Datavetenskap

Annika Billqvist och Katarina Friberg

MyDOF och Wireless Java

Examensarbete, C-nivå

2002:12

MyDOF och Wireless Java

Annika Billqvist och Katarina Friberg

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

Annika Billqvist

Katarina Friberg

Godkänd, 2002-06-04

Examinator och handledare: Martin Blom

Sammanfattning

Vi har studerat Telia Mobiles WAP-portal MyDOF för att undersöka om det är möjligt att använda Java 2 Micro Edition för att implementera olika tjänster på portalen. För att kunna undersöka detta har vi implementerat ett Yatzyspel avsett för flera deltagande spelare. Slutsatsen är att detta är fullt möjligt, men det finns restriktioner i Java 2 Micro Edition som innebär vissa begränsningar i användandet av detta språk. Vidare har vi undersökt möjligheten att använda Java 2 Micro Edition som ett verktyg för att skapa mer användarvänliga gränssnitt i de befintliga tjänsterna på portalen. Resultatet av denna undersökning visar att det inte vore ett bra alternativ idag, eftersom det skulle innebära en total omstrukturering av hela MyDOF-portalen och dess tjänster.

Däremot skulle man kunna bygga upp nya tjänster med Wireless Java som går att lägga upp på portalen. De användare som har WAP-telefoner med stöd för Java kan då få tillgång både till dessa tjänster samt de befintliga tjänsterna på portalen.

MyDOF and Wireless Java

Abstract

We have studied the WAP portal MyDOF at Telia Mobile AB, and examined whether it is possible in some way to use Java 2 Micro Edition for implementing the services therein. In order to do this, we implemented a multi user Yatzy game in Java 2 Micro Edition. The conclusion is, that this is possible. However, restrictions in Java 2 Micro Edition make the use of this language limited. Further on, we have to some extent examined the possibility to use Java 2 Micro Edition as a tool for implementing a more user friendly GUI to the existing WAP services. This does not seem to be a viable option as of today since this would call for a total recreation of the entire portal.

However, new services might be developed using Wireless Java and integrated into the existing portal. The users having access to a java enabled WAP phone, may then take advantage of these new java based services as well as the older WAP services.

Key words: Java 2 Micro Edition, MIDP, CLDC, KVM, WAP, WML and MyDOF

Förord

När vi började vårt examensarbete här på Telia Mobile AB i Karlstad, tyckte vi att 20 veckor var lång tid och att det inte skulle bli några problem med att skriva vår uppsats. Men utan Stefan Holmgren på Telia Mobile och Martin Blom på Karlstads universitet hade den aldrig blivit klar i tid. Därför vill vi speciellt tacka dem båda för all hjälp och stöd vi fått.

Vi vill också tacka övrig personal på Telia Mobile för deras vänliga bemötande av oss. Vi har under vår tid på Telia Mobile haft givande diskussioner med de andra examensarbetarna och vill därför tacka även dem för detta.

Vi vill även rikta ett tack till våra pojkvänner Niklas Modin och Ulf Back för att ha stått ut med oss under den här tiden.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Den tekniska utvecklingen av mobila terminaler har gjort det möjligt att kunna använda Internet som informationskälla även från en mobiltelefon. Ett flertal modeller av mobiltelefoner har stöd för WAP och idag finns många olika WAP-portaler tillgängliga för mobiltelefonanvändare. MyDOF är Telia Mobiles portal och innehåller en mängd olika WAP-tjänster där gränssnitten är utvecklade i WML. För användaren kan detta upplevas som långsamt då varje länk i gränssnittet representerar ett nytt gränssnitt som måste hämtas från server till telefonen.

Telia Mobile vill se ett ökande antal användare av MyDOF-portalen och vår uppgift är att undersöka om Wireless Java är en lämplig teknik för utveckling av befintliga och nya tjänster på portalen. Man ville också undersöka om det var möjligt att ta fram en tjänst med en klient/serverlösning. I kapitel 3 finns en mer detaljerad beskrivning av vår uppgift. Resultatet av vårt examensarbete som presenteras i denna rapport kommer att fungera som underlag för framtida utveckling av MyDOF-portalen.

För att undersöka om det går att förbättra användargränssnitten i de befintliga tjänsterna har vi jämfört Wireless Java-tekniken med WAP/WML och funnit att det är möjligt att realisera på längre sikt. Det kommer dock att innebära en total omkonstruktion av hela MyDOF-portalens arkitektur. Man vill också att vi skulle ta fram någon ny tjänst där vi använder den nya tekniken för att implementera den. Vi valde att göra ett Yatzyspel där flera användare kan spela Yatzy med varandra via en server, d v s en klient/serverlösning, där själva logiken finns på klienten.

2 Generell struktur

Rapporten är strukturerad på följande vis. Kapitel 3 presenterar själva problemformuleringen vilken innefattar bakgrund, problem och syfte. Kapitel 4 redogör för vilka metoder som använts för insamling av material. Kapitel 5 innehåller den tekniska bakgrunden som är indelad i tre stora delar. Den första delen är en beskrivning av WAP och WML (kapitel 5.1) som sedan följs av en redogörelse hur MyDOF-portalen fungerar (kapitel 5.2), för att avslutas med en redogörelse för Wireless Java (kapitel 5.3). Kapitel 6 är den utredande delen av rapporten som analyserar det insamlade materialet och besvarar de frågor som formulerats i problemformuleringen. Kapitel 7 är konstruktionsdelen av rapporten och här beskrivs implementationen av Yatzyspelet som vi konstruerat. Kapitel 8 redovisar resultatet av den utredande delen samt rekommendationer och erfarenheter från konstruktionsdelen. Slutligen ger vi i kapitel 9 en sammanfattning av vilka slutsatser vi har dragit under vårt examensarbete.

3 Problemformulering

Kapitlet inleds med en kort bakgrund om företaget Telia Mobile som vi gör vårt examensarbete hos. Därefter följer de problem och frågeställningar som Telia Mobile vill att vi ska besvara i vår rapport. Slutligen redogörs för syftet med examensarbetet.

3.1 Bakgrund

Telia Mobile är ett företag inom Teliakoncernen och ansvarar för det mobila nätet samt utveckling av integrerade fasta/mobila tjänster. 1999 lanserade man Sveriges första WAP-portal, MyDOF.[3]

MyDOF-portalen innehåller ett antal olika kategorier av mobila Internet-tjänster. Den största andelen av tjänsterna är informationstjänster, men det finns också olika typer av transaktions- och positioneringstjänster.

Tjänsterna på portalen har främst WML som grafiskt gränssnitt. Plattformen som används är Oracle Internet Application Server Wireless Edition. För teknisk bakgrund se kapitel 5.

3.2 Problem

För tjänsterna på MyDOF-portalen är det viktigt att gränssnitten är användarvänliga, så att det är lätt att använda dem vilket i sin tur leder till att fler använder tjänsterna. Problemet idag är att man använder WML som grafiskt gränssnitt. WML är ett metaspråk definierat i XML, vilket kan vara begränsande för vissa tjänster. Därför vill man ta fram en alternativ teknik. Dessutom vore det önskvärt om den alternativa tekniken kunde göra det möjligt att flytta logik från server till klient. Vår uppgift är att undersöka om Wireless Java är den alternativa teknik man söker. Vår rapport kommer att besvara följande frågeställningar:

- Kommer Wireless Java att medföra att man kan utöka och förenkla användandet av tjänsterna på MyDOF-portalen?
- Vilka nya tjänster blir möjliga att utveckla genom att använda Wireless Java?
- Vilka fördelar respektive nackdelar finns det med Wireless Java-tekniken?

3.3 Syfte

Telia Mobile vill bibehålla den höga kvalité som man har på MyDOF-portalen. Därför vill man utvärdera Wireless Java-tekniken och se vilka nya möjligheter tekniken kan erbjuda.

4 Metodbeskrivning

Vi började arbetet med att göra en tidsplan där vi delade in vår uppgift i flera arbetspaket och satte upp olika delmål för att kunna hålla oss till tidsplanen. De första veckorna ägnade vi oss åt att samla in material för att läsa in oss på ämnet. Den största delen av materialet har vi inhämtat från Internet men vi har även sökt information i annan litteratur se [1] och [2].

Efter inläsningsperioden fortsatte vi med att implementera ett Yatzyspel i syfte att kunna besvara frågeställningen om klient/serverlösningar med tunga klienter är genomförbart. För att utveckla Yatzyspelet använde vi utvecklingsmiljön Forte som också innehåller en testmiljö i form av en mobiltelefonemulator. I emulatorn har vi under hela projekttiden testat Yatzyspelet för att se hur det skulle fungera i en riktig mobiltelefon. Slutligen testade vi Yatzyspelet i riktiga mobiltelefoner som har stöd för Java. De modeller vi testade är Siemens SL 45i och Motorola Accompli 008.

5 Teknisk bakgrund

Kapitlet berör de tekniska aspekter som ligger till grund för utredningen av vår problemformulering. Vi kommer att redogöra för hur MyDOF-portalen fungerar, samt beskriva den teknik som används. Slutligen kommer Wireless Java att tas upp.

5.1 WAP – WML

WAP/WML är den teknik som används på portalen idag. Redogörelsen för denna teknik är av stor betydelse för att förstå hur MyDOF-portalen fungerar. En kort historisk bakgrund för WAP inleder kapitlet som sedan fortsätter med en beskrivning av WAP-arktitekturen och avslutas med en presentation av WML.

5.1.1 Historik

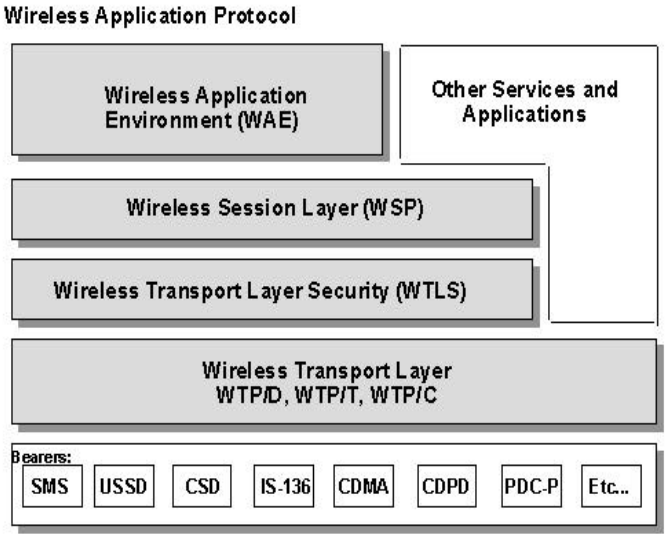
WAP (Wireless Application Protocol) är en protokollspecifikation framtagen av WAP-forum, en organisation som grundades 1997 av Ericsson, Motorola, Nokia och UnwiredPlanet [10]. Resultatet av deras arbete har definierat en standard för utveckling av applikationer för trådlösa nätverk.

Utvecklingen av olika nätverksteknologier har haft en enorm tillväxt under de senaste åren[4]. Allt fler använder Internet vilket har gett upphov till utveckling av nya informationstjänster. Problemet har varit att kunna utveckla Internet-applikationer för trådlösa terminaler som t ex mobiltelefoner och PDA:er (Personal Digital Assistant).

Det finns vissa begränsningar för att kunna använda Internet-teknik i mobiltelefoner. De har begränsad strömförsörjning och mycket lägre minneskapacitet än datorer. Dessutom är processorkraften inte lika stor, och framför allt är fönsterstorleken inte anpassad för att visa en Webbsida. För att hantera dessa begränsningar har WAP tagits fram.

5.1.2 WAP-arkitekturen

Strukturen för WAP bygger mycket på Internet-arkitekturen [4]och protokollstackarna ser snarlika ut. Protokollstackens översta lager i WAP är oberoende av de underliggande trådlösa näten vilket medför en stor skalbarhet med avseende på processorkraft, minne och bandbredd.



Figur 5-1: Protokollstacken för WAP

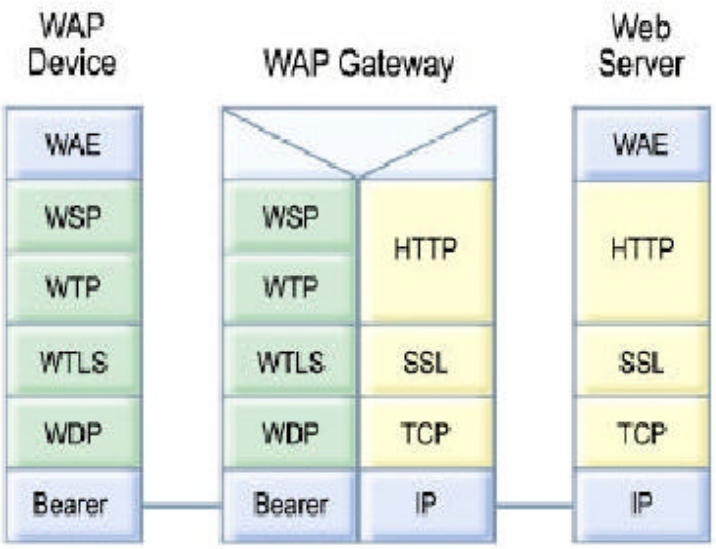
I Figur 5-1 ovan visas strukturen över protokollstacken för WAP. I det understa lagret ligger de olika bärarna som är helt oberoende av den övriga arkitekturen. Bärarna kan därför enkelt bytas ut, utan att man behöver göra om de överliggande lagren. Ovanför dessa ligger transportlagret som skickar data över det trådlösa nätet. Transportlagret (Wireless Transport Layer) har stöd för olika typer av protokoll för att skicka data över nätet, t ex datagramservicen där man använder UDP och WDP. För att sätta upp en förbindelse för IP-bärare används TCP som kan profileras för att klara av de egenskaper trådlösa nät har.

Säkerhetstransportlagret (Wireless Transport Layer Security) ansvarar för klient/server autentisering, integritet, och flera andra säkerhetsmekanismer. Protokollet WTLS (Wireless Transport Layer Security) är definierat för säker transport över datagramnät och TLS (Transport Layer Security) är definierat för säker transport i förbindelseorienterade nät t ex TCP/IP. Säkerhetsmekanismerna som dessa tjänster ger används även av de övriga lagren i protokollstacken.

Sessionslagret (Wireless Session Layer) innehåller olika tjänster för uppkoppling och nedkoppling av en förbindelse samt ser till att denna kan hålla ett tillstånd hos klient och server. Ett exempel är Push-sessionen som ser till att en WAP-terminal är redo att ta emot data från en gateway (server). WSP (Wireless Session Protocol) och WTP (Wireless Transaction Protocol) är de protokoll som finns definierade för överföring av information mellan olika nätverkselement se Figur 5-2.

WAE (Wireless Application Environment) är översta nivån vars huvudsakliga syfte är att vara en miljö för utveckling av applikationer och tjänster, som kan användas på många olika typer av trådlösa plattformar. WAE är en microbrowser-miljö som innehåller definitioner för

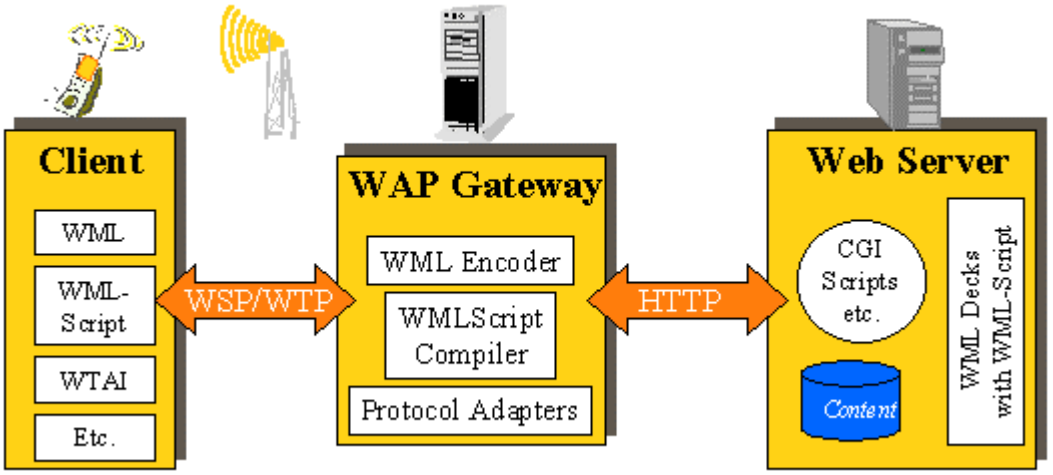
markup-, scripting- och stylesheetspråk samt telefontjänster och programmeringsinterfaces som är optimerade för mobila terminaler.



Figur 5-2: Exempel på protokollstacken

I mobila terminaler finns en microbrowser, som har minimala krav på hårdvara, minne och processorkraft, vilken motsvaras av en standard webbrowser i en vanlig terminal.

WAP kommunicerar på samma sätt som vanlig Internettrafik [9], med skillnaden att mellan klient (mobiltelefon) och server finns en gateway, som anpassar trafiken till det trådlösa nätverket. När den mobila terminalen skickar en informationsbegäran från en viss URL sker det över WSP och WTP se Figur 5-3.



Figur 5-3: WAP-arkitekturen

Den mobila terminalen skickar sin informationsbegäran till en gateway som tar emot och konverterar den från WSP till HTTP för att sedan vidarebefordras till Internet. Samma gateway utför också DNS-kontroll av servern utifrån klientens begäran. Från webbservern hämtas data och transformeras till ett lämpligt format, vanligtvis WML.

Vidare skickas data tillbaks till den mobila terminalen men då sker processen omvänt, från webbservern till en gateway skickas data via HTTP. Där sker en konvertering till WSP och data skickas vidare till den mobila terminalen där data visas i lämpligt format i displayfönstret.

Det finns även andra konfigurationer som WAP-arkitekturen kan anpassas till men vi kommer inte att beröra dessa i vår rapport.

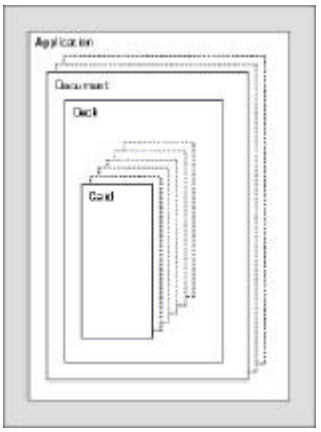
5.1.3 WML

För att utveckla WAP-applikationer används framför allt WML (Wireless Markup Language), ett presentationsspråk som är baserat på XML (Extensible Markup Language). WAP-Forum har tagit fram en formell specifikation för WML [8] som också inkluderar DTD:n (Document Type Definition). Denna specifikation definierar syntax, variabler och element som används i en giltig (valid) WML-fil. För att vara ett giltigt WML-dokument som kan köras måste det följa den DTD som finns specificerad.

WML är designat för att kunna användas som gränssnitt och visa data på trådlösa terminaler som mobiltelefoner och PDA:er d v s terminaler med små displayer och begränsade möjligheter för inmatning av data. En WML-sida för mobiltelefoner kan sägas

motsvara en HTML-sida för Internetbrowsern, d v s den innehåller kod för att microbrowsern ska veta vad som ska visas i displayen på telefonen.

En WML-sida kallas också för ett deck och innehåller ett eller flera cards se Figur 5-1, vilka kan innehålla t ex text, bilder, länkar och inmatningsfält [7]. Varje card specificerar en interaktion mellan mobiltelefon och användare.



Figur 5-1: Kortleksprincipen i WML

När man laddar ner en WML-sida från en WAP-server till en mobiltelefon laddar man alltså ner ett deck med alla dess cards. När decket tas emot aktiveras vanligtvis det första cardet i det, men man kan aktivera vilket som helst av dem. Det finns olika card-element som gör det möjligt att navigera mellan olika card i samma deck eller mellan två olika deck. Om inget annat än det första cardet är specificerat för att aktiveras, visas det första cardets innehåll i decket i displayen. När man hämtat ett deck från en server så kan man navigera mellan de olika carden som finns i decket utan att behöva tillgång till någon server.

Det finns inget krav på att alla card ska ligga i ett enda deck. Rekommendationen är att dela upp carden i flera deck och på så vis minska storleken på ett deck. Det beror på att olika mobiltelefoner har olika minneskapacitet. Även informationen i ett card kan behöva delas upp på flera card eftersom de kan innehålla fler rader än vad som är möjligt att visa i ett enda fönster. Om man inte delar upp i flera mindre card så kan användaren behöva scrolla nedåt i fönstret för att kunna se alla rader.

För att uppnå flexibilitet kan man även använda variabler i WML. På detta sätt kan man styra innehåll och gränssnitt beroende på inmatningsdata från användaren.

För mer information om WML, se [6] och [8].

5.2 MyDOF-portalen

MyDOF-portalen är Telias WAP-portal som innehåller ett stort tjänsteutbud för mobilt Internet. I detta kapitel redogör vi för hur MyDOF-portalen fungerar idag genom att beskriva den överliggande och underliggande strukturen.

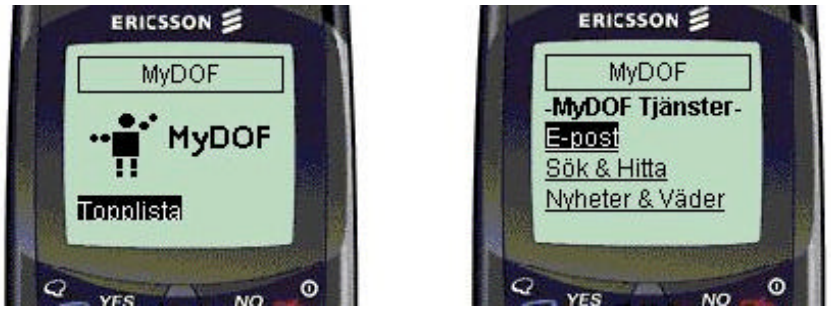
5.2.1 Portalens överliggande struktur

Som användare av portalen finns tre krav som måste vara uppfyllda. Användaren måste ha ett mobiltelefonabonnemang registrerat hos Telia Mobile och vara medlem i www.teliamobile.se samt ha tillgång till en WAP-telefon.

MyDOF är uppbyggd kring två centrala delar; webbplatsen och WAP-telefonen. På webbplatsen anger man vilka tjänster man vill nå när man loggar in på MyDOF med WAP-telefonen. För att navigera på portalen krävs vissa inställningar i WAP-telefonen.

På webbplatsen finns även möjlighet att ansluta e-postkonton så att man kan läsa och skicka e-post via telefonen. Man kan också skapa sin egen mobila hemsida, en WAP-sida.

Det finns olika kategorier av tjänster på portalen idag. Det finns informationstjänster som t ex *Finans*, *Hus & Hem* och *Nyheter & Väder*. Även ett antal positioneringstjänster finns tillgängliga t ex *BostadNäraDig*, *Friendfinder* och *RättVägNäraDig*, samt en samling spel.

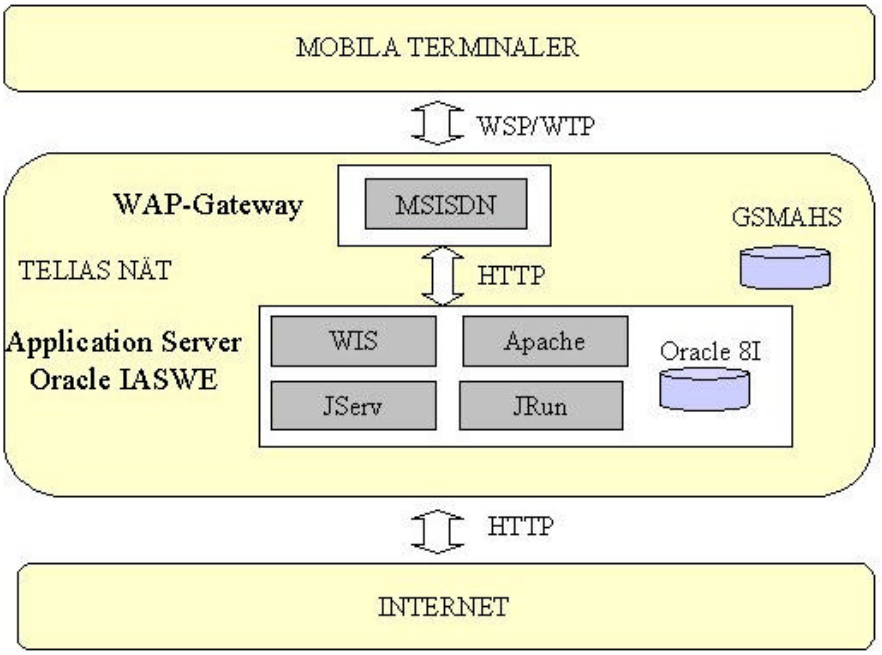


Figur 5-1: Exempel på gränssnitt på MyDOF-portalen

Gränssnittet som man använder på portalen är WML (se kapitel 5.1.3) och det består av länkar som man navigerar sig fram och tillbaka emellan. Startsidan för MyDOF (se Figur 5-1, vänstra bilden) innehåller ett antal länkar till de tjänster användaren valt att lägga till från webbplatsen. Om man klickar på MyDOF-tjänster (se Figur 5-1, högra bilden) visas de olika alternativen som länkar, t ex *Finans*, *Hus & Hem* och *Nyheter & Väder*. Om man vill läsa nyheter så klickar man på det tredje alternativet och förs vidare tills de önskade nyheterna visas i textformat i displayfönstret på telefonen.

5.2.2 Portalens underliggande struktur

Om man tittar på portalens underliggande struktur ser man att den består av tre stora delar med skilda innehåll: Mobila terminaler, Telias nät och Internet.



Figur 5-1: MyDOF-portalens struktur

5.2.2.1 Mobila terminaler

Den första delen består de trådlösa terminaler vilka stödjer WAP, som t ex mobiltelefoner och PDA:er. Det protokoll som används för att skicka en informationsbegäran är WSP/WTP, se Figur 5-1. Tillsammans med övrig header-information skickas även MSISDN (Mobile Station ISDN Number) med i informationsbegäran. MSISDN innehåller telefonnumret till terminalen.

5.2.2.2 Telias nät

Den andra delen består av en WAP-gateway, fem applikationsservrar med två databaser varav den ena innehåller redundant data, samt en databas som innehåller information om Telia Mobile-kunder, GSMAHS (Global Systems for Mobile Communications Abonnent Hanterings System).

En WAP-gateway tar emot informationsbegäran från terminalen och skickar den vidare till applikationsservern tillsammans med MSISDN, se Figur 5-1. Då endast Telia Mobiles kunder har tillgång till MyDOF-portalen måste en kontroll ske av telefonnumret. Denna kontroll görs mot databasen GSMAHS.

Applikationsserverarna som alla är av typen Oracle Integrated Application Server Wireless Edition, är uppbyggd av flera olika delar. Det finns två databaser, Oracle 8I, där alla MyDOF-tjänster finns lagrade samt information om användare och all konfigurering.

WIS (Web Integration Server) och Apache är två webbservrar som används för att hämta data som är av intresse från en html-sida på Internet. Inhämtning av data utförs av en adapter som hämtar data direkt från en html-sida och transformerar det till Simple Result som är ett XML-format. Vidare körs Simple Result genom en Device Transformer och genererar data till WML om det är en mobiltelefon som har skickat en begäran om informationen.

JServ är en servletmotor som fungerar tillsammans med webbservern Apache och för att exekvera JSP-kod (Java Server Page) används JRun vilken är en JSP-server.

JSP är ett skriptspråk som används för att utveckla webbapplikationer som är plattformsoberoende. Man kan med denna teknik utveckla dynamiska webbsidor som är lätta att underhålla och förändra utan att påverka användargränssnittet. JSP är en utvidgning av servletteknologin som underlättar utveckling av HTML- och XML-sidor. Detta gör det lättare att kombinera statiska gränssnitt med ett dynamiskt innehåll.

5.3 Wireless Java

I detta kapitel följer en redogörelse för hur Wireless Java är uppbyggt. Först kommer en kort historik för Java, därefter tar vi upp hur Java 2 Micro Edition är uppbyggd och vilka olika profiler som finns. Slutligen redovisar vi skillnader mellan Wireless Java och Java med avseende på olika programkonstruktioner.

5.3.1 Historik

När Sun utvecklade den senaste versionen av Java (Java 2) så splittrade man upp Java i tre delar, J2ME (Java 2 Micro Edition), J2SE (Java 2 Standard Edition) och J2EE (Java 2 Enterprise Edition), där J2SE används för de traditionella applikationerna. Sun har utökat J2SE jämfört med Java 1.2 så att den numer innehåller fler klasser och gränssnittspaketet Swing. J2ME är en förminskad version av J2SE och är tänkt att användas i små trådlösa terminaler, t ex mobiltelefoner.

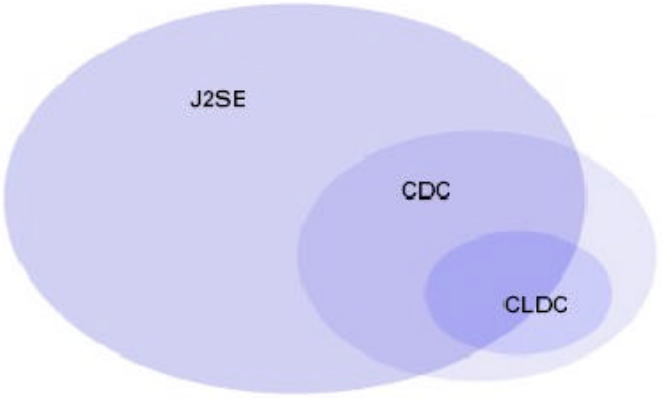
När Java splittrades i tre delar började man fundera på om man kunde förbättra den virtuella maskinen. J2SE stödjer fortfarande JVM (Java Virtual Machine), men man utvecklade en ny virtuell maskin som man kallar HotSpot VM (Virtual Machine). Efter att ha utvecklat HotSpot VM för J2SE började man förbättra den virtuella maskinen för J2ME som ledde till att man utvecklade KVM (Kilo Virtual Machine). KVM är en ny implementation av

JVM och är komprimerad för att vara bättre anpassade för små terminaler. J2ME stödjer nu både KVM och den klassiska virtuella maskinen.

J2ME-arkitekturen är uppbyggd av konfigurationer och profiler, vilket kommer att tas upp mer ingående i de nästkommande delkapitlen (kapitel 5.3.2 och 5.3.3).

5.3.2 J2ME-arkitekturen (Konfigurationer och Virtuella Maskiner)

I J2ME finns det idag två standardkonfigurationer, dessa är CDC (Connected Device Configuration) och CLDC (Connected Limited Device Configuration). I Figur 5-1 visas förhållandet mellan CDC och CLDC, samt deras förhållande till J2SE. CDC och CLDC har olika krav på de virtuella maskinerna och därför finns två olika specificerade CVM respektive KVM.



Figur 5-1: Samband mellan konfigurationerna och J2SE

5.3.2.1 CDC och CVM

CDC är en nerbantad version av J2SE med ett tillägg av CLDC klasserna och används i små terminaler t ex digitalTV-box och PDA:er. CVM liknar J2SE:s virtuella maskin och stödjer allt som finns med i Java 2 version 1.3 virtuella maskin, såsom säkerhet, JNI (Java Native Interface) och RMI (Remote Method Invocation).

Minimikraven som CDC ställer på terminalerna är att de måste ha en 32-bits processor och 2 MB minne. Dessutom måste nätverksuppkopplingen, vilken oftast är trådlös, ha en bandbredd på minst 9600 bps.

C:et i CVM stod från början för ”compact” men detta tyckte Sun var förvirrande likt K:et i KVM så därför står C:et i CVM inte för någonting alls.

5.3.2.2 CLDC och KVM

CLDC är den konfiguration som används mest i J2ME-världen, eftersom man kan köra CLDC-applikationer även på en CDC-terminal. CLDC är specificerad för mindre terminaler som har hårdare krav på kapacitet, minne och nätverkets bandbredd. I CLDC har man försökt att hitta den minsta gemensamma nämnaren för de små terminalerna. Det Sun har kommit fram till är att terminalerna måste ha 160 – 512 KB minne, en 16-bits eller 32-bits processor med en hastighet på 25 Mhz. Dessutom måste nätverksuppkopplingen vara en trådlös tvåvägs förbindelse med en begränsad bandbredd och en låg energiförbrukning, eftersom terminalerna oftast har batterier som energikälla. Terminaler som har dessa krav är t ex mobiltelefoner, navigationssystem, video- och ljudapparater.

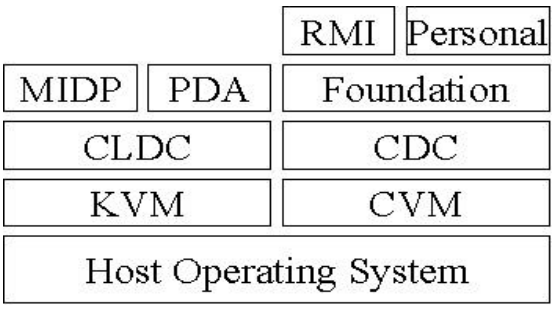
KVM har tagit bort stödet för flyttal, JNI, reflection och delar av garbage-collectorn, samt begränsat runtime error-klasserna till tre stycken (java.lang.OutOfMemoryError, java.lang.VirtualMachineError, java.lang.Error). För att inte ta kraft från terminalerna samt för att göra programmen säkrare, har man i KVM lagt verifikationen av programmen på servern, vilket kallas preverifikation.

CLDC har tagit 37 klasser från J2SE plattformen, som är hämtade från java.lang, java.io och java.util paketen. Från vissa av dessa klasser har man tagit bort funktionalitet för att J2ME ska bli mindre. Dessa klasser kastar vissa specifika exceptions vilket de måste få göra även i J2ME, så det finns 29 exception/error klasser.

I nästa version av CLDC kommer man försöka att göra det mer komplett, men ändå behålla bakåtkompatibiliteten. Man kommer att ta med stöd för flyttal, lägga till mer felhantering och en minimal säkerhetshanterare. CLDC ska fortsätta att ha fokus på små terminaler. När man har gjort dessa förändringar så är CLDC fortfarande väldigt litet jämfört med J2SE och om man vill köra större program ska man fortfarande använda CDC.

5.3.3 Profiler

Profilernas uppgift är att skapa ett gränssnitt till konfigurationerna. En konfiguration kan ha flera olika profiler beroende på vilka terminaler den stödjer. I Figur 5-1 ser man att profilerna ligger ovanpå konfigurationerna och är knutna till en specifik konfiguration. En viss terminal kan endast stödja en konfiguration men kan stödja ett flertal profiler. För CLDC finns MIDP (Mobile Information Device Profile) och PDA, CDC har profilerna Foundation, Personal och RMI. Idag är det bara MIDP som används då de andra är under utveckling.

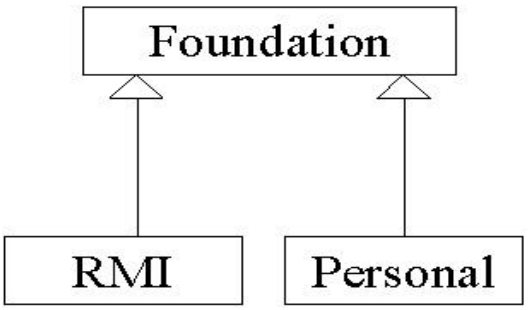


Figur 5-1: Profiler

MIDP-profilen är designad för att användas av mobila terminaler som mobiltelefoner. I profilen har man specificerat ett användargränssnitt och hur nätverkshantering ska fungera. Man kan ladda ner små applikationer, MIDlets (se kapitel 5.3.4) till slutanvändaren. För att MIDP ska fungera på ett tillfredsställande sätt behöver terminalen ha en display som är minst 96 x 54 pixlar stor. Minnet behöver vara 32 KB till Java plattformen, 128 KB sekundärminne för MIDP komponenter och 8 KB för applikationen att spara resultat på. Nätverket ska vara tvåvägskommunikation som är trådlös med begränsad bandbredd.

PDA-profilen har också till skillnad mot MIDP specificerat ett användargränssnitt vilket liknar AWT, men som inte är lika stort. PDA vänder sig till terminaler som har mer kapacitet än terminalerna som använder MIDP. Terminalerna ska ha 512 KB minne och ha en display som är minst 125 x 160 [MB2]pixlar stor.

Foundation-profilen är en stomme för andra profiler som t ex Personal och RMI, så här finns inget användargränssnitt specificerat. Terminalerna som ska använda Foundation måste ha 1024 KB primärminne och 512 KB sekundärminne.

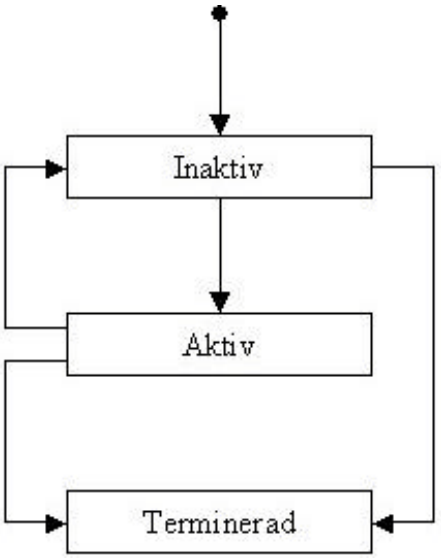


Figur 5-2: Arvshierarki

Personal-profilen ärver från Foundation (se Figur 5-2) och lägger till ett grafiskt användargränssnitt som har kapacitet att köra Java Web applets. Personal kräver att terminalerna har 2.5 MB primärminne och 1 MB sekundärminne vilket är mycket mer än vad Foundation kräver. Även RMI-profilen ärver Foundation och är kompatibel med J2SE RMI. Profilen RMI har samma krav på minne som Personal.

5.3.4 MIDlet och MIDlet Suite

MIDlet är en applikation som körs på en MIDP-terminal och som kan bestå av en eller flera Javaklasser. Dessa klasser packas till en JAR-fil (Java Archive) se kapitel 5.3.5. En MIDlet har en livscykel, se Figur 5-1, medan den exekverar på terminalen. Dess tillstånd är inaktiv, aktiv och terminerad. När en MIDlet startar hamnar applikationen i inaktivt tillstånd då mjukvaran kontrolleras så att den är okej. Därefter går applikationen över till aktivt tillstånd, då körs applikationen och användaren kan interagera med den. Applikationen kan härifrån återgå till inaktivt tillstånd antingen genom MIDP-systemet eller av applikationen [MB3]själv. Applikationen kan flyttas till tillståndet terminerad, både från inaktivt och aktivt tillstånd via MIDP-systemet eller av applikationen själv. En applikation som är terminerad måste lämna tillbaka alla resurser som används under programmets exekvering.



Figur 5-1: Tillståndsdiagram för en MIDlet

En MIDlet Suite är ett paket som innehåller en eller flera MIDlets, men kan också innehålla bilder och klasser. MIDlet Suite packas till en JAR-fil i samband med kompileringen. De MIDlets som ligger i samma MIDlet Suite har tillgång till alla klasser och resurser i de MIDlets som tillhör MIDlet Suite, och kan därför interagera direkt med varandra.

5.3.5 JAR-fil, JAD-fil och JAM

JAR-filerna innehåller oftast en manifestfil som definierar alla attribut som en MIDlet har. Dessa attribut används när terminalen ska identifiera och installera en MIDlet Suite.

Användningen av en manifestfil för att beskriva en MIDlet i en MIDlet Suite kan vara problematiskt. Innan man laddar en JAR-fil till terminalen kontrollerar JAM (Java Application Manager) om det finns tillräckligt med minne på terminalen för att klara av

programmet. Har man en manifestfil måste man ladda ner JAR-filen för att kunna läsa manifestfilen vilket inte är bra om man inte har tillräckligt med ledigt minne. För att undvika detta används en JAD-fil (Java Application Descriptor). En JAD-fil är en textfil som liknar en manifestfil fast den är inte inpackad i JAR-filen. JAD-filen definierar ett antal attribut och vissa av dessa måste vara lika i både manifestfilen och JAD-filen.

5.3.6 Programkonstruktion

Detta kapitel kommer att redogöra för hur man valt att implementera gränssnitt, händelsehanteringen, nätverksuppkopplingar och databashantering i Wireless Java.

5.3.6.1 Grafiskt gränssnitt

Gränssnittet som man specificerat för MIDP är en del i javax.microedition.lcdui paketet. MIDP UI (User Interface) är inte baserat på AWT eftersom interaktionssätten skiljer sig så pass mycket. AWT är väldigt minneskrävande och man skapar många objekt som lätt kan se till att garbage-collectorn får för mycket att göra för de små terminalerna.

Det finns två delar av gränssnitt, högnivå API och lågnivå API. Högnivå API har förbestämda komponenter som man använder sig av. Vill man till exempel ha en lista (List) skapar man en (se Figur 5-1) och lägger till de val man vill att användare ska välja emellan. Resultatet av koden i Figur 5-1 visas i Figur 5-2. Dessa komponenter rättar sig efter den telefon man kör programmet på. Som programmerare kan man inte påverka hur komponenterna ska se ut, de har en viss storlek och en viss stil. När man använder högnivå API finns det fördefinierade klasser som List, Display, TextField, Form och många fler.

```
public Lista() {
    list = new List("Lista", Choice.IMPLICIT);
    back = new Command("Back", Command.BACK, 1);

    list.addCommand(back);
    list.append("Val 1", null);
    list.append("Val 2", null);
    list.append("Val 3", null);
    list.append("Val 4", null);
}
```

Figur 5-1: Kodexempel på hur man skapar en lista

Figur 5-2: Lista som skapas med hjälp av klassen List som finns i högnivå API:et

Använder man istället lågnivå API så har man en Canvas-klass som visar upp en tom skärm som en MIDlet kan rita linjer och skriva text på (se Figur 5-4). När man ritar på skärmen så har översta vänstra hörnet koordinaterna 0,0. För att få reda på hur stor skärm man har att jobba med använder man sig av metoderna getHeight() och getWidth() som vardera returnerar en integer vilken talar om hur många pixlar skärmen består av. Med lågnivå API får man själv specificera komponenterna och det är då inte säkert att de ser likadana ut i olika terminaler, eftersom terminalerna kan ha olika storlek på displayerna.

```
public void paint(Graphics g) {
    int width = getWidth()/2;

    g.setColor(0,0,0);
    g.setFont(Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
        Font.SIZE_LARGE));
    g.drawString("Välkommen", width, 10, g.TOP | g.HCENTER);
    g.drawString("till", width, 30, g.TOP | g.HCENTER);
    g.drawString("YATZY", width, 50, g.TOP | g.HCENTER);
}
```

Figur 5-3: Kodexempel på hur man skriver text till en Canvas

Figur 5-4: Ett exempel på hur det ser ut när man skriver på en Canvas

Högnivå API används oftast till informationstjänster medan speltillverkarna som vill göra mer avancerade gränssnitt som inte högnivå API tillåter använder sig av lågnivå API.

5.3.6.2 Händelser

Det finns två typer av händelser, den ena är kopplat till högnivå API:et och den andra till lågnivå API:et. När en användare gör ett val så genereras en händelse. Används högnivå API måste klassen implementera något av interfacen CommandListener eller ItemStateListener för att ta hand om händelsen. För att registrera en lyssnare till en komponent så använder man sig av en av metoderna setCommandListener()/setItemStateListener() beroende på vilket interface man implementerat. En komponent som fått en lyssnare på sig kommer att skicka tillbaka en händelse till metoden commandAction()/itemStateChanged() när en användare trycker på komponenten.

Använder man lågnivå API:et har man för varje knapp på terminalen skapat en nyckelkod som registrerar ifall det sker en knapptryckning. Då genereras en händelse som tas om hand av metoder i Canvas klassen. Varje nyckelkod kan kopplas till en spelhändelse (game action), en spelhändelse kan ha flera nyckelkoder kopplat till sig.

5.3.6.3 Nätverk

I J2ME har man skapat ett helt nytt nätverkspaket, *Generic Connection Framework,* eftersom J2SE:s java.io och java.net-paketen inte är anpassade för att användas i små terminaler. Nätverkspaket är ett plattformsoberoende ramverk och innehåller bland annat klassen Connector. När man ska skapa en http-förbindelse använder man sig av den statiska metoden open() som finns i klassen Connector, se Figur 5-1.

Skapa en http-förbindelse:

```
Connector.open("http://www.kau.se");
```

Figur 5-1: Kodexempel på hur man skapar en http-förbindelse

Det går att skapa en förbindelse med hjälp av socket också, även då använder man metoden open(), se Figur 5-2.

Skapa en socket-förbindelse:

```
Connector.open("socket://www.kau.se:80");
```

Figur 5-2: Kodexempel på hur man skapar en socketförbindelse

MIDP ger support åt http-protokollet, eftersom man då kan använda både IP-protokoll och andra protokoll som t ex WAP och I-mode. Använder man sig av WAP eller I-mode så måste man gå via en gateway för att få access till Internet.

När data skickas över Internet är den utsatt för inkräktare precis som den är när man använder en vanlig dator för att skicka eller hämta något från Internet. Många terminaler klarar idag inte av att dekryptera ett krypterat meddelande eftersom det tar för lång tid och för mycket prestanda. Det är med andra ord en resursfråga [MB5]om man ska kryptera ett meddelande eller inte.

5.3.6.4 Databaser

När man exekverar ett program kan man ibland få fram data som man vill spara till ett annat exekveringstillfälle av samma eller ett annat program och denna data kan då sparas i en databas. J2ME har gjort en egen databashanterare som är mycket mindre än den som J2SE använder. J2ME kallar sin databashanterare för RMS (Record Management System).

RMS innehåller endast en klass RecordStore som är plattformsberoende, för att man ska kunna spara den på terminalen. För att skapa en databas använder man metoden openRecordStore() i klassen RecordStore, se Figur 5-1.

Skapa en databas:

```
RecordStore db;
try {
    db = RecordStore.openRecordStore("minDatabas", true);
}catch(...){}
```

Figur 5-1: Kodexempel på hur man skapar en databas

RMS försäkrar att accesserna blir atomära om man bara har en tråd. RMS försäkrar även att inga dubbletter förekommer. Om man däremot har flera trådar som gör accesser mot databasen måste programmeraren se till att synkronisera trådarna så de inte försöker förändra databasen samtidigt.

6 Utredning

I kapitlet utreder vi de frågor som ställs i problemformuleringen (se kapitel 3.2). Inledningsvis diskuterar vi vilka nya möjligheter Wireless Java-tekniken kan medföra för utvecklingen av MyDOF-portalen. Slutligen följer en beskrivning av vad som kan vara begränsande med Wireless Java-tekniken.

6.1 Nya möjligheter med Wireless Java

Telia Mobile är intresserad av att se vilka nya möjligheter Wireless Java-tekniken kan leda till för att förbättra MyDOF-portalen. I detta kapitel kommer vi att ta upp några av de möjligheter som Wireless Java-tekniken ger. Inledningsvis redogör vi för hur användargränssnitten kan förbättras. Sedan redogör vi för om man kan koppla upp klienten direkt mot servern genom att använda sig av en http-förbindelse. Därefter följer en redogörelse för om man kan flytta logik från server till klient. Slutligen kommer en sammanfattning av de tjänster som Wireless Java-tekniken kan ge upphov till.

6.1.1 Användargränssnitt

I MIDP-profilen (se kapitel 5.3.3) finns ett flertal klasser specificerade som tillhandahåller grafiska komponenter. Det finns 2 typer av API:er som man kan använda, högnivå och lågninvå (se kapitel **Fel! Hittar inte referenskälla.**). I detta kapitel kommer vi att ta upp vad de båda API:erna kan göra för att förenkla och förbättra användargränssnitten på MyDOF-portalen.

6.1.1.1 Högnivå

För att skapa ett bättre menysystem som underlättar för nya användare behöver man ett mer enhetligt system. Menyerna ska se likadana ut i alla nivåer för att de ska bli lätt att använda. Man ska lätt kunna koppla ner från MyDOF och inte behöva gå tillbaka flera steg. Avsluta bör alltså ligga i en meny som alltid finns tillgänglig. En direktknapp som gör att man kommer tillbaka till överliggande meny ska alltid visas i fönstret. Detta kan man implementera genom att använda klassen Command. Skapar man ett Command-objekt så kan man sätta prioritet på objektet (se Figur 6-1). Har man bara två Command-objekt läggs de på

varsin sida, har man däremot fler läggs den som har högst prioritet själv, medan de andra läggs i en undermeny se Figur 6-2.

```
public Meny() {
    Form form = new Form("Meny exempel");
    Command exit = new Command("Avsluta", Command.STOP, 2);
    Command back = new Command("Tillbaka", Command.BACK, 0);
    Command choice = new Command("Val", Command.SCREEN, 2);

    form.addCommand(exit);
    form.addCommand(back);
    form.addCommand(choice);
}
```

Figur 6-1: Kodexempel på hur man skapar en meny

I Figur 6-1 skapas tre Command-objekt, ”Avsluta”, ”Tillbaka” och ”Val”. Den första inparametern till Command-objektet är namnet på knappen, det andra är en fördefinierad typ och det tredje är den prioritet som ett objekt ska få. Tillbaka-knappen skapas med en högre prioritet än de andra två. Alltså kommer tillbakaknappen att ligga till vänster, se Figur 6-2. I den vänstra bilden syns inte avsluta- och valknappen utan enbart en menuknapp. Trycker man på menuknappen visas de dolda knapparna som en lista, se den högra bilden. Visas den högra bilden kan man trycka siffran till vänster och alternativet som motsvarar siffran blir valt. Om man exempelvis vill avsluta kan man trycka 1. Detta kan vara en fördel om man har många menyalternativ.

Figur 6-2: Exempel på menyer

Telia Mobile samarbetar med Föreningssparbanken så att man via en MyDOF–tjänst kan göra olika banktransaktioner. Där kan man kontrollera sitt saldo, göra överföringar mellan konton samt ta reda på den senaste transaktionen. Dessa tjänster har Föreningssparbanken utvecklat och har ansvaret för själva. För att få ett enhetligt system vore det önskvärt att även

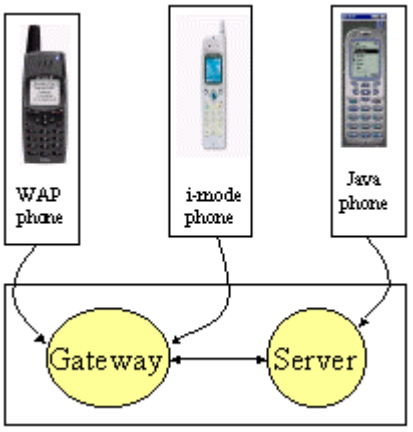
Föreningssparbanken anpassar sina tjänster till Wireless Java. När man loggar in till banken skriver man in personnummer och en personlig kod. Ett inloggningsformulär skulle kunna implementeras genom att använda klassen TextField. När man skapar ett TextField-objekt bestämmer man vilka inparametrar inmatningsfältet ska tillåta som t ex personnummer. Inloggningsformuläret kommer då att bestå av två textfältobjekt, ett för personnummer och ett för lösenord. Detta skulle sedan kunna laddas ner till telefonen för de som är kunder både hos Föreningssparbanken och Telia Mobile.

6.1.1.2 Lågnivå

För att kunna utveckla bättre och mer användarvänliga speltjänster behöver man använda lågnivå API:et. Detta på grund av att man inte har någon kontroll över hur olika komponenter visas på skärmen när man använder högnivå API. Fördelen med lågnivå API är att man kan skapa egna gränssnittskomponenter som man kan kombinera ihop till ett mera avancerat gränssnitt. Man kan rita egna bilder direkt i en Canvas eller importera bilder som finns sparade på telefonen. Genom att spara bilderna i png-format så minimeras minnesåtgången i telefonerna. För att interagera med gränssnittet använder man actionknapparna (se kapitel 5.3.6.2). När man använder actionknapparna behöver man inte tänka på vilken telefonmodell som programmet exekveras på, då alla telefoner har dessa actionknappar i någon form.

6.1.2 MyDOF utan gateway?

MyDOF-portalen idag använder sig av WAP/WML för att utveckla och tillhandahålla tjänster till sina kunder. För att kunna använda tjänsterna på portalen krävs att man har en mobiltelefon med stöd för WAP. WAP-protokollstacken (se Figur 5-2) visar ett exempel på när en gateway måste anpassa trafiken mellan det trådlösa nätverket och Internet. Eftersom MyDOF-portalen är (se kapitel 5.2.2) en WAP-portal måste trafiken mellan klient och server gå via en gateway (se Figur 6-1).



Figur 6-1: Fördelen med http

Med en telefon som stödjer java kan trafiken gå direkt till servern utan att gå via en gateway (se Figur 6-1). Det innebär att klienten kan kopplas upp direkt mot servern eftersom man kan skicka data över HTTP. I MIDP-profilen finns stöd för HTTP-protokoll (se kapitel 5.3.6.3) vilket medför att alla MIDP-applikationer kan köras på alla typer av MIDP-terminaler oavsett om det är en GSM-telefon med en WAP-stack, en I-mode-telefon eller en Palm. Av dagens mobiltelefoner är det endast Motorola Accompli 008 och Siemens SL45i som har stöd för MIDP-profilen (se kapitel 7.1).

Om MyDOF-portalen innehöll tjänster utvecklade med Wireless Java-tekiken skulle kunder med javatelefoner kunna ladda hem applikationen och spara den på telefonen. Det skulle innebära att användaren direkt från telefonen kan starta en http-förbindelse mot servern utan att gå via portalen. Fördelen för användaren är att man slipper väntetiderna mellan varje länk i WML-gränssnittet (se kapitel 5.1.3) och att man har tjänsten tillgänglig lokalt på telefonen. Gränssnittet kommer att bli mer användarvänligt eftersom applikationen kommer att ligga på klienten (se kapitel 6.1.1). Man kommer dock även att kunna använda WAP på en javatelefon.

6.1.3 Kan man flytta logik från server till klient och vad medför det?

Idag ligger all logik samt alla gränssnitt för tjänsterna på MyDOF-portalens server. För att besvara frågan om det går att flytta logik från server till klient har vi utvecklat ett Yatzyspel. Applikationen består av en klient och en server. Klienten har logiken för spelet samt alla gränssnitt och servern distribuerar resultatet mellan spelarna efter varje spelomgång samt kontrollerar vems tur det är att spela, för mer detaljer (se kapitel 7.2).

Vi kan konstatera att det är fullt möjligt att flytta logik från server till klient efter att i stort sett ha färdigställt Yatzyspelet. När man har gränssnittet på telefonen underlättar det för

användaren eftersom man genomför en spelomgång, max tre slag, varvid resultatet skickas till servern. Användaren kan också välja att köra applikationen lokalt på telefonen om han/hon så önskar. Trafiken mellan klient och server påverkas också positivt eftersom det inte genereras lika mycket trafik som när logiken ligger på servern.

6.1.4 Vilka nya tjänster kan man göra med hjälp av Wireless Java?

De tjänster som finns tillgängliga på MyDOF-portalen idag utgörs till största delen av olika typer av informationstjänster. Med Wireless Java-tekniken skulle man kunna utveckla ett flertal olika tjänster som vore attraktiva för MyDOF-portalens användare.

Man skulle kunna utveckla ett chatprogram, likt ICQ eller MSN-messenger, som användaren kan ladda hem från MyDOF-portalens server mot en rimlig kostnad. På klienten kommer hela applikationens gränssnitt att ligga och dataöverföringen kan ske via en socketuppkoppling (se kapitel 6.2.1) direkt mot servern. Servern kan hålla reda på vilka terminaler som kommunicerar med varandra och vidarebefordra all information mellan dem. Denna tjänst kräver en klient/serverlösning och i stort sett alla tjänster som har detta krav skulle man kunna utveckla med hjälp av Wireless Java-tekniken.

Spel är en annan typ av tjänst som alltid lockar många användare. Inom detta område skulle man kunna utveckla en mängd olika tjänster. Ett exempel på spel är Yatzyspelet som vi har implementerat. Det finns många liknande spel som man skulle kunna implementera på samma sätt, d v s med en klient/serverlösning. Man skulle också kunna utveckla spel som laddas ner till telefonen från portalen och där man sedan kan spela lokalt när man så önskar.

Ytterligare tjänster som man skulle kunna utveckla är olika typer av ordböcker. Man skulle kunna implementera en tjänst där användaren laddar hem en turistordbok med de allra vanligaste fraserna och orden för ett specifikt språk. Via gränssnittet som kommer att ligga på telefonen kan användaren söka de ord eller fraser han vill översätta. Fördelen för användaren är att man kan ladda hem tjänsten till telefonen innan man åker iväg på semester och då alltid ha tillgång till tjänsten eftersom den finns lokalt på telefonen.

Det skulle även kunna finnas andra tjänster på samma tema så som en ordbok för engelska tekniska termer eller en turistguide för ett specifikt resmål. De skulle kunna innehålla de 100 vanligaste fraserna, olika sevärdheter samt viktiga telefonnummer som t ex till sjukhus, ambassaden och taxi.

Dessa tjänster är förslag på tjänster som skulle kunna utvecklas med Wireless Java-tekniken. Om man utvecklar dessa tjänster på MyDOF-portalen idag skulle endast de användare som har en telefon som stödjer Java kunna få tillgång till dem. Detta är inte någon

begränsning eftersom utvecklingen av mobiltelefoner går snabbt framåt. Flera av de nya telefonmodellerna som lanserats men inte har kommit ut på marknaden än har stöd för J2ME. Det innebär att när allt fler kommer att ha mobiltelefoner som stödjer Java, skulle man kunna se ett ökat antal användare av MyDOF-portalen.

6.2 Begränsningar med Wireless Java

Telia Mobile är även intresserad av att se vilka begränsningar det finns med Wireless Java-tekniken för att veta om man trots dessa begränsningar vinner något på att använda Wireless Java-tekniken på MyDOF-portalen. I detta kapitel kommer vi att ta upp några av de begränsningar som finns med Wireless Java-tekniken idag. Inledningsvis redogör vi för nackdelarna som finns med att koppla upp en http-förbindelse, därefter redovisar vi de problem som man kan få om man skapar tunga klienter. Slutligen kommer en beskrivning av Wireless Javas obefintliga nätverkssäkerhet.

6.2.1 Http-förbindelse

När man kopplar upp en http-förbindelse så är det bara klienten som kan prata med servern. Servern vet inte vilka klienter som finns eller vart de finns, servern kan alltså bara svara på en informationsbegäran från en klient. Det vore önskvärt i många fall att även servern skulle kunna skicka ett meddelande till klienten för att berätta vad som hänt. För att lösa detta har man i Wireless Java gjort det möjligt att sätta upp en förbindelse med hjälp av en socket (se kapitel 5.3.6.3). Skapar man en förbindelse med en socket så får man en öppen kommunikation och servern kan skicka meddelanden till klienterna.

6.2.2 Tunga klienter

I kapitel 6.1.3 berättar vi att det går att lägga logik på klienten och att det underlättar att ha gränssnitten på telefonen istället för som det är nu att man måste hämta varje nytt gränssnitt på servern. Nackdel med att flytta logik till klienten från servern är frågan om hur mycket telefonerna som finns idag kommer att klara av att spara. Får verkligen hela MyDOF-portalens gränssnitt plats i telefonen? Lägger man hela gränssnittet kommer det då att bli väldigt långsamt att använda? Mycket talar för att det blir mycket att spara hela gränssnittet, framförallt för de yngre kunderna som kommer att vilja ladda ner spel till telefonen, dessa ska också sparas och kommer att ta upp minne. Vi tror att det idag kan bli svårt att överföra hela MyDOF-portalens gränssnitt och därtill ett antal spel eller andra applikationer. Utvecklingen av telefoner går dock väldigt fort och de nyare telefonerna får större minneskapacitet vilket

säkert kommer att leda till att telefonerna om ett antal år skulle kunna klara av att ha hela MyDOF-portalens gränssnitt och ett antal spel nerladdat utan att bli segt.

6.2.3 Säkerhet

I kapitel 6.1.1.1 tar vi upp att Telia Mobile samarbetar med bland annat Föreningssparbanken för att göra vissa banktjänster. Skulle man implementera tjänsten i Wireless Java som vi föreslår för att få ett enhetligt användargränssnitt måste man beakta de höga krav på säkerhet som en sådan tjänst kräver. Idag har Wireless Java obefintlig nätverkssäkerhet, eftersom att kryptering och dekryptering tar allt för mycket kraft av telefonerna. I nästa version av CLDC kommer man dock att utöka säkerhetshanteringen, se kapitel 5.3.2.2, vilket minskar detta problem.

7 Implementation

Telia Mobile vill att vi som en del i vårt examensarbete ska undersöka om det är möjligt att lägga en del logik på klienten (telefonen). För att se om det är möjligt valde vi att implementera ett Yatzyspel där man kopplar upp mot MyDOF-portalens server för att spela mot andra personer. För att göra spelet mer attraktivt valde vi att man även ska kunna ladda ner Yatzyspelet till telefonen och spela mot sig själv.

Detta kapitel kommer först att ta upp vilka förutsättningar och krav som finns på implementationen och därefter kommer designen att redogöras för. Slutligen kommer en beskrivning av implementeringen och de tester som gjorts.

7.1 Förutsättningar och krav

En MIDP-applikation ställer visa krav på telefonerna för att fungera tillfredställande. Dessa krav består av ett minneskrav på 160 – 512 KB minne samt en minsta skärmstorlek på 96x54 pixlar. De telefoner som kan användas för att köra Yatzyspelet är Motorola Accompli 008 och Siemens SL 45i. Dessa två mobiltelefoner är de enda som stödjer J2ME av de som finns tillgängliga på marknaden idag. För att jämföra de båda telefonerna har vi granskat den tekniska specifikationen för respektive telefon. Tabell 7-1 och Tabell 7-2 visar ett utdrag från dessa specifikationer [13] och [14]. Därifrån kan man utläsa att båda telefonerna stödjer MIDP och CLDC samt har stöd för HTTP- och socketförbindelser. Vissa jämförelser mellan telefonerna går inte att göra eftersom Siemens tekniska specifikation inte innehåller uppgifter om minnesutrymme, om JAD-filer krävs eller vilket bildformat som stöds.

J2ME Version Compability	CLDC 1.0, MIDP 1.0
Volatile Memory at Java Runtime	N/A
Memory	N/A
Maximum Midlet Suite Size	N/A
Midlet Download Methods	N/A
JAD File Required	N/A
Device Screen Size	101 x 80
Supported Connection Interfaces	HTTP and Socket
Number of Colors	Black and white
Image Formats Supported	N/A
Input Methods	N/A

Tabell 7-1: Utdrag ur teknisk specifikation för Siemens SL 45i

J2ME Version Compability	CLDC 1.0, MIDP 1.0
Volatile Memory at Java Runtime	662,114 bytes
Memory	For European Releases, approx 1.6 MB
Maximum Midlet Suite Size	N/A
Midlet Download Methods	Serial, Infrared, and OTA
JAD File Required	For serial and IR download, no. OTA download uses JAD file.
Device Screen Size	240 x 320
Supported Connection Interfaces	HTTP and Socket
Number of Colors	4 level grayscale
Image Formats Supported	PNG
Input Methods	Touchscreen and 5 keys returning J2ME keycodes

Tabell 7-2: Utdrag ur J2ME specifikation för Motorola Accompli 008

Det som skiljer de båda telefonerna åt är i första hand skärmstorleken. Motorolas bildskärm är dubbelt så bred och fyra gånger så hög i förhållande till Siemenstelefonens bildskärm. Dessutom har Motorola 4 olika gråskalor medan Siemens endast har färgerna svart och vit.

Utifrån uppgifterna i Tabell 7-1 och Tabell 7-2 har vi gjort vår jämförelse mellan de båda telefonerna och man kan konstatera att båda uppfyller minimikraven för att en MIDP-applikation ska kunna köras på dem. Således kan vårt Yatzyspel fungera på både Motorola Accompli 008 och Siemens SL 45i men med vissa begränsningar beträffande Siemens-telefonen. Orsaken är att gränssnittet som visar spelbordet med tärningarna, se Figur 7-2, kräver att det finns gråskalor för att kunna se om en tärning är markerad.

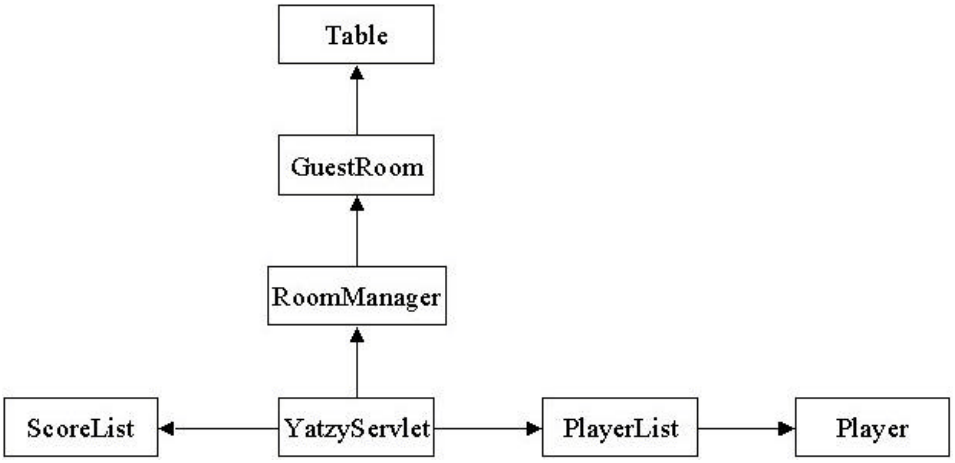
7.2 Beskrivning av konstruktionslösningen

Eftersom Telia Mobile vill se om det går att flytta logik från servern till klienten så valde vi att designa vårt Yatzyspel så att man ska kunna spela två till fyra spelare i varje spel, precis som det är i riktiga Yatzy..

För att kunna spela flera spelare mot samma Yatzyspel krävs att det finns en server som håller reda på vilka spelare som är med och spelar, och vems tur det är att slå. Servern ska också hålla reda på hur många poäng alla spelare har. Servern är implementerad som en servlet i Java. Klienten är implementerad i Wireless Java. Här nedan följer vilka klasser som finns på servern respektive klienten.

7.2.1 Design av servern

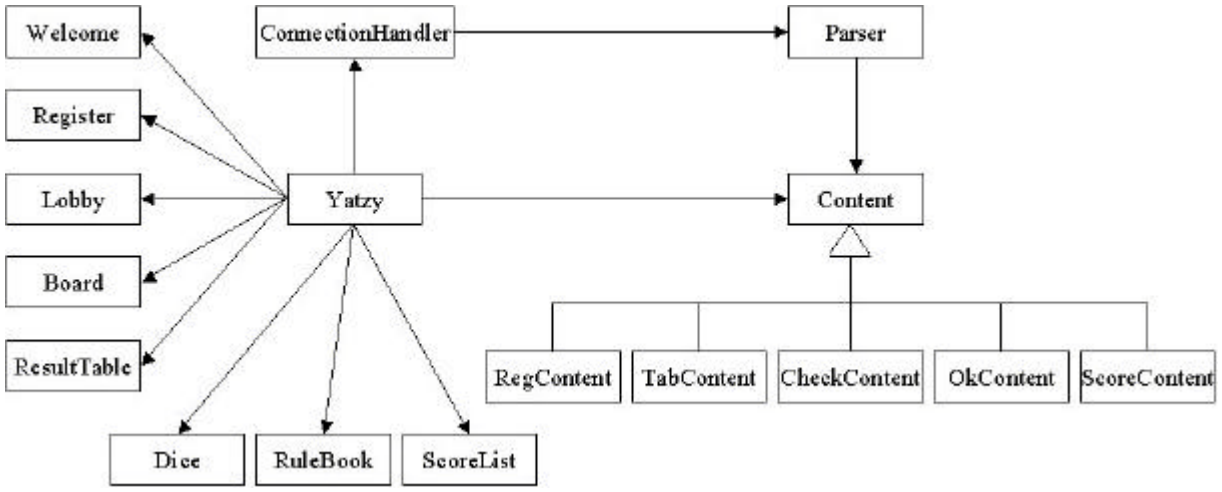
Servern består av sju klasser och deras samband visas i Figur 7-1. Det är serverns uppgift att besvara frågor från klienterna om hur resultatlistan ser ut och vems tur det är att spela. Servern kommer att ha en motor (**YatzyServlet**) som kommer att ha hand om kommunikationen med klienterna och veta vems tur det är att slå och registrera resultat (**ScoreList**). För att motorn ska kunna veta vems tur det är att slå måste det finnas en klass (**PlayerList**) som håller reda på vilka spelare (**Player**) som är uppkopplade mot servern. Klassen Player innehåller spelarens namn, id och vilket bord spelaren spelar vid. Klassen (**RoomManager**) vet vilka olika rum som finns att gå in i, just nu finns det bara ett gästrum (**GuestRoom**) och vilka bord (**Table**) som är lediga i respektive rum.



Figur 7-1: Klassdiagram för servern

7.2.2 Design av klienten

Klienten består av 17 klasser och deras samband visas i Figur 7-1. Klasserna på klienten sköter allt som har med själva spelet att göra och skickar resultat från en spelomgång till servern. De tar även emot resultat som de andra spelarna registrerat om man spelar online.



Figur 7-1: Klassdiagram för klienten

Det är klass (**Yatzy**) som sköter allt som har med Yatzyspelet att göra. Den slår tärningarna med hjälp av en tärningsklass (**Dice**) och kontrollerar om det man slagit motsvarar det man vill spara med hjälp av regelboken (**RuleBook**). Även på klienten kommer det att finnas en klass som registrera poäng (**ScoreList**). Klienten har ett flertal olika gränssnitt (GUI:n). När man väljer att börja spela får man först upp en välkomstskärm (**Welcome**), vilken visar en meny där man kan välja om man vill spela online, lokalt eller läsa om spelet (Figur 7-2).

Figur 7-2: Välkomstbild för Yatzyspelet

Väljer man att spela online så visas en ny skärmbild upp där man får registrera sitt namn (**Register**), när man har blivit registrerad på servern får man upp vilka bord det finns tillgängliga att spela vid (**Lobby**). Efter att man valt ett bord som man vill spela vid visas spelbordet (**Board**) där man ser tärningarna som slås (se Figur 7-3). **Board** innehåller ett antal olika bilder på tärningarna. Ett exempel på en sådan bild visas i Figur 7-3. Den första sexan

och den sista femman är gråmarkerade och alltså låsta. Eftersom dessa båda tärningar är låsta kommer de inte att slås om vid nästa slag. Den första sexan är även markerad som aktiv vilket syns på den svarta skuggkanten. En aktiv tärning är den tärning som man kan välja att låsa eller låsa upp till nästa slag. Med hjälp av piltangenterna på telefonen kan man förflytta sig mellan tärningarna och den tärning man står på är aktiv. [MB6]

Figur 7-3: Spelaren har slagit tärningarna två gånger

När spelaren är nöjd med sitt resultat väljer han att registrera poäng och får då fram en resultatlista (**ResultTable**). I resultatlistan (se Figur 7-4) ska man registrera vad man vill spara för något, från tärningarna i Figur 7-3 har spelaren valt att registrera två par. I resultatlistan kan man även se vad de andra spelarna valt för alternativ och hur många poäng de har. Vill man inte registrera poäng men ändå se hur man ligger till kan man välja att bara visa resultatlistan.

Figur 7-4: Resultatlistan när det är en spelare som spelar lokalt

Väljer man att spela lokalt istället för online enligt menyn i Figur 7-2 högra bilden så kommer man direkt att få upp spelbordet som visas i Figur 7-3, därefter spelar man precis som när man spelar online fast man slipper vänta medan de andra spelarna slår och resultatlistan innehåller bara ens egna resultat (se Figur 7-4). Man kan i Figur 7-2 också välja att läsa om spelet, då får man läsa hur man ska spela yatzy och vilka snabbkommandon som finns till spelet.

Klientens kommunikation med servern sker i en egen klass (**ConnectionHandler**). **Yatzy** anropar metoder i **ConnectionHandler** som skickar en informationsbegäran till servern som

sedan ställer sig och väntar på att det ska komma tillbaka ett svar. När svaret kommer från servern skickar **ConnectionHandler** strängen till en XML-parser som SUN distribuerat och vi har lagt till i vårt projekt. Klassen XMLParser kräver att man implementerar en klass (**Parser**) som ärver interfacet XMLEventListener. De metoder som måste implementeras är tagStarted(), tagEnded() och plaintextEncountered(). Klassen Parser delar upp strängen och kollar vad det är som kommer från servern, beroende på vad som kommer skapas det olika dataobjekt (**Content**). De olika dataobjekten som kan skapas är **RegContent**, **TabContent**, **CheckContent**, **OkContent** och **ScoreContent**. **Yatzy** hämtar sedan dataobjekten och läser informationen från servern.

7.3 Implementering och test

I detta kapitel kommer vi att ta upp hur vi valde att implementera vårt Yatzyspel för att det ska uppfylla kravet på en klient/serverlösning och att se om det är möjligt att överföra logik från servern till klienten.

Inledningsvis kommer en redogörelse för vad som valdes att läggas på klienten för att få en tung klient och därefter kommer en beskrivning av hur servern implementerades och hur den kommunicerar med klienterna. Sedan följer en beskrivning av de tester som gjordes på programmet under utvecklingen. Slutligen kommer en beskrivning av hur vårt program fungerade i en vanlig telefon jämfört med att köra det i en emulator.

7.3.1 Tung klient

Vi valde att implementera en tung klient som innehåller alla gränssnitt som ska visas på telefonen. Eftersom alla gränssnitt ligger på klienten slipper man onödig trafik för att hämta dessa. Vi har fem gränssnittsklasser som skapar olika gränssnitt beroende på vad som har hänt. Klienten sköter också om allt som har med själva Yatzyspelet att göra. Det är klienten som ser till att tärningarna slås och kontrollerar att det man har slagit motsvarar det man vill spara. Därefter skickar klienten resultatet av en spelomgång till servern som lägger in det i sin resultatlista (**ScoreList**), så att de andra klienterna kan hämta resultatet. Alla klienter kommer alltså ha en egen resultatlista (**ScoreList**) där man sparar resultatet som kommer från servern.

7.3.2 Klient/server lösning

Servern är implementerad i Java som en servlet. En servletmotor skapar endast en instans av en servlet, vilket betyder att alla klienter kommer att köra mot samma server. I Wireless Java kan man kommunicera med en server på två sätt. Det ena är via socket och det andra med en

http-förbindelse. Vi valde att implementera vårt program genom att sätta upp en http-förbindelse. Nackdelen med detta är att det inte finns någon öppen kommunikation mellan servern och klienten. Servern kan inte av sig själv tala om för klienten att det har hänt något. Alltså måste klienten ligga och fråga servern efter uppdateringar. Det är svårt att vet hur lång väntetid man ska ha mellan informationsbegäran till servern. Har man för lång tid mellan kan det bli ett väldigt långsamt program och det kan dröja innan spelaren får se resultatet som en annan spelare har registrerat. Skickar man däremot för tätt så kan det bli mycket trafik mellan klienten och servern vilket också kan göra att programmet blir långsamt. En annan nackdel med att skicka informationsbegäran för tätt efter varandra är att de kunder som idag har en GPRS-telefon betalar för mängden data som skickas till servern. Skickar man då för tätt kommer det att bli onödigt dyrt för dessa kunder. Det är då inte säkert att de kunder som har GPRS-telefoner kommer att utnyttja dessa tjänster och då har man tappat en kundkrets.

7.3.3 XMLParser

En fråga som vi ställdes inför var hur vi skulle skicka data till och från servern. När man ska skicka data till servern kan man lägga till en textsträng på url:en som servern sedan läser. Vi valde att skicka med vilken typ av tjänst vi vill att servern ska utföra åt oss. Beroende på vad som vi skickat utför servern det som finns specificerat att den ska göra, sen ska servern berätta hur det gått och då är den stora frågan hur servern ska svara. Ett alternativ var att bara skicka över en sträng som klienten känner till strukturen i och som den kan splittra och få fram resultatet. Detta fungerade bra när servern skulle skicka över små mängder data. Problemet uppstod när klienten frågade om det skett några förändringar i resultatlistan och det har skett. Ska man då bara skicka med det som sparades senast eller ska man skicka med hela resultatlistan. Skickar man bara den senaste uppdateringen går det bra att använda en vanlig textsträng som i de övriga fallen, men problemet är om en klient missar en uppdatering. Klienten kommer aldrig att få reda på det resultatet. Vi insåg att vi skulle vara tvungna att skicka hela resultatlistan till klienten. Det är visserligen möjligt att packa den till en textsträng med skiljetexten emellan attributen men en sådan sträng skulle bli väldigt stor och man skulle låsa servern och klienten till att alltid skicka data i samma ordning. Ett bättre sätt vore om man hade en XML-parser som kunde splittra upp textsträng. När vi började vårt examensarbete fanns det ingen tillgänglig XML-parser så vi funderade på att göra en egen, men vi insåg att det skulle bli komplext. [MB7]Under examensarbetes gång utvecklades dock ett antal XML-parsrar som Sun distribuerar på sin hemsida[12]. Det finns nu fyra XML-

parsrar som stödjer MIDP, dessa är av olika modeller och av olika storlekar. Vi valde en SAX-parser som *Al Suttons´ Software* har utvecklat.

SAX-parsern som vi använder heter *ASXMLP 020308* och den innehåller fem klasser och två interface. Vi använder oss av en klass som heter XMLParser som kräver att man skapar en egen klass (**Parser**) som implementerar interfacet XMLEventListener. Interfacet säger att man ska implementera metoderna tagStarted(), tagEnded() och plaintextEncountered(). Parsern fungerar så att den letar efter start- och sluttaggar. Hittas en starttag anropas metoden tagStarted(), hittas en sluttag anropas metoden tagEnded(). Beroende på vilka taggar som man hittat skapas det olika Content-objekt (se kapitel 7.2.2). Metoden plaintextEncountered() tar fram texten som står emellan en starttag och en sluttag och sparar det i Content-objektet.

7.3.4 Tester under utvecklingen

För att utveckla Yatzyspelet har vi använt utvecklingsmiljön Forte som har en emulator att exekvera programmen i. Under utvecklingens gång har vi testat vårt Yatzyspel i emulatorn för att se om det fungerat som vi har tänkt att det skulle göra. När Yatzyspelet inte har fungerat som det skulle har man kunnat skriva ut en vanlig spårutskrift och fått den utskriven i en medföljande logg till varje telefon. När vi skapade två telefoner som spelade mot varandra fick de en varsin logg att skriva i så man var alltså inte tvungen att hålla isär vem som skrev vad, något som annars kan vara ett problem.

På servern var det svårare att skriva spårutskrifter eftersom denna inte har någon egen logg som den kan skriva i. För att lösa detta skapade vi en klass (**Logger**) som öppnade en textfil och hade en metod log() som tar en sträng som inparameter och skriver den till textfilen. När något gick fel på servern, skapade vi ett Logger-objekt som sedan fick anropa metoden log() för att skriva ut vissa spårutskrifter för att kunna kontrollera vart det gick fel på servern. Nackdelen är att alla klienters bearbetningar skrivs i samma fil så det kan vara svårt att veta vilken klient som orsakar felet, men detta löses dock lätt genom att man skriver ut klientens id i alla utskrifter.

7.3.5 Hur fungerar Yatzyspelet i en riktig telefon?

Siemens SL45i och Motorola Accompli 008 är de telefoner vi valde att testa Yatzyspelet på. Det beror framförallt på att det är de telefoner som finns för testning hos Telia Mobile samt att dessa telefoner är de javatelefoner som finns på marknaden idag.

7.3.5.1 Siemens SL 45i

Det går att ladda ner och spela Yatzyspelet till Siemenstelefonen. Till att börja med testade vi om det gick att spela lokalt på telefonen för att se om det fungerade tillfredställande. Yatzyspelet innehåller ett gränssnitt som är en Canvas. Har man en Canvas som är större än fönsterstorleken på telefonen kommer inte hela Canvasen att visas. Detta inträffade när vi körde Yatzyspelet på telefonen. Följden blev att de två nedersta tärningarna endast var synliga till hälften. Koordinaterna för tärningarna är satta efter fönsterstorleken på emulatorn som vi använt i Forte och stämmer därför inte överens med Siemens fönsterstorlek.

Ett annat problem uppstod när vi skulle starta upp spelet första gången. Det gick väldigt långsamt när man skulle välja från menyn vad man ville göra. När valet att spela lokalt var gjort och gränssnittet med tärningarna visades försvann dock detta problem. En fördel med telefonen var när man ville registrera resultat i resultatlistan. När man scrollat ned till sista raden i listan, totalsumma, startade den om från första raden igen. Man behövde alltså inte gå tillbaka uppåt i listan för att komma åt ettorna om man var i slutet på den.

7.3.5.2 Motorola Accompli 008

Tyvärr går det inte att ladda ner vårt Yatzyspel till Motorola Accompli 008, på grund av att utvecklingsmiljön Forte skapar en manifestfil som telefonen inte kan läsa. Detta kom vi fram till efter att ha pratat med de andra på Telia Mobile som jobbar med Wireless Java. Hur man skulle förändra manifestfilen så att den var anpassad till Motorolan var det ingen som visste. Eftersom vår tid för examensarbetet börjar lida mot sitt slut valde vi att inte lägga ner mer tid på att få det att fungera på Motorolan utan nöjde oss med att det fungerade på Siemens SL 45i.

8 Resultat

I detta kapitel kommer vi att beröra de resultat som vi har kommit fram till under vårt examensarbete. Först kommer en redogörelse för hur vi tror att Wireless Java kommer att fungera på MyDOF-portalen. Slutligen tar vi upp de erfarenheter som vi fått genom att arbeta med Wireless Java och de rekommendationer vi kan ge till andra som vill utveckla tjänster i Wireless Java.

8.1 MyDOF med Wireless Java

En av de frågor vi skulle besvara i vår rapport är om Wireless Java är en teknik som skulle kunna användas för att utveckla mer användarvänliga gränssnitt på portalen. Vi har studerat hur MyDOF-portalen fungerar idag och undersökt vilka möjligheter respektive begränsningar som finns med Wireless Java-tekniken.

I kapitel 6.1.1 redogör vi för hur användargränssnitten skulle kunna förbättras för MyDOF-portalens tjänster. En fördel med att implementera användargränssnitten i J2ME kan vara att man som användare via en tyngre klient kan ha ständig tillgång till användargränssnittet. Detta leder till att man kan navigera sig fram till den tjänst man önskar och först där skulle uppkopplingen mot servern ske. På så vis slipper användaren väntetiderna som finns idag när man navigerar sig fram till en önskad tjänst genom att välja de länkar som finns i WML-gränssnittet. Med dagens teknik vore det dock omöjligt att lägga hela MyDOF-portalens gränssnitt på telefonen eftersom portalen består av så många tjänster och minnesutrymmet är begränsat.

Dock är det möjligt att utveckla nya typer av tjänster på portalen, se kapitel 6.1.4, eller implementera några av de redan befintliga tjänsterna med Wireless Java. De som har WAP-telefoner med stöd för J2ME kan då gå in på MyDOF-portalen och ladda ned de tjänster man vill ha i sin telefon. Dessa tjänster skulle även kunna erbjuda mer funktionalitet än dagens tjänster som i huvudsak fungerar som informationstjänster. Exempel på utökad funktionalitet skulle kunna vara olika typer av bearbetning av informationen den befintliga tjänsten tillhandahåller, såsom sortering och sökning.

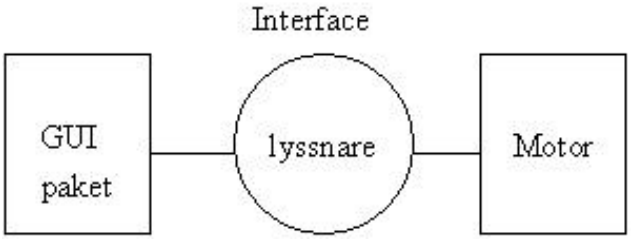
Att införa Wireless Java i MyDOF-portalen är således en fullt möjligt lösning, dock kommer det att kräva åtskillig omstrukturering av MyDOF-portalen vilket måste ställas mot den nytta detta innebär.

8.2 Erfarenheter

Vilka erfarenheter har vi fått genom att göra examensarbete i Wireless Java åt Telia Mobile? De är många och i detta kapitel kommer vi att ta upp ett antal av dessa och hoppas därigenom att andra på Telia Mobile slipper att göra om samma misstag som vi har gjort.

8.2.1 Designförbättringar

Om vi hade börjat skriva vårt examensarbete idag med den kunskap som vi har nu hade designen av Yatzyspelet sett lite annorlunda ut. De ändringar som vi skulle ha gjort på klienten är bland annat att alla gränssnittsklasser skulle ha lagts i ett eget gränssnittspaket, se Figur 8-1. Vi skulle även ha skapat en lyssnarklass istället för att motorn (**Yatzy**) ska ha hand om det.



Figur 8-1: Delar av den nya designen för yatzy

På servern skulle vi ha ändrat så att varje spelbord (**Table**) vet vilka spelare som sitter där, istället för att motorn (**Yatzy**) ska veta vilka spelare som finns registrerade och vilket bord de sitter vid.

8.2.2 Förbättringar av metoden commandAction

I vårt Yatzyspel har vi en CommandListener som ligger och lyssnar på alla våra gränssnitt. Det är klassen **Yatzy** som implementerar interfacet CommandListener. I metoden commandAction() har vi en selektion som tar reda på vilken knapptryckning det är som har skett. Detta görs genom att man tar reda på labeln som står på Command-objektet. Om man har två Command-objekt som har samma label men man inte vill göra samma sak, ställer detta till stora problem. T ex så vill man lätt kunna gå tillbaka till föregående gränssnitt och ett bra namn på ett sådant alternativ är ”tillbaka”, frågan är då hur man ska veta vilket som var det föregående gränssnittet. Ett sätt vore om man satte en variabel som sa vilket gränssnitt som visades innan det nuvarande eller om man istället för att kolla på labeln kan kolla på variabelnamnet för Command-objektet. Då skulle man bara kunna ha olika namn på de olika Command-objekten och på så sätt slippa ha en variabel som kommer ihåg vilket det

föregående gränssnittet var. Om det går att kolla på variabelnamnet vet vi inte riktigt för tiden har inte funnits till att kontrollera detta. Men det går att göra det i Java så det borde även gå att göra det i Wireless Java.

Ett annat problem med commandAction-metoden är att den bara kan ligga och lyssna på ett gränssnitt i taget. När vi i metoden försökte visa ett tillfälligt gränssnitt t ex en skärm som sa ”Väntar på fler spelare” och samtidigt skickade informationsbegäran till servern för att se om det har kommit in någon ny spelare som vill spela vid samma bord uppstod problem. Då commandAction inte är färdig med sin exekvering förrän det har kommit en ny spelare till bordet och man kan börja spela, kan inget nytt gränssnitt visas. När man börjar spela vill man inte längre visa skärmen ”Väntar på fler spelare” utan vill visa en bild på tärningarna som slås. En lösning på detta problem borde vara att skapa en tråd i commandAction som ligger och skickar informationsbegäran till servern för att få reda på om det kommer några fler spelare till bordet. [MB9]

8.2.3 Canvas

I vårt Yatzyspel har vi en klass (**Board**) som ärver ifrån klassen Canvas, som har en metod keyPressed(). Metoden tar hand om knapptryckningar på actionknapparna (se kapitel 5.3.6.2). Så som vi ville implementera vårt Yatzyspel skulle motorn (**Yatzy**) ligga och lyssna på alla gränssnitt som används, men metoden keyPressed() fungerade inte om den inte fick ligga i klassen som ärvde ifrån Canvas. Så vi fick lägga tillbaka keyPressed() i **Board** och den får anropa olika metoder i motorn (**Yatzy**) beroende på vad som har hänt.

8.2.4 Slumpa tärningar

När vi i vårt Yatzyspel skulle slå tärningarna och få fram olika siffror på tärningarna mellan varje slagomgång, är man tvungen att så ett frö till slumpmetoden. Detta går att göra på två olika sätt och vi valde att använda oss av antalet millisekunder som gått sedan 1 januari 1970. I klassen Date finns det en metod getTime() som returnerar detta, men av någon okänd anledning fungerade det inte att använda denna metod för att så ett frö. Vi fick istället använda oss av metoden currentTimeMillis() i klassen System.[MB10]

9 Slutsatser

Efter att ha undersökt om Wireless Java kan vara den alternativa teknik som Telia Mobile söker har vi kommit fram till att Wireless Java kan vara en möjlig lösning för MyDOF-

portalen i framtiden. I dagens läge är det inte ett alternativ eftersom man i så fall måste göra om hela strukturen på portalen. Det finns inget enkelt sätt att endast göra om gränssnittet för tjänsterna som finns idag, utan man är tvungen att göra om hela tjänsten. Eftersom det finns så många tjänster idag är det enligt vad vi har kommit fram till ogenomförbart att förändra alla tjänsterna samtidigt. Man måste ändå ha kvar alla tjänster implementerade i WML, eftersom det bara finns två telefoner idag som stödjer Wireless Java. Detta är ett faktum som medför att kostnaden kan bli större än vinsten om man använder Wireless Java för att implementera de befintliga tjänsterna på MyDOF-portalen.

Referenser

[1] Eric Giguère. Java™ 2 Micro Edition. John Wiley & sons, Inc., 2000.

[2] Qusay H. Mahmoud. *Learning Wireless Java*. O'Reilly & Associates, Inc., 2002.

[3] http://www.teliamobile.se, 2002-02-11

[4] http://www.wapforum.org/what/technical.htm, 2002-02-12

[5] http://www.wapdesign.org.uk/tutorial.html, 2002-02-12

[6] http://developer.phone.com/htmldoc/32w/wmlref/, 2002-02-13

[7] http://tutorials.findtutorials.com/read/category/79/id/69, 2002-02-14

[8] http://www.oasis-open.org/cover/wap-wml.html, 2002-02-14

[9] http://mikrodatorn.idg.se/guider/md9909/wap/, 2002-02-14

[10] http://www.wapforum.org/what/index.htm, 2002-02-13

[11] http://www.atiger.pp.se/dok/waptext.html, 2002-05-06

[12] http://www.wireless.java.sun.com/midp/articles/parsingxml, 2002-04-02

[13] http://developers.motorola.com/developers/wireless/products/matrix/dev_specs.asp?mode=j2me&id=15, 2002-05-07

[14] http://www.my-siemens.com/MySiemens/CDA/Index/0,2730,HQ_en_0_product%253AMW%252FHD%252FHD%252FSL45I%252Ftech,FF.html, 2002-05-07

A Sammanfattning av begrepp och förkortningar

A

AMS – Application Management System

API – Application Programming Interface

AWT – Abstract Window Toolkit

C

CDC – Connected Device Configuration

CLDC – Connected Limited Device Configuration

D

DTD – Document Type Definition

G

GSMAHS – Global Systems for Mobile Communications Abonnent Hanterings System

GUI – Graphical User Interface

H

HTML – Hyper Text Mark-up Language

HTTP – Hyper Text Transfer Protocol

I

IDE – Integrated Development Environment

IP – Internet Protocol

ISDN - Integrated Services Digital Network

J

JAD – Java Application Descriptor

JAM – Java Application Manager

JAR – Java Archive

JCP – Java Community Process

JRE – Java Runtime Environment

JNI – Java Native Interface

JSP – Java Server Page

JVM – Java Virtual Machine

JVMS – Java Virtual Machine Specification

J2EE – Java 2 Enterprise Edition

J2ME – Java 2 Micro Edition

J2SE – Java 2 Standard Edition

K

KVM – Kilo Virtual Machine (tidigare Kuaui Virtual Machine)

M

MIDP – Mobile Information Device Profile

MSISDN – Mobile Station ISDN Number

O

OTA – over-the-air

P

PDA – Personal Digital Assistant

R

RMI – Remote Method Invocation

RMS – Record Management System

T

TCP – Transmission Control Protocol

U

UDP – User Datagram Protocol

URL – Uniform Resource Location

W

WAE – Wireless Application Environment

WAP – Wireless Application Protocol

WIS – Web Integration Server

WML – Wireless Mark-up Language

WSP – Wireless Session Protocol

WTLS – Wireless Transport Layer Security

WTP – Wireless Transaction Protocol

X

XML – Extensible Mark-up Language

Sidan: 1

[MB1] Det kanske räcker om ni hänvisar till nummer och skippar själva texten. Det blir lite svårt att se var hänvisningen slutar och var texten börjar.

Sidan: 1

[MB2] Använd samma skrivsätt för alla profiler. Antingen XX*YY eller XX pixlar.

Sidan: 1

[MB3] Är det skillnad på applikation och program? Lite oklart i detta stycke.

Sidan: 1

[MB4] Är detta viktigt och kommer ni att avända er mer av detta? Annars kan ni ta bort detta.

Sidan: 1

[MB5] Resursfråga(?)

Sidan: 1

[MB6] Lite förvirrad beskrivning. Svårt att veta när, var och hur tärningarna har slagits. Är det t ex inte så att både första sexan och sista femman visar hur en markerad tärning ser ut?

Sidan: 1

[MB7] I princip börjar man aldrig en mening med ”men”, annat än i skönlitteratur och dikter. Kolla gärna igenom uppsatsen efter sånt. ”Och” börjar man heller aldrig med.

Sidan: 1

[MB8] Lite samma text som i introduktionen till kapitlet.

Sidan: 1

[MB9] När då?

Sidan: 1

[MB10] Jag förstår inte

Sidan: 1

[MB11] Skulle ni inte lägga till en förklarande mening?