IT 09 028
Examensarbete 30 hp
Augusti 2009

Mobilstudio för ShiShi TV

Andreas Pettersson

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Mobile Studio for ShiShi TV

Andreas Pettersson

During the fall of 2007 Uppsala University gave the course "Projekt DV". As a result of this class a publishing tool for real-time media came to be ShiShi TV. The purpose of this application was to introduce a new way to reach out to people. This was done with a video player on the Internet and a second one placed in a regular mobile phone. To render this possible some different languages were used. C# was used to write the PC-client which records and splits the material into small video pieces, to later transmit them to the ShiShi-server. Viewers from the Internet met an Adobe Flash-based video player which was placed on a PHP-page and used a MySQL-server to communicate with the ShiShi-server. And to view the contents on the mobile phone a Java ME-application was created.

This degree project has its focus on introducing an application that from a mobile phone can act as a publisher of data into the ShiShi-network.

To narrow the width of this project, the decision of targeting one specific mobile phone was made. By making a compilation of current mobile phones, I could draw the conclusion that the Nokia N95 with the Symbian S60 OS was the most suitable device as a test and target device for this project.

In the end of this degree project we could show that by using the same approach as the PC-client, we were able to send media in real-time, however with a much larger gap between the video files.

Handledare: Klas Eriksson
Ämnesgranskare: Olle Gällmo
Examinator: Anders Jansson
IT 09 028
Sponsor: Mobilebooster

Tryckt av: Reprocentralen ITC

Innehåll

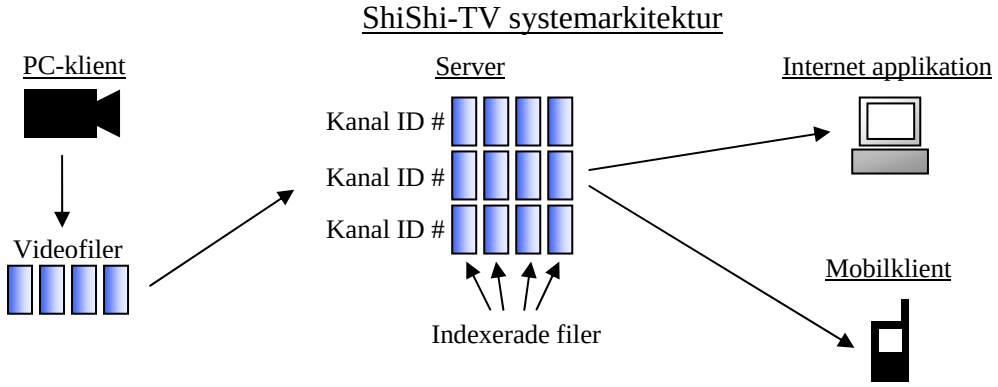
Inledning

Denna inledning är menad att ge en bakgrund till varifrån det ursprungliga systemet kommer och vad det består av. Den kommer även att beskriva syftet med examensarbetet. Men den kommer också att beskriva vad som inte blir behandlat, detta görs i avgränsningsavsnittet.

Bakgrund

Under höstterminen 2007 gavs kursen ”Projekt DV” på Uppsala Universitet. Ett resultat av denna kurs var ett publiceringsverktyg för direktsänd media, ShiShi-TV. Syftet med detta verktyg var att kunna strömma realtidsfångad video från en vanlig PC genom en server till en sida på Internet eller en mobilklient. Detta system består av: en C# PC-klient, en Erlang-server, en Java ME-mobilklient (Java Micro Edition) och en sida på Internet med en mediaspelare (Adobe Flash).

Det som skiljer denna lösning från mängden är att den inte använder sig av ett existerande protokoll för att strömma media, som till exempel RTSP (Real Time Streaming Protocol, ett av de nuvarande standardprotokollen för att strömma media över Internet). Istället försöker den simulera en ström genom att dela upp den kontinuerliga videoströmmen i flera småbitar, för att sedan skicka dem indexerade till en i förväg tilldelad kanal på servern. Därefter distribuerar servern videofilerna till de olika klienterna. Här är det dock klienternas eget ansvar att begära ”rätt” index då ShiShi-systemet tillhandahåller en buffert med äldre indexerade filer. Detta illustreras i Figur 1.



Figur 1 ShiShi-TVs systemarkitektur

Fördelarna med en sådan arkitektur är många. Dels kan den ofta ta sig igenom brandväggar eftersom det som sker är en vanlig filöverföring. Alltså utnyttjar vi faktumet att de flesta brandväggar släpper igenom Internettrafik och filtransaktioner. Detta gör att Windows-applikationen inte behöver några speciella portar öppna för videoöverföringen.
Dessutom kan inte äldre mobiltelefoner ta emot RTSP-strömmar, utan enheten försöker buffra all information i strömmen innan den spelas upp. Detta leder till att minnet i telefonen tar slut och ett applikationsfel uppstår i MIDlet-programmet.

Det finns även nackdelar med denna struktur. Ett av de största problemen är att kameror oftast är designade för endast en målfil. Skall denna sedan ändras måste de allokerade resurserna släppas för att sedan allokeras om med en ny målfil. Detta leder till att det kommer skapas hack i den realtidssimulerade strömmen varje gång detta byte sker.

Vid en direktsändning via ShiShi-nätverket upplevs dessa filbyten som hack i strömmen, det är därför av största vikt att försöka minimera dataförlusten under ett sådant glapp. Med en vanlig dator under kursen ”Projekt DV” lyckades studenterna att minimera glappet till cirka 0.20 sekunder.

Syfte

Syftet med detta examensarbete är att undersöka möjligheten att skapa en mobilstudio till ShiShi-TV. Om det ses som möjligt ska även en tänkbar lösning implementeras. De viktigaste frågorna som kommer att behandlas i en eventuell implementering är:

- Hur stort mellanrum kommer det att uppstå mellan de olika videofilerna?
- Kommer minnet i en telefon att klara av en sådan här lösning?
- Kommer en telefon klara av att sköta denna uppgift med den avsevärt mindre processorkraften?

Avgränsning

Arbetet kommer att begränsas till att undersöka möjligheten att skapa en mobilstudio till ShiShi-TV. Om det ses som möjligt kan det även bli aktuellt att välja en mobiltelefon som är relevant för den nuvarande marknaden och sedan utveckla programvara till just denna enhet. All kommunikation mellan ShiShi-TV och servern kommer att ske med existerande protokoll. Inga ändringar i det nuvarande systemet kommer att ske.

Analys

För att kunna välja en telefon måste vi först undersöka lite närmre de olika typer av nät som är tillgängliga. Här finns det två val, GPRS och 3G. Det som kommer att vara avgörande i valet är den hastighet uppströms som nätet kommer att kunna prestera. Detta beror på att den mängd datatrafik som kommer att alstras av applikationen (i största del video) till stor del är riktad ut mot nätet. Nu ställs den äldre teknologin GPRS mot den betydligt nyare 3G.

Vid valet av telefon är det inte enbart dess nätverkshastighet som spelar roll, utan även dess förmåga att kontrollera och hantera dess resurser på ett effektivt sätt. Detta är avgörande då, som tidigare beskrivet i inledningen, direktsändningen kommer delas upp i mindre videofiler och mellanrummet mellan dessa beror resurshanteringen. Det finns flera aspekter som kommer att vara avgörande för detta, men den som förmodligen har störst inverkan är vilket val av programmeringsspråk som görs. De kandidater som vi ställs inför är Java ME, Symbian OS och Microsoft Windows Mobile. Alla dessa språk har fördelar och nackdelar. Java ME har till exempel fördelen att programmet kommer att kunna köras på de flesta telefoner ute på marknaden, medan det har den stora nackdelen att applikationen exekveras i en virtuell motor vilket ökar svarstiderna. De två kvarvarande alternativen ser jag som likvärdiga. I båda språken arbetar applikationen nära mot systemresurserna, men som följd av detta erhåller de nackdelen att mängden mobiltelefoner som programmet kommer att kunna användas på avsevärt minskar.

Videoformatet har också det en inverkan på beslutet. Även fast detta inte kommer att vara en av de viktigaste delarna vid valet kommer det tas iakt. Anledningen till att formaten kommer att väga lättare i beslutet beror på att det existerande ShiShi-systemet ännu inte kan hantera flera videoformat, men en möjlighet för utbyggnad lämnas.

Då valet av en målenhet är gjord finns det krav på applikationen som kommer att utvecklas åt den. Det första kravet är utformningen av användarmiljön. För bästa möjliga upplevelse av programmet bör utformningen av interaktionen med mjukvaran vara väl igenomtänkt. Detta inkluderar hur information och de funktioner som användaren behöver presenteras på mobiltelefonen. Som en grund till användargränssnittet görs en användaranalys som skall agera likt en guide för utvecklingen av de visuella applikationsdelarna. Detta plus en nära kontakt i testfasen med en representativ mängd av slutanvändarna för de återkommande utvecklingscyklerna kommer att garantera ett användbart program.

För att förstå och designa en bra programstruktur görs en dataflödesanalys. Detta på grund av att programmet kommer att alstra en större mängd data som måste behandlas på ett tillfredsställande sätt, allt för att åstadkomma ett smidigt informationsflöde genom hela applikationen. Här kommer jag främst att titta på de större flödena av information, som till exempel nätverkstrafiken, kommunikation med filsystemet och kamerans informationssamlande.

Metod

Examensarbetet innehåller tre olika moment; förundersökning, programmering och användargränssnittsdesign. Därför kommer jag att använda tre olika metoder.

I den första fasen kommer jag att göra en omvärldsanalys för att inhämta information om de möjliga kandidaterna av utvecklingstelefon. Här granskas främst de marknadsledande företagen som Nokia, SonyEricsson, Samsung och LG Electronics. Detta på grund av att det är dessa företag som kommer att nå ut till störst volym av konsumenter, vilket i slutändan leder till en applikation som kan användas av störst mängd personer.

Då hårdvaran och en lämplig plattform för mjukvaran är vald kommer jag att använda mig av en agil metod för att lösa programmeringsproblem. Motivationen till detta val är att kravspecifikationen som gavs är vag, därför behövs en metod med en kort implementeringscykel vilket kommer ge en snabb återkoppling till arbetet, vilket i sin tur leder till en bättre slutprodukt.

Ytterligare en anledning till valet av den agila infallsvinkeln till problemlösningen var att jag själv hade en begränsad kunskap om de olika programmeringsspråken som stod till buds. Detta skapade problem eftersom det blev svårt för mig att uppskatta problemets magnitud och identifiera dess delar. Till följd av denna svårighet blir det komplicerat att använda sig av en mer strikt planerad metod, till exempel RUP, då uppgiften att bryta upp applikationen i mindre stycken för att sedan göra en tidsplan efter dessa skulle bli mycket svår, nästintill omöjlig. Istället sker en värdering av de olika delarna av applikationen i slutet av varje cykel som gör det lättare att fördela tiden rätt varefter projektet löper.

I fas tre av utvecklingen koncentrerar jag mig på användarinteraktion med systemet. En användaranalys görs för att identifiera den typiske användaren av systemet[1] som sedan med hjälp av denna analys skapa användningsfall som ligger till grund för konstruktionen av användargränssnittet[2].

[1] Jan Gulliksen och Bengt Göransson, Användarcentrerad systemutveckling, Studentlitteratur, 2002, sid 219.
[2] Jan Gulliksen och Bengt Göransson, Användarcentrerad systemutveckling, Studentlitteratur, 2002, sid 202.

Resultat

Nedan presenteras resultatet av examensarbetet. Det inkluderar förstudien, hur de viktigaste komponenterna i programmet är konstruerade och hur applikationsstrukturen är uppbyggd med hjälp av de olika designmönstren.

Förstudie

Förstudiens främsta uppgift var att hitta en mobiltelefon med en plattform som klarade av att lösa uppgiften effektivt. För att undersökningen skulle bli hanterbar gjordes en avgränsning av antalet telefoner. Denna avgränsning görs primärt med hjälp av telefonens hastighet, det är just hastigheten som spelar en avgörande roll då den bestämmer om video överhuvudtaget går att skicka i realtid mot Internet.

Här kan vi dela in mobiltelefonerna i två läger, de som är utrustade med 3G teknologi och de som inte är det. Som konstaterat i analysen är uppladdningshastigheten den kritiska punkten i kommunikationen.

En närmare studie av GPRS-protokollet visar att denna metod att skicka datatrafik har en teoretisk maxgräns vid 160 Kbps[3], vilket i sig är en ansenlig fart. Vid denna hastighet skulle vi kunna skicka 20 kilobyte av data upp mot Internet varje sekund. Detta möjliggörs genom en dynamisk resursallokering i GPRS-protokollet som sker vid varje förfrågan. Det bör dock noteras att denna hastighet sker då alla av de åtta möjliga kanalerna är konfigurerade för uppgående trafik och utan någon felkorrigering. I verkligheten är hastigheten som en GPRS-telefon kan uppnå tyvärr mycket mindre.

Om vi sedan jämför med den största videofilen i undersökningen, som var 40 Kb, se Tabell 2, så kommer det att krävas minst två sekunder att sända en fil med video. Detta lämnar en liten marginal för fel och ingen marginal för att kunna förbättra videoformatet. Därför föll valet på en begränsning till att enbart undersöka 3G-telefoner.

I kravspecifikationen fanns det tre huvudkrav för mobiltelefonen. Första kravet var förmågan att som minimum kunna fånga video i 3GP-format (3GP är en komprimeringsstandard skapat av Third Generation Partnership Project), men ett format med bättre kvalitet som MPEG-4 var att föredra. Andra kravet som ställdes var att mobilen kunde leverera den hastighet som krävdes för att strömma video till Internet. Sista kravet innefattade att enheten var väl dokumenterad och välkänd av andra utvecklare.

Kraven som ställdes på plattformen var att den skulle kunna hantera video effektivt plus att den även skulle var väl dokumenterad. Tidigt uteslöts Java ME. Detta gjordes på grund av de problem som uppstod i den hårt kontrollerade miljö som Java koden exekverades inom. Här var användaren tvungen att godkänna varje access till Internet och kameran som gjordes. Detta medförde då att den simulerade realtidsströmmen till servern inte gick att genomföra.

När väl Java ME visat sig vara otillräcklig finns det sparsmakat med språk att välja på, det vill säga om man inte ska inrikta sig till ett speciellt företags mobiltelefoner. De val som återstår är då Symbian OS, Windows Mobile och Linux. Windows Mobile finns i mobiler från

[3] Timo Halonen, Romero García Romero, Javier Romero, Juan Melero, GSM, GPRS and EDGE Performance, 2003, sid. 14.

Motorola och Samsung men också från märken som ej är allmänt kända som exempelvis Asus, Eten, HTC, Neonode, O2, Orange och HP. SonyEricsson kommer även att tillverka en mobil som använder sig av Windows Mobile, men den lanseras senare under våren. Intrycket av Windows Mobile är att operativsystemet är mer inriktat mot PDA- och Pocket PC-enheter.

Linux finns i olika skepnader från Maemo som används av Nokia till telefoner med öppen källkod som till exempel OpenMoko.

Symbian är det operativsystem som finns i överlägset flest telefoner i dagsläget, företag som använder sig av detta OS är bland annat; Nokia, SonyEricsson, Panasonic, LG Electronics och Samsung.

För att underlätta valet gjordes en sammanställning av tänkbara telefoner, Tabell 1.

Tillverkare	Modell	Plattform	Driftfrekvens	Notering
Nokia	N95	Symbian S60, 3rd Ed., feature pack 1	HSDPA	MPEG-4, 30 bilder/s.
	6120 Classic	Symbian S60, 3rd Ed., feature pack 1	HSDPA	
	E90 Communicator	Symbian S60, 3rd Ed., feature pack 1	HSDPA	
	E51	Symbian S60	HSDPA	
	N96	Symbian S60, 3rd Ed., feature pack 2	HSDPA	MPEG-4, 30 bilder/s. Lanseras våren -08.
SonyEricsson	P1i	Symbian	UMTS 2100	
	M600i	Symbian	UMTS 2100	
	P990i	Symbian	UMTS	
	W950i	Symbian	UMTS 2100	
	W960i	Symbian	UMTS 2100	
	X1	Windows Mobile	HSDPA, HSUPA, UMTS	Lanseras våren -08.
Samsung	SGH-i450	Symbian	HSDPA	
	SGH-i520	Symbian	HSDPA, UMTS	
	SGH-i550	Symbian	HSDPA	
	SGH-i560	Symbian	HSDPA	
	SGH-i600U	Windows Mobile 6	HSDPA	
	SGH-i780	Windows Mobile 6	HSDPA	
LG Electronics	KS10	Symbian S60	HSDPA, UMTS	

Tabell 1 Sammanställning av mobiltelefoner

I tabellen ovan finns det ett flertal olika driftsfrekvenser; UMTS, HSDPA och HSUPA. UMTS är den teknik som vi i vardagen refererar till som 3G. Hastigheten som uppnås med denna teknologi beror på hur och när den används. Vid användning inomhus och med en inomhusantenn får vi en maximal överföringshastighet på 2,048 Mbps. Men vid utomhusbruk och med en mottagare som sakta rör på sig sjunker hastigheten till 384 Kbps. När en

mottagare rör sig snabbare över större ytor sjunker hastigheten ännu mera, då kan vi maximalt uppnå en hastighet på 144 Kbps.[4]

Den lite mer avancerade tekniken, och den som är vanligast idag, HSDPA (High-Speed Downlink Packet Access), är en förbättrad version av UMTS-standarden. Här har man förbättrat hastigheten nedströms till 10 Mbps.[5]

Som en vidarepåbyggnad på HSDPA kommer HSUPA (High-Speed Uplink Packet Access). Detta leder till att alla produkter som stödjer HSUPDA även stödjer HSDPA eftersom det är ett krav i specifikationen. Det nya med HSUPA är en utökning av överföringshastigheten uppströms till mellan 0,72-5,76 Mbps.[6]

Enligt tabellsammanställningen är det bästa valet SonyEricssons X1. Detta på grund av den videokvalitet som den kan producera och att den är utrustad med HSUPA, vilket ger en god hastighet i strömmen av data vi kan skicka ut från mobilen. Det leder naturligtvis till att vi kan förbättra kvaliteten på den video vi strömmar ut från mobilen. Nu är detta val inte möjligt eftersom vid tiden då förstudien utfördes var ej mobilen i bruk än, den bästa telefonen för tillfället var Nokias N95. Den har de bästa anslutningsmöjligheterna och en kompetent kamera.

När jag sedan granskar operativsystemen som telefonerna använder sig av närmre framstår Symbian OS som det bästa valet. Det är utbrett över en mängd märken och modeller, dessutom kan vi skriva program som arbetar nära telefonens kärna. Vi slipper med andra ord de problem som kan uppstå med en virtuell motor som till exempel Java ME arbetar med.

Valet är dock inte bara grundat på själva enheten och mjukvaran som den kör, utan även på den möjlighet till hjälp som står till förfogande. Nokia har lyckats skapa en mycket bra portal för informationsutbyte mellan utvecklare. Detta är gjort genom sidor för lagring av öppen källkod, forum för utvecklingsdiskussioner och dynamiskt användaruppdaterade uppslagsverk (så kallade wiki-sidor). De har även utvecklat verktyget Carbide som är en påbyggnad av Eclipse (som är en öppen källkods IDE) för att underlätta att utveckla för Symbian.

Programmodell

Programmeringsspråket Symbian är vad man skulle kunna kalla en dialekt till C++. Språket är objektorienterat liksom C++ men har några mycket nyttiga påbyggnader. Ett sådant exempel är introduceringen av ”leaves”, dessa är inte helt olikt Javas ”exceptions”. Tanken är att varje funktion som kan misslyckas med sin exekvering skall omges av en såkallad ”trap” (en ”try” sats i Java) som fångar upp felet och hindrar programmet från att krascha och lämna en minnesläcka efter sig.

Med hjälp av denna nya funktion har ett designmönster formats för att förhindra minnesläckor vid skapandet av objekt. Detta är otroligt viktigt eftersom telefoner eller handdatorer ofta har en mycket lång drifttid innan omstart. Två-fasers konstruktorn, som designmönstret kallas, används vid skapandet av alla objekt som vill exekvera någon potentiell farlig kod vid dess konstruktion. Med farlig menas i detta fall operationer som allokerar/reserverar resurser i

[4] Alessandro Andreadis, Giovanni Giambene, Protocols for High-efficiency Wireless Networks, 2003, sid. 45.
[5] Alexander Joseph Huber, UMTS and Mobile Computing, 2002, sid. 124.
[6] Harri Holma, Antti Toskala, WCDMA for UMTS: HSPA Evolution and LTE, Sid. 416.

telefonen, exempel på sådana resurser kan vara minnesceller, en nätverksport eller kameran. Det här designmönstret gör att vi kan frigöra även de resurser som har allokerats innan felet har uppstått i konstruktionen av objektet. Ett exempel på detta mönster är koden i Figur 2.

`CFoo() {}`	En konstruktor som **aldrig** misslyckas.

`ConstructL() {`
`  ...`
`}`
	Här sker den ”farliga” exekveringen av kod.

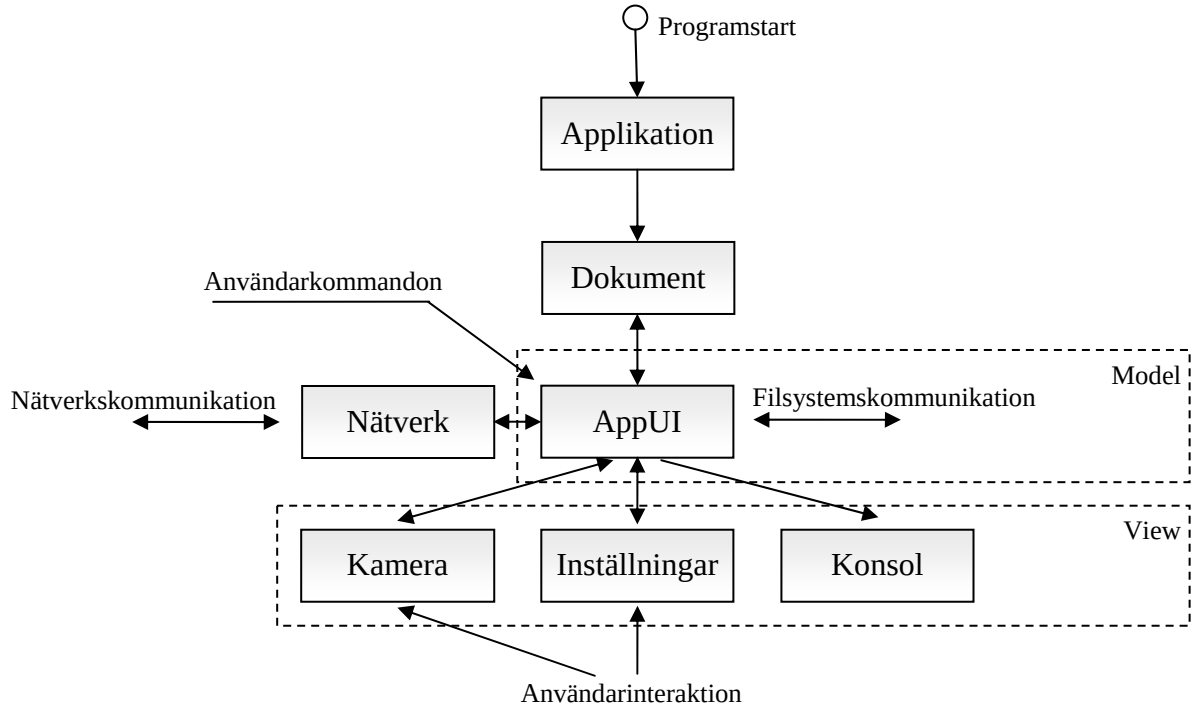
`static CFoo* NewL() {`
`  CFoo* self = new (ELeave) CFoo();`
`  CleanupStack::PushL( self );`
`  self->ConstructL();`
`  CleanupStack::Pop( self );`
`  return self;`
`}`
	En statisk funktion som skapar vårat objekt och lägg till det på Cleanup stacken. För att sedan rensabort det om konstruerandet av objektet lyckas.

Figur 2 Exempel på en två-fasers konstruktor

I exemplet ovan utnyttjas en till ny funktionalitet i Symbian, en ”Cleanup stack”. Denna stack innehåller pekare till objekt som allokerar minne på något sätt för att sedan kunna gå tillbaka och frigöra minnet om ett objekt orsakar ett fel.[7]

Till alla som börjar programmera applikationer till Symbian OS finns det ett flertal exempel och ramverk redan klara i den Symbian SDK (Software Development Kit) som finns tillgängligt. Jag har valt att använda mig av dessa ramverk. Den modell som jag har kommer att använda är till för att utveckla program med ett grafisktgränssnitt mot användaren. Denna mall är uppbyggd med hjälp av fyra klasser; Application-, Document-, ApplicationUI- och ApplicationView-klassen.

En överblicks bild av systemet uppbyggnad visas i Figur 3 nedan.



Figur 3 Övergripande översikt av programdesign

[7] Richard Harrison, Phil Northam, Symbian OS C++ for Mobile Phones, 2003, sid. 76.

Applikationsklassen är den första klassen som skapas när ett program startas. Det här objektet ärver de grundläggande beteendena av ett Symbian-program. Dess primära uppgift är att skapa ett eller flera dokument objekt och tillhandahålla diverse funktioner till dessa objekt. Klassen skall även ställa till förfogande programmets unika id till operativsystemet som används för att särskilja applikationen ifrån de andra som körs i telefonen.

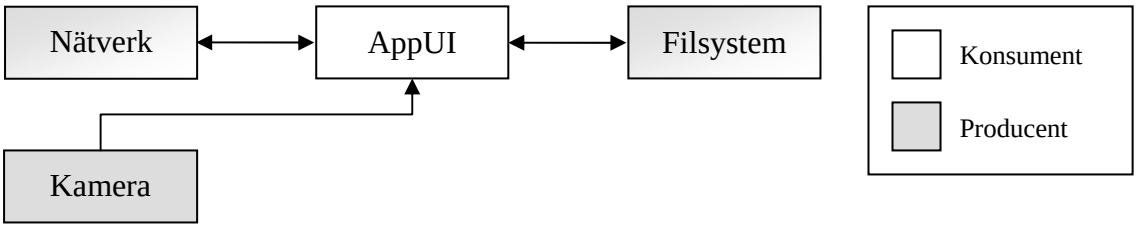
Dokumentklassens uppgift är att ladda och spara ett eller flera dokument från filsystemet vid start och avslut av applikationen. Då jag inte använder mig av denna funktionaliteten kommer klassen enbart skapa mitt programs användargränssnitt.

Användargränssnittet är baserat på det vanliga Model-View-Controller mönstret.[8] Här kombineras kontroller och modell i klassen AppUI. Modellen tar emot och behandlar informationen som kommer ifrån nätverket, kameran och det lokala filsystemet. Kontrollerns uppgift är att hanterar fördefinierade användarkommandon från menyer och CBA knappar i telefonen. Allt detta presenteras sedan i den aktuella vyn.

Det här är den grundläggande strukturen för alla Symbian-program med användargränssnitt mot användaren. Från ett rent programmeringsperspektiv är fördelarna många med att använda en sådan här standardmall, mycket arbete är redan gjort i förväg och många småfel kan undvikas. I det ramverket jag har valt blir programmeraren dessutom uppmuntrad till att redan från start tänka på språkstöd. Detta genom att tillhandahålla ett strängbibliotek i enskild fil, vilket gör det lättare att översätta programmet till andra språk.

Dataflödesanalys

Genom att göra en analys över hur data kommer att produceras och konsumeras i min programmodell kan jag lättare designa mina klasser och bestämma vilken typ av kommunikation mellan dem som kommer att vara lämplig. Resultatet av denna analys visas i Figur 4.



Figur 4 Dataflödesdiagram

Ovan finns de fyra klasserna som antingen producerar eller konsumerar information i programmodellen. Konsumenter i vitt och producenter i grått.

Filsystemet finns representerat även fast den inte skapar någon ny eller förbrukar information. Detta är på grund av att den hämtar upp information vid programstart och sparar ner det applikationen vill behålla vid programavslut.

Nätverket är liksom filsystemet både producent och konsument av information. Genom att ta emot nätverkstrafik från Internet producerar klassen data, och som konsument ansvarar den

[8] Martin Fowler, David Rice, Patterns of Enterprise Application Architecture, 2003, sid. 330.

för att omvärlden får ta del av den information som programmet skapar. AppUI står som den enda rena konsumenten, detta är naturligt eftersom det är själva kontrollklassen som tar emot information och sedan exekverar åtgärder baserad på den. Sedan återstår bara den självklara producenten, kameran som vid sändningstillfällena producerar video i en konstant ström.

Dataflödet från dessa klasser kommer att ske asynkront, det vill säga en lösning med bara en tråd för programmet kommer inte att fungera. Här finns det några olika sätt att lösa problemet. Den klassiska lösningen var att använda en eller flera nya trådar till programmet, men denna lösning medför också merarbete då kommunikationen mellan dem måste lösas av programmeraren själv. Utvecklarna av Symbian har identifierat detta problem med multitrådande program och skapat ett nytt sätt att lösa uppgiften. Lösningen innefattar en ny typ av objekt, ett så kallat aktivt objekt.

Ett aktivt objekt är en klass som arbetar asynkront mot andra delar av applikationen. Den funktionaliteten möjliggörs med hjälp av en schemaläggare i operativsystemet. Resultatet blir ett beteende som är snarlikt ett multitrådande program.

Tack vare att vi använder oss av en schemaläggare kan objektet självt bestämma när det vill arbeta asynkront mot de andra objekten i applikationen. Vanligtvis görs detta när operationer mot filsystem, nätverk eller någon annan systemresurs görs. Anledningen till att det oftast är mot dessa resurser aktiva objekt används är på grund av de ibland långa svars- eller exekverings- tider som förknippade med dem.

När ett aktivt objekt väljer att koppla sig löst från resterande delar av ett program notifierar den schemaläggaren att den vill börja sin exekvering och givetvis även då också vilket anrop som skall behandlas i bakgrunden. Då programkoden i bakgrunden har kört klart anropas en ”callback” funktion i objektet som gjorde begäran från första början för att meddela hur exekveringen gick.[9]

Om detta utvecklingssätt används uppstår inte problemet med kommunikationen mellan trådar. Nu behövs bara ett lämpligt designmönster för kommunikationslösningen.

I och med att det finns en ren konsument i modellen har jag valt att lösa kommunikationen mellan de olika objekten på enklast möjliga sätt. I mitt fall är det med hjälp av aktiva objekt och ett Observermönster, även känt som publish/subscribe. Genom min dataflödesanalys kan vi konstatera att lämpligaste implementering av detta mönster är att göra AppUI-klassen till subscriber. Därefter gör jag nätverket och kameran till publishers mot AppUI.[10]

Klassdesign

Många av de klasser som jag har använt mig av är från början en del av ramverket. Därför kommer jag inte att gå närmre in på dem i detalj eftersom de redan är väl dokumenterade av Nokia i deras nybörjarmanual.[11] De klasser som jag har lagt ner mest tid på att designa är nätverks-, kamera- och vy-klasserna.

---

[9] Ben Morris, The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS, 2007, sid. 60.
[10] Alan Shalloway, James Trott, Design Patterns Explained: A New Perspective on Object-oriented Design, sid. 263.
[11] Symbian OS Basic Workbook, http://www.forum.nokia.com/info/sw.nokia.com/id/981e8e7b-668c-449a-b0e0-e7e5f534c39d/Symbian_OS_Basics_Workbook.html, 198-215.

Vyklasserna

Det viktigaste att lägga märke till hos en vyklass är hur den har implementerat funktionen *InputCapabilities*. Denna funktion bestämmer vilken interaktion med vyn som kommer att vara tillåten.

Nätverketet

Nätverket i Symbian OS är uppbyggt med en klient server struktur.[12] Här måste varje applikation som vill använda nätverket använda sig av två klasser; RSocketServ och RSocket. RSocketServ behövs för att skapa en session med den socketserver som finns på telefonen, när denna session har skapats kan vi sedan ansluta med applikationens socket objekt (RSocket).

För att denna procedur skall flyta smidigt och inte frysa programmet krävs det att nätverkskoden ligger i ett aktivt objekt. Sedan utnyttjar vi faktumet att funktionen *RunL* anropas efter varje aktiv fas för att ta anslutningen igenom dess olika stadier. Denna egenskap gör att de flesta aktiva objekt är designade som en automat.

Nätverkstrafiken använder sig av en ansluten TCP/IP socket, allt enligt den ursprungliga ShiShi-TV kommunikationsspecifikationen. Eftersom trafiken sköts asynkront med de inbyggda läs och skriv operatorerna i nätverksobjektet får man som följd att anrop till klassen går att göras även då det inte kan hantera dem. Det vill säga när data håller på att skickas eller tas emot. Eftersom denna möjlighet inte är önskvärd då den kommer att ställa applikationen i ett paniktillstånd, avvärjes den enkelt med att implementera en kö. I detta fall en vanlig FIFO-kö.[13] Då vi har lagt till en kö skapar vi även ett nytt tillstånd i nätverksautomaten som tar hand om medelanden som väntar i kön.

Kameran

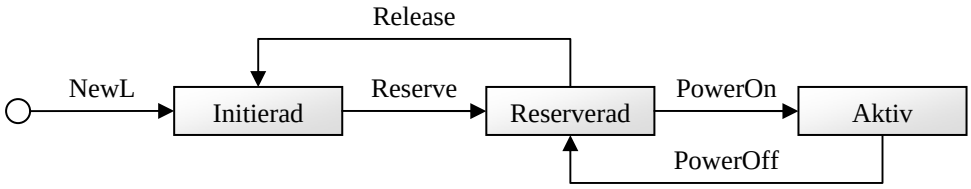
I den här klassen utnyttjas Symbians funktionalitet med multipla arv. Det underlättar avsevärt då objektet har flera olika typer av uppgifter att tillgodo se i applikationen. CCoeControl ärvs då den gör det möjligt att visa innehåll på telefonens skärm och MCameraObserver är klassen som måste implementeras för att programmet skall kunna kommunicera med kameran. För att sedan kunna spela in video används arvet av MVideoRecorderUtilityObserver. Då vi sedan önskar en pulserande beteende, detta då vi önskar att dela upp videon i delar, ärvs MBeating som gör det möjligt att ta emot klockpulser.

Det är inte alltid säkert att en telefon med Symbian OS som plattfrom innehar en kontroll som stödjer videoinspelning, därför undersöks även detta vid skapandet av objektet. Om det visar sig att telefonen inte har stöd för inspelning kommer konstruktorn att misslyckas och lämna ett felmeddelande till den som försöker skapa objektet, den kommer att generera en ”leave”.

Kameramodulen är en del av användargränssnittet, en View i Model-View-Controller mönstret, därför ärvs CCoeConstrol. Det gör att vi kan använda klassen som en klassisk sökare i en systemkamera.

[12] Ben Morris, The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS, 2007, sid. 57.
[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, 2001, sid. 202.

För att kunna använda kameran måste den allokeras. Innan det kan göras krävs det att det objektet som vill reservera resursen implementerar klassen CCamera. CCamera kräver i sig en prenumerant som har implementerat MCameraObserver. Detta beror på att kameran har en initieringscykel den måste ta sig igenom innan den är klar för användning. Stegen i cykeln visas i Figur 5 nedan.



Figur 5 Statuscykel för CCamera

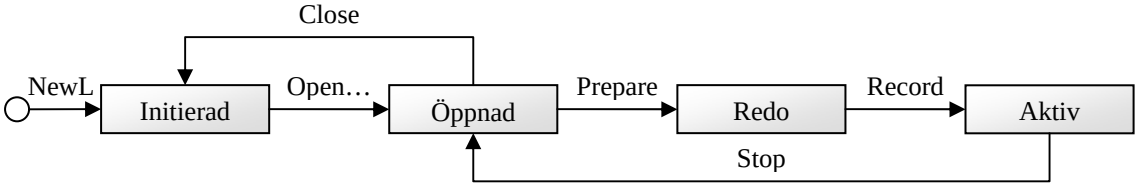
När ett kameraobjekt skapas intar det initierade stadiet, i det här läget har det bara skapats en koppling mot den valda kameran på telefonen. Det enda som är säkerställt här är att det finns en kamera med det ID på mobiltelefonen.

Nästa steg är att försöka reservera kameran. Om detta lyckas får applikationen exklusiva rättigheter till kameran, inget annat program kommer att kunna använda kameran undertiden applikationen har den reserverad. Men om enheten redan är allokerad av ett annat program kommer vi att stöta på problem. Det enda som återstår då är att be användaren att stänga av det program som har hunnit reservera kameran före oss och sedan försöka igen. Detta scenario är inte helt otroligt eftersom de flesta telefoner redan från fabriken har ett flertal programvaror som har stöd för kamera användning.

Om reservationen har lyckats måste vi se till att enheten är försedd med ström innan användning. Då detta har skett kan kameran börja användas i applikationen för att ta stillbilder eller använda andra funktioner som kameran erbjuder.

Då programvaran har använt kameran klart är det viktigt att frigöra resursen, eftersom vi tidigare har noterat att kameran exklusivt reserverad vilket gör det omöjligt för andra applikationer att använda resursen. Nu ser vi även vikten med att följa de rekommenderade designmönstren, här tänker jag främst på tvåfas konstruktionen. Om inte mönstret används och av någon anledning programmet kraschar kommer kameran att vara obrukbar till nästa omstart av mobiltelefonen.

För att ta upp video används verktyget CVideoRecorderUtility. Denna klass kräver att den som utnyttjar dess funktionalitet ärver MVideoRecorderUtilityObserver. Även detta objekt har en initieringscykel som en användare tar sig igenom för användning. Cykeln innehåller fyra stadier som ses i Figur 6.



Figur 6 Statuscykel för CVideoRecorderUtility

Första steget i statuscykeln skapar objektet med en tvåfas konstruktorn. Därefter måste ytterligare information läggas till. De finns nu tre vitala bitar av information; adressen till kameran som står redo (ett så kallat ”handle”), en kontroll som hanterar den data som kameran skapar och tillsist ett format som videon skall spelas in i. När denna information finns tillgänglig kan en av flertalet funktioner för att reservera en plats att spela in till användas.

Efter processen är slutförd och innan inspelningen har börjat får användaren av verktyget en chans att gör de sista inställningarna. Dessa kommer inte att gå att ändra efter inspelningsstart. Ett exempel på en sådan inställning kan vara om ljud skall inkluderas i videon eller om det skall finnas en maximal storlek på videofilen.

När denna komplicerade reservation är avslutad är det bara att initiera inspelningen med ”Record”-funktionen. Videoinspelningen avslutas då den fördefinierade maxgränsen har nåtts eller när stoppfunktionen anropas. Det är här viktigt att notera att om stoppfunktionen används kommer inte avslutningshändelsen att registreras i ”callback” funktionen. Notera även att det är nödvändigt att stänga den öppna videodestinationen för att all information skall sparas korrekt.

En essentiell del i ShiShi-systemet är att videon delas upp i flera små videofiler som sedan sammanfogas då materialet skall visas. I det nuvarande systemet sker denna uppdelning genom att vi stannar videoinspelningen och byter målfil till en ny videofil. Ett alternativ till denna metod är att en primärfil används för videoinspelning och sedan extraheras delar ur denna för att skapa de mindre bitarna. Metoden ser lovande ut på papper då vi kommer att undgå en förlust av data då byte av fil sker i kameran. Men när tillvägagångssättet sattes på prov visade det sig att den videofil som kameran använder är låst. När läsförsök tvingades igenom ändå resulterade det i ett systemfel.

Den sista centrala delen till kameramodulens beteende är arvet av MBeating. Detta arv möjliggör ett pulserande beteende (genom att implementera CHeartbeat). Nu kommer objektet att motta en puls i förutbestämda tidsintervall (i vårt fall varje sekund). Härifrån läggs ansvaret på klassen att synkronisera och skapa den fördefinierade längden på videofilerna.

Användargränssnitt

Användaranalys

Resultatet av min användaranalys visar att det finns två stora användargrupper.

1. Förstagångsanvändare
2. Återkommande användare

Den klart största gruppen kommer att vara förstagångsanvändarna. Till denna grupp räknas de användare som använder applikationen för första gången, men även de användare som inte kommer använda ShiShi-TV regelbundet. Då system inte är tänkt att användas dagligen eller heller regelbundet kommer den första gruppen vara dominerande.

I min granskning försökte jag även ta reda på om det skulle kunna finnas en speciell typ av användargrupp. Om det hade visat sig var fallet skulle designen av användargränssnittet kunnat anpassas mot denna grupp. Jag misslyckades dock att isolera en sådan typ av användargrupp, detta medför att inlärningskurvan till systemet kommer att vara väldigt lång.

Då jag varken har en stor återkommande grupp av användare eller en viss typ av användargrupp kan jag inte riktiga designen av gränssnittet mot dem. Det leder till att interaktionen med applikationen måste bli enkel för att programmet ska bli användbart.

Användningsfall

Utifrån användaranalysen konstruerar jag två användningsfall som syns i Figur 7 och Figur 8.[14] Här visas interaktionen med systemet i svart och det som sker transparent mot användaren i grått.

Det första som en användare av programvaran kommer att se av systemet är en begäran om inloggningsuppgifter. Dessa uppgifter skall redan vara införskaffade genom ShiShi-TV portalen. Här interagerar användaren för första gången genom att mata in uppgifterna och ger sedan en accept på detta.

Därefter ber programmet användaren att välja anslutningssätt till Internet. Då detta val har skett ansluter systemet till Internet och verifierar inloggningsuppgifterna. Visar det sig att uppgifterna är riktiga slussas användaren vidare till den vy som tillhandahåller interaktionen med kameran. Om det däremot visar sig att uppgifterna var inkorrekta meddelas användaren om den felaktiga inmatningen och tillfälle för en ny inmatning ges.

Från kameravyn kan användaren sedan börja sin utsändning till åskådarna med ett enkelt knapptryck. För att sedan avsluta sändningen med samma knappsekvens. Vid sändningsavslutet förflyttas användaren av applikationen automatiskt tillbaka till konsolen som ger en sammanfattning av händelseförloppet i klartext.

När sedan kommando för applikationsavslut ges sparas de essentiella inställningarna som till exempel användaruppgifterna av applikationen. Sedan frigörs de resurser som har allokerats och det riktiga programavslutet sker.

[14] David Benyon, Phil Turner, Susan Turner, Designing Interactive Systems, 2005 sid. 198.

Den stora skillnaden mellan användningsfallen är att personer som återkommer till applikationen inte behöver ange sina inloggningsuppgifter vid programstart. Därefter följer de samma mönster.

Nu får användningsfallet för flergångsanvändare i Figur 8 tala för sig självt.

Förstagångsanvändare	System
	Session startar
	Inmatning av inloggningsuppgifter begärs
Uppgifterna matas in och ett OK ges till systemet	
	Anslutnings-alternativ presenteras
Val görs, OK ges	
	Inloggning mot ShiShi servern sker
	Kameravyn presenteras då en acceptans ges på inloggningen
Börjar sändningen med kommando	
	Sändning börjar
Sändning avslutas med kommando	
	Sändningen avslutas
	Konsolvyn presenteras
Avslut av systemet begärs	
	Inställningar sparas och sessionsavslut sker

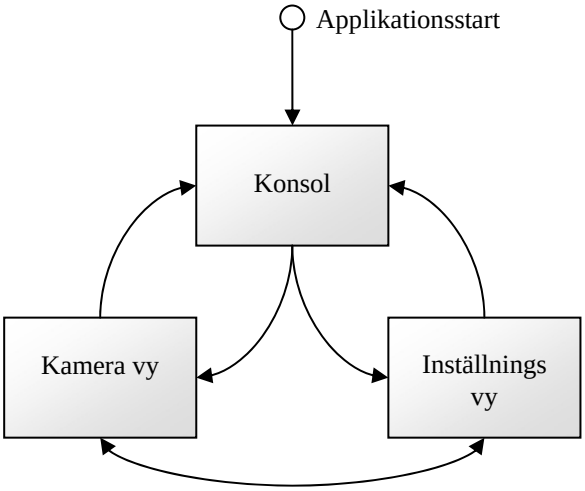
Figur 7 Användningsfall för förstagångsanvändare

Van användare	System
	Session startar
	Anslutnings-alternativ presenteras
Val av anslutning görs och accept ges	
	Inloggning mot ShiShi servern sker med i förhand sparade uppgifter
	Kameravyn presenteras då en acceptans ges på inloggningen
Börjar sändningen med kommando	
	Sändning börjar
Sändning avslutas med kommando	
	Sändningen avslutas
	Konsolvyn presenteras
Avslut av systemet begärs	
	Eventuella ändringar av inställningarna sparas och sessionsavslut sker

Figur 8 Användningsfall för flergångsanvänare

Gränssnittsdesign

Användargränssnittet har en väldigt enkel struktur. Detta är ett medvetet val eftersom användaranalysen visade att den största användargruppen kommer vara förstagångsanvändare. Av den orsaken har jag valt att bygga upp programmet med hjälp av tre vyer baserat på de användningsfall som tagits fram, illustrerat i
Figur 9.



Figur 9 Användargränssnittets struktur

Navigering mellan de olika vyerna sker antingen automatiskt eller med menyer som aktiveras med CBA knapparna.

Den automatiska navigeringen sker mellan konsol- och kameravyn. Detta görs vid två tillfällen. Första övergången sker från konsol- till kameravyn, detta händer då en anslutning har skapats till ShiShi servern och användaren har identifierat sig med ett giltigt användarnamn och lösenord. Det andra skiftet sker då en användare avslutar en utsändning. Då sker bytet automatisk från kamera- till konsol vyn.

Den manuella navigeringen sker via en meny som aktiveras med den vänstra CBA knappen (som ses i Figur 10). Menyn har sex alternativ som användaren kan välja i mellan; Console, Camera, Settings, Authenticate, About och Exit. Alternativen Console, Camera och Settings ger möjligheten att förflytta sig till respektive vy.

De återstående menyalternativen ger chansen till; att visa information om produkten (About), att avsluta programmet (med Exit alternativet) eller att återidentifiera sig om en felaktig inmatning av användarnamn eller lösenord skett (med Authenticate alternativet).

Figur 10 CBA meny

Observera att om det inte finns en enhet på

telefonen som stöder inspelning av media kommer Camera alternativet vara aktivt men inte förflytta användaren från den vyn som är aktuell för tillfället.

Konsolvyn

Denna informationsskärm finns till för användare ska få en enkel och lättförståelig återkoppling till vad som sker och har skett.

Figur 11 Exempel på konsol vyn

Innehållet i meddelandena innefattar det som håller på att ske, återkoppling av lyckade operationer likväl de som misslyckas. Även de meddelanden som visas separat i nya fönster sparas i konsolen.

I Figur 11, till höger, kan vi se ett exempel på hur konsolen fungerar. Där visas det senaste meddelandet till användaren längst upp mot övre vänstra hörnet och följer i kronologisk ordning i nedåtstigande led. I exemplet kan vi även se att det första användaren försökte göra vara var att allokera en kamera, men detta misslyckades då inga enheter gick att hitta. Sedan skapades en internanslutning mot nätverksservern i telefonen. Detta får vi sedan en positiv återkopplig till då operationen lyckades. Därefter skapades den egentliga länken mellan mobiltelefonen och ShiShi servern. När detta var gjort begärde servern att vi skulle identifiera oss, detta gjorde vi med ett lyckat resultat.

Kameravyn

Denna vy är uppbyggd av huvudsakligen en enda komponent, kamerans sökare (eller viewfinder på engelska). Sökaren är placerad i mitten på telefonens skärm och aktiveras då kameravyn aktiveras.

Uppe i det högra hörnet finns en indikator som visar om sändningen har påbörjats mot ShiShi servern eller ej (”on air”).

Själva utsändningen startas och avslutas med att knappen i mitten av telefonens styrkors.

När vi befinner oss i denna vy ändras även CBA knapparna. Istället för den vanliga uppsättningen med en ”Options” knapp och de resterande knapparna blanka. Ersätts nu den CBA knappen längst till höger med en ”Get Info” knapp. Denna funktion aktiveras då en direktsändning har påbörjats, då den hämtar information om den pågående sändningen. Informationen innehåller hur många som tittar på utsändningen och antal applåder och bu rop som tittarna har skickat in. Då all information är insamlad presenteras resultatet i ett poppuppfönster för användaren.

Inställningsvyn

Vyn för inställningar är uppbyggd med två enkla textfält för interaktion. Där det ena fältet representerar användarnamnet och det andra fältet representerar lösenordet. Dessa kommer gemensamt att skapa den unika nyckel som används när identifiering mot servern sker.

I exemplet till höger, Figur 12, ser vi de två textfälten med enkla etiketter över dem för att visa vad som skall skrivas in i dem. Vi ser även att användarfältet är fokuserat och texten i fältet är markerad.

För att navigera sig mellan de olika kontrollerna används upp och ner knapparna på mobiltelefonenstyrkors.

Figur 12 Exempel på inställnings vyn

Diskussion

Denna diskussion är uppdelad i två delar. Första delen återkopplar jag till mitt syfte med examensarbetet för att undersöka om jag har besvarat de frågor som fanns i frågeställningen. Den andra delen har jag använt till att redogöra för vissa speciella problem som jag har stött på under utvecklingen av programvaran. Detta är gjort i avsnittet problemlösnings nedan.

Som en inledning till denna diskussion skulle jag vilja understryka att denna lösning kommer vara den enda fungerande för att strömma video till äldre mobiltelefoner. Detta på grund av att den äldre Java videospelarna inte stödjer RTSP protokollet. Detta gör att spelaren alltid kommer att vänta till den datamängd som den skall spela upp har ett korrekt slut. Vilket i sin tur leder till att en lösning där en riktig ström används ej kommer att fungera.

Syftesåterkoppling

För att återkoppla till den första frågan som ställdes i syftesformuleringen, gjordes en undersökning med den aktuella test telefonen. Testet utfördes med videoklipp som hade en bestämd längd på tre sekunder. Resultatet visas i Tabell 2 nedan.

Start	Slut	Skillnad	Filstorlek i Kb
N/A	00:00:20.36	N/A	33
00:00:21.49	00:00:24.45	1,12	34
00:00:25.27	00:00:28.36	0,82	40
00:00:29.57	00:00:32.44	1,21	39
00:00:33.51	00:00:36.47	1,07	35
00:00:37.33	00:00:40.43	0,86	39
00:00:41.36	00:00:44.33	0,93	39
00:00:45.45	00:00:48.51	1,12	37
00:00:49.72	00:00:52.43	1,21	36
00:00:53.72	00:00:56.31	1,29	36
00:00:57.71	N/A	1,40	38
		Medel: 1,103	Medel: 36,909

Tabell 2 Mätvärden av videoglapp och videostorlek

När ett medel tas av dessa tio mät punkter ser vi att ett glapp i storleksordningen 1,103 sekunder är att vänta vid filväxling. Denna siffra är hög men inte missvisande. Av egen erfarenhet av att ha sett en direktsändning i ShiShi nätverket finns det små glapp.

En kritisk granskning av undersökning bör också göras. Tänkbara felkällor som har inverkat till ett tveksamt resultat kan vara att; det stoppur som användes vid undersökningstillfället var gjort i Flash och möjligtvis för långsam i sin uppdatering eller att videospelaren som mätvärdena sedan lästes från ej hade en bild för bild funktion vilket medförde att det var svårt att avgöra exakt start och slut för filen.

För att minska mellanrummet ytterligare skulle en tänkbar implementering av en till instans av CVideoRecorderUtility klassen. Här är det dock stor risk att operativsystemet på telefonen nekar åtkomst till ena av de två objekten då de kommer att försöka att använda kameran samtidigt eller nästintill samtidigt.

Om vi sedan granskar den andra frågan som ifrågasatte om minnet på mobiltelefonen skulle räcka till visar det sig en farhåga som var helt onödig. Test telefonen har ett interntminne på 160 Mb[15] som används dynamiskt till de applikationer och tjänster som körs för tillfället. Om vi sedan tittar på genomsnittsstorleken på de filer som vi skapar, 36,909 Kb, är det lätt att räkna ut att om inte telefonen kör allt för många program samtidigt kommer minnet att gott och väl räcka.

Det sista ämnet som behandlades i syftesformuleringen ifrågasatte processorkraften hos mobilen. Detta visade sig vara ett mindre problem än väntat. Eftersom programvaran för att sända video in till ShiShi systemet från en stationär dator krävde en konvertering till 3GP formatet, medan i telefonen går det direkt att spara video till detta format. I och med denna fördel kan vi spara in en stor mängd beräkningar och gör att programvaran är mindre processorkrävande i telefonen. Men om det hade funnits mer processorkraft tillgänglig skulle vi möjligtvis kunnat minska det blanka mellanrum som uppstår mellan de olika filerna ytterligare.

Problemlösning

I och med att jag började implementera nätverkskommunikationen var det naturligt att det var där jag mötte de första problemen.

När data skall läsas från en socket görs den till en åttabitars deskriptor. En deskriptor är i standardutförande en enkel buffert. Datatypen håller reda på hur mycket som finns i bufferten och hur stor den kan bli. Dessa två värden kan kontrolleras med funktionerna Length() och MaxLength(), där Length() representerar hur långt vi har fyllt bufferten och MaxLength() är hur stor bufferten kan bli. Dessa två värden sparas i början av datatypen. Här lyder då följande förhållande mellan de omtalade funktionerna:

```
0 <= Length() <= MaxLength()
```

För att läsa data från en nätverksändpunkt används följande funktion:

```
void RSocket::Read( TDes8& aDesc, TRequestStatus& aStatus )
```

Eftersom meddelanden kan vara av variabel längd beroende på om de skall skickas eller om det är ett meddelande som skall tas emot, använder jag klassen HBufC, vilket allokerar minne i telefonens heap. Det är här problemet uppstår, när allokeringen sker görs den i jämna ord (word aligned). Detta leder till att man kan i princip få mera utrymme i sin deskriptor än vad som efterfrågades. Det leder till stora problem eftersom funktionen som läser bytes från nätverket och placerar dem i bufferten jämför med MaxLength() värdet. Men eftersom vi nu kan komma att ha en buffert som är större än det vad vi begärde, kommer läs funktionen att ibland fastna i en evig exekvering då den väntar på information som aldrig kommer att skickas.

Lösningen till det här problemet är inte helt klart för en person som inte har satt sig in i Symbian och hur de som har utvecklat Symbian vill att programmerare skall använda språket. Eftersom grundtanken är alla alltid skall använda de nya datatyperna som Symbian har infört ges inte möjligheten till att använda de äldre datatyperna i kommunikationen mellan de olika lagren i Symbians arkitektur. Men som tur är har de även implementerat en datatyp som omsluter pekare, TPtr. Detta blir vår räddning.

[15] Nokia.com, http://europe.nokia.com/A4323297, 2008-06-10.

Vi skapar och allokerar minne själva till en array med bytes. Sedan skapar vi ett TPtr8 objekt och omsluter en pekare till första positionen i vår array. Därefter sätter vi manuellt en max längd på det nya pekar objektet, max längden i det här fallet är längden på det inkommande meddelandet. När detta är gjort använder vi det nya objektet till att anropa nätverket läs funktion. Genom detta har vi tagit oss förbi problemet med möjligheten att få för mycket utrymme då lagring på heapen sker genom att istället lägga meddelandet på stacken.[16]

Under utvecklingen av användargränssnittet stötte jag på ett annat intressant problem. Problemet är intressant på grund av att det är en av de vanligaste frågorna som ställs på de hjälpforum som jag har besökt.

Det är teckeninmatningen på telefoner som ställer till svårighet för många programmerare. För att erhålla en fungerande inmatande av tecken till en kontroll krävs det inte bara att kontrollen tillåter tecken utan också att den vy som kontrollen ägs av har vissa egenskaper, vilket många missar. Till att börja med behöver vyn implementera en till funktion som returnerar de olika typerna av inmatning den tillåter, som i grundinställning bara är siffror. Denna funktion visas i Figur 13 under.

```
TCoeInputCapabilities InputCapabilities() const {
     TCoeInputCapabilities iIC( TCoeInputCapabilities::ENavigation |
                                TCoeInputCapabilities::EAllText );
     iIC.MergeWith( CCoeControl::InputCapabilities() );
     return iIC;
}
```

Figur 13 Den implementerade funktionen i vyn

Som ni ser i funktionen över aktiverar jag även möjligheten att ta emot inmatning från navigeringstangenter, detta på grund av att fönstret kommer att innehålla flera kontroller och dessa tangenter används för att förflytta sig i mellan dem. Sedan lägger jag även till de inställningar som redan finns och sammanfogar dem med de nya för att de äldre inställningarna som har gjorts inte skall gå förlorade.

När detta är gjort måste även vyn läggas till på stacken som ägs av ApplicationUI klassen och implementera funktionen *OfferKeyEventL* för att någon inmatning från början skall kunna ske. Därefter måste den nämnda funktion reglera vilka kontroller i fönstret som skall få inmatningen och i vilken ordning det skall ske. Som grundinställning tillåts inte teckeninmatning till text- och sifferinmatnings- kontrollen CEikEdwin. Här måste inmatningssättet ändras vid initieringen av kontrollen i *ConstructL* funktion, dessa variabler går nämligen inte att ändra på då objektet har gått igenom hela sin tvåfas kontruktor.

Att kunna lita på dokumentationen från Symbian för ett API som företag själv har utvecklat trodde jag var en självklarhet. Men detta är tyvärr inte fallet. När jag skrev modulen för videoinspelning hittade jag en funktion, *CVideoRecorderUtility:OpenDesL*, som lät idealisk. Den spelar in data direkt till en buffert i minnet. Om jag använder denna funktion skulle telefonen slippa besväret med att interagera med filsystemet, vilket i sin tur skulle minska mellanrummet mellan video filerna. En riktigt bra lösning helt enkelt.

[16] Wlu, Reading given number of bytes with RSocket::Read(), http://www.newlc.com/Reading-given-number-of-bytes-with.html, 2007-07-15.

Vid en närmare granskning av funktionen visade det sig att den aldrig blev implementerad, men fullt dokumenterad. Detta gällde även de flesta andra funktionerna i CVideoRecorderUtility som var till för att spela in video, den enda funktionen som fungerade var *OpenFileL*[17].

[17] http://wiki.forum.nokia.com/index.php/KIS000590_-_Unsupported_methods_in_CVideoRecorderUtility, 2007-08-34.

Slutsatser

Den enskilt största slutsatsen vi kan dra av detta examensarbete är att det är möjligt att genomföra en direktsändning från en mobiltelefon. Jag kan även sträcka mig till att vid perfekta förhållanden kommer denna teknologi att fungera över GPRS nätet. Detta då under förutsättning att den sämsta kvalitén av video används och perfekta signalförutsättningar till GSM masten finns. Det som är positivt med detta är att det kommer att finnas gott med utrymme för vidare förbättring av bild och ljudkvalité av sändningen, då den valda 3G teknologin ger plats för detta.

Det är också viktigt att understryka att för att en sådan här applikation skall kunna fungera i ett skarpt läge måste algoritmen för video växling ses över. Kanske bör även ett eget filter eller liknande skrivas då en växling mellan videofiler bör hanteras på en lägre nivå än vad nuvarande sker. Allt detta för att minimera det mellanrum som uppstår.

En annan slutsats som är viktig att dra är att denna metod *alltid* kommer att leda till en informationsförlust vid data växling. Viket i min mening är en av de största nackdelarna med lösningen till problemet. I det långa loppet kommer denna förlust i sin tur leda till en försämrad helhets upplevelse av produkten.

Litteraturförteckning

[1] **Användarcentrerad systemdesign**
Jan Gulliksen, Bengt Göransson
Studentlitteratur, 2002
91-44-02029-5

[2] **Designing Interactive Systems**
David Benyon, Phil Turner, Susan Turner
Addison-Wesley, 2005
0-321-11629-1

[3] **Symbian OS C++ for Mobile Phones Volume 2**
Richard Harrison
Symbian Press, 2004
0-470-87108-3

[4] **GSM, GPRS and EDGE Performance**
Timo Halonen, Romero García Romero, Javier Romero, Juan Melero
John Wiley and Sons, 2003
0-470-86694-2

[5] **Protocols for High-efficiency Wireless Networks**
Alessandro Andreadis, Giovanni Giambene
Springer, 2003
1-402-07326-7

[6] **UMTS and Mobile Computing**
Alexander Joseph Huber
Artech House, 2002
1-580-53264-0

[7] **WCDMA for UMTS: HSPA Evolution and LTE**
Harri Holma, Antti Toskala
John Wiley and Sons, 2007
0-470-31933-X

[8] **Symbian OS C++ for Mobile Phones**
Richard Harrison, Phil Northam
John Wiley and Sons, 2003
0-470-85611-4

[9] **The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS**
Ben Morris
John Wiley and Sons, 2007
0-470-01846-1

[10] **Design Patterns Explained: A New Perspective on Object-oriented Design**
Alan Shalloway, James Trott
Addison-Wesley, 2002
0-201-71594-5

[11] **Patterns of Enterprise Application Architecture**
Martin Fowler, David Rice
Addison-Wesley, 2003
0-321-12742-0

[12] **Introduction to Algorithms**
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
MIT Press, 2001
0-262-03293-7

För de som är vidare intresserade och vill lära sig mer om Symbian finns information på dessa hemsidor.

[1] **Symbian OS Basics, Workbook**
http://www.forum.nokia.com/info/sw.nokia.com/id/981e8e7b-668c-449a-b0e0e7e5f534c39d/Symbian_OS_Basics_Workbook.html
2008-06-25

[2] **Nokia's Developer Community Wikipedia**
http://wiki.forum.nokia.com/index.php/Wiki_Home
2008-06-25

[3] **Nokia's Developer Community Forum**
http://discussion.forum.nokia.com/forum
2008-02-20

[4] **NewLC**
http://www.newlc.com
2008-06-25

[5] **Symbian Developer Network**
http://developer.symbian.com/main/index.jsp
2008-03-13

[6] **Symbain SDK Documentation**
http://www.symbian.com/developer/techlib/v70sdocs/doc_source/index.html
2008-06-25