom Windows Server, CONE och UIkon men även andra funktioner såsom ljud med mera. [1]



Figur 4.2: Ui control framework

Windows Server Symbian använder sig av en Windows server vilken exekverar som en separat applikation till vilken applikationer får registrera sig för att få tillgång till skärmen. Detta ses som en server-klient förhållande. Då flera applikationer kan exekveras samtidigt så har varje applikation ett statiskt tilldelat unikt ID som Windows server använder sig av för att kunna hantera händelser mellan applikationen och skärmen. Windows server tillhandahåller även vissa specialeffekter som applikationer kan använda sig av (skuggor, animation, scrollningshantering med mera). Dessa operationer varierar dock beroende på vilka grafiska ramverk som används, UIkon, UIQ med flera.
Windows server tillhandahåller dessutom vissa grundläggande APIer för de interaktionsmöjligheter som finns (tex skärm, pekpinne, tangentbord). Dessa APIer är dock svåra att hantera varför de flesta applikationer använder sig av CONEs API. Se figur 4.2 för en schematisk bild på hur de olika delarna förhåller sig till varandra.[27] [13]

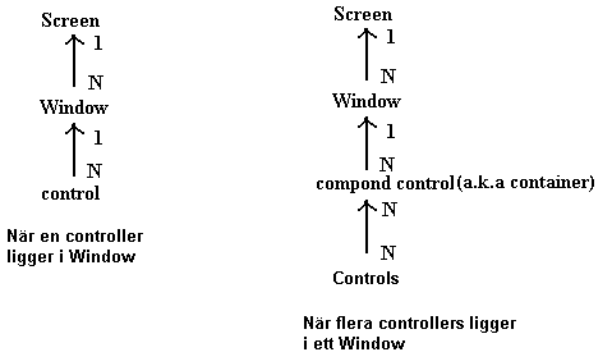
UI appliktion ramverk UI Applikation ramverk är ett nytt namn för CONE[18]. Eftersom Windows server har mycket generella och komplexa APIn så fungerar CONE som ett skal. CONE döljer mycket av komplexiteten och tar därmed bort en del av generaliteten för applikationsprogrammeraren, då CONE i sig själv gör designbeslut mot fönsterservern. Den kanske främsta målet för det grafiska ramverket hos Symbian är att skapa en infrastuktur för applikationer att interagera med varandra och användare. Med hjälp av denna infrastruktur skapas ett flexibelt systemet för grafiska applikationer. Det skapar även ett gemensam ramverk för hur alla UI bibliotek bör se ut och gör detta utan att påtvinga ett speciell LAF (Look-and-Feel) för applikationer. Detta medför bland annat att portningar mellan olika grafiska ramverk i Symbian OS till exempel Uikon, UIQ lätt kan göras. De funktioner som ramverket tillhandahåller är:

- Händelsedrivet GUI ramverk
- Fönstersystem för att dela skärmen, tangentbord och andra sorters inmatningar, mellan applikationer.
- Direct Navigation Link (DNL) som ger applikationer möjligt att interager med andra applikationer.
- Mekanismer för att ändra och hanter LAF för en användare, kan tex vara beroende av organisation eller den uppgift som apparaten skall lösa).
- Möjlighet att känna igen skriven text
- Multipla semitransparanta fönster

CONE ger även tillgång till vissa primitiva rittyper, såsom streck, pennor med mera från skärmservern. En appliktion behöver inte använda sig av CONE, men om applikationen inte gör det så ökar komplexiteten för applikationen. [17]

UI bibliotek UI biblioteken använder CONE för att underlätta kommunikationen med Window servern, den kanske vanligaste funktionen är att skapa så kallade kontroller. Vid grafiska operationer så används ofta de konkreta och för CONE underliggande APIerna, såsom UIQ eller UIkon för att utföra operationer som kommando hantering, filhantering med mera. [19]

Kontroller All grafik som ritas ut på skärmen ritas ut på så kallade kontroller. En kontroll är en rektangulär yta på skärmen som kan uppta hela eller delar av den. Applikationer får tillgång till kontroller genom CONE eller UI biblioteken som till exempel UIkon. En kontroll kan bestå av grafiska figurer, exempelvis en cirkel, men kan även innehålla ytterligare kontroller varvid de kallas för 'compound controls'. Andra kontroller kan exempelvis vara container dessa kan innehålla ritobjekt som till exempel knappar. Generellt sätt så kan man säga att alla UI element är kontroller även om det finns vissa undantag. Förutom att hantera grafik så hanterar en kontroll även användarrespons, som till exempel knapptryckning i en kontroll. [19] Figuren 4.3 visar en schematisk bild på hur kontroller förhåller sig till skärmen och Windows servern.

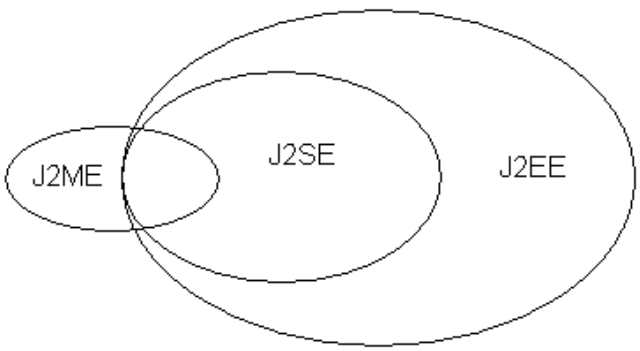


Figur 4.3: Skärm,window,control,container,compound struktur

UIkon Uikon är det grafiska UI API som alltid finns tillgängligt i Symbian OS. Detta UI API kan kompletteras av andra UI APIn såsom UIQ. UI APIt ger stöd för en mängd grafikfunktioner som till exempel dialog rutor, rull-listor, bilder, knappar med mera. [20] Det finns även stöd för att skapa egna typer av grafiska funktioner.

4.2 SUN J2ME

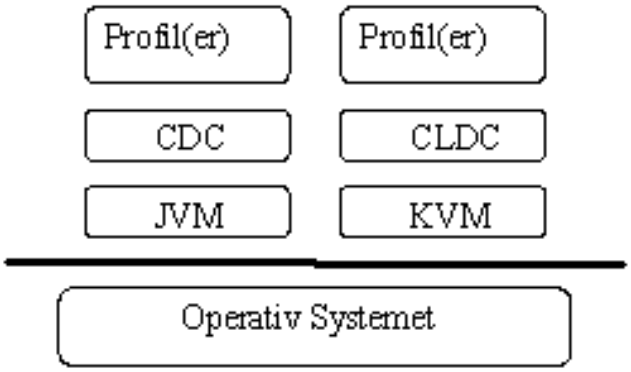
Översikt Genom tiderna har SUN utvecklat ett antal java platformar. Efter att ha utvecklat en del java versioner så bytte man namn till Java 2, denna version i sig släpptes i tre utgåvor. Detta gjordes på grund av att de olika utgåvorna skulle kunna passa i olika miljöer. De tre olika utgåvorna är mer kända som J2SE (Java 2 Standard Edition), J2EE (Java 2 Enterprise Edition) och J2ME (Java 2 Micro Edition). [9]
I denna uppsats behandlas J2ME. Figur 4.4 visar hur dessa tre utgåvor hänger ihop.



Figur 4.4: Sambanden mellan SUN's Java 2 utgåvor

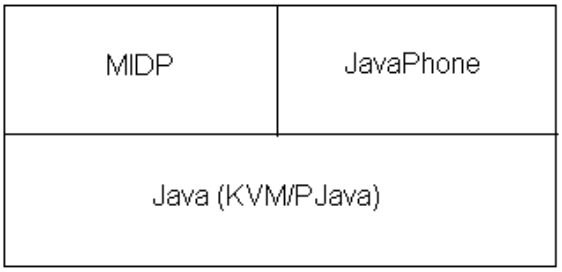
J2ME utgåvan ses egentligen inte som en enda plattform utan som en mängd av plattformar där de olika plattformarna riktar in sig mot olika typer av apparater, med det gemensamma att de riktar in sig mot apparater som har någon typ av fysisk begränsning.

De olika J2ME plattformarna består av en konfiguration och en profil. Mer om detta senare. I figur 4.5 kan man se arkitekturen för hur detta är uppbyggt. [9]



Figur 4.5: SUN's J2ME konfiguration

Värt att nämna är att innan J2ME introducerades på marknaden så utvecklade SUN en annan plattform för mobila enheter. Plattformen som utvecklades kallas för PersonalJava och bygger på SUNs JDK 1.1. PersonalJava plattformen kan byggas ut ytterligare med JavaPhone APIt. I mobila enheter som bygger på Symbian OSet finns det till exempel möjligheter att använda sig av både detta API och J2ME APIt. Dock så används de inte i samma applikation. [6] [33] I figur 4.6 visas en konfiguration som använder både J2ME och PersonalJava.



Figur 4.6: Java konfiguration i Symbian OS

J2ME Konfigurationer I nuläget finns det två konfigurationer, CDC (Connected Device Configuration) och CLDC (Connected Limited Device Configuration). CDCn lämpar sig mer för apparater som har lite större mängd minne. Sådana kan till exempel vara high-end-handhelds, navigationssystem i bilar och så vidare. CLDCn lämpar sig däremot för enklare mobiltelefoner, low-end-handhelds och så vidare, helt enkelt apparater som är ännu mer begränsade. Konfigurationerna i J2ME definierar det minsta kravet för en J2ME plattform. Kraven som definieras av en konfiguration kan bland annat vara sådant som tillgängligt minne i apparaten, vad det finns för möjligheter för inmatning/utmatning och vilka klasser som existerar.[33] [31]

J2ME Profiler Varje konfiguration (CDC, CLDC) i J2ME har ett antal profiler som ligger ovanför själva konfigurationen och på så sätt utökas de konfigurationerna som finns. Med hjälp av de olika profilerna kan man på så sätt definiera en Java plattform för en specifik familj av apparater. Till exempel så kan mobiltelefoner definieras av ett antal profiler på samma sätt kan tvättmaskiner definieras av ett antal andra profiler. Ett begränsat antal profiler finns idag definierade och några exempel på sådana är: [33][31]

- MIDP (Mobile Information Device Profile)

- PDAP (Personal Digital Assistant Profile)
- PBP (Personal Basis Profile)

GUI i J2ME Profiler utvecklas konstant och mer information om dessa finns att tillgå på *www.jcp.org* (Java Community Process). I uppsatsen kommer nu MIDP profilens API för grafiska användargränssnitt att behandlas. MIDP profilen i J2ME definierar ett antal klasser för uppbyggnad och manipulation av ett grafiskt användargränssnitt.
Dessa klasser är uppdelade i två delar vilka kallas för low-level UI API och high-level UI API. Det finns dock ett tredje API som egentligen inte har definierats av MIDP, enligt [9] så kallas denna för Common UI API. I detta API finns klasser som används av både low-level UI API och high-level UI API. Exempel på en sådan klass är Command klassen.

Low-level API Detta API är designat för applikationer som behöver precis placering och kontroll av de grafiska elementen. Med hjälp av detta API så kan man till exempel skapa en applikation som kan:

- Kontrollera vad som ritas på skärmen
- Hantera primitiva händelser, så som knapptryckningar
- Få tillgång till konkreta knappar och andra inmatningsapparater

Problemet med denna low-level API är att portabiliteten hos applikationerna kan bli lidande. Detta kan dock undvikas genom att applikationen endast använder den del av APIt som är plattformsoberoende. Man ska alltså helt enkelt inte ta för givet att alla apparater har knappar så som siffror 1-9, '*' och '#', till exempel så har inte alla handdatorer tillgång till sådant.

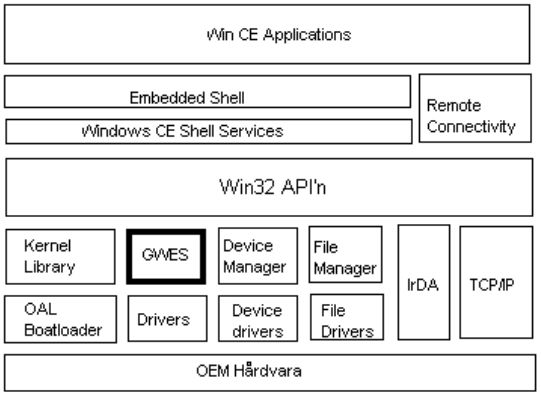
High-level API Detta API är mest lämpad att användas i så kallade business applikationer. APIt ger inte någon som helst kontroll över vad som ritas på skärmen, dock så får

man en hög grad av portabilitet. Visningen av GUI komponenter sköts i detta fall istället av själva apparaten som applikationen körs på. Applikationen bestämmer helt enkelt inte utseendet av dess komponenter.

I en MIDP applikation är det fullt möjligt att använda sig av båda APIerna. Dock är det inte så att man kan använda dem i ett och samma fönster. Om man till exempel utvecklar ett spel så finns det möjligheter att använda high-level APIt då användaren skall få göra val som antal spelare och svårighetsgrad, därefter så använder man low-level APIt då man ritar upp spelet på skärmen. Dessa två skall dock användas på separata fönster. Det skall påpekas att man inte kan bestämma utseendet på de grafiska komponenterna. Det finns inte heller någon möjlighet att bestämma vart de olika knapparna hamnar på skärmen utan det sköts helt och hållet av själva apparaten som applikationen körs på. Det finns till exempel inga möjligheter att skapa menyer och på så sätt bestämma att vissa funktioner skall hamna i denna. [31] [33]

4.3 Windows CE

Vissa människor tror att Windows CE är en produkt som portades från Windows 95. Detta beror på att den första produkten som använde Windows CE som operativsystem hade ett grafiskt användargränssnitt likt det i Windows 95. Det skall därför påpekas att Windows CE är ett unikt operativsystem, den har sin egen kodbas, skriven från grunden och den är speciellt designad för inbyggda system.[30]



Figur 4.7: Windows CE systemarkitektur

Windows CE är ett 32-bitars skalbart operativsystem, designat för apparater med begränsad funktionalitet. På grund av detta har man byggt upp Windows CE med en mängd diskreta moduler, som i sin tur kan innehålla ett antal valbara komponenter. På detta sätt kan tillverkare av handdatorer eller liknande apparater kontrollera storleken av operativsystemet på sina apparater. [11][5]
Genom att välja en minimal mängd av moduler och komponenter så kan man alltså minimera hårdvaruresurserna.

De moduler som inkluderar en eller flera komponenter som inte är obligatoriska i konfigurationen kallas för komponentmoduler. Till exempel så är Coredll och GWES komponentmoduler. Dessa moduler har alltså en eller flera komponenter som till exempel handdatortillverkare inte behöver ha med i deras konfigurationer av Windows CE. De komponenter i Coredll som inte är obligatoriska är sådana komponenter som stödjer telefoni, multimedia

och GDI (Graphical Device Interface) grafik. [4]
I uppsatsen kommer nu GWES (Graphics, Windowing and Event Components System) modulen att behandlas lite kort eftersom det är här som all GUI funktionalitet stöds. I figur 4.7 ser man vart denna modul förbehåller sig i operativsystemet.

GWES I Windows CE har User och GDI subsytemen från Win32 APIt kombinerats in i en enda komponentmodul, GWES. Händelsehanteraren och fönsterhanteraren är likartade som de i Win32. GDI subsystemet är däremot ersatt med ett mindre multi-platform GDI (MGDI) detta för att passa för mindre och begränsade apparater. GWES modulen har hand om användargränssnittet och fönsterfinesserna i Windows CE. GWES är även ansvarig för visning av grafik och att utföra avancerade grafiska manipulationer. GWES komponenterna kan organiseras på följande sätt. [4]

- Grundläggande GWES komponenter (Krav)
- Meddelandekökomponenter (Krav)
- GDI komponenter (Valbart)
- Fönsterhanteringskomponenter (Valbart)
- Dialoger (Valbart)
- Kontroller (Valbart)
- Font support (Valbart)
- Användarinmatningskomponenter (Valbart)
- Power-management komponenter (Valbart)

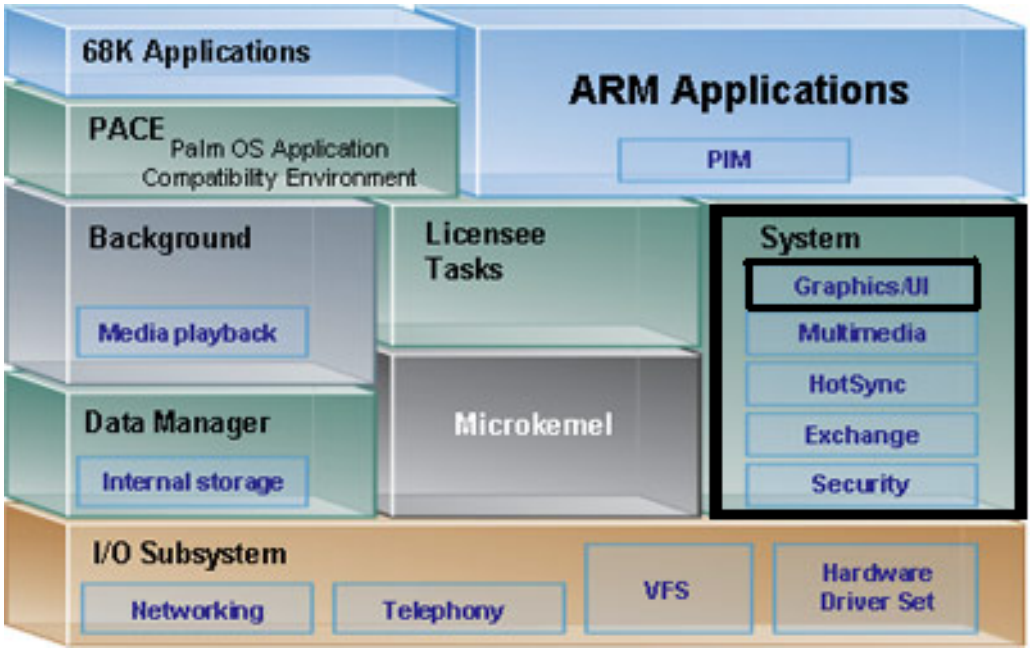
Genom GWES APIt tillhandahålls möjligheter att skapa fönster, dialog boxar, kontroller, menyer, och andra grafiska komponenter. Genom GWES tillhandahålls även andra resurser

så som ikoner, text och annat. Även om de flesta komponenterna i GWES är valbara så är vissa komponenter obligatoriska. De apparater som inte har något fönster där de kan visa ett grafiskt användargränssnitt måste ändå implementera vissa grundläggande GWES komponenter. [8]

Windows CE tillhandahåller ett antal förtestade komponentkonfigurationer som tillverkare kan välja bland. Dessa kan delas in i tre kategorier, minimal-featured, moderately-featured och full-featured. Den minimala konfigurationen som Microsoft tillhandahåller visar inget användargränssnitt. Den mellersta konfigurationen moderately-featured har stöd för ett antal av GWES komponenterna men inte alla. Full-featured konfigurationen har däremot fullt stöd för alla moduler och dess komponenter. [7]

4.4 Palm OS

Palm OS är en av de största inom handdatormarknaden. De har tillverkat ett antal versioner av Palm OS operativsystemet och till en början så tillverkade de även sina egna handdatorer. Fram till och med Palm OS version 4 har de använt Motorolas MC68000 chip som processor. Då denna processor var långsam så har de sedan version 5 bytt till ARM baserade processorer och då förändrades även arkitekturen för operativsystemet något. [12] I figur 4.8 visas en figur på operativsystemets arkitektur.



Figur 4.8: Palm OS Cobalt (Palm OS 6) system arkitektur

De nya versionerna sedan Palm OS 5, är bakåtkompatibla och det är alltså fullt möjligt att så kallade 68K applikationer kan köras på det nya OSet. 68K applikationer kan köras på det nya OSet genom att man använder något som kallas för PACE (Palm Application Compatibility Environment). PACE simulerar inte på något sätt ett gammalt operativ system eller en 68K processor utan den översätter 68K instruktionerna till ARM instruktioner.
Nu kommer uppsatsen att fortsätta med att behandla det grafiska APIt i Palm OS.
Som man ser i figur 4.8 så består Palm OS av en kärna, som är hjärtat i operativsystemet, resterande delen av Palm OS är organiserat i ett antal moduler som kallas för managers. En manager kontrollerar en specifik del av apparaten som Palm OS exekverar på. De vanligaste managers är de för användargränssnittet, för systemet, för minneshanteringen och för kommunikationen. Varje manager tillhandahåller en mängd av APIer. [12]

I System blocket i Palm OS så ingår ett antal moduler eller managers så som multimedia,

security och så vidare. I detta block så finns även moduler för grafik och användarsgränssnitt. De managers som har hand om grafik och användargränssnitt är window manager respektive form manager. Det är window manager som tillgodoser med APIt så att man bland annat kan rita linjer, rektanglar, bitmaps och så vidare.
Form manager däremot tillgodoser med ett API för att kunna ta hand om användargränssnittskomponenter såsom knappar, listor och så vidare. Vidare så kan man genom form manager APIt även skapa egendefinierade användargränssnittskomponenter, som kallas för gadgets.[14]
På detta sätt kan man helt enkelt skapa egna knappar som man själv bestämmer hur de ska bete sig.

5 Utvärdering av API

I föregående kapitel presenterades de APIer som i detta kapitel kommer att analyseras med hjälp av ett antal utvärderingskriterier. Detta kapitel börjar med en kort introduktion om metoden som använts vid val av API. Senare så beskrivs de kriterier som använts vid analysen och slutligen analyseras de APIer som presenterades i föregående kapitel.

5.1 Val av APIer

I början av projekter så tog vi fram kriterier som vi tänkte använda för att ta fram och särkilja på de APIer som vi sedan skulle välja. Vi märke dock efter ett tag att de APIer som främst användes var de som gjordes av respektive operativsystemstillverkare. Detta ledde oss till att vi valde ut de operativsystemstillverkare som vi uppfattade som störst på handdatorer. Den här uppfattningen baserar vi på personlig erfarenheter och det allmänna intrycket som vi fick på Internet när vi letade efter APIer. Vi tillämpade sedan våra kriterier på de utvalda opertivsystemstillverkarnas standard APIer. Nedan presenteras de kriterier som vi tog fram tillsammans med en kort motivering till varför vi fann detta kriterie viktigt, samt en förklaring av kriteriet i det fall vi fann att detta krävdes.

5.2 Utvärderingskriterier

Här presenteras de utvärderingskriterierna som tagits fram för att särskilja de APIer som vi har hittat, samt en kort förklaring till varför vi valt ut dessa kriterier.

- Hur svårt är det att förstå APIt och dess funktioner?
  Hur lätt är det att göra ett program med hjälp av APIt? Krävs det några speciella kunskaper förutom baskunskaper i programmeringsspråket som APIt använder? Detta bör ses som en viktigt fråga varför den valdes att tas med bland kriterierna för val av API.

- Komplexitet?

  Den nivå av komplexitet som API kan tillhandahålla till programmeraren, tillhandahåller APIt till exempel färdiga knappar eller måste/kan egna definieras.

- Programmeringsspråk?

  Detta kriterie anger vilket språk som APIt är baserat på. Att ett API är på ett visst språk kan det underlätta för utvecklare, det kan vara så att programmeraren inte har en aning om hur Java språket fungerar. Då man inte förstår språket som APIt är baserat på så kan detta medföra ytterligare komplikationer.

- Är APIt plattformsoberoende?

  Att ett API är plattformsoberoende tillför att applikationen kan nå ut till fler användare. Istället för att behöva anpassa en applikation till olika APIer i olika plattformar så räcker det då att ha ett API.

- Portning?

  Finns det bra stöd om man vill flytta program från en plattform till en annan.

- Prestanda hos APIt?

  Kan man till exempel använda APIt för att skapa högpresterande applikationer såsom spel?

- Pris?

  Vad kostar APIt att använda. Om ett API kostar pengar kan det vara värt att tänka på att man kanske ska använda ett annat API som finns tillgängligt. För ett företag med begränsad budget kan detta vara ett viktigt kriterie.

- Finns det emulatorer till APIt?

  Emulatorer är viktigt för framställandet av ett bra program. En bra emulator ger möjlighet till att få dels en grafisk vy om hur en applikationen kommer att se ut på en handdator, men även att de specifikationer som finns för programmet uppfylls. Det

kan finnas både tekniska och praktiska svårigheter med att kontrollera applikationer direkt på handdatorn.

- Utvecklingsverktyg för APIt.
  Vad finns det för utvecklingsverktyg till APIt? Med utvecklingsverktyg så menas till exempel gratis kompilatorer, IDEer med mera. Då det kan vara enklare att hålla reda på ett stort projekt så är utvecklingsverktyg bra att ha till hands. På detta sätt så kan utvecklingsförfarandet bli enklare.

- Finns tillgänglig dokumentation av API?
  Att det finns lättillgänglig dokumentation av ett API är också en punkt som medför till att utvecklingsförfarandet blir enklare. På så sätt kan man lägga ner tid på att göra en välfungerande applikation istället för att försöka förstå sig på APIt.

5.3 Analys av varje API

I detta delkapitel följer nu en analys av de APIer som presenterades i kapitel 4. Analysen utförs med hjälp av de utvärderingskriterier som presenterades i föregående delkapitel.

5.3.1 Symbian SDK

- Hur svårt är det att förstå APIt och dess funktioner?
  Vi tyckte det var svårt, mycket berodde på den språkspecialisering som måste göras på grund av de begränsade resurser som finns. Bland annat så måste man känna till att man har definierat om primitiva typen integer. Klassnamn har specifika språkkonventioner.

- Komplexitet?
  APIt tillhandahåller ett stort antal fördefinierade grafiska komponenter, ett exempel är OK knapp. Möjlighet finns även för att kunna definiera egna komponenter.

Men tanke på hur många olika sätt programmeringskod kan uppnå samma resultat så är det uppsatsförfattarnas uppfattning att språket bör anses vara förhållandevis komplext.

- Programmeringsspråk?
  Det språk som används för att programmera mot APIt är C++.

- Är APIt platformsoberoende?
  Nej det är bundet till Symbian OS.

- Portning?
  Detta har vi inte kunnat hitta någon information om.

- Prestanda hos APIt?
  Detta har vi inte kunnat hitta någon information om.

- Pris?
  Symbian tillhandahåller gratis SDK för utvecklare.

- Finns det emulatorer till APIt?
  Symbian tillhandahåller gratis emulator för utvecklare, vidare så tillhandahåller även licenstagare av Symbian OS sin version av Symbians emulatorn.

- Utvecklingsverktyg för APIt.
  Det finns en mängd av utvecklingsverktyg, både gratis och sådana som kostar. Symbian rekommenderar dock att man använder MetroWerks Codewarrior IDE.

- Finns tillgänglig dokumentation av APIt?
  Symbian och andra organisationer tillhandahåller mycket bra och gratis dokumentation till dess utvecklare, inte bara om rent kodspecifika frågor utan även om GUI design och dylikt.

5.3.2 Sun J2ME

- Hur svårt är det att förstå APIt och dess funktioner?

  Skillnaderna mellan J2SE och J2ME är mycket små, omställningen för en J2SE programmerare till att programmera J2ME applikationer bedömer vi därför vara minimal.

- Komplexitet?

  APIt tillhandahåller ett stort antal fördefinierade komponenter, huruvida egna komponenter kan skapas är vi osäkra på.

- Programmeringsspråk?

  Programmeringsspråket mot APIt är Java.

- Är APIt platformsoberoende?

  J2ME kan exekveras på alla plattformar som har en KVM (Kilo Virtual Machine).

- Portning?

  Förutsatt att applikationer inte använder sig av plattformsberoende egenskaper så går det bra.

- Prestanda hos APIt?

  Detta har vi inte funnit information om.

- Pris?

  SUN tillhandhåller en SDK gratis för utvecklare.

- Finns det emulatorer till APIt?

  SUN tillhandahåller en gratis emulator för utvecklare.

- Utvecklingsverktyg för APIt.

  Det finns en mängd utvecklingsverktyg. SUN tillhandahåller gratis IDEer för applikationsutveckling, även här rekomenderar många utvecklare MetroWerks CodeWarrior.

- Finns tillgänglig dokumentation av APIt?

  SUN tillhandahåller gratis dokumentation för utvecklare.

5.3.3 Windows CE

- Hur svårt är det att förstå APIt och dess funktioner?

  Vi tyckte att det var ett API som var svårt att förstå. Dock om man har förkunskaper i Windows programmering så ska övergången enligt Microsoft inte vara svår.

- Komplexitet?

  APIt tillhandahåller ett stort antal fördefinierade grafiska komponenter, ett exempel är Cancel knapp. Möjlighet finns även för att kunna definiera egna komponenter.

- Programmeringsspråk?

  Det språk som används för att programmera mot APIt är C++.

- Är APIt platformsoberoende?

  Nej, det är bundet till Windows CE.

- Portning?

  Detta har vi inte kunnat hitta någon information om.

- Prestanda hos APIt?

  Detta har vi inte kunnat hitta någon information om.

- Pris?

  Microsoft tillhandahåller gratis SDK för utvecklare.

- Finns det emulatorer till APIt?

  Microsoft tillhandahåller gratis emulator för utvecklare.

- Utvecklingsverktyg för APIt.

  Microsoft tillhandahåller utvecklingsverktyg för APIt. Men rekomenderar Microsoft Visual Studio.

- Finns tillgänglig dokumentation av API?

  Microsoft tillhandahåller bra gratis dokummentation som inte bara berör kodspecifika aspekter utan även dokumentation om hur operativssystemet är uppbyggt och hur man designar GUI.

5.3.4 PalmOS SDK

- Hur svårt är det att förstå APIt och dess funktioner?

  Palms APIer är förhållandevis enkla att använda för programutveckling. Detta är något som verkar vara en medveten strategi från Palm för att det skall var lätt att nästan direkt kunna skapa applikationer.

- Komplexitet?

  APIt tillhandahåller ett stort antal fördefinierade komponenter och det är även möjligt att skapa egna komponenter.

- Programmeringsspråk?

  Programmeringsspråket vid programmering mot APIt är C.

- Är APIt platformsoberoende?

  APIt är bundet till Palm OS.

- Portning?

  Detta har vi inte kunnat bilda oss en uppfattning om.

- Prestanda hos APIt?

  Detta har vi inte kunnat bilda oss en uppfattning om.

- Pris?

  Palm tillhandahåller SDK gratis för utvecklare.

- Finns det emulatorer till APIt?

  Palm tillhandahåller en emulator gratis för utvecklare.

- Utvecklingsverktyg för APIt.

  Det finns en mängd utvecklingsverktyg att tillgå. Dock rekommenderar Palm att man använder MetroWerks CodeWarrior.

- Finns tillgänglig dokumentation av API?

  Palm tillhandahåller gratis dokumentation för utvecklare, men tillskillnad från Windows CE och Symbian så förefaller det som Palm inriktat mycket av sin dokumentation mot GUI design. Dokumentation inriktar sig mot att man snabbt skall kunna skapa applikationer.

6 Erfarenhet och rekommendationer

I detta kapitel presenteras rekommendationer för val av API, detta baserat på våra erfarenheter. Vidare presenteras scenarion som kan underlätta vid val av utvecklings API. Kapitlet börjar med att beskriva våra erfarenheter kring de fyra APIer som presenterades i tidigare kapitel. Därefter presenteras tre scenarier som ger olika perspektiv på vilket API man helst ska välja beroende på vilken klassifikation projektet har (stort/litet). Vidare så presenteras det API som vi tycker är bäst att använda.

6.1 Erfarenhet

Efter att ha en undersökt de fyra APIer som finns beskrivna i kapitel 4 samt kapitel 5 så kunde vi få en uppfattning om vilket API som vi helst skulle rekommendera. Dock så kan denna uppfattning skilja sig något beroende på utvecklaren. Här presenteras nu därför våra synpunkter på de APIer som presenterats tidigare.

6.1.1 Symbian SDK

Symbian har en klar strävan att göra så bra, funktionsdugliga och användarvänliga applikationer som möjligt. De lösningar som presenteras återföljs ofta av ett resonemang till varför lösningen implementerades på en bra teknisk nivå. Detta bidrar till att ge en bättre förståelse till varför och hur en programmerare skapar bra program.
Symbians API är trots detta ganska svår att använda. Det krävs att man sätter sig in de namn- och kodskrivnings-konventioner som flitigt används för att kunna skapa applikationer med hjälp av APIerna. Symbian har dessutom inte fördelen att såsom Windows CE påminna om andra och kanske mer välkända kodmiljöer. Konventionerna är dock till stor fördel när man väl har lärt sig dem, då det bland annat blir lättare att förstå andras programkod.
Symbian tillhandahåller även feltestning av kod med hjälp av så kallad Defensive program-

ming (assertion och invarianter).
Symbians förefaller dessutom ha en strävan i sin dokumentation att ge en applikationsutvecklare dels information om hur ett användarvänligt GUI bör utformas och dels hur operativsystemet fungerar i stort. Detta troligen med intentionen att en utvecklare skall få en djupare insikt om hur applikationer fungerar och därmed kunna gör sina applikationer mer resurssnåla.

6.1.2 J2ME

Då vi läste en del om J2ME och sett en del kodexempel så kunde vi bilda oss en uppfattning om vad det innebär att programmera ett grafiskt användargränssnitt i J2ME miljön.
Bland annat så insåg vi ganska snart att det knappt var någon skillnad på att utveckla applikationer i J2ME miljön och J2SE miljön. Man behöver dock exempelvis känna till att det finns vissa restriktioner. Det hello-exjobb exempel som har tagits med i appendix A visar tydligare hur enkelt det är att skapa en grafisk applikation. Allt fungerar i princip på samma sätt som en applikation skriven för J2SE miljön, dock enklare. Med baskunskaper i Java och kunskap om kompileringsproceduren får man snabbt en fungerande applikation. Som det framgår av exemplet är det inte heller många rader kod. Skall man dock utveckla andra applikationer förutom hello-exjobb så blir det genast mer komplicerat som vi har sett på en del andra exempel. [31] Då det blir fler och fler klasser så kan det bli jobbigt att få överblick på det hela, det handlar dock inte om kodmässig svårighet utan själva projektets storlek. De kodexempel som vi har sett var alla ändå lätta att förstå.

6.1.3 Windows CE

Efter en del inläsning av Windows CE och en hel del utforskning av kodexempel så kunde vi bilda oss en uppfattning om hur det är att programmera en grafisk applikation till Windows CE baserade apparater.
Vi tyckte att i princip ingen del av koden på något sätt kunde göra sig förståelig, inte minst

den del av kod som behandlade de grafiska aspekterna. Ingen av oss har någon tidigare erfarenhet av Windows programmering vilket förmodligen var den avgörande faktorn i att vi inte kunde behandla koden lika enkelt som med Java. Vi inser dock att om man har kunskaper inom Windows programmering så kommer koden i exemplet se appendix B inte att vara något problem att förstå. Även i detta fall så har man en del skillnader från traditionell programmering till Windows baserade system, detta på grund av att man har begränsade resurser. Om man läser igenom hello-world exemplet som finns med i appendix B så inser man snart att det behövs en hel del mer kunskap om programmeringstypen än vad som krävs i J2ME fallet. Man måste ta hänsyn till fler detaljer då man programmerar ett grafiskt användargränssnitt. Man måste till exempel definiera att det är en viss typ av knapp man ska ha, man måste själv bestämma menyer och så vidare. Detta är dock fullt förståeligt eftersom Windows CE och de andra operativsystemen till skillnad från J2ME måste även hantera utseendet av det grafiska användargränssnittet. Som vi tidigare har nämnt så sköts utseendet av det grafiska användargränssnittet av det operativ system som exekverar på handdatorn.
Även om utvecklaren har kunskaper i Windows programmering så tycker vi fortfarande att det krävs större engagemang av utvecklaren då denne skall skapa en grafisk applikation.
Andra aspekter så som typdefinition och variabelnamn är också sådant som ställer till för en ny utvecklare. Till exempel så använder man *TCHAR* istället för *char*. Variabelnamn börjar alltid med någon prefix. I hello-world exemplet ser man till exempel att man har prefixet *sz* framför en variabel av typen *TCHAR*.

6.1.4 PalmOS SDK

Användare gör man lättast glada genom ett bra GUI och lätthanterliga applikationer. Palm insåg detta tidigt och utformade lättförståeliga och detaljerade riktlinjer för hur GUI bör se för att användare skall finna dess applikationer lätta att använda. [2]
Som det framgår av Palm skall det gå snabbt att komma igång med att utveckla applika-

tioner.

Av den kod som vi har sett så verkar programmering för PalmOS vara mycket intuitiv, med en lättförståelig namnstandard. I en stor del av dokumentation som handlar om programmering för Palm kommer man väldigt snabbt fram till det första kodexemplet. Detta har inneburit att Palm idag har ett mycket stort antal utvecklare och därmed har ett stort antal applikationer, allt från program för att se stjärnkonstellation till excel program. Denna bredd av applikationer är något som vi tror att många handdatoranvändare efterfrågar, som användare så kan du använda din Palm såväl på jobbet för att skriva dokument och email som på golfbana för att hålla reda på scorekortet.

6.2 Scenarior

I detta kapitel presenterar vi ett antal scenarior för att på detta sätt kunna ge rekommendationer för ett specifikt fall.

6.2.1 Student

Vi tror att J2MEs API är det mest passande för studenter, främst på grund av att inlärningströskeln är mycket lägre än för de andra APIerna. Java gömmer mycket av den komplexitet som annars måste tas om hand av programmeraren. J2ME blir ännu enklare att förstå om utvecklaren har grundläggande kunskaper i Java programmering. Den tid det tar att sätta sig in i de andra APIernas semantik tror vi är bättre lagd på andra kursmoment såsom GUI design och programmering paradigmer (inriktat mot handdatorer). Den låga inlärningskurvan tillsammans med tillgången till välskriven dokumentation och gratis utvecklingsverktyg från SUN [29], får oss att komma till slutsatsen att J2ME troligen är den bästa miljön att introduceras till handdator programmering. Att de Java applikationer som programmeras kan exekvera på ett stort antal plattformar ser vi inte heller som något negativt.

6.2.2 Mindre projekt

Mindre projekt förutsätter vi har mindre resurser och därför har en smalarare kunskapsbas. Med mindre resurser och en smalare kunskapsbas så kan det vara lämpligt att välja det API som är enklast att komma igång med. Med J2ME så behöver man inte bry sig om detaljer i det grafiska användargränssnittet, dessutom så når man en större mängd med användare då J2ME är portabelt. Vi tycker därför att J2ME skulle passa bättre i detta fall.

6.2.3 Större projekt

Större projekt tror vi innebär längre projekttider och därmed större resurser att lägga ner på projektet. Här har man till exempel tid att lära sig plattformsberoende APIer. Då vi tror att Symbian kommer att vara en av de största operativsystemen på handdatormarknaden så rekommenderar vi dess API. Den kräver längre inlärningstid än till exempel Palm, men detta tror vi dock inte skall vara ett bekymmer i ett större projekt.

6.3 Rekommendation

Då vi hade haft en diskussion kring varje API och tänkt mera på de scenarier som vi beskrev i förregående avsnitt så kom vi fram till ett beslut angående vilket API vi skulle rekommendera. Vi har bland annat studerat enklare exempel likt hello-world exempel och andra mer avancerade kodexempel. Vi har insett att J2ME är det enklaste av dem att komma igång med och skapa en fungerande applikation. Palm är förhållandevis enklare än Symbian och Windows CE men jämfört med J2ME så var Palm svårare. Windows CE behöver inte vara svårt att förstå om man har förkunskaper i Windows programmering. Av de fyra APIer vi har studerat har Symbian varit den svåraste att förstå, dock så är det även så att vi tycker att den och J2ME är de APIer som vi personligen skulle rekommendera. J2ME för att den är så enkel att komma igång med även om man själv inte kan bestämma

mycket i utseendet. Symbian för att man förmodligen kommer att nå en stor kundmassa (både SonyEricsson och Nokia använder Symbian OS).

7 Slutsats och sammanfattning

Projektet handlar om att analysera olika grafiska gränssnitts APIer och redogöra för grafisk design med inriktning mot handdatorer. Vi har dessutom åskådliggjort den grafiska strukturen för olika handdatorplattformar då vi tror att detta kan ge en djupare förståelse för hur APIerna kan användas. Under projekts gång har vi även kommit fram till att programmering för handdatorer skiljer sig markant från programmering för vanliga datorer. Detta inte minst beroende på att målgruppen för handdatorer inte är samma som den målgrupp som använder sig utav den traditionella hemdatorn, varför den grafiska designen blir annorlunda. Den begränsning som hårdvaran på handdatorer skapar speglas också i utvecklingen av applikationer. I form av en rekommendation till vilket programmerings-API som applikations utveckling bör ske i, fann vi att det berodde på de förutsättningar som fanns. Detta är varför vi inte utser någon 'vinnare' bland APIerna utan hänvisar till de scenarier som tidigare specificerat i uppsatsen. En överraskning under projektets gång har varit att vår tidsplan har sviktat ibland. Dock har detta alltid löst sig på ett eller annat sätt. Under projektets gång har vi, författarna, fått större intresse för applikations utveckling till handdatorer, vi ser detta projekt som en god fortsättning för vidare fördjupning i något av dessa APIer. Till exempel har vi kommit fram till att man skulle kunna skapa en laboration med något av de APIer som behandlats i uppsatsen. Vidare skulle man kunna fördjupa sig i något av de operativsystemen och ge en mera detaljerad information.

Referenser

[1] *UIQ Style Guide.* http://uiq.velocitytech.net/UIQStyleGuide/index.html, 20040219.

[2] *Palm OS Application Design.* http://www.palmos.com/dev/support/docs/ui/UI_Design.html, 20040220.

[3] *Palm OS User Interface Guidelines.* http://www.palmos.com/dev/support/docs/ui/UIGuidelinesTOC.html, 20040220.

[4] *Understanding Modularity in Microsoft Windows CE 2.1.* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/duce21/html/modularity.asp, 20040318.

[5] *Windows CE Modules and Components.* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceintro/html/wceintroWindowsCEModulesandComponents.asp, 20040318.

[6] *Differences between PersonalJava and MIDP Java Environments.* http://www.symbian.com/developer/techlib/papers/PJAE_MIDP/PJAE_MIDP_2.pdf, 20040319.

[7] *GWES Component Model.* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceui/html/_wcesdk_GWES_Component_Model.asp, 20040319.

[8] *Introduction to User Interface Services on Windows CE.* http://msdn.microsoft.com/library/default.asp?=/library/en-us/wceui/html/_wcesdk_Introduction_to_User_Interface_Services_onWindows_CE.asp, 20040319.

[9] *Introduction to J2ME Programming using MIDP.* http://www.codewarriorU.com, 20040324. Gratis registrering krävs för att få tillgång till kurser.

[10] *UI Guidelines vs Usability testing.* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwui/html/uiguide.asp, 20040415.

[11] *About Windows CE.* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceintro/html/wceintroAboutWindowsCE.asp, 20040417.

[12] *Palm OS.* http://www.codewarriorU.com, 20040420. Gratis registrering krävs för att få tillgång till kurser.

[13] *Symbian OS overview.* http://www.symbian.com/technology/OSoverview/OSoverview_040304.exe, 20040420.

[14] *Palm OS Programmers Companion.* http://text.staticfree.info/palmdev/Palm%20OS%20Companion.pdf, 20040421.

[15] *The History of Psion (from 1980 to 1997).* http://3lib.ukonline.co.uk/historyofpsion.htm, 20040511.

[16] *Palm OS: An Enterprise Powerhouse.* http://www.palmsource.com/developers/enterprise_powerhouse.html, 20040511.

[17] *Simpliefying access to the Window Server in UI Control Framework.* http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/ApplicationFramework/UIControlFrameworkGuide/UIControlFrameworkGuide1/SimplifyWindowServer.guide.html#UIControlFrameworkGuide1%2eSimplifyWindowServer, 20040511.

[18] *UI Control Framework in Using UI Control Framework.* http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/ApplicationFramework/UIControlFrameworkGuide/UIControlFrameworkGuide1/UIControlFrameworkIntroduction.guide.html#UIControlFrameworkGuide1%2eintroduction, 20040511.

[19] *UI Control Framework Overview in Application Framework.* http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/ApplicationFramework/UIControlFrameworkOverview.guide.html#ApplicationFrameworkOverview%2eUIControlFrameworkOverview%2emain, 20040511.

[20] *Uikon Core Controls Overview in Application Framework.* http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/ApplicationFramework/UikonCoreControlsOverview.guide.html#ApplicationFrameworkOverview%2eUikonOverview%2eUikonCoreControlsOverview%2emain, 20040511. På grund av tekniska fel så är länken INTE klickbar!

[21] *PSION: History and Business.* http://www.investorrelations.co.uk/psion/corp_history.jsp, 20040520.

[22] *The Dynabook Revisited.* http://www.honco.net/os/kay.html, 20040529.

[23] *PalmSource | Press Room.* http://www.palmsource.com/press/2004/031804_q3fy04.html, 20040529.

[24] *Symbian: Press office: Symbian Limited Q3 2003 results.* http://www.symbian.com/press-office/2003/pr031120.html, 20040529.

[25] *Windows mobile home.* http://www.microsoft.com/windowsmobile/default.mspx, 20040529.

[26] Eric Bergman. *Information Appliances and beyond.* Academic Press, 1st edition, 2000.

[27] Richard Harrison. *Symbian OS C++ for mobile phones.* Wiley, 2003. En mycket bra bok om man vill börja programmera Symbian applikationer.

[28] Bruce Thomas Konrad Baumann. *User Interface Design For Electronic Appliances.* Taylor and Francis, 1st edition, 2001.

[29] John W. Muchow. *Core J2ME Technology MIDP.* Sun Microsystems, 1st edition, 2002.

[30] John Murray. *Inside Microsoft Windows CE.* Microsoft, 1st edition, 1998.

[31] Mark VandenBrink Roger Riggs, Antero Taivalsaari. *Programming Wireless Devices with the Java2 Platform, Micro Edition.* Addison Wesley, 1st edition, 2001.

[32] Joel Spolsky. *User Interface Design for Programmers.* Apress, 1st edition, 2001.

[33] Kim Topley. *J2ME IN A NUTSHELL A Desktop Quick Reference.* O'Reilly, 1st edition, 2002.

A J2ME exempel

```
 1
 2 /*
 3  * HelloExjobb.java
 4  *
 5  * Created on den 4 april 2004, 17:37
 6  */
 7
 8 //package Exjobb;
 9
10 import javax.microedition.midlet.*; import javax.microedition.lcdui.*;
11
12 /**
13  * An example MIDlet with simple "Hello" text and an Exit command.
14  * Refer to the startApp, pauseApp, and destroyApp
15  * methods so see how each handles the requested transition.
16  *
17  * @author  Administrator
18  * @version
19  */
20  public class HelloExjobb extends MIDlet implements CommandListener
21  {
22     private Command exitCommand; // The exit command
23     private Display display;    // The display for this MIDlet
24
25     public HelloExjobb()
26     {
27
28         display = Display.getDisplay(this);
29         exitCommand = new Command("Exit", Command.SCREEN, 2);
30
31     }
32
33     /**
34      * Start up the Hello MIDlet by creating the TextBox and associating
35      * the exit command and listener.
36      */
37      public void startApp()
38      {
39             TextBox t = new TextBox("Hello Exjobb MIDlet", "Hello godkänt Exjobb", 256,↙
      0);
40
41             t.addCommand(exitCommand); t.setCommandListener(this);
42
43             display.setCurrent(t);
44
45      }
46
47     /**
48      * Pause is a no-op since there are no background activities or
49      * record stores that need to be closed.
50      */
51      public void pauseApp()
52      {
53      }
54
55     /**
56      * Destroy must cleanup everything not handled by the garbage collector.
57      * In this case there is nothing to cleanup.
58      */
59      public void destroyApp(boolean unconditional)
60      {
61      }
62
63     /*
64      * Respond to commands, including exit
65      * On the exit command, cleanup and notify that the MIDlet has been destroyed.
66      */
67      public void commandAction(Command c, Displayable s)
68      {
69         if (c == exitCommand)
70         {
71             destroyApp(false);
72             notifyDestroyed();
73
```

```
74          }
75
76      }
77
78 }
79
```

B Windows CE exempel

```
1 /*****************************************************************
2
3 THIS CODE AND INFORMATION IS PROVIDED AS IS WITHOUT WARRANTY OF
4 ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING
5 BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY
6 AND/OR FITNESS FOR A PARTICULAR PURPOSE.
7
8 Copyright  1994-1997  Microsoft Corporation.  All Rights Reserved.
9
10
11 Module Name:
12
13     Hello.cpp
14
15 Abstract:
16
17 This file contains the source code of the "Hello" application, which will be used to ↙
       illustrate part of the
18 functionality of Pocket PC. The application creates a window and draws "Hello Pocket ↙
       PC" text. This example
19 resembles to the Hello example of the WIN32_WCE. But there is neither exit nor       ↙
       minimize functionality on the
20 window because of the properties of Pocket PC applications.
21
22 ****************************************************************/
23 #include <windows.h>
24 #include <windowsx.h>
25 #include "resource.h"
26
27 HINSTANCE hInst = NULL;     // Local copy of hInstance
28 HWND hwndMain = NULL;      // Handle to Main window
29 returnedfrom CreateWindow
30
31 TCHAR szAppName[] = TEXT("Hello Pocket PC Application");
32 TCHAR szTitle[] = TEXT("Hello Pocket PC");
33
34 int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int↙
       CmdShow); BOOL
35  InitApplication ( HINSTANCE hInstance ); BOOL InitInstance (HINSTANCE hInstance, int ↙
       CmdShow );
36    LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp);
37
38
39 /*****************************************************************************
40
41   WinMain
42
43   ****************************************************************************/
44
45 int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int↙
       CmdShow)
46 {
47     MSG msg;
48     HWND hHelloWnd = NULL;
49      HACCEL hAccel = NULL;
50
51     //Check if Hello.exe is running. If it's running then focus on the window
52     hHelloWnd = FindWindow(szAppName, szTitle); if (hHelloWnd) { SetForegroundWindow (↙
       hHelloWnd); return 0; }
53
54     if ( !hPrevInstance ) { if ( !InitApplication ( hInstance ) ) { return (FALSE); } ↙
        } if ( !InitInstance(
55     hInstance, CmdShow )  ) { return (FALSE); } hAccel = LoadAccelerators(hInstance,
56     MAKEINTRESOURCE(IDR_ACCELERATOR1));
57
58     while ( GetMessage( &msg, NULL, 0,0 ) == TRUE ) { if (!TranslateAccelerator(      ↙
       hwndMain,hAccel, &msg)) {
59     TranslateMessage (&msg); DispatchMessage(&msg); } } return (msg.wParam); }
60
61 /*****************************************************************************
62
63    InitApplication
64
65   ****************************************************************************/
```

```
 66
 67 BOOL InitApplication ( HINSTANCE hInstance ) { WNDCLASS wc; BOOL f;
 68
 69     wc.style = CS_HREDRAW | CS_VREDRAW ; wc.lpfnWndProc = (WNDPROC)WndProc; wc.
        cbClsExtra = 0; wc.cbWndExtra =
 70     0; wc.hIcon = NULL; wc.hInstance = hInstance; wc.hCursor = NULL; wc.hbrBackground
        = (HBRUSH) GetStockObject(
 71     WHITE_BRUSH ); wc.lpszMenuName = NULL; wc.lpszClassName = szAppName;
 72
 73     f = (RegisterClass(&wc));
 74
 75     return f; }
 76
 77 /**********************************************************************
 78
 79   InitInstance
 80
 81 **********************************************************************/
 82
 83 BOOL InitInstance (HINSTANCE hInstance, int CmdShow ) { hInst = hInstance; hwndMain =
     CreateWindow(szAppName,
 84 szTitle, WS_VISIBLE, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL,
      NULL, hInstance, NULL );
 85
 86     if ( !hwndMain ) { return FALSE; } ShowWindow(hwndMain, CmdShow ); UpdateWindow(
        hwndMain); return TRUE; }
 87
 88 /**********************************************************************************
     *
 89
 90    WndProc
 91
 92  **********************************************************************************
     */ LRESULT CALLBACK
 93  WndProc(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp) { LRESULT lResult = TRUE; HDC hdc;
      PAINTSTRUCT ps; RECT
 94  rect;
 95
 96     switch(msg) { case WM_COMMAND: switch (GET_WM_COMMAND_ID(wp,lp)) { case IDOK:
 97     SendMessage(hwnd,WM_CLOSE,0,0); break; default: return DefWindowProc(hwnd, msg, wp
        , lp); } break; case
 98     WM_PAINT: { hdc = BeginPaint (hwnd, &ps); GetClientRect (hwnd, &rect);
 99
100             DrawText (hdc, L"Hello Pocket PC!", -1, &rect, DT_SINGLELINE | DT_CENTER |
       DT_VCENTER);
101
102             EndPaint (hwnd, &ps); } break;
103
104         case WM_CLOSE: DestroyWindow(hwnd); break;
105
106         case WM_DESTROY: PostQuitMessage(0); break;
107
108         default: lResult = DefWindowProc(hwnd, msg, wp, lp); break; } return (lResult)
     ; }
109
110
111 // END HELLO.CPP
112
```

C Symbian OS exempel

```
Filen: BLD.INF
// BLD.INF
//
// Copyright (c) 2000 Symbian Ltd.  All rights reserved.
//



PRJ_MMPFILES

HelloWorld.mmp
##########

Filen: DISTRIBUTION.policy
Category G
##########

Filen: HelloWord.h
// HelloWorld.h
// ------------
//
// Copyright (c) 2000 Symbian Ltd.  All rights reserved.
//

////////////////////////////////////////////////////////////////////
// HelloWorld
// ----------
//
//
// The class definitions for the simple example application
// containing a single view with the text "Hello World !" drawn
// on it.
//
// The class definitions are:
//
// CExampleApplication
// CExampleAppUi
// CExampleAppView
// CExampleDocument
//
//
////////////////////////////////////////////////////////////////////
#ifndef __HELLOWORLD_H
#define __HELLOWORLD_H

#include <coeccntx.h>

#include <eikenv.h>
#include <eikappui.h>
#include <eikapp.h>
#include <eikdoc.h>
#include <eikmenup.h>

#include <eikon.hrh>

#include <helloworld.rsg>
#include "helloworld.hrh"



////////////////////////////////////////////////////////////////////
//
// CExampleApplication
//
////////////////////////////////////////////////////////////////////

class CExampleApplication : public CEikApplication
    {
private:
                // Inherited from class CApaApplication
    CApaDocument* CreateDocumentL();
    TUid AppDllUid() const;
    };

```

```
 75 ////////////////////////////////////////////////////////////////////////
 76 //
 77 // CExampleAppView
 78 //
 79 ////////////////////////////////////////////////////////////////////////
 80 class CExampleAppView : public CCoeControl
 81     {
 82 public:
 83     static CExampleAppView* NewL(const TRect& aRect);
 84     CExampleAppView();
 85     ~CExampleAppView();
 86     void ConstructL(const TRect& aRect);
 87
 88 private:
 89                 // Inherited from CCoeControl
 90     void Draw(const TRect& /*aRect*/) const;
 91
 92 private:
 93     HBufC*  iExampleText;
 94     };
 95
 96
 97 ////////////////////////////////////////////////////////////////////////
 98 //
 99 // CExampleAppUi
100 //
101 ////////////////////////////////////////////////////////////////////////
102 class CExampleAppUi : public CEikAppUi
103     {
104 public:
105     void ConstructL();
106     ~CExampleAppUi();
107
108 private:
109                // Inherirted from class CEikAppUi
110     void HandleCommandL(TInt aCommand);
111
112 private:
113     CCoeControl* iAppView;
114     };
115
116
117 ////////////////////////////////////////////////////////////////////////
118 //
119 // CExampleDocument
120 //
121 ////////////////////////////////////////////////////////////////////////
122 class CExampleDocument : public CEikDocument
123     {
124 public:
125     static CExampleDocument* NewL(CEikApplication& aApp);
126     CExampleDocument(CEikApplication& aApp);
127     void ConstructL();
128 private:
129                 // Inherited from CEikDocument
130     CEikAppUi* CreateAppUiL();
131     };
132
133
134 #endif
135 ####################
136
137 Filen: HelloWorld.hrh
138
139 // HelloWorld.hrh
140 //
141 // Copyright (c) 2000 Symbian Ltd.  All rights reserved.
142 //
143
144
145 enum TExampleMenuCommands
146     {
147     EExampleItem0 = 200,
148     EExampleItem1,
```

```
149     EExampleItem2
150     };
151
152
153 #############3
154 Filen: HelloWorld.mmp
155
156 // HelloWorld.mmp
157 //
158 // Copyright (c) 2000 Symbian Ltd.  All rights reserved.
159 //
160
161 // using relative paths for sourcepath and user includes
162
163 TARGET        HelloWorld.app
164 TARGETTYPE    app
165 UID           0x100039CE 0X10008ACE
166 TARGETPATH    \system\apps\HelloWorld
167 SOURCEPATH    .
168 SOURCE        HelloWorld_Main.cpp
169 SOURCE        HelloWorld_Application.cpp
170 SOURCE        HelloWorld_Document.cpp
171 SOURCE        HelloWorld_AppUi.cpp
172 SOURCE        HelloWorld_AppView.cpp
173 USERINCLUDE   .
174 SYSTEMINCLUDE \epoc32\include
175 SYSTEMINCLUDE \epoc32\include\techview
176
177 RESOURCE      HelloWorld.rss
178 LIBRARY       euser.lib apparc.lib cone.lib eikcore.lib
179 ############################
180
181 Filen: HelloWorld.RSS
182 // HelloWorld.RSS
183 //
184 // Copyright (c) 1997-1999 Symbian Ltd.  All rights reserved.
185 //
186
187 NAME HEWO
188
189 #include <eikon.rh>
190 #include <eikcore.rsg>
191
192 #include "helloworld.hrh"
193
194 RESOURCE RSS_SIGNATURE { }
195
196 RESOURCE TBUF { buf=""; }
197
198 RESOURCE EIK_APP_INFO
199     {
200     hotkeys=r_example_hotkeys;
201     menubar=r_example_menubar;
202     }
203
204
205
206 RESOURCE HOTKEYS r_example_hotkeys
207     {
208     control=
209         {
210         HOTKEY { command=EEikCmdExit; key='e'; }
211         };
212     }
213
214 RESOURCE MENU_BAR r_example_menubar
215     {
216     titles=
217         {
218         MENU_TITLE { menu_pane=r_example_first_menu; txt="HelloWld"; }
219         };
220     }
221
222 RESOURCE MENU_PANE r_example_first_menu
```

```
223     {
224     items=
225         {
226         MENU_ITEM { command=EExampleItem0; txt="Item 0"; },
227         MENU_ITEM { command=EExampleItem1; txt="Item 1"; },
228         MENU_ITEM { command=EExampleItem2; txt="Item 2"; },
229         MENU_ITEM { command=EEikCmdExit; txt="Close"; }
230         };
231     }
232
233
234 RESOURCE TBUF r_example_text_Hello { buf="Hello World!"; }
235 RESOURCE TBUF r_example_text_Item0 { buf="Item 0"; }
236 RESOURCE TBUF r_example_text_Item1 { buf="Item 1"; }
237 RESOURCE TBUF r_example_text_Item2 { buf="Item 2"; }
238 ###############
239 Filen: HelloWorld_CExampleApplication.cpp
240
241 // HelloWorld_CExampleApplication.cpp
242 // ----------------------------------
243 //
244 // Copyright (c) 2000 Symbian Ltd.  All rights reserved.
245 //
246
247 ////////////////////////////////////////////////////////////////////////
248 //
249 // Source file for the implementation of the
250 // application class - CExampleApplication
251 //
252 ////////////////////////////////////////////////////////////////////////
253
254 #include "HelloWorld.h"
255
256 const TUid KUidHelloWorld = { 0X10008ACE };
257
258 //              The function is called by the UI framework to ask for the
259 //              application's UID. The returned value is defined by the
260 //              constant KUidHelloWorlde and must match the second value
261 //              defined in the project definition file.
262 //
263 TUid CExampleApplication::AppDllUid() const
264     {
265     return KUidHelloWorld;
266     }
267
268 //              This function is called by the UI framework at
269 //              application start-up. It creates an instance of the
270 //              document class.
271 //
272 CApaDocument* CExampleApplication::CreateDocumentL()
273     {
274     return new (ELeave) CExampleDocument(*this);
275     }// HelloWorld_CExampleAppUi.cpp
276 // ----------------------------
277 //
278 // Copyright (c) 2000 Symbian Ltd.  All rights reserved.
279 //
280
281
282
283 ////////////////////////////////////////////////////////////////////////
284 //
285 // Source file for the implementation of the
286 // application UI class - CExampleAppUi
287 //
288 ////////////////////////////////////////////////////////////////////////
289
290 #include "HelloWorld.h"
291
292 //              The second phase constructor of the application UI class.
293 //              The application UI creates and owns the one and only view.
294 //
295 void CExampleAppUi::ConstructL()
296     {
```

```
297                 // BaseConstructL() completes the UI framework's
298                 // construction of the App UI.
299     BaseConstructL();
300                 // Create the single application view in which to
301                 // draw the text "Hello World!", passing into it
302                 // the rectangle available to it.
303     iAppView = CExampleAppView::NewL(ClientRect());
304     }
305
306
307 //              The app Ui owns the two views and is.
308 //              therefore, responsible for destroying them
309 //
310 CExampleAppUi::~CExampleAppUi()
311     {
312     delete iAppView;
313     }
314
315
316 //              Called by the UI framework when a command has been issued.
317 //              In this example, a command can originate through a
318 //              hot-key press or by selection of a menu item.
319 //              The command Ids are defined in the .hrh file
320 //              and are 'connected' to the hot-key and menu item in the
321 //              resource file.
322 //              Note that the EEikCmdExit is defined by the UI
323 //              framework and is pulled in by including eikon.hrh
324 //
325 void CExampleAppUi::HandleCommandL(TInt aCommand)
326     {
327     switch (aCommand)
328         {
329               // Just issue simple info messages to show that
330               // the menu items have been selected
331     case EExampleItem0:
332         iEikonEnv->InfoMsg(R_EXAMPLE_TEXT_ITEM0);
333         break;
334
335
336     case EExampleItem1:
337         iEikonEnv->InfoMsg(R_EXAMPLE_TEXT_ITEM1);
338         break;
339
340     case EExampleItem2:
341         iEikonEnv->InfoMsg(R_EXAMPLE_TEXT_ITEM2);
342         break;
343                 // Exit the application. The call is
344                 // implemented by the UI framework.
345
346     case EEikCmdExit:
347         Exit();
348         break;
349         }
350     }
351 ###############
352 Filen: HelloWorld_CExampleAppView.cpp
353
354 // HelloWorld_CExampleAppView.cpp
355 // ------------------------------
356 //
357 // Copyright (c) 2000 Symbian Ltd.  All rights reserved.
358 //
359
360 ////////////////////////////////////////////////////////////////////////
361 //
362 // Source file for the implementation of the
363 // application view class - CExampleAppView
364 //
365 ////////////////////////////////////////////////////////////////////////
366
367 #include "HelloWorld.h"
368
369 //
370 //              Constructor for the view.
```

```
371 //
372 CExampleAppView::CExampleAppView()
373     {
374     }
375
376
377 //              Static NewL() function to start the standard two
378 //              phase construction.
379 //
380 CExampleAppView* CExampleAppView::NewL(const TRect& aRect)
381     {
382     CExampleAppView* self = new(ELeave) CExampleAppView();
383     CleanupStack::PushL(self);
384     self->ConstructL(aRect);
385     CleanupStack::Pop();
386     return self;
387     }
388
389
390 //
391 //              Destructor for the view.
392 //
393 CExampleAppView::~CExampleAppView()
394     {
395     delete iExampleText;
396     }
397
398
399 //              Second phase construction.
400 //
401 void CExampleAppView::ConstructL(const TRect& aRect)
402     {
403                 // Fetch the text from the resource file.
404     iExampleText = iEikonEnv->AllocReadResourceL(R_EXAMPLE_TEXT_HELLO);
405                 // Control is a window owning control
406     CreateWindowL();
407                 // Extent of the control. This is
408                 // the whole rectangle available to application.
409                 // The rectangle is passed to us from the application UI.
410     SetRect(aRect);
411                 // At this stage, the control is ready to draw so
412                 // we tell the UI framework by activating it.
413     ActivateL();
414     }
415
416
417 //              Drawing the view - in this example,
418 //              consists of drawing a simple outline rectangle
419 //              and then drawing the text in the middle.
420 //              We use the Normal font supplied by the UI.
421 //
422 //              In this example, we don't use the redraw
423 //              region because it's easier to redraw to
424 //              the whole client area.
425 //
426 void CExampleAppView::Draw(const TRect& /*aRect*/) const
427     {
428                 // Window graphics context
429     CWindowGc& gc = SystemGc();
430                 // Area in which we shall draw
431     TRect      drawRect = Rect();
432                 // Font used for drawing text
433     const CFont*     fontUsed;
434
435                 // Start with a clear screen
436     gc.Clear();
437                 // Draw an outline rectangle (the default pen
438                 // and brush styles ensure this) slightly
439                 // smaller than the drawing area.
440     drawRect.Shrink(10,10);
441     gc.DrawRect(drawRect);
442                 // Use the title font supplied by the UI
443     fontUsed = iEikonEnv->TitleFont();
444     gc.UseFont(fontUsed);
```

```
445                 // Draw the text in the middle of the rectangle.
446     TInt   baselineOffset=(drawRect.Height() - fontUsed->HeightInPixels())/2;
447     gc.DrawText(*iExampleText,drawRect,baselineOffset,CGraphicsContext::ECenter, 0);
448                 // Finished using the font
449     gc.DiscardFont();
450     }
451
452
453
454 // HelloWorld_CExampleDocument.cpp
455 // -------------------------------
456 //
457 // Copyright (c) 2000 Symbian Ltd.  All rights reserved.
458 //
459
460 ////////////////////////////////////////////////////////////////////////
461 //
462 // Source file for the implementation of the
463 // document class - CExampleDocument
464 //
465 ////////////////////////////////////////////////////////////////////////
466
467 #include "HelloWorld.h"
468
469 //              The constructor of the document class just passes the
470 //              supplied reference to the constructor initialisation list.
471 //              The document has no real work to do in this application.
472 //
473 CExampleDocument::CExampleDocument(CEikApplication& aApp)
474         : CEikDocument(aApp)
475     {
476     }
477
478
479 //              This is called by the UI framework as soon as the
480 //              document has been created. It creates an instance
481 //              of the ApplicationUI. The Application UI class is
482 //              an instance of a CEikAppUi derived class.
483 //
484 CEikAppUi* CExampleDocument::CreateAppUiL()
485     {
486     return new(ELeave) CExampleAppUi;
487     }
488 ##############
489 Filen: HelloWorld_Main.cpp
490
491 // HelloWorld_Main.cpp
492 // -------------------
493 //
494 // Copyright (c) 2000 Symbian Ltd.  All rights reserved.
495 //
496
497 ////////////////////////////////////////////////////////////////////////
498 //
499 // HelloWorld
500 // ----------
501 //
502 //
503 // The example is a simple application containing a single view with
504 // the text "Hello World !" drawn on it.
505 //
506 // The example includes code for displaying a very simple menu.
507 //
508 // ---------------------------------------------------------------------
509 //
510 // This source file contains the single exported function required by
511 // all UI applications and the E32Dll function which is also required
512 // but is not used here.
513 //
514 ////////////////////////////////////////////////////////////////////////
515
516
517 #include "HelloWorld.h"
518
```

```
519 //             The entry point for the application code. It creates
520 //             an instance of the CApaApplication derived
521 //             class, CExampleApplication.
522 //
523 EXPORT_C CApaApplication* NewApplication()
524     {
525     return new CExampleApplication;
526     }
527
528 //             This function is required by all Symbian OS DLLs. In this
529 //             example, it does nothing.
530 //
531 GLDEF_C TInt E32Dll(TDllReason)
532     {
533     return KErrNone;
534     }
535
536
```