Examensarbete i informatik

GapiDraw för Symbian

Jonas Bylund & Erik Ruisniemi
Göteborg, Sweden 2004

GapiDraw för Symbian

Portning av ett grafikbibliotek från Windows till Symbian

JONAS BYLUND

ERIK RUISNIEMI

Report no 2004:27

ISSN: 1651-4769

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

Box 8718

SE – 402 75 Göteborg

Sweden

Telephone + 46 (0)31-772 4895

Chalmers repro

Göteborg, Sweden 2004

REPORT 2004:27

GapiDraw för Symbian

Portning av ett grafikbibliotek från Windows till Symbian

JONAS BYLUND
ERIK RUISNIEMI

Handledare: Johan Sanneblad
Examinator: Urban Nuldén

Department of Applied Information Technology
IT UNIVERSITY OF GÖTEBORG
GÖTEBORG UNIVERSITY AND CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2004

GapiDraw för Symbian

Portning av ett grafikbibliotek från Windows till Symbian

JONAS BYLUND & ERIK RUISNIEMI

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

SUMMARY

The purpose of this thesis is to investigate the possibility of developing applications with high performance graphics for mobile phones and handheld computers with different operating systems running on different hardware, using a common code base. The problems involved in programming for multiple platforms are also described, including an overview of the available multiplatform graphics libraries currently available, which are mainly for stationary computers.

Porting the graphics library GapiDraw from the Win32 platform (stationary and handheld computers) to the Symbian platform would be a great step in the right direction since two of the three most common mobile operating systems could use the same code base for graphics. The problems encountered during this process are further discussed and results are evaluated according to performance and the possibility of producing portable games.

The report is written in Swedish.

Keywords: real time graphics, multi platform, mobile phones, mobile gaming.

INNEHÅLLSFÖRTECKNING

FIGURINDEX

1 Inledning

Spel till mobila enheter omsatte år 2002 500 miljoner USD, och omsättningen väntas stiga till två miljarder USD till 2006 enligt en rapport från analysföretaget Alexander Resources [39].

Mobiltelefoner blir allt mer avancerade, och de innehåller allt fler funktioner, såsom kalendrar, webbläsare och kameror, och inte minst spel. Många nya mobiltelefoner är så avancerade att de kräver ett operativsystem för att ha kontroll på alla funktioner.

Det finns idag flera olika operativsystem till mobila enheter. Till handdatorer finns exempelvis Palm OS, Linux och Microsoft Pocket PC och till mobiltelefoner finns bland annat BREW, Symbian OS och Microsoft SmartPhone. Olika enheter har dessutom olika hårdvaruförutsättningar. Bildskärmarna är olika, med olika storlekar, olika antal pixlar, olika antal färger. Även där antalet färger är samma kan kodningen av färgerna vara olika. Sättet användaren interagerar med enheten kan också vara olika. Vissa enheter har tryckkänslig skärm där användaren kan använda sig av en särskild penna för att göra inmatning medan andra enheter enbart har knappar. Hur många dessa knappar är och var på enheten de sitter skiljer sig också.

Olika operativsystem har olika krav på de program som körs. Som vi senare ska se i fallet Symbian innebär det exempelvis krav på att variabler inte får vara globala eller statiska, samt att minnesutrymmet på stacken är mer begränsat än på operativsystem för stationära datorer.

Sammantaget gör detta att den som utvecklar en applikation måste utveckla stora delar av den från grunden för varje nytt operativsystem hon eller han vill stödja, och även i viss mån anpassa applikationen efter varje existerande enhet, även om operativsystemet skulle vara detsamma.

1.1 Syfte

Syftet med detta arbete är att undersöka om man kan utöka det existerande grafikbiblioteket GapiDraw, så att det även skall kunna användas för utveckling av spel och andra applikationer med krav på hög grafisk prestanda, för mobiltelefoner och handdatorer som kör Symbian, men ändå använda sig av en gemensam kodbas. GapiDraw fungerar i dagsläget på stationära datorer, handdatorer och mobiltelefoner med operativsystemen Microsoft Windows, Microsoft Pocket PC respektive Microsoft SmartPhone. Dessa operativsystem har samma tillverkare, och vi vill se om man kan utöka detta till att även fungera för mobiltelefoner med Symbian OS, som på flera sätt skiljer sig från dessa som redan stöds. Dessutom vill vi belysa och dokumentera problem som kan uppstå när man portar ett program från en plattform till en annan, i detta fall från Windows till Symbian.

1.2 Problemformulering

Är det möjligt att utöka grafikplattformen GapiDraw så att den går att använda för andra operativsystem till mobiltelefoner än de som redan stöds?

1.3 Problemavgränsning

.
Att få ett program att köra på ett annat operativsystem kan innebära många förändringar. Exempelvis har de olika fönstersystem, ibland en annan kompilator och extra bibliotek som finns till en plattform kan saknas på andra. För att lägga till ytterligare en plattform till GapiDraw valde vi att implementera GapiDraw på Symbian. Detta val gjordes i samråd med vår handledare då Symbian i skrivande stund var det enda alternativ till Microsoft Smartphone som hade utvecklingsmöjligheter i C++ för mobiltelefoner. Palm har även börjat tillhandahålla sitt operativsystem för mobiltelefoner, men det är inget som finns i Sverige ännu och BREW är inte ett öppet system.

1.4 Målgrupp

GapiDraws målgrupp är utvecklare av grafiska applikationer med höga krav på prestanda. Typiska sådana är spelutvecklare, som hellre lägger tiden på att utvecklar själva spelet än implementerar grafikrutinerna. Genom att stödja flera plattformar kan man vidga denna definition till utvecklare som dessutom vill skriva program för flera plattformar utan att behöva göra de anpassningar som annars krävs.

1.5 Ordlista

3G	Tredje generationens mobiltelefoner.
API	Application Programmer Interface, uppsättning funktioner och parametrar för en programmerare som använder sig av exempelvis ett bibliotek eller operativsystem.
Bluetooth	En teknik för att skicka data via radio över korta distanser.
BMP	Bitmap, punktavbildad bild.
dll	Dynamic Link Library, ett bibliotek som ett eller flera program kan läsa in i minnet och anropa funktioner och hämta data ifrån.
fps	Frames Per Second, antalet bildskärmsuppdateringar som sker per sekund i en applikation.
GPRS	General Packet Radio Service, en teknik för att skicka paketdata via radiovågor.

LCD	Liquid Crystal Display, bildskärm som bygger på flytande kristaller, vars optiska axels riktning ändras när de utsätts för en elektrisk spänning
MDI	Människa-dator-interaktion
Pixel	En bildpunkt.
PNG	Portable Network Graphics, en fri algoritm för att komprimera bildfiler.
RGB	Red Green Blue, en färgkodningsstandard som bygger på att man anger hur mycket rött, grönt och blått en färg består av.
SDK	Software Development Kit, en uppsättning program och verktyg för att utveckla programvara till exempelvis ett annat program eller till ett operativsystem.
Series 60	Ett användargränssnitt för Symbian utvecklat av Nokia.
Sprite	En (liten) bild som används i ett datorspel, t ex som en del av en större bild.
UIQ	Ett användargränssnitt för Symbian utvecklat av UIQ Technology.
WLAN	Wireless Local Area Network, trådlöst nätverk.

2 Bakgrund och relaterat arbete

I detta kapitel tar vi upp relaterad forskning och annat arbete som är relevant för detta arbete. Vi tar upp hur spel har använts inom forskningen, beskriver utvecklingen av mobila spel, mobila operativsystem och grafikbibliotek med stöd för flera plattformar. Genom att beskriva dessa områden som ligger nära vårt arbete vill vi ge en bakgrund till vårt arbete och sätta in det i ett större sammanhang, och visa på vad som redan är gjort.

2.1 Historik - utveckling av mobila spel

Mobila spel, i bemärkelsen datorspel som är lätta att ta med sig och spela var som helst, har funnits i över 25 år. Utvecklingen av mobila spel är intressant i sammanhanget, då GapiDraw ofta används just till spelutveckling. Bland de tidiga speltillverkarna fanns flera leksakstillverkare, som Nintendo och Mattel. Nintendo satsar numera nästan helt på datorspel, medan Mattel har övergivit den branschen. De första elektroniska spelen hade lysdioder på olika positioner som tändes och släcktes i spelet. Några år därefter kom LCD-spelen där skärmen istället bestod av flytande kristaller, men det var fortfarande bara vissa förutbestämda former som kunde visas på skärmen. Med GameBoy kommer den första handhållna enheten med utbytbara spel, och sedan Lynx med färgskärm. Mobila spel finns även på enheter som inte främst är avsedda för spel. Exempel på sådana enheter är handdatorer och mobiltelefoner, och ett tidigt exempel är Nokias 6100-serie.

Nedan följer några viktiga milstolpar inom utvecklingen av mobila spel.

1976 Mattel släpper Auto Race, det första helt elektroniska handhållna spelet, utan rörliga delar, och med lysdioder som visade bilarnas position. [23]

1980 Nintendo släpper 5 titlar i Serien Game & Watch, med LCD-skärmar. [19]

Figur 1 – Oil Panic

1983 Nintendo släpper Oil panic, i serien Game & Watch. Nu med dubbla skärmar.

1989 Nintendo släpper GameBoy, den första mobila spelenheten med utbytbara spel. GameBoy kan visa fyra gråskalor och har fyra kanalers ljud. 160*144 pixlar. Upp till fyra enheter kan kopplas samman via kabel. [29,18]
Atari Corp släpper samma år Lynx, den första handhållna spelenheten med färgskärm, och även den med utbytbara spel. 4096 färger med max 16 färger per rad på skärmen och fyra kanalers ljud. [22]

1990 Sega släpper GameGear som kan visa 32 färger och med en skärmstorlek på 160*146 pixlar.

1993 Apple Computer, Inc släpper Newton och myntar uttrycket personal digital assistant, PDA. Den första handdatorn med handskriftsigenkänning. Även spel utvecklades för Newton. Svartvit skärm. [28]

1998 Nokia 6100-serie släpps, med tre spel, Snake, Logic och Memory. Bland de första telefonerna med inbyggda spel.
Nintendo släpper GameBoy Color, med upp till 56 färger på skärmen. I övrigt i stort sett samma som den äldre versionen.

2000 Cybiko Inc släpper Cybiko, en maskin för trådlösa spel och kommunikation. Förutom spela spel, kan man chatta med andra användare inom räckhåll (upp till 100 m).

2001 Nintendo släpper GameBoy Advance, en bärbar spelkonsol som visar 32000 färger och 240*160 pixlar.

.

Figur 2 - Nokia N-Gage

2003 Nokia släpper N-Gage, en mobiltelefon med Symbian OS designad för spel. [26]

2.2 Mobila operativsystem

För avancerade mobiltelefoner finns idag tre större operativsystem, Symbian OS (Series 60 och UIQ), Microsoft Windows Mobile och PalmOS för handdatorer med

telefonfunktion. MontaVista erbjuder Linux i en mobiltelefonversion och SavaJe ett operativsystem helt i Java. Dessutom finns DMSS/BREW som är ett stängt operativsystem där endast program godkända av mobiloperatören kan installeras.

Analysföretaget Nomura förutspår i en rapport [8] att Symbians andel av operativsystemsmarknaden för avancerade mobiltelefoner kommer att ligga stabilt runt 40 %.

Mobiltelefoner är små enheter med begränsade resurser, vilket ställer krav både på operativsystemet och på de program som körs. Processorsnabbhet, minne och risken för att batteriet tar slut när som helst är faktorer som måste beaktas.

2.2.1 Symbian

Symbian är utvecklat ur EPOC, operativsystemet för Psions handdatorer. Symbiantelefoner har, förutom själva operativsystemet, ett användargränssnitt. Detta användargränssnitt finns i olika varianter, Series 60 och Series 90 som är utvecklat av Nokia, men som även licenseras till andra tillverkare, och UIQ, som främst används av Sony Ericsson och Motorola. Det gör att alla funktioner för interaktion med användare inte finns tillgängliga på alla telefoner, eller är implementerade på olika sätt.

2.2.2 PalmOS

Palm OS[41] är operativsystemet i Palms handdatorer sedan 1996, och är ett av de större mobila operativsystemen. Sedan 2002 är Palm uppdelat i två bolag, Palm One som tillverkar handdatorer och Palm Source som tillverkar operativsystemet. Palm Source licensierar operativsystemet även till andra tillverkare, som använder det i sina handdatorer och mobiltelefoner.

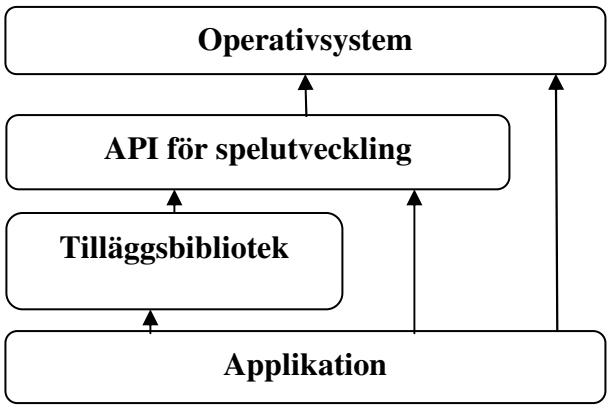
2.2.3 Windows Mobile

Windows Mobile är ett samlingsnamn för olika varianter av Windows CE som är ett operativsystem framtaget av Microsoft. Den första versionen av Windows CE släpptes 1996[40], men det var först med version 2.1 som Windows CE innehöll stöd för palm-sized PC:s, enheter som vi idag kallar för PDA:er. Tidigare hade endast stöd för så kallade handheld pc:s funnits, men nu började Microsoft konkurrera med Palm som tidigare satsat på denna typ av datorer.
För PDA:er med telefonifunktioner finns Pocket PC phone edition och för mobiltelefoner med viss handdatorfunktionalitet finns Smartphone. Båda varianterna har Windows CE som grund, men användargränssnittet och funktionaliteten skiljer sig åt på olika sätt.

2.3 Grafikbibliotek med stöd för flera plattformar

I det här stycket kommer vi att ta upp några exempel på grafikbibliotek som har lite olika typer av plattformsobeoroende. Anledningen till att vi tar upp det här är för att visa vad som finns och vad de olika biblioteken har att erbjuda om man vill utveckla grafiska applikationer för flera plattformar. Gemensamt för alla är den något förenklade situationen som visas nedan i nästkommande bild. Situationen är oftast att applikationsprogrammeraren jobbar mest mot spelutvecklingsbiblioteket, men ibland måste använda funktioner direkt från operativsystemet. Orsaken till att programmeraren måste göra detta kan till exempel bero på att funktionen denne vill använda inte finns med i det använda biblioteket. Så fort man använder sådana funktioner blir följden troligtvis att programmet inte längre går att kompilera för andra operativsystem. Givetvis kan applikationsprogrammeraren välja att göra på olika sätt för olika plattformar, men att sätta sig in i en ny plattform är ofta tidsödande och arbetskrävande.

För att få sitt API så plattformsoberoende som möjligt bör så mycket som möjligt ingå för att försöka ta bort pilen längst till höger i bilden nedan och därmed det direkta beroendet av operativsystemet.



Figur 3 - Arkitektur för plattformsoberoende spelbibliotek

2.3.1 Java

Java är ett programspråk utvecklat av Sun Microsystems, Inc, och idén är att man ska kunna skriva ett program och sedan kompilerar det till bytekod, som sedan kan köras på flera plattformar i en virtuell maskin. Den virtuella maskinens uppgift är att översätta bytekoden till anrop hos det aktuella operativsystemet som den körs på. Det finns en särskild version av Java för enheter med begränsade resurser, Micro Edition (J2ME).

Java är alltså inte ett grafikbibliotek. Det är ett programmeringsspråk framtaget för portabilitet. Vi tar ändå upp Java då det har många färdiga klasser för grafikhantering,

och sålunda kan användas för spel på flera plattformar. Det innehåller funktioner för bildinläsning, bildmanipulering och bilduppritning.

Nomura [8] anser att Java kommer att få det svårt p g a prestandaproblem och att det flera tillverkare har olika implementationer, vilket leder till att idén om att Java skulle vara plattformsoberoende faller i viss mån. Även Sangappa med flera pekar på prestandaproblem och skillnader i minnesåtgång på olika hårdvaruplattformar även om man använder sig av så kallad Just In Time-kompilering [10].

2.3.2 Mophun

Mophun är en spelutvecklingsplattform för mobila spel och är utvecklad av Synergenix [36]. I likhet med Java kompilerar man sitt program för en virtuell maskin (Mophun RTE) och därigenom uppnås ett plattformsoberoende. De enheter som stöds i dagsläget är samtliga mobiltelefoner som använder version 6.1 eller högre av Symbian samt några andra av Sony Ericssons modeller. Mophun består av fyra delar:

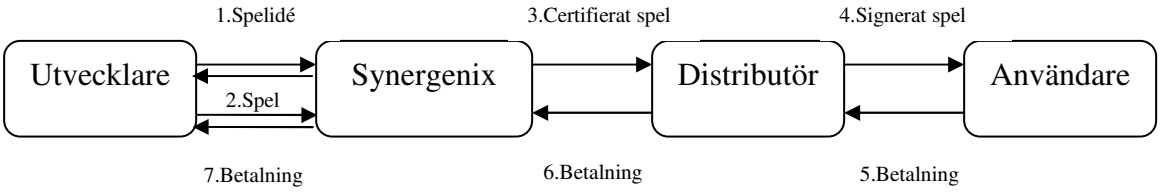
Mophun *RTE* (Run Time Engine)
- En virtuell maskin som körs på telefonen

Mophun *API*
- Ett API som ger tillgång till valda delar av Mophun RTE

Mophun *SDK*
- Paket med verktyg för programutveckling som ger tillgång till hela Mophun API

Mophun *VST* (Vendor Signing Tool)
- Ett verktyg för distributörer som låser applikationen till en enheten och en RTE samt att den gör applikationen spelbar.

Mophun är inte bara en utvecklingsmiljö, utan även ett distributionssystem där digitalt signerade spel har verifierats och godkänts av Synergenix, och nedladdade spel är låsta och kan sedan endast köras på en viss mobiltelefon. För att kunna göra kommersiella spel som använder Mophun (även kallade gamelets), måste man som utvecklare gå med på vissa villkor som bland annat innebär att man måste låta Synergenix certifiera spelen. Man måste dessutom distribuera spelen antingen direkt genom Synergenix eller via en auktoriserad distributör. Denna distributör är innehavare av en Mophun VST och signerar varje spel till beställaren (låser spelet till en enhet). Synergenix tillhandahåller alltså inte enbart ett utvecklingspaket, utan även en slags kvalitetskontroll och ett standardiserat sätt att komma ut med produkten på marknaden.



Figur 4 - Modell över distributionen av Mophun gamelets

Själva Mophun API:t är gjort för C och består av många olika moduler. Förutom grafikmoduler för 2D och 3D finns moduler för ljud, trådning, minneshantering, strömmar, http, sms med mera. Delen av Mophun API som behandlar 2D-grafik innehåller dels vanliga operationer som att fylla rektanglar, sätta pixlar, rita linjer och polygoner, men även stöd för sprites (inklusive kollisionsdetektering) och animerade tilemaps.
Alla resurser såsom till exempel bilder och ljud kompileras med en speciell resurskompilator och länkas sedan in i binärfilen för programmet.

2.3.3 SDL

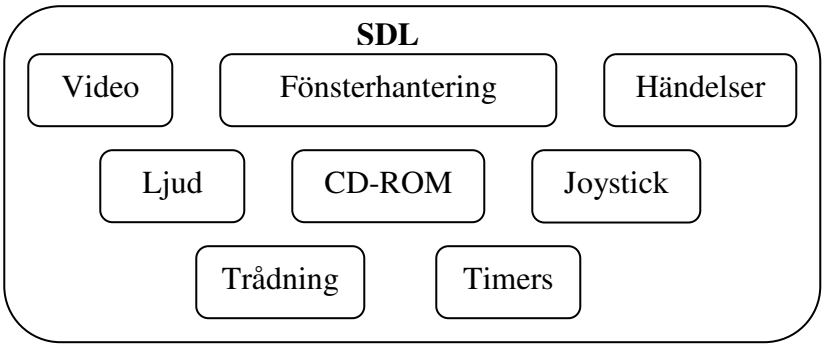
1998 påbörjade Sam Lantinga sitt arbete med Simple DirectMedia Layer [30]. Han fick idén till biblioteket när han portade en applikation från Windows till Macintosh. Kort därefter började han arbeta på Loki Games som senare släppte titeln ”Civilization: Call to Power” för Linux som bygger på SDL. Efter det har projektet över åren utvecklats till att innefatta ett hundratal personer som bidrar med olika saker. Vissa med programmering (bl.a. portning till andra plattformar), andra med dokumentation och/eller annat administrativt arbete.

SDL, är ett flerplattformsbibliotek som ger tillgång till tangentbord, ljud, bildskärm, cd-läsare, multitrådning, timrar och spelkontroller. Skärmdelen innehåller funktioner som hanterar skärmåtkomst och operationer på surfacestrukturer. Bland dessa operationer finns till exempel möjlighet att rita enskilda pixlar och rektanglar, beskärning av bilder, blitfunktionalitet med maskfärg och alphablandning samt inläsning av bmp-filer till en surfacestruktur.

Önskar man ytterligare funktionalitet som till exempel bildinläsare för andra bildformat finns påbyggnadsbibliotek till SDL att tillgå (i detta fall SDL_Image). Bland övriga tilläggsbibliotek kan nämnas SDL_net som är till för nätverksapplikationer och SDL_mixer som är till för ljudmixning.

I SDL finns även möjligheten att använda sig av OpenGL, ett bibliotek som finns till Windows, MacOS, UNIX och Linux, och som ger tillgång till 3D-grafik genom ett standardiserat API. SDL har, förutom stöd för nämnda plattformar, visst stöd för andra

plattformar inofficiellt. Bland dessa kan nämnas versioner för Windows CE och Symbian[31].



Figur 5 - SDL:s delar

2.3.4 ClanLib

ClanLib är ett flerpattformsbibliotek för spelutveckling, med stöd för Windows 98, Windows 2000, Windows XP och Linux, och det arbetas på en version för MacOS. Det licensieras till utvecklare under GNU Lesser General Public License, LGPL, vilket kortfattat innebär att ClanLibs källkod är fritt tillgänglig, och att biblioteket är gratis att använda, men att de spel som använder och har utvecklats med hjälp av ClanLib inte behöver vara gratis och kan låta källkoden förbli hemlig.

Det hela började som ett enmansprojekt med Nordlib 1995, som grundades på Microsoft Game SDK (DirectX 1.0), berättar grundaren Magnus Norddahl[1]. Utvecklarna var inte helt nöjda med Nordlib, och började istället utveckla ClanLib utifrån detta, och man lade nu också in stöd för Linux, med DirectDraw och XLib.

På en låg nivå hanterar biblioteket bildskärm, ljud, inmatning, nätverk, filer och exekveringstrådar. På en högre nivå finns stöd för resurshantering, grafiska användargränssnitt och spelskript. Detta gör att spelutvecklaren slipper hantera plattformsberoende saker som bildinläsning och skapande av fönster, utan kan koncentrera sig på utvecklingen av spelet mot ett API som är likadant på de stödda plattformarna.

Spel som skapats med ClanLib kan skrivas med plattformsoberoende kod och sedan kompileras för de olika plattformarna.

[1] Intervju med Magnus Norddahl utförd på IRC (#clanlib,irc.freenode.net) 2004-01-22

2.3.5 MobileCore

MobileCore [17,24] är ett relativt nytt (från oktober 2003) bibliotek för mobila enheter utvecklat av Jacco Bikker, och för närvarande stöds PocketPC och Symbian. Det går även att köra på stationära Windowsdatorer, men det är inte optimerat för detta, utan det är främst för teständamål.

MobileCore distribueras under GNU General Public License, GPL, vilket innebär att om man använder sig av det får man inte ta betalt för sitt program, och man måste låta sin källkod vara allmänt tillgänglig.

Bibliotekets syfte är att abstrahera skillnaderna i de operativsystemberoende delarna, såsom bildskärmsminne, knapptryckningar och ljud. Det realiserar detta genom att ha två klasser, core och surface, med samma gränssnitt för alla plattformar. Klassen core tar hand om de delar som nämns ovan, medan surface hand har inläsandet av bilder från fil. Surface har också funktioner för att rita linjer, rektanglar, att kopiera bilder, som dock inte är plattformsberoende.

2.4 Mobila speltillämpningar

Starner med flera menar på att datorspel är ett unikt protoypverktyg för studier av människa-dator-interaktion [9]. De menar att underhållningsvärdet som spel för med sig gör att användare är mer villiga att utforska nya saker, som metaforer och teknik. Vidare listar författarna en del områden, som till exempel artificiell intelligens, computer supported collaborative work (CSCW) med mera, där nöjesplattformar har haft betydelse för upptäckter.

Vi listar här nedan några exempel på sådan forskning.

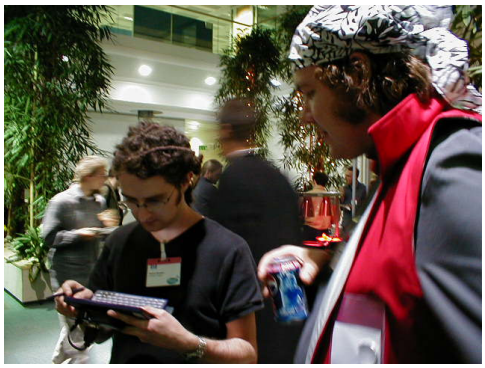
2.4.1 Pirates!

Gruppen PLAY på Interaktiva Institutet byggde 2000 ett fleranvändarspel för handhållna enheter som kommunicerade med varandra trådlöst via WLAN. Spelet kallades för ”Pirates!” [6], där varje spelare var kapten på ett eget skepp. Uppdrag i spelet bestod till exempel av att leta efter skatter på olika öar, idka byteshandel med varor och utkämpa sjöslag mot andra spelare.

Syftet med Pirates var att visa på hur man med teknik som kan känna av närhet till andra föremål, kan få in nya designimplikationer till datorspelsdesign. Detta ska bidra till en ökad spelupplevelse sett ur ett socialt perspektiv och mer likna traditionella spel.

För att kunna undersöka detta användes handdatorer utrustade med sensorer för att känna av att andra enheter var i närheten. För att kunna utföra vissa saker, som till exempel gå i

land på en ö för att leta efter skatter, var man tvungen att fysiskt förflytta sig i rummet till en plats som symboliserade den ön. Detsamma gällde för att ge sig in i ett sjöslag med en annan spelare. All interaktion med spelet sköttes via ett grafiskt gränssnitt på den handhållna datorn.



Figur 6 - En spelare och sitt skepp i Pirates!

Figur 7 - En av öarna i Pirates!

2.4.2 OpenTrek

Fler och fler mobila enheter har inbyggt stöd för någon form av trådlös kommunikation såsom GPRS, 3G, WLAN och Bluetooth. Johan Sanneblad och Lars Erik Holmquist [11] menar att dessa egenskaper skapar nya möjligheter för interaktivt nöje och samarbete. De nämner Cybiko [14] som den enda spelmaskinen som designats i huvudsak för trådlösa spel men spår samtidigt att fler liknande maskiner kommer i framtiden när tekniken fått ordentligt fotfäste och mycket riktigt finns idag fler enheter på marknaden [26,38].

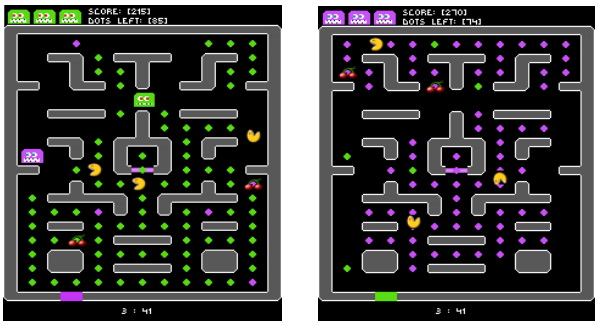
Författarna menar även att det är mycket svårare att komma igång med utveckling på dessa mobila enheter jämfört med stationära datorer på grund av avsaknaden av bibliotek, som till exempel DirectX[15], SDL och ClanLib (beskrivs senare i rapporten). Detta medför i sin tur att det blir svårt att hinna med att utveckla applikationer som använder sig av denna teknik i tidsbegränsade akademiska projekt, såsom uppsatser och kurser. Att man trots tidsbegränsningen måste använda sig av själva enheterna istället för att simulera applikationer på stationära datorer vid test av mobila MDI-tekniker, beror enligt Sanneblad och Holmquist på faktorer som andra förutsättningar för användarinput(t.ex. tangentbord mot pekpenna) och andra användningssituationer (t.ex. stillasittande framför datorn mot scenarion som bussresor och promenader).

Mjukvaruplattformen OpenTrek togs därför fram på Viktoria institutet för att underlätta utveckling av applikationer för mobila enheter, främst PocketPC, som är i behov interaktiv grafik och olika typer av nätverkskopplingar. Numera finns även DirectPlay, en delmängd av DirectX, som har hand om nätverkskommunikation, även till PocketPC. OpenTrek består av två oberoende delar. En grafikdel, GapiDraw, som vi beskriver mer utförligt i nästa kapitel och en nätverksdel som har hand om all nätverkskommunikation.

2.4.3 Multipla skärmar

Danesh med flera beskriver Geney [7], en applikation som är framtagen för att bland annat studera effekten som handhållna datorer har på kollaborativt lärande. Hela Geney består av en central dator och ett flertal Palmdatorer och är tänkt att hjälpa skolbarn att förstå genetiska koncept och att lära sig problemlösning och samarbete. Detta skall ske genom att tillsammans försöka avla fram en fisk med en bestämd uppsättning gener. Eleverna har var sin handdator som var för sig representerar en fisk med en speciell uppsättning arvsanlag. För att klara av uppgiften måste eleverna avla fram den förutbestämda fisken genom att titta på varandras skärmar och jämföra fiskar för att därefter para lämpliga fiskar med varandra. Denna parning görs genom att rikta två enheters IR-portar mot varandra.

Future Applications Lab på Viktoriainstitutet forskar också en del på trådlösa flerspelarspel och hur man kan designa spel för att främja samarbete mellan spelarna [12]. Ett spel som de har studerat är ”Pac-Man must die!” för PocketPC. Spelet, som vid en första anblick ser ut precis som den gamla klassikern Pac-Man, skiljer sig från originalet genom att två till fyra spelare styr var sitt spöke istället för Pac-Man. Målet med spelet är att vara först med att ta samtliga prickar som har samma färg som sitt eget spöke och dessa prickar är utspridda över alla deltagares skärmar. För att klara spelet måste man som spelare därför placera spöket på speciella teleportzoner som skickar iväg spöket till en annan deltagares skärm. Spöket styrs fortfarande med den egna handdatorn men för att kunna veta vart man ska styra måste man titta på den skärm dit spöket förflyttats.



Figur 8 - Pacman must Die! - Två spelare och deras skärmar.

Bild återgiven med tillstånda av Johan Sanneblad

Interaktionen som denna speldesign ger upphov till diskuteras av Johan Sanneblad [13] och de tror även att man kan generalisera resultaten till andra domäner som till exempel applikationer som stödjer samarbete för folk som befinner sig på samma plats.

3 GapiDraw

I det här stycket beskriver vi vad GapiDraw är för någonting. Vi berättar om hur det har utvecklats och hur det är uppbyggt. Det började som en påbyggnad på Microsofts GameAPI (GAPI) och därför börjar vi med en liten kort beskrivning av GAPI.

3.1 GAPI

Det finns ett SDK för Pocket PC och SmartPhone som heter GAPI (GameAPI)[20] vilket innehåller en funktion som kan användas för att få en pekare direkt till minnet för enhetens display. En annan av GAPI:s funktioner kan användas för att ta reda på antal byte man ska stega i minnet för att ta sig en pixel till höger på skärmen och samma sak för en pixel neråt (x- och yPitch). Med hjälp av GAPI kan man även ta reda på andra modellberoende egenskaper såsom till exempel färgdjup, bredd och höjd på skärmen i pixlar och huruvida displayen är buffrad eller inte.

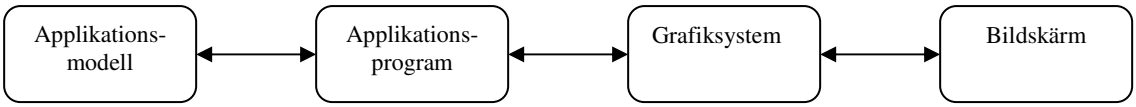
Förutom funktioner för displayen har GAPI även en funktion som låser samtliga hårdvaruknappar och skickar knapphändelserna till sin egen applikation istället för att t.ex. föra ett annat program till förgrunden.

GAPI är väldigt hårdvarunära och innehåller inga grafikrutiner för att rita linjer eller blitta bilder. Att bara sätta en viss pixel till en vald färg kräver att man måste känna till dels hur man skall stega i minnet för att komma till pixeln (x- och yPitch), dels den aktuella displayens pixelformat.

3.2 Vad är GapiDraw?

GapiDraw är ett grafikbibliotek för Microsoft Windows, Microsoft PocketPC och Microsoft SmartPhone. Man kan använda samma programkod och kompilera för samtliga plattformar. Biblioteket hanterar de olika skärmstorlekar, färgdjup, pixelformat och skärmvridningar som olika enheter har [21].

Foley m fl [1] beskriver ett ramverk för interaktiv grafik enligt figur 9.



Figur 9 - Ramverk för interaktiv grafik

GapiDraw, och även de grafikbibliotek som beskrivits ovan, OpenGL och SDL:s grafikdel lägger sig i denna modell mellan applikationsprogrammet och grafiksystemet.

3.3 GapiDraws historia

Första versionen av GapiDraw släpptes 22 mars 2002, men historien börjar inte där, utan GapiDraw bygger på GapiTools, som togs fram för en kurs på IT-universitetet hösten 2001. GapiTools var dock inte optimerat för hög prestanda på handdatorer, utan det skrevs om för att öka prestandan och underlätta integration med nätverksplattformen OpenTrek. I samband med de stora förändringarna bytte GapiTools namn till GapiDraw.

GapiDraw implementerades i tre steg. Första steget var att få det att fungera med alla skärmvarianter. Denna tidiga version var inte prestandaoptimerad. Det andra steget var att optimera alla grafikrutiner genom att skapa funktions- och klassmallar för grafikrutinerna, så att kompilatorn genererar specifik kod för alla skärmformat. Det tredje och sista steget var att skapa GapiDraws struktur, med ett programmerargränssnitt som är lätt att använda för utvecklare.

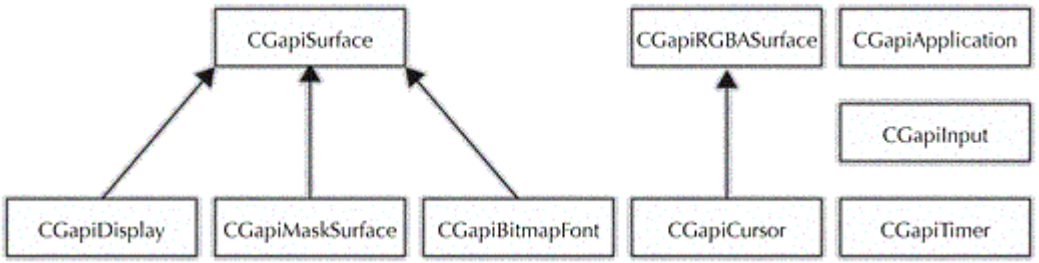
Utvecklingen av GapiDraw har sedan skett både för att optimera koden för snabbare grafikoperationer och för att lägga till fler funktioner. Nya funktioner har bland annat tillkommit efter önskemål från användare. Versionen som släpptes 9 maj 2003 hade så många nya funktioner och förbättringar att man valde att kalla den 2.0. Bland de nya funktionerna märks hårdvaruacceleration på stationära datorer med stöd för detta och kraftiga prestandaförbättringar.

GapiDraw har inte bara använts inom undervisning och forskning. Över hundra kommersiella spel har utvecklats för PocketPC med hjälp av GapiDraw, bland andra Warlords II – PocketPC edition och Pocket EverQuest.

3.4 GapiDraws uppbyggnad

GapiDraws kärna är ett dynamiskt länkbibliotek, en dll, med ett antal exporterade funktioner för hantering av grafik, bildskärm och inmatning, som ett program kan anropa. Till detta kommer CGapiApplication, som kan användas för att styra dll:en, och den beskrivs närmare nedan.

Följande klassdiagram visar GapiDraws uppbyggnad. De ingående klasserna kommer att beskrivas nedan.



Figur 10 - GapiDraws klasser

3.4.1 CGapiApplication

Klassen CGapiApplication är den del av GapiDraw som finns tillgänglig med källkod. Den innehåller funktioner som att ta hand om systemmeddelanden såsom knapptryckningar, penntryckningar på skärmen och inkommande samtal. Den låser enhetens knappar till applikationen och öppnar applikationen i fullskärmsläge. Detta sker olika på olika plattformar, så CGapiApplication ser olika ut beroende på operativsystem. Det står varje utvecklare fritt att ändra i CGapiApplication om denne vill att programmet ska bete sig annorlunda, och man behöver inte använda CGapiApplication för att kunna använda GapiDraw. Det rekommenderas att man skapar en egen underklass som ärver från CGapiApplication.

Teixeira de Sousa tar upp spelloopen, eller meddelandeloopen, som är typisk för spel i händelsestyrda operativsystem. Loopen ser ut enligt följande:

1. Se om det finns något meddelanden från systemet. Om det finns, hantera dem.
2. Kör spellogik, såsom förflyttningar, poängberäkning.
3. Rita upp på skärmen

Dessa steg gås sedan igenom tills programmet minimeras eller avslutas. CGapiApplication arbetar enligt denna princip.

3.4.2 CGapiInput

I denna klass sker ovannämnda låsningen av knappar genom funktionen OpenInput. GapiDraw har stöd för roteringen av skärmen (90, 180 eller 270 grader), och CGapiInput ser till att upp-, ner- och sidoknappar mappas om till skärmvridningen.

3.4.3 CGapiSurface

En bild i GapiDraw kallas surface, yta. En surface är ett objekt med en uppsättning funktioner för att initialisera och manipulera bilder. Exempel på manipuleringsfunktioner i GapiDraw är att sätta enskilda pixlar, rita linjer, skriva text och att rita fyllda och ofyllda

rektanglar. Man kan applicera olika effekter på utritningarna, och bland dessa kan nämnas genomskinlighet för en vald färg, rotationer, zoomningar, transparens och alphablandning. Bland övriga funktioner kan nämnas kollisionskontroll, som gör att man kan se om olika objekt kolliderar.
CGapiSurface hanterar även de olika pixelformaten som förekommer, d v s hur färger representeras i minnet. För närvarande stöds varianterna 565 (65536 färger), 555 (32768 färger), och 444 (4096 färger). Dessa siffror anger hur många bitar varje färg representeras med i RGB. I t ex 565 representeras rött med fem bitar, grönt med sex bitar och blått med fem bitar. Detta är oftast ingenting som användaren av en CGapiSurface behöver bekymra sig om då alla operationer där man skall ange en färg tar in en COLORREF som består av 8-bitar vardera för rött, grönt och blått. Konverteringen till det aktuella pixelformatet sköts av CGapiSurface.

3.4.4 CGapiDisplay

Bildskärmen hanteras med klassen CGapiDisplay. För att visa något på skärmen ritar man upp det via CGapiDisplay. CGapiDisplay hanterar också de olika möjliga inbyggda skärmbuffertarna. För enheter utan egen skärmbuffert skapar CGapiDisplay en egen medlemssurface som buffert och för enheter med inbyggd skärmbuffert pekar den bufferten på denna. Skärmbuffertens syfte är att undvika att man ritar direkt till bildskärmsminnet, eftersom det skulle leda till flimmer på skärmen.

GapiDraws objektorienterade struktur möjliggör att CGapiDisplay är en underklass till CGapiSurface. Man kan alltså göra samma operationer på CGapiDisplay som på någon annan surface.

3.4.5 CGapiTimer

På handdatorer, till skillnad från på stationära datorer, går det inte att låsa skärmuppdateringen till skärmens inbyggda uppdateringsfrekvens. Man vill göra detta för att få en konstant uppdateringscykel i applikationen oavsett processorhastighet. Gör man på det här sättet undviker man att skriva till skärmminnet under tiden skärmen uppdateras vilket eliminerar så kallad tearing.

För att få sin applikation att flyta med jämn hastighet utan möjligheten till skärmuppdateringssynkronisering finns CGapiTimer. Klassen löser detta genom att vid varje uppdatering av skärmen jämföra den faktiska tiden det tog jämfört med den avsedda. Om tiden det tog var kortare än önskat väntar den tills det är dags att köra igen.

3.4.6 CGapiBitmapFont

Med CGapiBitmapFont går det att skapa typsnitt för utskrift på skärmen. Även CGapiBitmapFont är en underklass till CGapiSurface. I CGapiDisplay finns två inbyggda

instanser av CGapiBitmapFont för att skriva text till skärmen för den som inte har behov av att skapa egna typsnitt.

3.4.7 CGapiMaskSurface

CGapiMaskSurface är en underklass till CGapiSurface som är speciellt lämpad för kollisionsdetektering.

3.4.8 CGapiRGBASurface

Detta är en särskild typ av surface, som inte är en underklass till CGapiSurface. Denna typ av surface stödjer bilder med en särskild alphakanal, och är särskilt lämplig för alphablandningar. Alphakanalen anger graden av genomskinlighet hos varje pixel i bilden, så om en bild med alphakanal ritas upp på en annan bild med hjälp av alphablandning lyser olika delar av den underliggande bilden upp olika mycket.

3.4.9 CGapiCursor

På stationära datorer kan CGapiCursor användas för uppritning av muspekare på skärmen. CGapiCursor är en underklass till CGapiRGBASurface.

4 Metod

Vi beskriver i detta kapitel hur vi har gått till väga för att svara på den inledande frågan och lösa vårt problem. Detta arbete är av teknisk karaktär, och därför är mycket av vår metodik inriktad på att hitta tekniska lösningar på problemet.

4.1 Förstudie

I förstudien fokuserade vi på att undersöka Symbian och dess förutsättningar. Vi studerade även GapiDraws uppbyggnad för att se vilka delar som gick att använda oförändrade, och vilka som måste modifieras för att passa samtliga plattformar. Detta arbete pågick parallellt med litteraturstudien.

4.1.1 Litteraturstudie

Vi har studerat dels tidigare relaterat arbete, såsom andra plattformsoberoende projekt och problem med plattformsoberoende utveckling, dels rent tekniskt hur Symbian och dess enheter är uppbyggd, och hur det skiljer sig från de plattformar där GapiDraw redan finns. Det senare har utgjort en betydande del av litteraturstudien. Det finns få relevanta böcker inom området, utan mycket av de eftersökningar vi har gjort har vi gjort på Internet [27, 16, 32], på tillverkares hemsidor, som Nokias och Ericssons hemsidor och på utvecklarforum såsom NewLC, och Forum Nokia, och naturligtvis Symbians hemsida. Vi har också studerat litteratur om datorgrafik.

4.2 Analys

Till att börja med måste vi ha klart för oss vad vi skulle göra och vad biblioteket skulle klara av, funktionellt och prestandamässigt samt vilka som ska använda sig av det. Därför analyserade vi problemet och olika lösningsmöjligheter. De slutgiltiga kraven har justerats under arbetets gång och under studier av vad vilka möjligheter Symbian erbjuder och inte erbjuder.

4.2.1 Bibliotekets uppgift och krav

GapiDraw ska vara ett högpresterande grafikbibliotek för de plattformar som stöds. Med högpresterande menar vi att det ska vara i nivå med eller bättre än de lösningar som redan finns. Det ska också vara plattformsoberoende i största möjliga mån. Med detta menar vi att det ut användaren, d v s den programmerare som använder sig av GapiDraw, ska krävas minimalt med arbete för att få en applikation som fungerar på en plattform att även fungera på en annan, t ex om man har gjort ett spel för Microsoft PocketPC ska det gå att få samma spel att fungera på exempelvis Symbian med minimalt arbete i form av kodändringar. Programmerarens gränssnitt ska vara likadant för alla plattformar.

4.3 Design

Eftersom det bibliotek vi valt att göra redan finns för andra plattformar gällde inte designen i någon större mån biblioteket som sådant. I designen av biblioteket har vi dock fått ta hänsyn till vilka funktioner som har varit möjliga och meningsfulla att implementera. Vi har också utvärderat vad i den existerande designen som var nödvändigt att förändra för att det ska fungera på olika plattformar. För att kunna göra detta har vi utfört en baklängeskonstruktion (reverse engineering) av GapiDraw.

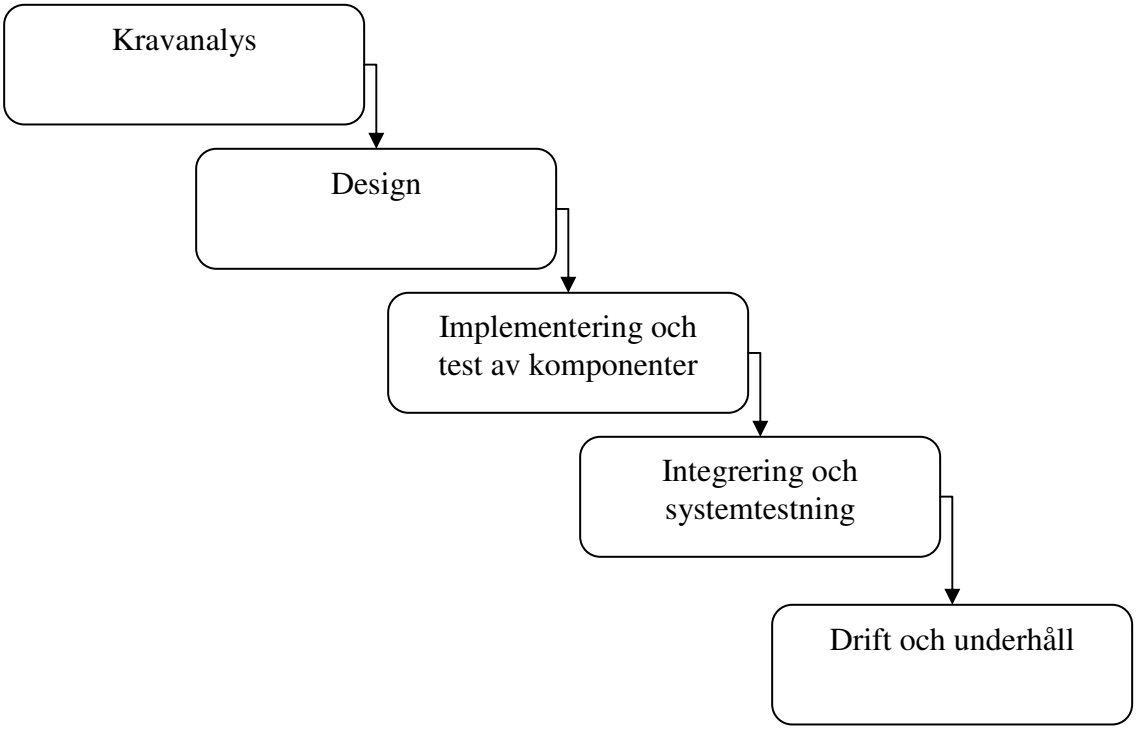
Designen har också varit tämligen låst av existerande API. Detta skulle bevaras i den mån det var möjligt, och inte förändras mer än absolut nödvändigt. I det fall förändringar vore nödvändiga skulle dessa även fungera på existerande plattformar.

4.4 Utvecklingsmetoder

Vid utveckling av program finns flera etablerade metoder. Vi beskriver några vanliga metoder här och sedan vilken vi valde och varför.

4.4.1 Vattenfallsmodellen

Vattenfallsmodellen är den första publicerade modellen för programvaruutveckling. Sommerville [4] beskriver vattenfallsmodellen som en femstegsmodell där varje steg slutförs innan nästa påbörjas, därav namnet vattenfallsmodellen, där vattnet rinner från en häll till en annan, och när man har kommit ner ett steg går man inte tillbaka till tidigare steg. Fördelen med metoden är att allt är väldefinierat och väldokumenterat när man går vidare till nästa steg, vilket gör att den passar bra för stora projekt med många inblandade. Nackdelen är att den kan vara oflexibel då man inte får gå tillbaka till ett tidigare steg, och det är svårt att ta med nya krav på programmet som kommer in senare i projektet.



Figur 11 - Vattenfallsmodellens olika steg

4.4.2 Boehms spiralmodell

I Boehms spiralmodell arbetar man sig, enligt Sommerville, flera steg genom en spiral, och går då igenom fyra viktiga steg i varje iteration:

1. *Målbestämmande*
   Mål för fasen definieras, och risker identifieras

2. *Riskbedömning*
   Riskerna bedöms och steg vidtas för att minska dessa risker

3. *Utveckling och verifiering*
   Utvecklingsmetod väljs, och arbetas efter, och resultatet verifieras

4. *Planering*
   Utvärdering av projektet. Om man väljer att fortsätta planeras för nästa varv i spiralen

Man går sedan igenom dessa steg om och om igen tills projektet anses färdigt. Själva utvecklings- och verifieringssteget kan bestå av en annan formell modell, exempelvis vattenfallsmodellen.

En central idé i modellen är riskanalysen, och åtgärder för att minimera riskerna. Med risker avses saker som kan tänkas gå fel i utvecklingen.

4.4.3 Extremprogrammering

Extremprogrammering är en relativt ny programmeringsmetod. Extremprogrammeringen är dock inte bara en metod, den innehåller allehanda tips för ett lyckat projekt.

Det finns tolv viktiga punkter i extremprogrammering [37]:

1. *Planering*
   Projektbeställaren berättar vad den vill ha och hur mycket värda de olika delarna är för denne, och bedömer vad som måste göras och vad som kan skjutas på framtiden.

2. *Små versioner*
   Släpp en enkel version snabbt och uppdatera den ofta under utvecklingen. Fel upptäcks tidigt och återkoppling från beställaren kommer in tidigt i processen.

3. *Metaforer*
   Använd metaforer under utvecklingen för att göra problemet begripligt.

4. *Enkel design*
   Designen ska anpassas efter dagens krav och inte bygga för vad som eventuellt kan komma senare.

5. *Testning*
   Programmet ska testas hela tiden under arbetets gång.

6. *Omstrukturering*
   Systemets design utvecklas ständigt. Detta görs genom att hålla koden ren och utan att duplicera kod.

7. *Parprogrammering*
   För att producera kod av högre kvalitet programmerar man två och två vid samma dator.

8. *Gemensamt ägarskap*
   All kod tillhör alla, och alla i projektgruppen ska kunna arbeta med vilken del som helst av systemet.

9. *Kontinuerlig integration*
   Systemets delar ska integreras flera gånger om dagen, för att tester ska kunna ske ofta.

10. *Fyrtiotimmarsvecka*
    Trötta programmerare producerar sämre kod, så övertid ska undvikas.

11. *Beställaren ska finnas på plats*
    Projektbeställaren ska ingå i projektgruppen, eller åtminstone finnas tillgänglig, då den som ska använda produkten är den som kan komma med krav och återkoppling

12. *Kodstandard*
    För att alla ska kunna äga koden gemensamt måste gemensamma kodstandarder användas.

4.4.4 Val av utvecklingsmetod

Vi har arbetat på ett sätt som ligger nära extremprogrammeringen. Vi har släppt versioner och testat dem kontinuerligt, och då integrerat nya delar i systemet, vi har tvingats omstrukturera designen när det har krävts och vi har delvis programmerat i par. Anledningen till att vi valde att arbeta på ett sätt som liknade extremprogrammering är att det passade bra i och med att det redan fanns ett färdigt system där vi kunde lyfta över små bitar åt gången och testa att dessa fungerade. Dessutom var det vitalt att alla ändringar vi gjorde inte bröt kompatibiliteten med de plattformar som redan stöddes, så alla ändringar i den gemensamma kodbasen kunde verifieras på samtliga plattformar.

4.5 Evaluering och prestandamätningar

Vi har verifierat funktionen av GapiDraw för Symbian genom att överföra och testköra existerande program för PocketPC till Symbian. Vi har också låtit utvalda externa betatestare testa biblioteket för att få reda på eventuella andra fel som kan ha uppstått. Dessa testare var 10 till antalet varav de flesta var bekanta med GapiDraw sedan tidigare (då inom utveckling för PocketPC) och handplockades från GapiDraws supportforum. Vid urvalet såg vi till att vi fick med användare av både Series60- och UIQ-telefoner. Vi skickade dels med testapplikationer för att se att provapplikationerna som vi gjorde fungerade på samtliga modeller men vi ansåg att de flesta buggar skulle upptäckas när användarna gjorde sina egna program.

Vi har jämfört GapiDraws prestanda med prestandan hos existerande grafikrutiner som finns inbyggda i Symbian samt att vi jämfört prestandan mot GapiDraw för SmartPhone.

Vi har verifierat att biblioteket fungerar på olika Symbianenheter, både UIQ och Series60.

5 Genomförande

Den första milstolpen vi satte upp i själva utvecklingsarbetet bestod av att, på ett klart och tydligt sätt, få klart för oss vilka delar av GapiDraw som skulle fungera utan ändringar på Symbian och vilka delar som var plattformsberoende och skulle behöva skrivas om. För att åstadkomma detta krävdes dels att vi satte oss in i den dåvarande källkoden för GapiDraw samt att vi fick utforska utvecklingsmöjligheterna i Symbian.

Vi ägnade de första veckorna åt att växelvis läsa in oss på Symbians C++ SDK och läsa igenom systemdokumentationen och källkoden för GapiDraw. Vi byggde även små testapplikationer för att bekanta oss med utvecklingsmiljön för Symbian.

5.1 Teknisk plattform

De verktyg vi använt i utvecklingsfasen av arbetet består av C++-SDK för Nokia Series 60 version 1.2 och UIQ:s version 2.0 och 2.1. I dessa ingår GCC som kompilator för ARM (kompilering för telefonerna) samt att vi använt Microsofts Visual C++ för att kompilera för Series60-emulatorn och Metrowerks CodeWarrior för att bygga för UIQ-emulatorn.

5.2 Grafik i Symbian

I Symbian finns det en inbyggd fönsterserver som hanterar ritning till skärmen, där bildskärmen är abstraherad till en `CWsScreenDevice`. Grafikoperationer i sin tur görs på en grafikmiljö, `CWindowGC`, med möjlighet att rita linjer, polygoner, bilder m m. Fönsterservern kan sedan uppdatera de delar av skärmen där en applikation som är synlig och har förändrats. Detta kräver att fönsterservern har koll på alla aktiva program ser till att inte flera program skriver till samma del av skärmen samtidigt.

Förfarandet ovan fungerar exempelvis menybaserade applikationer, som inte kräver snabb uppdatering av skärmen och där det är acceptabelt att uppdateringen kan dröja något. I exempelvis spel är detta sällan acceptabelt. Symbian har löst detta genom Direct Screen Access, där man begär tillgång till skärmen från fönsterservern, och får ett område tilldelat, d v s ett område som inte används av någon högre prioriterad tråd. CDirectScreenAccess tar hand om kommunikationen med fönsterservern, så att man därigenom blir varse när någon annan applikation vill använda sig av skärmen, t ex vid inkommande samtal.

Normalt sett kan applikationer inte komma åt hårdvaran på symbiansystem. Bildskärmen är ett undantag, där man kan få adressen till bildskärmsminnet. Man kan på så sätt helt gå förbi fönsterservern, men man måste då se till att hantera annat som vill använda skärmen, vid exempelvis inkommande samtal.

5.2.1 Grafikrutiner i Symbian

Symbian har ett antal funktioner för att rita olika geometriska figurer, text och bilder till skärmen. Dessa inkluderar sätta enskilda pixlar, rita linjer och kurvor, rita fyllda och ofyllda rektanglar, ellipser och polygoner.

Bilder, bitmapar, kan dras ut och tryckas ihop till ett visst område på skärmen, den kan också beskäras så att bara delar av den ritas ut. Delar av en bitmap kan göras genomskinlig genom att en extra bild av samma storlek, en maskbitmap, läses in där de områden som ska vara genomskinliga är vita.

5.3 Datatyper i GapiDraw

Vid en första genomgång av källkoden kunde vi konstatera att många typer och makron som användes inte ingick i standard C++ utan var windowsspecifika. Alla typer var dock antingen omdefinitioner av primitiva datatyper eller structar vilka helt enkelt kunde definieras om i en headerfil som vi började sätta ihop på en gång. Exempel på typer vi definierade i den filen är t.ex. HRESULT, TCHAR, RECT, DWORD med flera som vi satte till ekvivalenta typer i Symbian.
Vi såg även att stränghantering och minnesallokering endast använde sig standardbiblioteket i C vilket även finns att tillgå i Symbian. TCHAR var dock en sak vi fick fundera lite extra på. TCHAR är en windowstyp för tecken vilken kan vara av antingen ASCII eller UNICODE beroende på huruvida operativsystemet stödjer UNICODE eller inte och det bestäms när man kompilerar. Symbian stödjer i och för sig UNICODE, men standardbibliotekets funktioner för stränghantering använder sig enbart av vanlig ASCII. Eftersom vi ville behålla så mycket av den ursprungliga koden som möjligt definierade vi helt enkelt en TCHAR som en char och de funktioner som opererar på TCHARs till deras motsvarande ASCII-variant i vår definitionsfil. Anledningen till att vi befattade oss med TCHARs över huvudtaget var att man som användare av GapiDraw inte skall behöva göra mer ändringar i sin ursprungliga kod än nödvändigt för att kunna kompilera programmet för de olika plattformarna.

5.4 Operativsystemberoende delar

De flesta funktioner i GapiDraw är funktioner som gör olika operationer på minnesområden. Dessa operationer är inte på något sätt beroende av vilket operativsystem som körs utan är helt generiska. CGapiApplication, basklassen för applikationer som använder sig av GapiDraw måste dock skrivas om helt eftersom den innehåller anrop för att skapa fönster, ta emot meddelanden och annat som görs på olika sätt beroende på operativsystem.
Det vi identifierade i källkoden som skiljer sig åt mellan plattformarna var:

- Hantering av globala variabler i dll:er
- Hur man får åtkomst till skärmen
- Inläsning av bildfiler
- Låsning av hårdvaruknappar
- Trådhantering
- CGapiApplication

Efter detta första steg gjorde vi upp en handlingsplan för resten av portningsarbetet. Vi bestämde oss i samråd med vår handledare för att lösa problemen i ovan nämnda ordning. En mer ingående beskrivning på problemen och våra lösningar följer i efterföljande stycken.

5.4.1 Dll:er i Symbian och globala variabler

Symbian förutsätter att dll:er kan befinna sig i ROM även om det skulle kunna vara så att den i själva verket laddas i RAM. Eftersom man inte kan skriva till ROM så har dll:er inget datasegment och kan inte innehålla skrivbart statisk data.
För att komma runt det problemet finns ett antal lösningar beskriva och Symbians utvecklardokumentation listar följande tre sätt [35]:

- Lägga till den globala som en parameter till varje funktion som använder sig av den. Denna metod kan innebära väldigt stora förändringar i koden då det ofta måste ändras i mellanliggande funktioner också.

- Använda sig av thread-local storage (TLS). Varje dll får ett utrymme som rymmer en dataadress tilldelat för skrivbar global data. Detta utrymme kan användas för att innehålla en pekare till en struct vars medlemmar består av alla globala variabler. Detta tillvägagångssätt har dock en liten effekt på prestandan då man måste göra ett systemanrop som tar ungefär 20 instruktioner att utföra istället för en instruktion för att hämta en pekare [3].

- Spara pekaren till TLS i structar och klasser där man behöver den. Då slipper man prestandaförlusten som nämns i föregående lösning.

Det står även skrivet i dokumentationen att en kombination av de tre ovanstående teknikerna ofta används i praktiken för att minimera både omskrivning av kod och prestandaförluster. Tilläggas bör också att ovanstående strategier oftast endast används när man har redan färdigt skriven kod som skall portas till Symbian. Skriver man ny kod bör man designa med dessa restriktioner i åtanke och undvika statisk skrivbar data. Dessutom finns det en hel del andra tips för hur man bör skriva C++-kod som är så portabel som möjligt för att den skall kunna kompileras med olika kompilatorer [25, 2].

5.4.2 Vårt val av strategi

I GapiDraw används som sagt globala variabler och vi var tvungna att bestämma oss för hur vi skulle gå till väga för att göra koden symbiankompatibel. Vår första tanke var att vi skulle analysera koden och försöka uppskatta om vi skulle kunna använda oss av den första strategin, alltså göra sig av med de globala variablerna genom att istället skicka med alla nödvändiga variabler som parametrar tvärs igenom hela GapiDraw.

Efter cirka en veckas skissande på olika designförslag kom vi i samråd med vår handledare fram till att lämna denna strategi. Det skulle ta allt för mycket tid och innebära allt för mycket omstrukturering för att få till en snygg design. Även om vi inte använde oss av den här metoden så är det bra att ha det liggandes i bakhuvudet så att man tänker efter en extra gång innan man använder globala variabler vid framtida plattformsoberoende projekt.

Strategi tre, där man sparar undan pekaren till thread-local storage skulle inte heller fungera utan rätt stora ändringar då många ställen i programmet där globala variabler används är i funktioner som inte tillhör någon klass eller struct utan ligger för sig själva i olika namnrymder. Det man skulle behöva göra för att utföra den här strategin är att göra om alla dessa namnrymder (eng. namespaces) till klasser där funktionerna då skulle bli metoder och överallt där de tidigare funktionerna användes skulle en medlemsvariabel av denna nya klass behöva finnas. Efter någon dags skissande tyckte vi inte heller att den här varianten såg speciellt bra ut utan en större omskrivning av hela strukturen på GapiDraw vilket vi inte hade tid att genomföra.

Vårt val blev till slut att använda oss av thread-local storage. Tillvägagångssättet vi jobbade efter är det samma som användes för att porta webbläsaren Opera till Symbian [34]. Det går till som så att man först gör en struct (används enbart i symbianversionen) som innehåller alla globala variabler.

```
struct {
      BOOL g_initialised;
      DWORD g_xPitch;
      DWORD g_yPitch;
     etc.
} Globals;
```

Sedan gör man definesatser för att koppla namnet på variabeln som används i den ursprungliga koden till den som nu finns i thread-local storage:

```
#define g_initialised  Globalref(g_initialised)
#define g_xPitch       Globalref(g_xPitch)
#define g_yPitch       Globalref(g_yPitch)

#define Globalref(v)   ((Globals*)Dll::Tls())->v
```

Slutligen går man igenom koden och ser till att alla ställen där man deklarerar variabler som externa ändras till följande:

```
#ifdef __SYMBIAN32__
#include ”Globals.h”  // Fil där man har ovanstående deklarationer
#else
extern BOOL g_initialised;
etc.
#endif
```

För att snygga till koden lite gjorde vi så att vi alltid inkluderar en fil, “GapiDrawGlobals.h”, i källkodsfiler som använder någon global variabel oavsett vilket operativsystem vi ska kompilera för. I denna fil gör vi sedan val vid kompileringstillfället och bestämmer hur vi skall representera variablerna (riktiga globala på windowsplattformen och pseudoglobala på Symbian). Gör man på detta sätt slipper man många #ifdef makron utspritt över de olika källkodsfilerna.

5.5 Åtkomst till skärmen

För att kunna rita upp de färger man vill måste man även veta hur pixelformatet ser ut på de olika enheterna. När vi började med arbetet fanns det bara symbiantelefoner som klarar 4096 färger (12-bitars färgdjup) på marknaden. Även om bara 12 bitar används för färginformation tar varje pixel upp 2 bytes där de fyra första bitarna i varje pixel är oanvända, följt av fyra bitar för rött, fyra bitar för grönt och fyra bitar för grönt (xxxxRRRR.GGGGBBBB). Att det var så här det var upplagt fanns inte att hitta i någon användardokumentation, utan det hittade vi på olika diskussionsforum på nätet.

För att slippa gå via fönsterservern vid ritning till displayen på symbianenheter finns en odokumenterad klass TScreenInfoV01, som man kan använda sig av för att komma åt minnesadressen till skärmen samt skärmbredd och höjd i pixlar. För att ta reda på färgdjup kan man använda sig av klassen CWsScreenDevice. Färgdjupet är intressant för att veta vilket pixelformat man ska använda sig av. Med GAPI får man enkelt reda på det aktuella formatet genom ett funktionsanrop, men på Symbian kan man endast ta reda på antalet bitar som används för en färg, ingenting om hur pixlarna ligger. Det verkar dock vara så att samtliga 4096-färgers telefoner använder sig av xxxxRRRR.BBBBGGGG och att de nyare telefonerna med 65536-färger kör ett vanligt 565-format RRRRRGGG.GGGBBBBB.

Efter att man har minnesadressen till displayen, pixelformatet och skärmstorleken behövs ytterligare en sak innan man kan börja sätta pixlar på valfritt ställe på skärmen och det är x- och yPitch. Pitch-värden används för att specificera hur många byte man skall avancera i minnet för att flytta sig ett steg i x eller y-riktning på skärmen. Både xPitch och yPitch kan vara negativa. På PocketPC får man dessa värden genom GAPI, men det finns inte något motsvarande på Symbian. Vi förutsatte att adressen som man får i TScreenInfoV01 pekar till pixeln längst upp till vänster på skärmen och att xPitch är 2

(storleken i bytes för en pixel) samt att yPitch skulle vara xPitch * skärmens bredd i pixlar, vilket har fungerat på alla enheter vi har testat på.

Minnesadressen som returneras i TScreenInfoV01 skiljer sig dessutom åt på olika telefoner. Nokias Series60-telefoner returnerar alltid en pekare till en buffert som man måste tömma explicit för att rita om skärmen, medan Sony Ericssons UIQ-telefoner och Nokias communicator returnerar en adress direkt till bildskärmsminnet. På Nokias modeller med 4096 färger består dessutom de 32 första byten i adressen av en färgpalett som måste hoppas över för att komma till själva skärmminnet. Eftersom vi ville att GapiDraw skulle fungera på så många olika modeller som möjligt, behövdes ett sätt att kunna identifiera telefonmodellen under exekvering och detta går att lösa med hjälp av hårdvaruabstraktionsklassen HAL.

När vi kommit så här långt skrev vi ihop ett litet testprogram som hämtade adressen till skärmen och provade fylla skärmen med omväxlande svart, rött, grönt, blått och vitt för att se att vi hade korrekt pixelformat och skärmadress. Följande kodexempel fyller skärmen med en röd färg på en Sony Ericsson P800:

```
// Get address to display
TScreenInfoV01 screenInfo;
TPckg<TScreenInfoV01> sInfo(screenInfo);
UserSvr::ScreenInfo(sInfo);

// Validate address
TUint16* screenAddress = screenInfo.iScreenAddressValid ?
     (TUint16*)screenInfo.iScreenAddress : NULL;

int width  = screenInfo.iScreenSize.iWidth;
int height = screenInfo.iScreenSize.iHeight;

// Define colors (pixelformat 444)
//
//   byte1    byte2
// xxxxRRRR.GGGGBBBB

TUint16 black = 0;
TUint16 white = 0x0FFF;
TUint16 red = 0x0F00;
TUint16 green = 0x00F0;
TUint16 blue = 0x000F;
.
.
.
// Fill screen with red
for(int i=0; i < width * height; i++)
      screenAddress[i] = red;
```

För att ovanstående exempel ska fungera på en Nokia måste screenAddress plussas med 16 (32 bytes) för att hoppa över paletten samt att man måste skicka en redraw event till systemet på Series60 för att tömma skärmbuffern.

Efter att ha skrivit koden för att komma åt bildskärmsminnet var nästa steg att lägga in koden i en egen bildskärmsklass för Symbian och börja pussla ihop den med resten av GapiDraw. Eftersom GapiDraw redan hade stöd för två olika pixelformat (555 och 565) så var det inget större problem att lägga till ytterligare ett (444). Man bestämmer vid kompileringstillfället vilka pixelformat man vill stödja och när användaren skapar ett bildskärmsobjekt och anropar metoden OpenDisplay på det objektet, använder GapiDraw ett plattformsberoende anrop för att ta reda på den aktuella enhetens pixelformat och sätter en flagga för att markera detta. När användaren sedan anropar de olika metoderna i biblioteket kollas det upp på olika ställen vilket format man använder och denna information skickas vidare till en surfaceadapterklass. Denna klass tar sedan hjälp av olika konverteringsmetoder mellan RGB-värden (8 bitar vardera för rött, grönt och blått) och det aktuella pixelformatet, alphablandningar, och allt annat som är beroende av pixelformat. Surfaceadaptern i sig fungerar oberoende av pixelformat och använder sig av klasser som implementerar dessa ovanstående konverteringsmetoder och väljer den aktuella klassen som passar för enheten som koden körs på för tillfället.

5.5.1 Bildinläsning

Inläsning av bilder till CGapiSurface-objekt kan göras på olika sätt. Gemensamt för de olika windowsplattformarna är specialskrivna bildinläsare för bmp och png. Dessa läser in bilderna med minimal minnesåtgång genom att använda så kallade minnesmappade filer. Den konvertering som behövs för att få bilden på samma format som enheten man kör på utförs och resultatet sätts in direkt i surfaceobjektet. För att läsa in andra bildformat än bmp och png används COM-interfacet IPicture på stationära windowsdatorer och ett bildkonverteringsbibliotek ”imgdecmp” på windows CE. Då man använder sig av dessa alternativ används mer minne än om man använder bmp- eller pngbilder eftersom minne reserveras för temporära windowsbitmappar.

I Symbian finns något som kallas för en mediaserver som kan användas för att läsa in och konvertera ljud och bild till en applikation [33]. Vi ger användaren av GapiDraw två olika alternativ för att läsa in bildfiler. Antingen kompilerar man en symbianmultibitmapfil som innehåller ett antal olika bilder som sedan kan läsas in till symbianbitmappar (CFbsBitmap) eller så läser man in bildfiler av något av de stödda formaten direkt från valfri plats i filsystemet och använder mediaserverns konverteringsmetoder för att få den till en CFbsBitmap. Efter att ha fått den önskade bilden i en CFbsBitmap för vi över själva pixelinformationen i bilden till en CGapiSurface som är på samma format som displayen.

Önskar man av någon anledning inte använda sig av bildinläsarna som vi har implementerat går det utmärkt att implementera sina egna. Då får man skapa en CGapiSurface med samma bredd och höjd som bilden man ska läsa in och därefter anropa CGapiSurface::GetBuffer() för att få tillgång till pitch, pixelformat och själva databufferten.

5.5.2 Låsning av hårdvaruknappar

När det gäller möjligheter för inmatning från användaren av telefonerna skiljer sig Series60 och UIQ från varandra. Series60 är ett knapporienterat användargränssnitt där all interaktion sköts med knappsatsen till telefonen. UIQ-gränssnittet är först och främst menat att användas med en pekpenna, men några hårdvaruknappar brukar finnas på telefonerna.

I likhet med PocketPC finns det snabbknappar på de flesta symbianenheter som används för att t ex starta kalendern, öppna webbläsaren, starta kameran osv. Detta beteende är ofta inte önskvärt i fullskärmsspel, då man gärna vill använda så många knappar som möjligt till sitt spel (speciellt på Sony Ericssons P800/900 som bara har två knappar och en s k jog dial). Microsoft har löst problemet med hårdvaruknapparna i GAPI där man kan anropa en funktion som låser samtliga knappar och skickar knapphändelserna till sin egen applikation istället för att föra ett annat program till förgrunden. På Symbian finns dock inte någon motsvarande funktion, utan det man måste göra är att explicit anropa fönsterservern och låsa varje knapp för sig. Normalt får en applikation tre olika meddelanden för varje knapptryckning: EEventKeyDown, EEventKey och EEventKeyUp. För att undvika hårdvaruknapparnas normala beteende (t ex applikationsbyte) krävs att man tar hand om samtliga meddelanden och man måste låsa upp- och nertryckningar för sig och själva knapphändelsen för sig. Detta innebär alltså att för att låsa en knapp måste man dels meddela fönsterservern att man vill ha meddelanden för själva upp- och nertryckningarna av knappen och dels meddela att man vill ha själva knapphändelsen (motsvarande måste göras vid upplåsning av knappen).

Ett problem med det här tillvägagångssättet uppkommer i och med att olika telefonmodeller har olika identifierare för sina hårdvaruknappar. Eftersom det inte finns ett enkelt sätt att låsa samtliga knappar måste man antingen känna till dessa identifierare eller låsa samtliga möjliga knappar i Symbian vilket är omkring 200 stycken. Vi valde det senare tillvägagångssättet då det inte är möjligt att förutse knappuppsättningar på kommande enheter, och man skulle även vara tvungen att behandla varje enhet som stöds som ett specialfall.

5.5.3 Trådhantering

CGapiTimer är en klass i GapiDraw som innehåller funktioner för att få applikationen att köra i en konstant hastighet. De viktigaste funktionerna är StartTimer() där man anger hur många fps man vill att applikationen skall köra i och WaitForNextFrame() som sover så många millisekunder som det behövs för att man ska hålla sin fps. Det man gör i praktiken är då att starta timern innan man går in i sin spelloop, och efter att man gjort alla beräkningar och flippat backbuffern till skärmen anropar man WaitForNextFrame() sist i loopen.

Problemet på symbianplattformen är att det inte finns någon timer med tillräckligt hög noggrannhet. För att suspendera en aktiv tråds exekvering finns ett systemanrop User::After() som tar ett tidsintervall i mikrosekunder som argument. Man kan dock inte suspendera tråden med en större precision än ungefär 15ms på grund av att timern är kopplad till frekvensen på hårdvaruklockan som är 64Hz på existerande enheter (10Hz på emulatorerna). Detta innebär att ett avbrott genereras en gång var 1000/64≈15,5ms och vill man göra en User::After(1*1000) som skulle innebära suspendering 1ms kan resultatet bli att tråden sover ända upp till 15,5ms beroende på hur nära klockans avbrott man befinner sig.

På grund av ovanstående är det inte att rekommendera att använda denna metod för att kontrollera sin fps. Lösningen vi bestämde oss för att använda istället var att inte porta CGapiTimer över huvudtaget, utan använda oss av ett så kallat aktivt objekt, CIdle, som anropar CGapiApplication::ProcessNextFrame() så ofta den bara kan, det vill säga så länge inget aktivt objekt med högre prioritet har sagt till systemet att det vill köra, och låta användaren av GapiDraw basera sina animationer, spellogik m m på tid istället för att räkna bildskärmsuppdateringar.

5.5.4 CGapiApplication

Efter att grafikrutinerna och knapplåsningarna var implementerade i GapiDraw var nästa steg att skriva CGapiApplication, en klass som använder sig av själva GapiDraw-dll:en. Denna klass fungerar som en basklass som spel och applikationer ärver ifrån. All hantering av systemspecifika händelser som t ex inkommande telefonsamtal, varningar för låg batterinivå, minimering av applikationen o.s.v. sköts i CGapiApplication och applikationsutvecklaren kan koncentrera sig på själva applikationen och slippa ta hänsyn till dessa saker då de oftast skall hanteras på samma sätt. CGapiApplication gör alltså det absolut nödvändiga som t ex plocka ut meddelanden från meddelandekön, klassificera meddelandet och anropa en lämplig funktion för att ta hand om det. Många av dessa funktioner är virtuella och kan överlagras i applikationsklassen som ärver den. Som ett exempel på hur det fungerar beskrivs nedan hur en nedtryckning av pennan på skärmen behandlas:

1. Pennan trycks ner
2. CGapiApplication::HandleWsEventL() anropas av Symbians systemstruktur.
3. En koll görs på vad för slags händelse som har kommit och vi ser att det är en pekarhändelse.
4. CGapiDisplay::DisplayToLogical() anropas för att översätta punkten som händelsen inträffade på till den logiskt riktiga koordinaten (dessa är inte samma om vi har roterat skärmen).
5. Efter att ha klassificerat händelsen som en nedtryckning av pennan skickar vi punkten vi fick i steg 4 till den virtuella metoden CGapiApplication::StylusDown() och där bestämmer underklassen (applikationen) vad som skall ske härnäst.

6 Evaluering

I detta kapitel kommer vi att ta upp resultatet av vårt arbete, och försöker svara på vår forskningsfråga. Vi beskriver vad vi har överfört, hur portabel programkoden blir och redogör för de prestandamätningar vi har gjort.

6.1 Resultat

Vi har portat stora delar av GapiDraw till Symbian. De delar vi inte har portat är CGapiTimer, CGapiCursor och CGapiRGBASurface. CGapiTimer är inte portat då de inbyggda timerkretsarna i Symbianenheter har en precision på 1/64 s, vilket ger alltför dålig noggrannhet, och klassen skulle då inte uppfylla sitt syfte. CGapiRGBASurface är inte portat då Symbians bildinläsare inte har stöd för variabel alphainformation. Detta skulle dock gå att lösa genom att skriva en egen bildinläsare som stödjer detta. CGapiCursor är inte portad då den inte är meningsfull om man inte har någon mus eller annat pekdon för skärmen, och är dessutom en underklass till CGapiRGBASurface, som inte portades.

Eftersom vi går direkt på framebuffern i våra operationer mot displayen så fungerar det inte i emulatorerna. Det är dock genomförbart att skapa en lösning som fungerar på både riktig hårdvara och emulatorerna. Tillvägagångssättet hade i så fall varit att vi sparat backbuffern i Symbians egna bitmapformat CFbsBitmap och låtit Symbians egna grafikrutiner ta hand om den sista blitten till bildskärmen. Genom att ta ut pekaren till pixeldatan i symbianbitmapen kan vi spara undan den pekaren i en CGapiSurface av samma storlek och sedan använda den som vilken CGapiSurface som helst. Vi har gjort en sådan implementation, men den ingår inte ännu eftersom vi inte hunnit testa den tillräckligt väl för att kunna ingå i den första releasen av biblioteket.

CGapiApplication uppför sig precis likadant på symbianplattformen som i windowsversionen. Alla metoder och operationer som t ex InitInstance, CreateSysMemSurfaces och ProcessNextFrame anropas där de skulle ha anropats under Windows. Den största skillnaden man märker mellan plattformarna som användare av CGapiApplication är restriktionen mot globala variabler i symbianvarianten. Detta beror som sagt på att dll:er i Symbian inte får ha globala variabler, men hade man baserat CGapiApplication på en exe istället för en app hade det varit möjligt att använda sig av globala variabler. Detta hade medfört en mycket större arbetsinsats på grund av att man får väldigt mycket gratis genom att använda sig av en app och hade inte varit möjligt för oss att genomföra på den tid vi hade till förfogande. Men har man den här restriktionen i åtanke när man designar sin applikation så är det inget problem att få en kod som fungerar på bägge plattformarna.

6.2 Plattformsoberoende

Ett program skrivet för Win32-plattformen kan fortfarande inte kompileras rakt av för Symbian. Vissa förändringar krävs fortfarande, samt vissa tillägg. Se appendix B ”Porta en applikation från Win32 till Symbian” för ett exempel på vad som måste ändras för att applikationen ska kunna köras.

Att CGapiTimer inte är portat får konsekvensen att spel och andra applikationer som är tidsstyrda måste använda operativsystemets egna anrop för tidsfunktioner, och dessa ser olika ut i de olika operativsystemen.

Om man när man skriver sitt program för en plattform tänker på att göra sin kod portabel och använder sig av den struktur som ges i CGapiApplication är det dock endast små förändringar som måste göras. När det gäller att skriva sin kod portabel ska man framför allt tänka på att inte använda sig av globala variabler och att minimera utnyttjandet av programstacken. Man bör också spara bilderna som filer för inläsning istället för som resurser. Har man skrivit sin applikation med detta i åtanke är det få detaljer som måste ändras, och dessa gäller enbart skapandet av själva applikationen och öppnandet och initialiserandet av bildskärmen (CGapiDisplay::OpenDisplay() tar olika parametrar i Symbian- och Win32-versionen), samt, som nämns ovan, tidsfunktionanropen.

6.3 Prestandamätningar

Vi har testat GapiDraws prestanda jämfört med funktioner som redan finns i Symbian, för att se om vi har uppfyllt vårt krav att det ska vara i nivå med eller snabbare än de redan existerande funktionerna samt att givetvis försöka svara på vår frågeställning. Testen utfördes genom att se hur lång tid det tog att göra ett stort antal upprepade operationer, för inte drabbas av onoggrannheten i de inbyggda tidtagningskretsarna.

```
Ttime before;
TTime after;
TUint width=backbuffer->GetWidth();
TUint height=backbuffer->GetHeight();
TRect screensize=TRect(0,0,width,height);
before.UniversalTime();
for(int i=0;i<1000;i++)
{
      m_BenchSurface->FillRect(NULL, RGB(128, 128, 128), 0, NULL);
}
after.UniversalTime();
diff1=(after.Int64().GetTReal() - before.Int64().GetTReal()) / 1000.0;

m_Gc->SetBrushColor(TRgb(128,128,128));
before.UniversalTime();
for(int i=0;i<1000;i++)
{
      m_Gc->DrawRect(screensize);
}
after.UniversalTime();
diff2=(after.Int64().GetTReal() - before.Int64().GetTReal()) / 1000.0;
```

Kodexempel på hur testerna gick till

Dessa tester avser inte ritning till skärmen, utan operationer på de olika klasserna, CGapiSurface och CFbsBitmap i systemminnet, där varje instans av CGapiSurface och CFbsBitmap hade en storlek som motsvarar bildskärmens storlek på en SonyEricsson P900, d v s 208*320 pixlar. Omskalningstesterna utfördes genom att ta en kvadrat på 100*100 pixlar och rita upp den som 200*200 pixlar. Testerna utfördes på en Sony Ericsson P900 och en Qtek 7070.

Test	**Symbianbitmaps**	**GD Symbian**	**GD Smartphone**
Rita fylld rektangel	1,5 ms	1,3 ms	2,6 ms
Rita bild	5,1 ms	3,8 ms	7,9 ms
Rita maskad bild	29,7 ms	7,5 ms	6,6 ms
Omskalning av bild	51,5 ms	5,2 ms	6,8 ms

Vi valde att göra just dessa tester då de är de funktioner som direkt motsvarar varandra i GapiDraw och Symbian. Ritning av en maskad bild sker dock på två olika sätt. GapiDraw använder sig av en maskfärg, d v s en valfri färg i bilden som är transparent, medan Symbian använder sig av en maskbild där de områden som ska vara transparenta i motsvarande bild är utritade.

6.4 Resultatanalys

Toki Tori (se http://www.aimproductions.be/) är ett plattformsspel för smartphone som använder sig av GapiDraw för alla grafikoperationer. Toki Tori utnyttjar resurskrävande flerlagersscrolling för bakgrunden som innebär att man måste utföra flera stycken fullskärmsblittar (en för varje lager).

Figur 12 – Skärmdumpar från Toki Tori

Eftersom spelet finns ute och fungerar bra kan man säga att GapiDraw klarar av att vara grafikmotor för spel till smartphones. Resultaten från vår evaluering visar att GapiDraw på Symbianplattformen har liknande prestanda som på Smartphone, vilket innebär att plattformen bör lämpa sig lika bra för alla typer av applikationer med höga krav på grafisk prestanda.

Vår forskningsfråga var: Är det möjligt att utöka grafikplattformen GapiDraw så att den går att använda för andra operativsystem till mobiltelefoner än de som redan stöds? Vårt arbete visar att det är möjligt för Symbian och borde även gå för andra operativsystem. Problem som är plattformsspecifika kan vara svåra att förutse och kommer sannolikt att dyka upp för varje ny plattform så det är väldigt svårt att generalisera svaret ytterligare. Något som är viktigt att tänka på när man inför ändringar är att kontinuerligt testa och göra prestandamätningar på samtliga plattformar. Vårt sätt att arbeta låg väldigt nära extremprogrammeringen och så här i efterhand tyckte vi att vi gjorde ett bra val då det ingick i metoden att kontinuerligt testa små bitar av programmet efterhand.
Den externa betatestningen fungerade inte riktigt tillfredsställande dock. Vi fick väldigt lite feedback och det som kom fram visade sig oftast inte vara direkt kopplat till GapiDraw. Några talade dock om stora prestandavinster mot andra kommersiella grafikbibliotek, men det är inte något vi hunnit verifiera på egen hand. En mer

genomtänkt testprocedur för externa testare är något som vi borde ha gjort för att få in mer användbar data känns det som så här i efterhand.

7 Slutsatser

Vår problemformulering var: Är det möjligt att utöka grafikplattformen GapiDraw så att den går att använda för andra operativsystem till mobiltelefoner än de som redan stöds? För att undersöka detta utökade och modifierade vi GapiDraw för att även stödja Symbian. Vi ställde som krav på GapiDraw att det skulle prestera minst lika bra som de inbyggda grafikrutinerna i Symbian.

I och med GapiDraw kan man skriva källkod för grafik som är till den största delen plattformsoberoende. Olika skärmstorlekar och olika format på resursfiler med mera måste fortfarande behandlas vid utvecklingen men det är svårt att komma ifrån. De prestandamätningar vi har utfört visar att GapiDraw presterar bättre än de inbyggda rutinerna i Symbian i samtliga testfall och likvärdigt med GapiDraw på windows smartphone.

Vi har även belyst vissa problem som en portning kan innebära, som till exempel problemet med globala variabler.

7.1 Framtida arbete

Utökar man GapiDraw till att innehålla mer än bara grafikrutiner skulle man kunna få ännu mer plattformsoberoende källkod. Detta skulle till exempel kunna vara ett API för ljud, multitrådning och nätverkskommunikation. Vi tror att det skulle vara intressant att undersöka vad mer man behöver stödja för att underlätta utveckling av plattformsoberoende mobila spel.

8 Referenser

Böcker

[1] Foley, J.D. m fl (1990). *Computer Graphics: Principles and Practice*, andra utgåvan, Addison-Wesley Publishing Company, USA.

[2] Giencke, P (1996) *Portable C++*, McGraw-Hill, USA

[3] Harrison, R (2003) *Symbian OS C++ for Mobile Phones* John Wiley & Sons, USA

[4] Sommerville, I. (2001), *Software Engineering*, sjätte utgåvan, Pearson Education Limited, Essex, Storbritannien

[5] Teixeira de Sousa, B. M. (2002), *Game Programming All in One*, Premier Press, Inc, USA

Artiklar

[6] – Björk, S, J Falk, R Hansson, P Ljungstrand,. (2001). *Pirates! Using the Physical World as a Game Board* Proceedings of Interact 2001

[7] Danesh, A., K Inkpen, F Lau, K Shu, K Booth, (2001) *Geney: Designing a Collaborative Activity for the Palm Handheld Computer*, in Proceedings of CHI 2001.

[8] – Windsor, R *Mobile software, Unchartered territory* (2003)
http://www.telecomsintelligence.com/Mobile%20OS%20mkt.pdf

[9] – Starner T, B Leibe B Singletary, K Lyons, M Gandy, J Pair. (2000) *Towards augumented reality gaming*. Proceedings of IMAGINA 2000

[10] Sangappa, S, K Palaniappan, R Tollerton, (2002) *Benchmarking Java against C/C++ for interactive scientific visualization*,
Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande

[11] – Sanneblad, J och L E Holmquist (2003)
OpenTrek: A Platform for Developing Interactive Networked Games on Mobile Devices
Proceedings of Mobile HCI 2003

[12] – Sanneblad, J och L E Holmquist (2003): *Designing collaborative games*
Proceedings of the SIGGRAPH 2003 conference on Sketches & applications: in conjunction with the 30th annual conference on Computer graphics and interactive techniques

[13] – Johan Sanneblad, J *"Why is everyone inside me?!" Interacting with Multiple Shared Displays*

Webbreferenser

[14] Cybiko - http://www.cybiko.com 040122

[15] DirectX - www.microsoft.com/directx/ 040206

[16] Forum Nokia - http://forum.nokia.com 040206

[17] Mobile Core - http://www.gamasutra.com/features/20031117/bikker_pfv.htm 040122

[18] Gamefaqs - http://www.gamefaqs.com/systems.html 040120

[19] Game & Watch - http://www.gameandwatch.com/ 031103

[20] GAPI - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/guide_ppc/htm/gx_bvrh.asp 040206

[21] GapiDraw - http://www.gapidraw.com 040206

[22] Lynx - http://www.atarilynx.com/faq.shtml 031103

[23] Handheld Museum- http://www.handheldmuseum.com/ 040120

[24] Mobile Core - http://sourceforge.net/projects/mobilecore 040122

[25] Mozilla portable c++ guide - http://www.mozilla.org/hacking/portable-cpp.html 040206

[26] N-Gage - www.n-gage.com 040206

[27] NewLC - http://www.newlc.com 040206

[28] Apple Newton- http://www-db.stanford.edu/pub/voy/museum/pictures/display/0-2-PDAs.htm 040122

[29] Nintendo - http://www.nintendo.com/corp/history.jsp 031103

[30] SDL - www.libsdl.org 040206

[31] SDL-Symbian - http://koti.mbnet.fi/~haviital/ 040206

[32] Symbian - http://www.symbian.com 040206

[33] Symbian (Media server) - http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/MediaServer/MediaServerOverview.guide.html 040206

[34] Symbian (Operas portningsstrategi) - http://www.symbian.com/developer/techlib/papers/Khopera/Opera_KeithHollis.htm 031120

[35] Symbian (Statisk data) - http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/EssentialIdioms/StaticData.html 031119

[36] Synergenix (Mophun) - http://www.synergenix.com 040206

[37] Extremprogrammering - http://www.xprogramming.com/what_is_xp.htm 040206

[38] Zodiac - www.tapwave.com 040206

[39] Analysrapport från Alexander Resources - http://www.alexanderresources.com/News%20&%20PR/$1.93%20Billion%20Mobile%20Games%20Market%20Will%20Be%20Dominated%20By%20a%20Few%20Key%20Players%20PR.htm 040120

[40] WindowsCE historik - http://www.hpcfactor.com/support/windowsce/ 040212

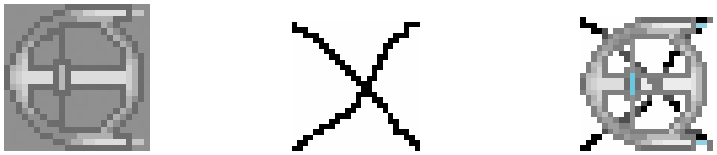
[41] PalmSource, PalmOne – http://www.palm.com/ 040212

Appendix A - Grafikkoncept

Teixeira de Sousa, B. M. [5] tar upp flera grafikkoncept som används inom spelutveckling. Nedan följer begrepp som tas upp för tvådimensionella spel. GapiDraw har inbyggt stöd för samtliga grafikkoncept som beskrivs nedan.

Maskfärg

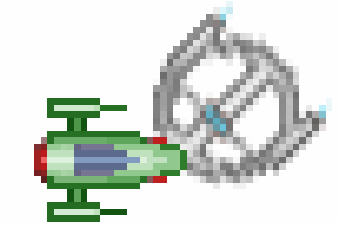
Bilder är i regel rektangulära. Om bilden i exemplet nedan skulle ritas upp som den är på skärmen skulle även bakgrunden komma med. Om man däremot sätter en viss maskfärg kan man välja att inte ta med denna färg vid uppritningen på skärmen. Nedan ses ett rymdskepp från spelet Earth Defenders, ett kryss, och hur det ser ut när rymdskeppet ritas upp på krysset med bakgrunden som maskfärg.



Exempel på användning av maskfärg

Kollisionsdetektering

Kollisionsdetektering går ut på att se om två bilder krockar, d v s överlappar varandra. Detta används t ex för att se om en spelare går in i en vägg eller en fiende blir träffad av ett skott.

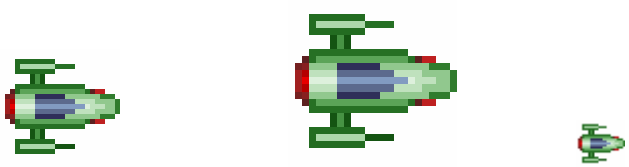


Två bilder ritas över varandra, och en kollision uppstår

Sidoförflyttning

I spel förflyttas ofta spritar för att ge känsla av animation och rörelse. Detta kan ske relativt, d v s att man ger den nya positionen relativt den gamla i antal pixlar, eller absolut, genom att man anger den nya positionen.

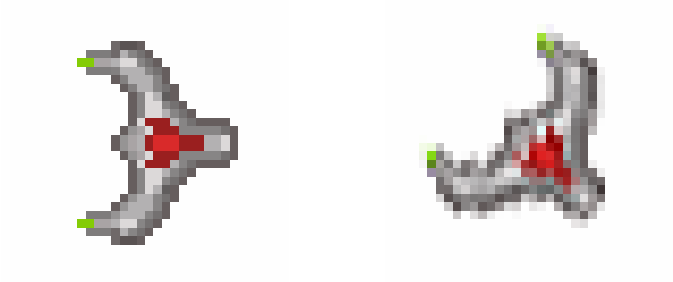
Omskalning



Exempel på omskalning.

Man kan skala om bilder genom att ange dess nya storlek, antingen absolut, genom att ange dess mått, eller relativt genom att multiplicera den ursprungliga bildens storlek med en skalfaktor, och på så sätt få en ny storlek på bilden.

Rotation



Den vänstra bilden har roterats 45 grader

Enligt Teixeira de Sousa är rotation av bilder bland det svårare när det gäller tvådimensionell grafikprogrammering. Det räcker inte att bara rotera varje bildpunkt med ett visst gradtal, då dessa är representeras av heltal, utan mer komplicerade algoritmer krävs för att det inte exempelvis ska bli hål i bilden. Vi går inte in på dessa här.

Appendix B – Porta ett spel från Win32 till Symbian

Nedan följer en readmefil som följer med installationen av GapiDraw. Den tar upp lite ändringar i API:t samt en beskrivning av hur man kan gå tillväga för att porta en existerande applikation till Symbian.

Readme

```
Please read the section "known issues" below before sending any bug
reports.


==================================
Symbian Changelog for GapiDraw 3.0
==================================

Changes since GapiDraw 2.10 beta1

* Added support for all Symbian devices running Symbian 6.1 or 7.0,
that is devices either buffered with palette, or not buffer with or
without palette.

* Added Minimize() in CGapiApplication.

* Improved support for lid close on P800/P900 in CGapiApplication
(application will minimize)

* Multiple keypresses now supported on Series 60 devices if compiled
with SERIES60 defined (put MACRO SERIES in the mmp file). An app
compiled with this option needs to be recompiled without this define to
work on UIQ devices. An app compiled without this define will also work
on a Series 60 device, but will not handle multiple key presses.

* CGapiDisplay::OpenDisplay() is now OpenDisplay(RWsSession&
windowSession, DWORD dwFlags), where dwFlags specifies the Symbian
device family. This change is made to allow compatibility with future
Symbian devices, which we haven't tested, but could easily be added
when the device family is known.

* CGapiDraw::CreateGapiDraw() removed.
```

```
========
Contents
========

1. What you need to start developing games with GapiDraw for the
Symbian platform
2. API changes in GapiDraw 2.1
3. Emulator support
4. Known Issues
5. Some problems and solution suggestions
6. Useful links to some Symbian pages
7. Steps to port a simple application to Symbian


================================================================
1. What you need to start developing games with GapiDraw for the
Symbian platform
================================================================

* A SDK for the telephones you are targeting. If you want to build for
Nokia 3650/7650, get the series60 SDK for Symbian OS 6.1. Get the UIQ
2.0 for Sony Ericsson P800. SDKs can be found here:
http://www.symbian.com/developer/SDKs.html

* If Active perl is not included in the SDK for your phone, you need it
as well. The scripts in the SDK tools use perl, so get it at
http://www.activestate.com/Products/ActivePerl/

* Make sure that you are able to build and install one of the provided
example projects in the SDK. There is a nice tutorial at newlc.com
about getting started with symbian programming at
http://www.newlc.com/article.php3?id_article=134

* Symbian version of GapiDraw including a dll, lib file and headers:
http://www.gapidraw.com/

* To port a game from pocketPC (or XP), follow the guidelines from the
porting of the sample project "simple"

* The gd210.lib file should be moved to
%epocroot%/epoc32/release/armi/urel/
```

```
==============================
2. API changes in GapiDraw 3.0
==============================

All platforms
=============

New class CGapiDraw
-------------------

CGapiDraw performs some initialization and cleanup. Just create an
object m_gapidraw on the heap before using other parts of GapiDraw
delete it when you are all done with GapiDraw. If you're using
CGapiApplication this is done for you.


New namespace gd444
-------------------

GapiDrawExtension.h now includes a namespace gd444 with full support
for surfaces with pixel format 444 (i.e. devices with 4096 colour
displays).


CGapiInput
----------

OpenInput() is no longer called in the constructor of a CGapiInput
object. You have to call CGapiInput::OpenInput() yourself. If you're
using CGapiApplication this is done for you.


Symbian version
===============


CGapiSurface
------------

Since the Symbian tool chain does not allow you to include bitmaps,
sounds etc. directly in your main binary, the method CreateSurface that
takes a resource ID is not included in GapiDraw for Symbian. However,
you can store all your game graphics in a multi bitmap file (.mbm) and
create your surfaces from there instead. The option of providing a
filename to CreateSurface is still there if you want to just include
your game graphics as is without worrying about the Symbian bitmap
compiler. It is not possible to load a image file from memory and the
SaveSurface() method is not implemented yet. The GetDC() and
ReleaseDC() methods are not included.
```

CGapiDisplay

OpenDisplay() in GapiDraw for Symbian doesn't take a HWND parameter as the win32 versions, but a session to the window server. CGapiDisplay::OpenDisplay(RWsSession windowSession, DWORD dwFlags). The display is always opened in fullscreen mode.

CGapiInput

OpenInput/CloseInput takes a RWindowGroup as a parameter. This parameter should be the window group of your application. If you are using CGapiApplication, this is done for you.

Timers / animation

Not included:
Timers in Symbian are a real drawback. It is impossible to sleep with a greater accuracy than 15 ms, although the documentation for the Symbian equivalent of win32's Sleep() takes a microsecond argument so this is rather a limitation of current devices. Most cell phones running Symbian OS generates a clock tick every 1/64 of a second and the equivalent to Sleep(1) simply waits until the next clock tick. Therefore, it is virtually impossible to set a constant fps other than 16,21,32,64 so the approach used in CGapiApplication for Symbian is simply to render graphics when there are no other higher priority threads scheduled. This practically means that you should never base your in-game animation on how many frames that have passed, but rather on time. This approach is discussed in a few threads at http://forums.pocketmatrix.com

CGapiRGBASurface

Not included

CGapiCursor

Not included

==========
3. Emulator support
==========

Emulator support is not included in GapiDraw for Symbian because of mainly three reasons:

1. Emulator support for UIQ require you as a developer to purchase Codewarrior for Symbian

2. The performance of timers in the emulator is horrible. Clock ticks are generated only 1/10 of a second. (See the section "Timers / animation" above)

3. You can always build your game for win32 and create a window of the same size as the display of the phone and use the same graphics, sounds etc. as in your Symbian version. Then you will know that your game logic is ok.

==========
4. Known Issues
==========

When running an application derived from CGapiApplication on the 7650, the menu program terminated about 10-15 seconds after starting, because of ViewSrv 11. It seems that the scheduler on the 7650 does not give enough priority to the menu program, and the menu program doesn't respond to the View server in time. The GapiDraw application does not crash however and this problem does not occur on any other Symbian device we have tested.

The imageloader does not work if you build an exe. If you need to build an exe instead of an app, and you want to load images, you would need to provide your own image loading, and convert the loaded image to a CGapiSurface using CGapiSurface::GetBuffer().

==========
5. Some problems and solution suggestions
==========

Problem: the compiler/linker says that there are initialized data in my dll

Solution: You are probably using global/static variables in your program. Apps/dlls in Symbian are not allowed to use these, so you have to organize thing differently to get rid of these, i.e. have them as member variables in your class rather than static variables in functions.

Problem: I get "undefined symbol __chkstk" when I try to build my app

Solution: You are putting to much on the stack. The stack size in Symbian is very limited so try to use the heap as much as possible.

Problem: The menu program crashes 15 seconds after my app has started. My app runs fine though.

Solution: Se section known issues

6. Useful links to some Symbian pages

Forums:

* Symbian C++ - http://www.newlc.com/forums/

* Development for Nokia devices - http://discussion.forum.nokia.com/forum/forumdisplay.php?s=&forumid=40

* Development for SE P800 - http://www.sonyericsson.com/developer/show_forums.do

* General Symbian forum - http://www.allaboutsymbian.com/forum/

General:

* Developer pages at symbian.com - http://www.symbian.com/developer/

* Nice site about Symbian - http://www.newlc.com

* Symbian news + forums - http://my-symbian.com/

```
=======================================================
7. Steps to port a simple application to Symbian (UIQ)
=======================================================

Here's a brief description of the steps required to port the simple
example from win32 to Symbian. The Symbian SDK has to be installed on
your computer. This description is not intended to be an introduction
to Symbian programming, and it doesn't try to explain any Symbian
specific details, its sole purpose is to show the amount of work to get
a small existing application up and running using GapiDraw for Symbian.
If you are interested in the various tools invoked in the build process
for Symbian, check out the online documentation at symbian.com:
http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/To
olsAndUtilities/Build-ref/index.html

0. Copy the files gd210.lib and gd210.dll (from GapiDraw/lib/Symbian
and GapiDraw/dll/Symbian) to %epocroot%/epoc32/release/armi/urel

1. Copy the content of "common".

2. Rename myapplication.h and .cpp to something else, and copy the
Symbian version of minimal myapplicationfiles to the new directory.

3. Create bld.inf, which should contain the following lines:

// File bld.inf

PRJ_MMPFILES
simpleUIQ.mmp

// End bld.inf

4. Create the file simple.mmp:

// File simple.mmp

TARGET            Simple.app
TARGETTYPE        app
TARGETPATH        \system\apps\simple
UID               0x100039ce 0x0101F415
RESOURCE          Simple.rss
SOURCEPATH        .
SOURCE            myapplication.cpp
SOURCEPATH        ..\..\..\include\symbian
SOURCE            GapiApplication.cpp
USERINCLUDE       ..\..\..\include ..\..\..\include\symbian .
SYSTEMINCLUDE     \epoc32\include \epoc32\include\libc
LIBRARY           euser.lib apparc.lib cone.lib eikcore.lib ws32.lib
LIBRARY           bitgdi.lib estlib.lib etel.lib gd210.lib

AIF simple.aif res simpleaif.rss c8 gapidrawicon.bmp
      gapidrawiconmask.bmp

// End simple.mmp
```

Note that the above UID 0x0101F411 is a test UID (the range is 0x01000000 to 0x0fffffff). You need to obtain a real UID from Symbian before releasing your app. When changing UIDs, make sure you change it in all relevant places, or your app will probably not run. The places you have to change the UIDs are: in MyApplication.h, your .mmp file and in your .pkg files and the .rss file for AIFs.

5. Create an rss resource file.

```
// File simple.rss

NAME SIMP

#include <eikon.rh>
#include <eikon.rsg>

RESOURCE RSS_SIGNATURE { }

RESOURCE TBUF { buf=""; }

RESOURCE EIK_APP_INFO
{
}

RESOURCE HOTKEYS r_example_hotkeys
{
}

RESOURCE MENU_BAR r_example_menubar
{
}

RESOURCE MENU_PANE r_example_file_menu
{
}

// End simple.rss
```

(rss files defines how menus, hotkeys and other stuff that we don't really care about when writing fullscreen games with GapiDraw. However, the Symbian framework insists that every application has a compiled resource file, so we make a really slimmed version of one).

6. Create make files with the command bldmake bldfiles

7. If you're using Visual C++ 6.0 you can use abld makefile vc6 to build vc6 project files. Your .dsw file will end up in %epocroot%\Epoc32\BUILD\{path-to-mmp-file} (for example C:\Symbian\6.1\Series60\Epoc32\BUILD\SOURCE\GAPIDRAW\MYPROJECTS\SYMBIAN\SIMPLE\WINS) together with a .dsp file.

ABOUT VC++6.0: You should see Visual C++ 6.0 just as a file editor for the files included in the mmp file of your project and nothing else. You still have to build the project from the command line.
If you create new source files and want to include them in your build, you still have to add them in a SOURCE directive in your mmp file. It's not enough to just add them in your visual C++ 6.0 project. You can add a file to your workspace like this: 1. Create an empty file in your project directory 2. add this file on a SOURCE line in your mmp file and save. 3. Run the command abld makefile vc6 4. Visual C++ will ask if you want to reload your workspace, click yes and now the new files are added in visual c++ as well.

8. Copy missing members from the W32 version to the Symbian version. The m_cursor member needs to be omitted since GapiDraw for Symbian doesn't support CGapiCursor, and the same for goes for DrawFrameInfo, because of CGapiTimer. Also, we want to have the member surfaces as pointers.

9. In the .cpp file:
 a Note that there are fewer things to set in the config in the Symbian version. ConstructL is called by the Symbian framework instead of WinMain. It also constructs the application.
 b Create surfaces must a bit altered. In the Win32 version they were created from resources, whereas in the Symbian version they will be created from files. Enter the relative or absolute path to the files on the device. The cursor surface is of course omitted too. SetRect has to be changed too. The parameter display is now a pointer.
 c ProcessNextFrame: Remove the timer parts and change the SetRect. Parameters display and backbuffer are now pointers. We can't have static variables, they need to become member variables instead.
 d StylusDown: Change the way to calculate if the stylus is within the bounds of a rect, since we cant use the w32 specific functions.

10. Compile your work with the command abld build armi urel

11. Create the simpleUIQ.pkg file to specify which files to include in the sis file, for example for UIQ

```
#{"Simple"},(0x0101f415),1,0,0
(0x101F617B), 2, 0, 0, {"UIQ20ProductID"}

"\Symbian\UIQ_70\epoc32\release\armi\urel\Simple.app"-
"!:\system\apps\Simple\Simple.app"
"\Symbian\UIQ_70\epoc32\data\z\system\apps\simple\Simple.rsc"-
"!:\system\apps\Simple\Simple.rsc"
"\Symbian\UIQ_70\epoc32\data\z\system\apps\simple\Simple.aif"-
"!:\system\apps\Simple\Simple.aif"
"\Symbian\UIQ_70\epoc32\release\armi\urel\gd210.dll"-
"!:\system\apps\Simple\gd210.dll"
"res\ipaq.png"-"!:\system\apps\Simple\ipaq.png"
"res\SimpleFont.png"-"!:\system\apps\Simple\SimpleFont.png"
"res\taskbar.png"-"!:\system\apps\Simple\taskbar.png"
```

12. Create a sis file with the command makesis simpleUIQ.pkg

13. Install the sis file on your phone, either by IR or by using a PC connectivity suite and a data cable.

These are the steps that were used to create the simple Symbian example in the samples folder.