Datavetenskap

Stefan Rickfjord

Portering av SMS Notifier till Microsoft.NET

Examensarbete

2004:10

Portering av SMS Notifier till Microsoft.NET

Stefan Rickfjord

© 2004 Författarna och Karlstads Universitet

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 mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Stefan Rickfjord

Godkänd, 2004-06-03

Handledare: Robin Staxhammar

Examinator: Tim Heyer

Sammanfattning

Detta examensarbete behandlar porteringen av SMS Notifier från programspråket Java till C#. SMS Notifier är en applikation, utvecklad av Devo IT i Karlstad, för att sända sms från en pc till en mobiltelefon. Smsmeddelandet sänds via en operatör som t.ex. Vodafone.

Målet med examensarbetet var att portera SMS Notifiers kärna. Kärnan är den modul i SMS Notifier som handhar kärnfunktionerna som t.ex. att sända sms. Två andra viktiga mål var även att ta fram en kravspecifikation för den färdiga SMS Notifer samt en dokumentation av den ursprungliga Javaversionen.

SMS Notifier skulle porteras p.g.a. Devo ITs ändrade behov av SMS Notifier. Detta arbete påbörjades manuellt genom att "översätta" Javakoden till C#. Efter att den manuella porteringen pågått i ca sju veckor hittades ett verktyg som porterade nästan all kod automatiskt. Verktyget heter Microsoft Java Language Conversion Assistant. Tre implementationsdetaljer porterades dock inte av verktyget. Den första av detaljerna som inte porterades berörde Javas händelsehantering, den andra gällde tid/datum-formatering och den tredje och sista var en hanterare av XML-dokument. Dessa tre detaljer implementerades istället manuellt.

Vid slutet av examensarbetet hade de tre målen uppnåtts. Den porterade C#-koden kompilerades utan fel, kravspecifikationen var färdigställd och Javaversionen av SMS Notifier var dokumenterad.

Porting the SMS Notifier to Microsoft.NET

This thesis covers the porting of SMS Notifier from the programminglanguage Java to C#. SMS Notifier is an application, developed by Devo IT in Karlstad, for sending sms messages from an ordinary pc to a cellular phone. The sms message is sent via an operator such as Vodafone.

The main goal of this bachelors project was to port the kernel of SMS Notifier. The kernel is the module in SMS Notifier that handles the basic functionality, i.e sending a sms. Two other essential goals were to create a specification of demands for the final version of SMS Notifier and documentation for the original Javaversion.

SMS Notifier was to be ported because of the changed needs that Devo IT had on SMS Notifier. After manually porting the code for seven weeks, a tool was found that automatically ports Java sourcecode to C#. The tool is Microsoft Java Language Conversion Assistant and it ported almost all of the code automatically. Three implementation-issues that couldn't be ported arose. The first one concerned the event handling of Java, the second issue concerned the formatting of time/date strings and the third was a reader of XML-documents. All of these had to be implemented manually.

By the end of this project, the three goals were accomplished. The ported C#-code was compiled without any errors, the specification of demands was complete and the Java version of SMS Notifier was documented.

Tack

Jag vill rikta ett stort tack till personalen på Devo IT i Karlstad för att de givit mig möjligheten att genomföra detta examensarbete.

Innehåll

1 Inledning **1**

1.1 Bakgrund 1

1.1.1 Teknisk beskrivning 1

1.2 Mål 2

2 Beskrivning av SMS Notifier **5**

2.1 Att logga in 5

2.2 Väl inloggad 6

2.3 Brister i dagens version 7

3 Kravspecifikation **9**

3.1 Sändning av sms 9

3.2 Mottagning av sms 10

3.3 Lägga upp nummerregister 11

3.4 Lägga upp sändlistor 11

3.5 Logga skickade och mottagna sms 12

3.6 Säkerhet 12

3.7 Debiteringsinfo 13

3.8 Administratörsfunktion 13

3.9 Integration 13

4 Kravanalys för porteringen **15**

4.1 Implementation av protokollet CIMD-2 15

4.2 Implementation av kärnan 15

4.2.1 Sändenheten - “Dispatcher”-modulen 15

4.2.2 Mottagarenheten - Incoming handler 16

5 Beskrivning av konstruktionsuppbyggnad 17
5.1 Customer Applications 17
5.2 API 18
5.3 Kernel 18
5.3.1 Dispatcher 18
5.3.2 Incoming handler 18
5.3.3 Persistent message que 18
5.3.4 Billing 19
5.4 SMS-C protocols 19
5.5 Management 19

6 Nuvarande implementation av SMS Notifier 21
6.1 Strukturen i SMS Notifier 21
6.2 Databashanteringsklasserna i db 24
6.3 Smsprotokoll för Vodafone - CIMD-2 25
6.4 Sändmodulen Dispatcher 30
6.4.1 SmsNotDispatcherApi 34
6.4.2 SmsNotDispatcherApiSendDispatcher 36
6.4.3 SmsClient 37
6.4.4 SmsNotDispatcherApiSmsStatusReportQue 38
6.4.5 SmsNotDispatcherApiSendQue 39
6.4.6 SmsNotDispatcherApiStatisticMessage 40
6.4.7 SmsNotDispatcherApiStatisticWriter 41
6.4.8 ServerStatus 41
6.4.9 Namnrymden que 42
6.4.10 Namnrymden supervisory 44
6.4.11 Namnrymden thread 48
6.5 Hjälpklasserna i util 51

7 Porteringsprocessen 55
7.1 Strukturen hos den porterade SMS Notifier . . . 57
7.1.1 Kort om modularisering . . . 57
7.1.2 Huvudnamnrymden *smsnotifier* . . . 58
7.1.3 Protokollnamnrymden *protocols.cimd2* . . . 60
7.1.4 Dispatchernamnrymden *smsnotdispatcher* . . . 62
7.2 Förslag till ytterligare ändringar . . . 64

8 Manuell implementation vid porteringen 65
8.1 PropertyChange . . . 65
8.2 SimpleDateFormat . . . 69
8.3 jdom . . . 70

9 Resultat och rekommendationer 73
9.1 Huvudresultat . . . 73
9.1.1 Kravspecifikation . . . 73
9.1.2 Dokumentation . . . 73
9.1.3 Porteringen . . . 73
9.1.4 Webgränssnitt för test . . . 74
9.2 Rekommendationer . . . 74
9.2.1 Förstå produkten . . . 74
9.2.2 Undersökning . . . 74

10 Summering av projektet 75

Referenser 77

A Delmål till den färdiga SMS Notifier 79

Figurer

5.1 Moduluppbyggnaden av nya SMS Notifier. 17
6.1 Namnrymdsstrukturen hos SMS Notifier. 21
6.2 De olika dbklasserna. 24
6.3 Klassberoenden för CIMD2-protokollet. 26
6.4 Klasserna som utgör Dispatchermodulen. 30
6.5 Klassberoenden för dispatchermodulen. 31
6.6 Klasserna som utgör incoming handler-modulen. 32
6.7 Klassberoenden för smsnotdispatcherapi. 35
6.8 Klassberoenden för smsnotdispatcherapisenddispatcher. 36
6.9 Klassberoenden för smsclient. 37
6.10 Klassberoenden för smsnotdispatcherapismsstatusreportque. 38
6.11 Klassberoenden för smsnotdispatcherapisendque. 39
6.12 Klassberoenden för smsnotdispatcherapistatisticmessage. 40
6.13 Klassberoenden för smsnotdispatcherapistatisticwriter. 41
6.14 Klassberoenden för serverstatus. 41
6.15 Klassberoenden för namnrymden que. 42
6.16 Klassberoenden för queklassen. 42
6.17 Klassberoenden för compundqueobjectklassen. 43
6.18 Klassberoenden för namnrymden supervisory. 44
6.19 Klassberoenden för clientklassen. 45
6.20 Klassberoenden för supervisoryclientklassen. 46
6.21 Klassberoenden för supervisorycommandklassen. 47
6.22 Klassberoenden för namnrymden thread. 48
6.23 Klassberoenden för threadparametersklassen. 49
6.24 Klassberoenden för timestampedthreadklassen. 50
6.25 De olika hjälpklasserna. 51

7.1 Namnrymdsstrukturen hos den porterade SMS Notifier. 58
8.1 En händelseavsändare och lyssnare. 69

1 Inledning

1.1 Bakgrund

SMS Notifier är en produkt utvecklad av Devo IT för massutskick av SMS via datorn. På grund av den höga andelen mobiltelefoner idag som stöder SMS, är detta en mycket enkel, snabb och effektiv meddelandetjänst. Genom att använda datorn för att skicka SMS ökas effektiviteten Händelsehanteringen implementerades m.h.a. klassen EventArgs i C#. Tid/datumformateringen implementerades med klasserna CultureInfo och DateTimeFormat. För XML-dokumenthanteringen användes i C# den inbyggda hanteraren XmlValidatingReader. . Det går inte bara snabbare att skriva ett SMS. Den största fördelen får de som använder SMS mycket t.ex. för att delge en kundgrupp viktig information snabbt och enkelt. Detta är styrkan hos SMS Notifier. Genom att skriva det meddelande som ska skickas och enkelt välja mottagare ur sin adressbok går det snabbt och enkelt att sprida informationen. Exempel på lämpade informationsflöden via detta medium är t.ex. bokningsverifikationer för hotellrum, reklammeddelanden till utvalda kunder eller helt enkelt en julhälsning till kunderna vilket ger en något mer personlig prägel än ett epostmeddelande.

1.1.1 Teknisk beskrivning

Dagens version är skriven i Java och använder serialiserade filer för lagring av objekt. Objekten innehåller bl.a. telefonnummer och sms. Sms där sändningen lyckats eller misslyckats sparas i en databas. SMS Notifier kan användas både via en webbserver och som ett fristående program men kan även integreras med andra produkter m.h.a. ett API mot kärnan. Kärnan kallas den del av SMS Notifier som sänder och tar emot sms. Den består av tre delar, en del som tar emot sms från användaren och sparar säkerhetskopior av dem i databasen. Den andra delen sänder de meddelanden som användaren sänt samt den tredje delen som lyssnar efter eventuella meddelanden *till* SMS Notifier. Kärnan utgör alltså

SMS Notifiers *kärnfunktionalitet*. Beroende på önskad operatör hanterar SMS Notifier olika protokoll för SMS-tjänster. Idag finns stöd för bla Nokias SMS-protokoll CIMD-2, för t.ex. Vodafone, och CMGs UCP50-protokoll för t.ex. Telia. Uppkoppling mot SMS Notifier sker via nätverksprotokollet TCP/IP.

På grund av ändrade behov skall applikationen implementeras i Microsofts utvecklingsmiljö .Net. Programspråket som ska användas är det objektorienterade C# (C-Sharp). På den färdiga produkten SMS Notifier ställs inte bara krav från kunder utan även från Devo ITs sida. Detta innebär att förutom porteringen måste även en kravspecifikation tas fram för att tillgodose dessa krav.

1.2 Mål

Det primära målet med arbetet är att att portera applikationens kärna. Ett annat viktigt mål är att sätta samman en kravspecifikation för den slutliga produkten SMS Notifier. Den porterade kärnan skall kunna skicka och ta emot sms. SMS Notifier använder idag sms-protokollet CIMD-2 för att skapa en uppkoppling mot telefonoperatören Vodafone och sända sms. Även i den porterade versionen kommer operatören Vodafone att användas för detta ändamål. Även protokollet kommer fortsättningsvis användas då det är det enda protokoll som operatören Vodafone stöder. Följande är en lista över de delmål som måste nås för den porterade versionen[1].

1. **Sammanställa en kravspecifikation.**

   Kravspecifikationen ska beskriva vad den slutliga produkten skall klara av och vilka funktioner som ska finnas.

2. **Dokumentera dagens SMS Notifier.**

   Att dokumentera SMS Notifier blir en del av uppgiften då det vid examensarbetets början ej finns någon dokumentation tillgänglig. Detta är nödvändigt för att

[1] Se appendix A för de kompletterande delmål som måste nås för den färdiga produkten

underlätta porteringen men även för att förbättringar lättare skall kunna göras. En dokumentation önskas även från Devo ITs sida då det, som tidigare nämnts, ej finns någon att tillgå.

3. **Portera befintlig Javakod till C#.**

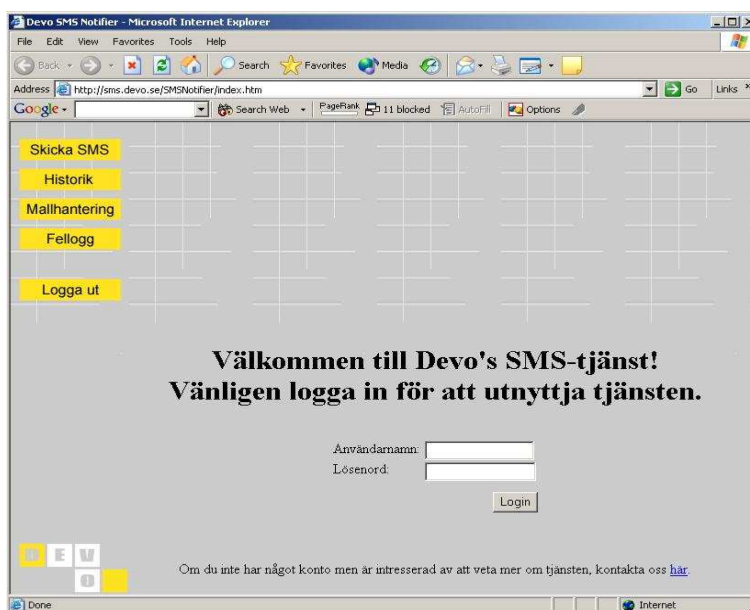
   Javakoden skall porteras till C#. Den porterade koden skall vara funktionell i den mån att kärnan skall kunna sända samt ta emot smsmeddelanden. För detta skall operatören Vodafone användas.

4. **Ett enkelt webgränssnitt för att testa kärnan.**

   Detta webgränssnitt ska vara enkelt där ett telefonnummmer och ett meddelande kan skrivas innan detta sänds. Gränssnittet kommer endast att användas för testning av den porterade koden, d.v.s. efter att delmål 3 slutförts.

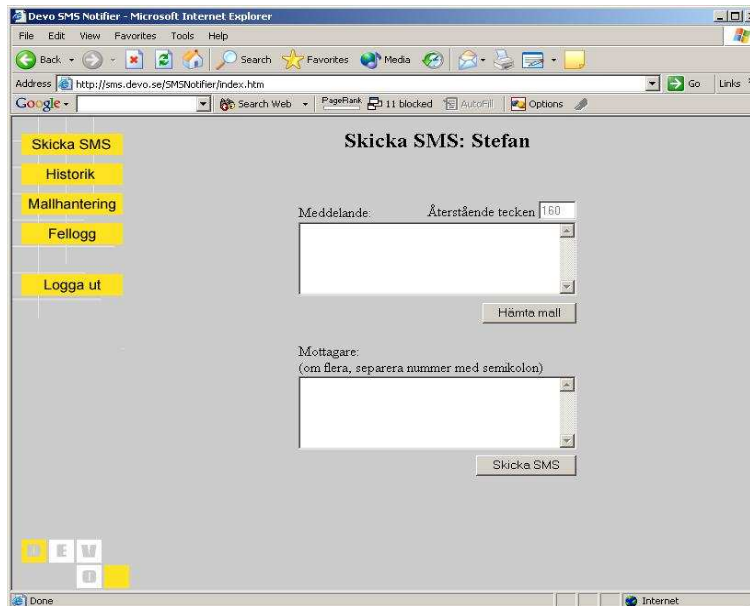
2 Beskrivning av SMS Notifier

Dagens SMS Notifier är, som tidigare nämnt, skriven i Java. Applikationen består av en motor(kärnan) och ett webgränssnitt för att utföra de tjänster som kärnan tillhandahåller. Här nedan följer en kort beskrivning av användargränssnittet i dagens SMS Notifier. Detta kapitel skall ge läsaren en översiktlig bild över hur dagens SMS Notifier i stora drag fungerar, ser ut och även påpeka dess brister.



2.1 Att logga in

Det första användaren möts av är en beskrivande skärm där denne ombeds att ange ett användarnamn och ett lösenord för att få tillgång till SMS Notifier. Fem knappar finns till vänster, men dessa är ej funktionella så länge användaren inte är inloggad med godkända uppgifter. Om det användarnamn och lösenord som anges är ogiltiga fortsätter SMS Notifier att be om dessa genom att ladda om inloggningsskärmen med tomma fält. Om de uppgifter användaren anger är giltiga får användaren börja använda applikationen.



2.2 Väl inloggad

Nu välkomnas användaren av två tomma rutor för meddelandet och de telefonnummer som meddelandet skall sändas till. Nu är även de fem knapparna till vänster funktionella.

- **Skicka sms:** Visar en sida där användaren kan skriva meddelandet och ange mottagarens(mottagarnas) telefonnummer.
- **Historik:** Här kan användaren undersöka sin historia. Genom att ange två datum visar SMS Notifier alla sms som sändes *mellan* dessa datum, dess text, det mottagande telefonnumret och datumet.
- **Mallhantering:** Denna knapp aktiverar en tjänst där användaren kan skapa och hantera mallar för sms som sänds ofta. Här kan också mallar för telefonnummer lagras som en mycket enkel telefonbok.
- **Fellogg:** Här sparas alla eventuella fel, fel på sändningar etc tillsammans med datum.
- **Logga ut:** Användaren loggas ut.

2.3 Brister i dagens version

Här beskrivs kort de brister som finns i dagens version av SMS Notifier.

- En “riktig” telefonbok saknas. Den lilla funktionalitet mallhanteraren bistår med är att telefonnummer kan skrivas in. Detta sker dock utan att på något sätt knytas an till mottagarens/mottagarnas namn. Telefonnumren blir snabbt oöverskådliga utan något sätt att skilja dem åt.
- Det finns ingen möjlighet att gruppera mottagare till sändlistor.
- En fungerande metod för att ta emot sms, som t.ex. leveransrapporter eller svar saknas.
- Användaren har ytterst begränsade inställningsmöjligheter. T.ex. kan inte användaren ange hur många sms som får skickas under en tidsperiod eller bestämma vad som ska loggas i databasen.
- Dagens SMS Notifier är inte integrerad med applikationer som Microsoft Outlook eller Lotus Notes. Detta ger inte den lättillgänglighet en kund kan önska.

3 Kravspecifikation

I detta kapitel kommer kravspecifikationen för den färdiga SMS Notifier presenteras. Detta specificerar vad den slutliga produkten skall prestera och skall ge läsaren en inblick i vad SMS Notifier kan användas till. Specifikationen har tagits fram tillsammans med personal på Devo IT genom möten, analyser av dagens version och önskemål från kunder. För att uppfylla de delmål som specificerades i avsnitt 1.2 på sidan 2 är det framförallt avsnitten 3.1(*sändning av sms*) och i mån av tid 3.2(*mottagande av sms*) som kommer att behandlas i denna rapport. Kraven i dessa två avsnitt är de som den porterade versionen skall uppfylla.

3.1 Sändning av sms

Då en användare skall sända ett sms med SMS Notifier, kommer ett antal funktioner finnas att tillgå.

- Ett sms skall kunna skickas till en eller flera mottagare, d.v.s ett eller flera telefonnummer kan anges i följd som mottagare av ett sms.
- I SMS Notifier skall man kunna lagra de personer man ofta skickar sms till som kontakter för att sedan ange dessa som mottagare.
- Även möjligheten att gruppera kontakter till en sändlista skall finnas. Detta kan sedan utnyttjas genom att enkelt ange sändlistans namn som mottagare.
- Sms-kroppen, d.v.s själva meddelandetexten skall max vara 480 tecken. Ett normalt sms innehåller max 160 tecken, alltså kommer SMS Notifier att dela upp ett sms i maximalt tre sms och skicka dessa ett och ett. Denna gräns är endast satt av praktiska skäl, behöver man mer utrymme kan ett samtal eller ett epostmeddelande vara lämpligare.
- Om avsändaren önskar svar av mottagaren skickas detta önskemål tillsammans med den maximala tid som användaren kan vänta till SMS Notifier. Kärnan skall då lyssna

efter ett svar på det skickade meddelandet under den tid som användaren angivit.

- Om avsändaren vill ha svaret till en plats annan än den som angivits i inställningarna som lagrats i databasen, skall detta kunna anges till SMS Notifier då användaren skickar sitt sms. Mottagarplatsen kan då förutom databasen vara en mapp med XML-filer, en mobiltelefon eller en epostadress.

- Då avsändaren skickar ett meddelande skall kvittens på meddelandet kunna begäras. Om avsändaren begärt detta begär SMS Notifier leveransrapporter från operatören som skickas vidare till avsändarens angivna svarsplats.

3.2 Mottagning av sms

Om användaren önskar ett svar på ett sänt meddelande anges detta vid sändning av meddelandet. I detta fall skall applikationen lyssna efter ett svar på just detta meddelande.

- Identifiering av inkommande sms krävs om flera sms skickas till samma telefonnummer under en kort tidsperiod. Detta är viktigt om användaren sänt flera meddelanden till mottagaren och i ett eller flera av dessa begärt ett svar.

- SMS Notifier skall lagra meddelandet på den av användaren önskade platsen, meddelandet kan alltså skickas vidare till ett specificerat telefonnummer, sändas via epost till användaren eller loggas som XML-fil för avläsning av en extern produkt.

- Det mottagna meddelandet skall kunna presenteras för användaren.

- Om användaren anger en tidsperiod som han eller hon är villig att vänta på ett svar gäller följande:

    1. SMS Notifier skall vänta på ett svar så länge användaren angivit, om inget svar kommer lagras ett meddelande om detta i databasen.

2. Om svaret inkommer efter tidsperiodens slut lagras detta i databasen tillsammans med meddelandet.
3. Om svar inkommer sparas svarsmeddelandet i databasen tillsammans med meddelandet att svaret anlänt inom tidsramen.
4. Om mottagaren svarar genom att t.ex. ringa kan användaren själv skriva in att svaret har inkommit, vad personen svarade och på så sätt avbryta bevakningen av svaret.

3.3 Lägga upp nummerregister

För att underlätta för användaren skall denne kunna lagra sina kontaktpersoner i ett personligt nummerregister. En kund har förutom sina användares nummerregister även ett gemensamt nummerregister för företaget att tillgå.

- En kunds olika användare skall kunna nyttja både gemensamma och personliga listor. Detta innebär att användaren kan välja mottagarnummer både ur sitt personliga och ur de gemensamma nummerregistren.
- Namn och telefonnummer skall kunna kopplas till varandra i SMS Notifiers databas där telefonnumret utgör en unik nyckel. Även andra unika fakta om en person kan lagras och användas som identifieringsinformation, som t.ex personens e-post adress.
- Administrationsfunktioner som att lägga till och ta bort kontakter i nummerregistret skall också finnas.

3.4 Lägga upp sändlistor

En sändlista är en lista som grupperar en mängd kontakter i nummerregistret.

- Ett eget namn skall kunna sättas på den lista användaren skapat, t.ex. “Nattskiftslaget” eller “Kompisar”.

- För hantering av sändlistor skall användaren erbjudas normala administrationsfunktioner som att lägga till nya sändlistor och kontakter, byta namn, ta bort kontakt ur sändlista eller ta bort en hel lista.

3.5 Logga skickade och mottagna sms

Behovet av att kunna kontrollera meddelandehistorien gör att all information skall lagras av SMS Notifier i databasen, följande lista är den information som skall lagras i databasen:

- Det ursprungliga meddelandet.
- Mottagarens telefonnummer och/eller namn.
- Avsändarens identitet.
- Om avsändaren begärt ett svar. (J/N)
- Mottagarens svarsmeddelande, om sådant inkommer.
- Tidpunkterna då meddelandet sändes och ett eventuellt svar togs emot.

All denna information skall gå att visa för den person som administrerar kundkontot. Detta kan vara t.ex. chefen för företaget eller en tillordnad administratör.

3.6 Säkerhet

SMS Notifier skall ha säkerhetsfunktioner som gör att produkten skall vara säker för en mängd tänkbara scenarion.

- Meddelanden som skickats från och till SMS Notifier skall lagras persistent, d.v.s meddelanden får inte tappas inte bort p.g.a någon typ av fel eller systemkrasch.
- Applikationen skall ha intrångsskydd i form av lösenordsverifikation.

- Olika användare av produkten skall kunna ges specifika tillstånd för tillåtna operationer. Med detta menas t.ex. att endast en grupp anställda får skicka sms med valfri text medan övriga endast får skicka färdigskrivna meddelanden.

3.7 Debiteringsinfo

För att underlätta för kunden skall det finnas funktioner för att kontinuerligt följa upp statistik över kostnader, antalet skickade sms och den aktuella prissättningen.

3.8 Administratörsfunktion

Administratörsfunktionerna som kundens kontoadministratör skall ha tillgång till är ett antal funktioner för att kunna anpassa produkten efter kundens behov.

- Maxgränsen för skickade sms per tidsperiod.
- Ovan nämnda privilegier för anställda, vilken grupp anställda som får skicka vad.
- Undersökning av meddelandehistoria.

3.9 Integration

SMS Notifier skall med sitt API ge bra möjligheter att kunna integreras i andra applikationer eller i t.ex. intranät.

4 Kravanalys för porteringen

I avsnitten 3.1, sidan 9, och 3.2, sidan 10, beskrevs kraven för kärnan i den färdiga versionen av SMS Notifier. Kraven i dessa två avsnitt gäller även den porterade versionen. I detta avsnitt analyseras vilka av dessa krav som måste tillgodoses för att målet med porteringen och målen med detta examensarbete skall uppfyllas. Dessa mål kan ses i avsnitt 1.2 på sidan 2.

4.1 Implementation av protokollet CIMD-2

Kravet för protokollet är att kunna kommunicera med sms-centralen. Detta innebär att protokollet skall sända rätt information till sms-centralen enligt de specifikationer som finns för CIMD-2 protokollet[1]. Dessa specifikationer tas ej upp i denna rapport då det ej ingår i examensarbetet att analysera och förbättra implementationen av detta protokoll utan endast att portera det existerande.

4.2 Implementation av kärnan

4.2.1 Sändenheten - "Dispatcher"-modulen

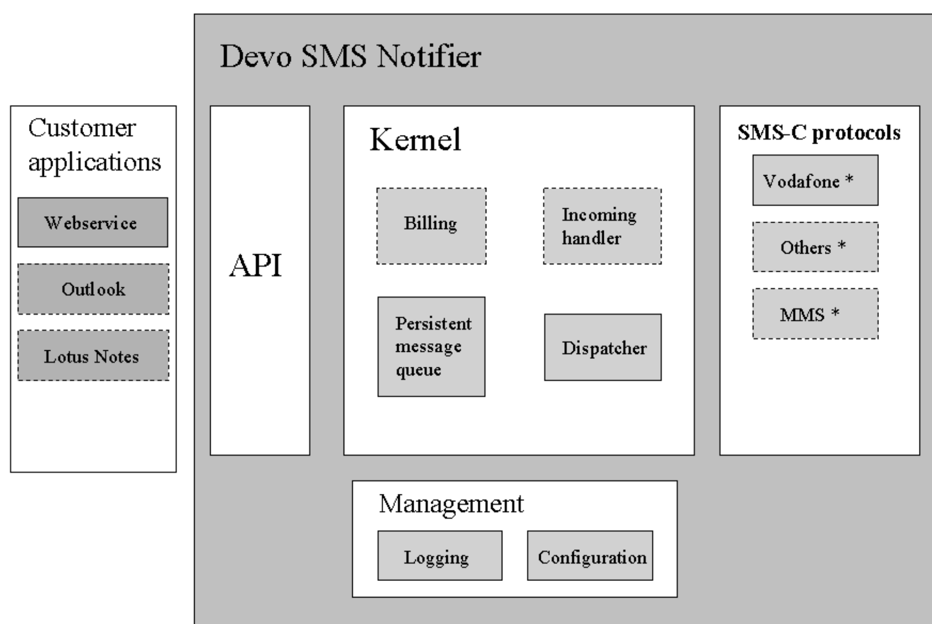
Det första målet som måste måste nås med kärnan i applikationen är att kunna sända sms. För att detta skall kunna uppfyllas måste endast ett krav säkerställas ur avsnitt 3.1; att sända sms. De övriga kraven uppfylls antingen genom att använda sig av sändenheten(t.ex. att skicka ett sms) eller utan inblandning av sändenheten(t.ex. att skapa ett nummerregister). Detta gör att de övriga kraven ej påverkar sändenhetens basfunktion att sända sms. T.ex. så uppfylls första kravet att sända samma meddelande till flera mottagare genom att sändenheten anropas för varje mottagare som anges.

4.2.2 Mottagarenheten - Incoming handler

Det andra målet som måste nås med kärnan är att mottagardelen av kärnan kan ta emot ett sms. För att uppfylla detta mål måste modulen för att ta emot sms fungera korrekt.

5 Beskrivning av konstruktionsuppbyggnad

I detta avsnitt beskrivs CIMD2-protokollet och kärnan närmare. Dessa togs upp i föregående kapitel. Här förklaras hur SMS Notifier skall byggas upp och vad de olika delarna har för uppgifter. Detta för att ge läsaren en bild av delarnas funktioner och eventuella delmoduler som t.ex. kärnans dispatchermodul. Den struktur som dessa olika moduler skall ha har sammanställts i figur 5.1 nedan. Här ser man tydligt hur de olika modulerna är tänkta att förhålla sig till varandra. De moduler som är streckade är de moduler som ej är operativa i dagens version av SMS Notifier. Detta beror antingen på att de ej finns implementerade alternativt att implementationerna ej är slutförda.



Figur 5.1: Moduluppbyggnaden av nya SMS Notifier.

5.1 Customer Applications

Customer applications är tänkt som ett samlingsnamn för de utomstående produkter som använder sig av SMS Notifier. Exempel på dessa är t.ex. Lotus Notes, Microsoft Outlook

eller det webgränssnitt som Devo IT bifogar SMS Notifier.

5.2 API

API(Application Programmers Interface) skall vara den modul som tillhandahåller SMS Notifiers interna funktioner till Customer applications. API är alltså porten för all kommunikation till och från kärnan och managementdelen. Kommunikation till och från protokollmodulen sker via kärnan då det är här funktionerna för att skicka och hämta sms skall finnas.

5.3 Kernel

Kernel är namnet på kärnan, den modul där bearbetningen av mottagna användardata skall ske. Kärnan skall innefatta fyra mindre moduler:

5.3.1 Dispatcher

Dispatchern är den modul i SMS Notifier som hanterar sändningen av själva meddelandet. När dispatchern skapat ett sms utifrån de data användaren angivit skall meddelandet sändas m.h.a ett SMS-C protokoll.

5.3.2 Incoming handler

Detta är den modul som periodiskt skall undersöka om något sms har kommit in *till* SMS Notifier. Inkommande meddelanden kan t.ex. vara leveransrapporter eller svar på tidigare skickade sms. Incoming handler är inte implementerad i dagens version av SMS Notifier.

5.3.3 Persistent message que

Persistent message que skall vara en meddelandekö där alla smsmeddelanden mellanlagras efter att användaren tryckt på “skicka”, men innan dispatchern har sänt dem till smscen-

tralen. Med persistent menas att de meddelanden som lagrats i sändkön lagras permanent i databasen tills de skickats. Endast då de skickats tas de bort ur kön och lagras istället i databasen som ett sänt sms. Detta är en av skillnaderna från den äldre versionen av SMS Notifier där meddelandekön var uppdelad i två delar varav en av dessa var persistent och lagrades på hårddisk medan den andra låg i minnet. Inkommande sms hanteras av en databaslagrad kö i modulen Incoming handler.

5.3.4 Billing

Denna modul skall se till att rätt kund debiteras för varje sms kunden sänder. Om kunden själv installerat produkten i sitt lokala system läggs istället kostnaden för meddelandet till den användare hos kunden som sänt meddelandet. Denna funktion är ej operativ i Javaversionen av SMS Notifier.

5.4 SMS-C protocols

Dessa är de tänkta protokollen för att sända sms. Varje protokoll utgör en modul som innebär att olika operatörer och/eller olika meddelandetyper används för att sända meddelandet.

5.5 Management

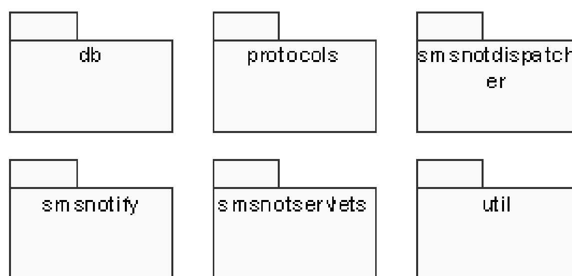
Managementmodulen skall ha två delmoduler, en för loggboken och en för konfiguration. Loggboken skall lagras i en databas där information om händelser som t.ex. skickade sms, inkomna sms och fel i SMS Notifier lagras. Konfigurationsmodulen skall vara en modul där varje användare kan konfigurera sitt konto i SMS Notifier.

6 Nuvarande implementation av SMS Notifier

I detta avsnitt kommer jag att behandla dagens version av SMS Notifier. I kapitlet kommer jag att ge en dokumentationsinriktad funktionell beskrivning av de klasser som ingår i SMS Notifiers namnrymder. Detta dels för att ge läsaren en detaljerad bild av hur SMS Notifier fungerar och är uppbyggd men främst för att tillgodose önskemålen från Devo IT och uppfylla målet att ta fram en dokumentation för produkten. Detta kommer att underlätta förståelsen av nästa kapitel där jag behandlar de ändringar jag gjort till porteringen, både strukturella och kodmässiga.

I nedanstående beskrivning kommer jag särskilt att belysa de klasser som utgör kärnan-(Kernel) i SMS Notifier. Detta p.g.a. att det är kärnan som innehar den största funktionaliteten i applikationen.

6.1 Strukturen i SMS Notifier



Figur 6.1: Namnrymdsstrukturen hos SMS Notifier.

Figur 6.1 illustrerar namnrymdsstrukuren i den ursprungliga SMS Notifier. Den är uppdelad i sex stycken huvudnamnrymder där varje namnrymd innehåller klasser för att utföra en speciell uppgift eller ytterligare namnrymdsnivåer till klasserna.

- △ **SMS Notifier** : Huvudnamnrymd
  - △ **db** : Innerhåller klasser för att hantera kommunikationen mot en databas.

- △ **failed**: Verktyg för att spara meddelanden som misslyckats vid sändning i en databas.
- △ **history**: Verktyg för att spara meddelanden vars sändning lyckats i en databas.
- △ **msgtemplates**: Verktyg för att spara meddelandemallar i en databas.

△ **protocols** : Innehåller de klasser som ingår i de protokoll SMS Notifier kan använda. (För tillfället endast CIMD-2 men kan utökas i framtiden.)

- △ **serialphone**: Klasser för att sända sms via en mobiltelefon kopplad till serieporten. Används ej.
  - △ **EricssonT28**: Innehåller klasser för att sända sms via en Ericsson T28 mobiltelefon.
  - △ **SiemensSL45**: Innehåller klasser för att sända sms via en Siemens SL45 mobiltelefon.
- △ **telia**: Implementation av CMG's smsprotokoll UCP-50. Används ej.
- △ **vodafone**: Implementation av Nokias smsprotokoll CIMD-2. Fungerar ej.
- △ **vodafonetest**: Implementationen av Nokias smsprotokoll CIMD-2.

△ **smsnotdispatcher** : Huvudnamnrymden för sändenheten.

- △ **que**: Innehåller en köimplementation.
- △ **supervisory**: Innehåller klasser för att hantera tillstånd för olika delar av SMS Notifier. Ex. på tillstånd är t.ex. om uppkopplingen mot sms-centralen är aktiv eller inte.
- △ **thread**: Utökade trådegenskaper, loggar t.ex. tidsstämplar i samband med händelser.

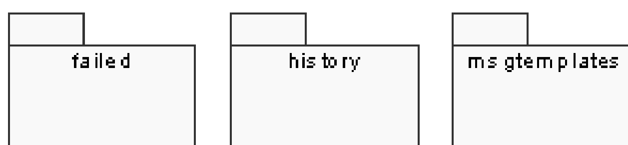
△ **smsnotify** : De klasser som knyter samman alla klasser till SMS Notifier.

△ **smsnotservlets** : De klasser som ger funktionalitet åt användarna.

- △ **util** : Små hjälpklasser som loggfilshanterare, konfigurationsverktyg etc.
  - △ **alarm**: Klass för att underrätta andra klasser och framförallt administratören att t.ex. en en databasförbindelse ej går att sätta upp. Används ej.
  - △ **authenticate**: Verifierar och hanterar användarkonton.
  - △ **billing**: Verktyg för att hålla reda på kostnader, används ej.
  - △ **config**: Läser in konfigurationsinställningar från en XML-fil.
  - △ **debug**: Skriver ut meddelanden med bl.a. tidsstämpel på systemkonsollen.
  - △ **file**: Allmän datafil för att lagra data som t.ex. användare etc.
  - △ **log**: Implementerar en loggfil samt skriver den till hårddisken.
  - △ **message**: Innehåller klasser för att implementera korta meddelanden, t.ex. sms-meddelanden och leveransrapporter.
  - △ **session**: Genererar ett sessionsnummer till varje nyinloggad användare.
  - △ **string**: Strängverktyg för att t.ex. ta bort mellanslag etc.

6.2 Databashanteringsklasserna i db

I namnrymden *db* finns tre namnrymder med klasser framtagna för att möjliggöra kommunikation mot en databas. Nedan visas ett klassdiagram över namnrymden och korta beskrivningar av klasserna.



Figur 6.2: De olika dbklasserna.

△ ***failed*** : Huvudnamnrymd

- ▷ **SmsFailedVO** : Objekt som innehåller de data som lagras för en misslyckad sändning.
- ▷ **SmsFailedDBHdlr** : Klass för att bl.a spara meddelandeobjekt vars sändning misslyckats i en databas.

△ ***history*** : Huvudnamnrymd

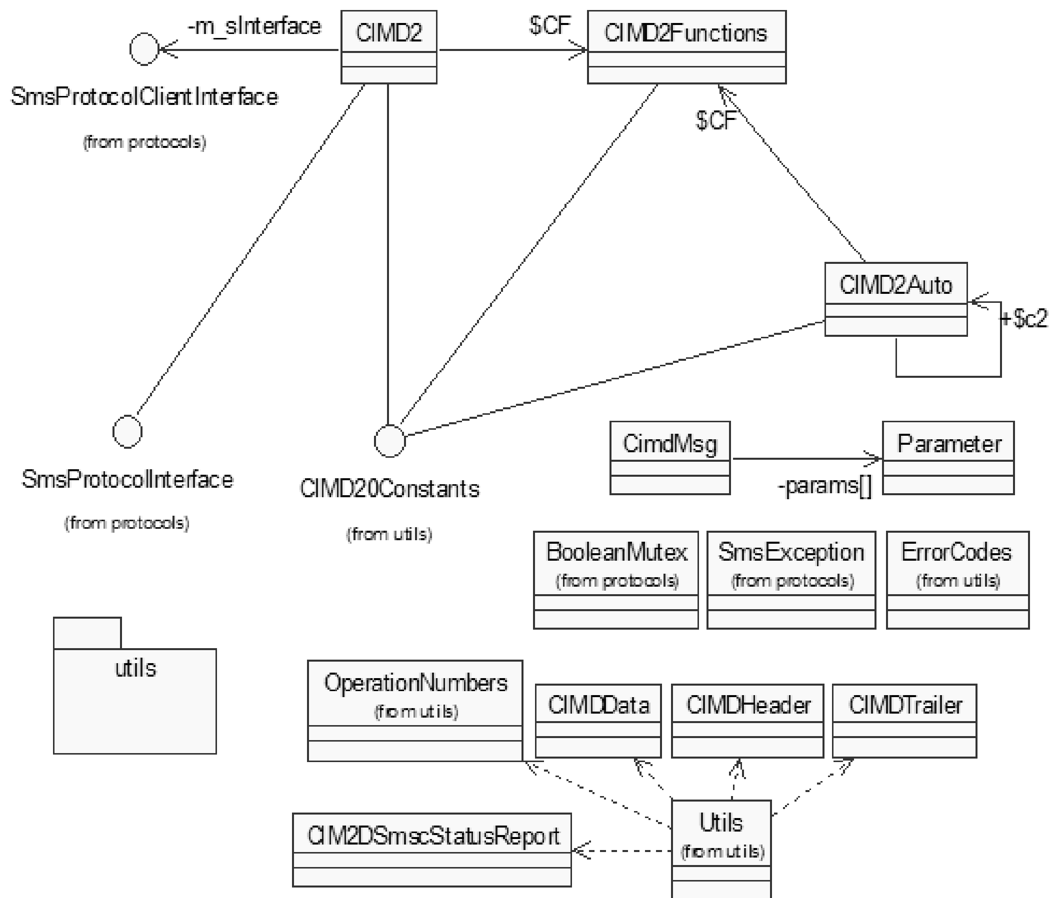
- ▷ **SmsHistoryVO** : Objekt som innehåller de data som lagras för ett sänt sms.
- ▷ **SmsHistoryDBHdlr** : Sparar meddelandeobjekt som sänts.

△ ***msgtemplates*** : Huvudnamnrymd

- ▷ **SmsMsgTempVO** : Objekt som implementerar en meddelandemall.
- ▷ **SmsMsgTmpDBHdlr** : Sparar mallobjekt.

6.3 Smsprotokoll för Vodafone - CIMD-2

Protokollet CIMD-2 är utvecklat av Nokia. Som tidigare nämnt behövde ej protokollet förbättras då det redan fungerade tillfredsställande. Detta gör att protokollets klassdiagram i figur6.3 ser likadant ut för både C# och för Java. Den enda skillnaden är att de klasser som ingår i namnrymden *vodafone* i listan nedan istället ligger i namnrymden *protocols.cimd2* eller *protocols.cimd2.utils*. Detta kommer att förklaras närmare i nästa kapitel. I diagrammet ser vi att klassen CIMD2 är central för funktionen och att att utils-klassen instansierar CIMDHeader, CIMDData, CIMDTrailer, CIM2DSmscStatusReport samt OperationNumbers för att utföra sina uppgifter. CIMDMsg hämtar information från ErrorCodes men instansierar inte klassen. CIMD2Auto instansierar CIMD2Functions och implementerar CIMD20Constants men används ej i praktiken och är inte heller testad av Devo IT. Anledningen till att Devo IT ej testat att ta emot sms m.h.a. klassen CIMD2Auto är att det abonnemang Devo IT innehar hos operatören Vodafone ej stöder detta.



Figur 6.3: Klassberoenden för CIMD2-protokollet.

Nedan följer en förklaring till klasserna och namnrymderna i namnrymden *protocols*. I tabellen symboliserar en uppil en namnrymd och högerpilen representerar en klass eller ett gränssnitt(skrivs ut).

△ ***protocols*** : Huvudnamnrymd med gränssnitt och klasser som används i CIMD-2 men som även kan användas i andra framtida protokoll t.ex. ett mms-protokoll etc.

▷ **BooleanMutex** : En semaforklass för att synkronisera trådar i applikationen. Används ej.

- ▷ **SmsException** : Ett exception som definierar vad som gått fel då ett undantag kastats.
- ▷ **SmsProtocolInterface** : Ett protokollgränssnitt för att skicka sms.
- ▷ **SmsProtocolClientInterface** : Ett protokollgränssnitt för att ta emot sms och leveransrapporter.

△ ***serialphone*** : Huvudnamnrymden för att sända sms via en mobiltelefon kopplad till serieporten. Används ej.

- ▷ **Commands** : Innehåller konstanter för kommandon.
- ▷ **DeliverPDU** : Konstanter som t.ex. talar om om ett sms är inkommande eller ej.
- ▷ **SendQueue** : Sänder en kö av sms.
- ▷ **SubmitPDU** : Konstanter som t.ex. talar om om ett sms är inkommande eller ej.
- ▷ **Utils** : Hjälpmetoder för att t.ex. göra om tecken i ett meddelande till teckenkoder för att sända i sms.

△ ***EricssonT28*** : Huvudnamnrymden för att sända sms via en Ericsson T28 mobiltelefon kopplad till serieporten. Används ej.

- ▷ **EricssonT28** : Klass för att hantera smstrafik via en Ericsson T28 mobiltelefon.

△ ***SiemensSL45*** : Huvudnamnrymden för att sända sms via en Siemens SL45 mobiltelefon kopplad till serieporten. Används ej.

- ▷ **SiemensSL45** : Klass för att hantera smstrafik via en Siemens SL45 mobiltelefon.

△ ***telia*** : Huvudnamnrymden för UCP-50 protokollet. Ej testat av Devo IT, används ej.

- ▷ **Telia** : Sänder sms via operatören Telia, gör bl.a. om tecken till dess motsvarande teckenkod i sms-protokollet UCP-50.
- ▷ **UCP_OC-51** : Innehåller en mängd fält som t.ex. mottagarens mobiltelefonnummer etc.
- ▷ **UCPHeader** : Implementerar headern där information som t.ex. hur många tecken som finns i meddelandet som skall sändas.
- ▷ **ValueHolder** : Hjälpklass, lagrar tecken i meddelandet som ej stöds av UCP-50 protokollet.

△ ***vodafonetest*** : Huvudnamnrymden för det fungerande CIMD-2 protokollet.

- ▷ **CIMD20Constants** : Gränssnitt med konstanter som används i protokollet.
- ▷ **CIMD2** : Huvudklass för att logga in på sms-centralen och sända sms.
- ▷ **CIMD2Auto** : Klass för att automatiskt ta emot sms. Används ej.
- ▷ **CIMD2Functions** : Innehåller funktioner som behövs i andra klasser som att skriva meddelandet till nätverksströmmen.
- ▷ **CIMDMsg** : Implementerar det meddelande SMS Notifier skickar till sms-centralen.
- ▷ **Parameter** : En klass som sparar parametrar b.l.a meddelandetexten och avsändaren.

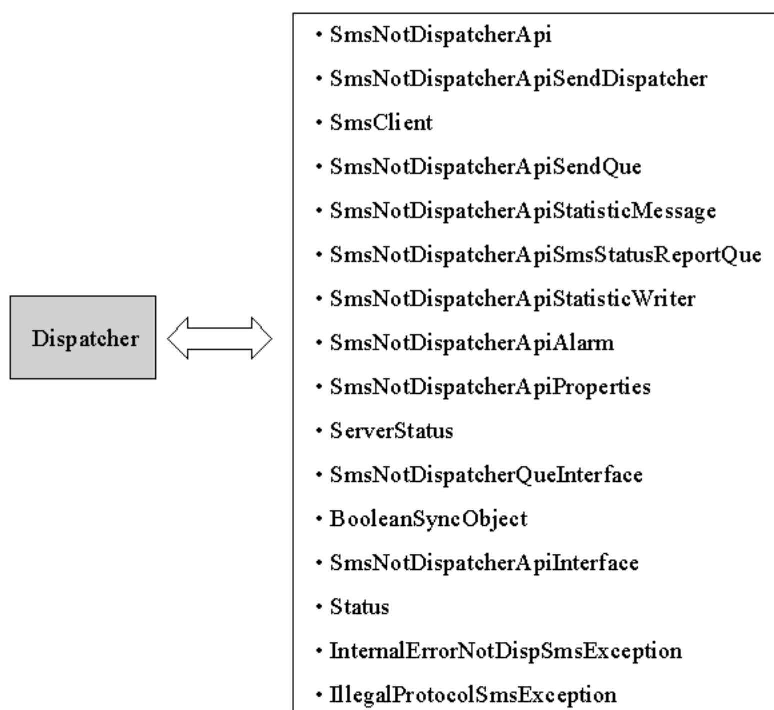
△ ***vodafone*** : Huvudnamnrymden för det icke fungerande CIMD-2 protokollet.

- ▷ **vodafone** : Huvudklass för det icke fungerande CIMD-2 protokollet.
- ▷ **CIM2DSmscStatusReport** : Ärver SmscStatusReport och lägger till funktionalitet som är specifik för CIMD-2 protokollet.
- ▷ **CIMDHeader** : Innehåller paketnummer och en kod som indikerar vad operatören skall göra med det data som kommer. Kan t.ex. vara inloggningsinformation.

- ▷ **CIMDData** : Innehåller det meddelande som användaren sänder.
- ▷ **CIMDTrailer** : Innehåller checksumman för paketet.
- ▷ **ErrorCodes** : Felkoder med motsvarande textbeskrivningar för protokollet.
- ▷ **OperationNumbers** : Operationskoder med textbeskrivningar..
- ▷ **Utils** : Verktyg för t.ex. checksummeberäkning.

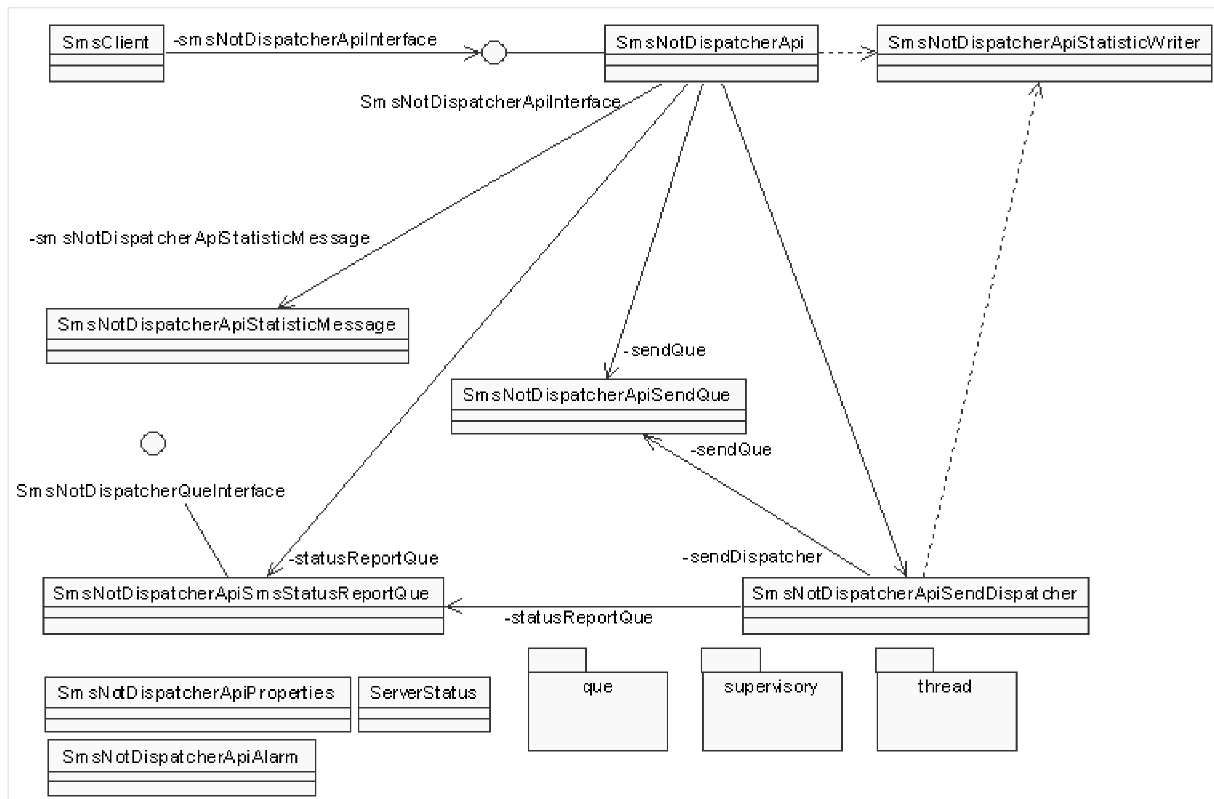
6.4 Sändmodulen Dispatcher

Dispatcher är en mycket stor del av SMS Notifier. Det är denna del som är länken mellan användaren och protokollet. Den handhar kärnfunktionerna som att ta emot sms från användaren, lagra dem i en kö för att sedan sända dem. I sändenheten ingår även Incoming Handler. Nedan visas en figur där klasserna som utgör Dispatchern illustreras samt ett klassdiagram för kärnmodulerna Dispatcher och Incoming handler som demonstrerades i figur 5.1 på sidan 17. Modulerna Billing och Persistent message que ingår också i kärnan men dessa är ej lika mycket i fokus som modulerna Incoming handler och Dispatcher. Billing och Persistent message que är dessutom relativt enkelt uppbyggda och består endast av hjälpklasserna i *util.billing*-namnrymden respektive SmsNotDispatcherApiSendQue i *smsnotdispatcher*-namnrymden.



Figur 6.4: Klasserna som utgör Dispatchermodulen.

Som man ser i figur 6.5 är SmsNotDispatcherApi central i kärnans Dispatchermo-



Figur 6.5: Klassberoenden för dispatchermodulen.

dul. Denna instansierar de övriga klasserna och använder dessa för att knyta samman en fungerande sändenhet. ServerStatus hanterar händelser baserade på serverns (den sms-mottagande och sändande delen av SMS Notifier) tillstånd. D.v.s om en klass skulle stänga ned servern skickas detta som en händelse till övriga klasser som använder servern. SmsNotDispatcherApiProperties och SmsNotDispatcherApiAlarm används ej.

Figur 6.6: Klasserna som utgör incoming handler-modulen.

I figur 6.6 ser vi Att Incoming handler är relativt enkel. Den består i princip bara av SmsNotDispatcherApi och SmsClient. Det är SmsNotDispatcherApi som tar emot meddelandet.

Dispatchern består av klasserna i nedanstående lista. Alla klasser i listan utom BooleanSyncObject beskrivs med ett klassdiagram samt en beskrivning. BooleanSyncObject och gränssnittet i namnrymden beskrivs istället kort i listan.

△ **smsnotdispatcher** : Huvudnamnrymd för sändenheten

- ▷ **SmsNotDispatcherApi** : Se avsnitt 6.4.1
- ▷ **SmsNotDispatcherApiSendDispatcher** : Se avsnitt 6.4.2
- ▷ **SmsClient** : Se avsnitt 6.4.3
- ▷ **SmsNotDispatcherApiSmsStatusReportQue** : Se avsnitt 6.4.4
- ▷ **SmsNotDispatcherApiSendQue** : Se avsnitt 6.4.5
- ▷ **SmsNotDispatcherApiStatisticMessage** : Se avsnitt 6.4.6
- ▷ **SmsNotDispatcherApiStatisticWriter** : Se avsnitt 6.4.7
- ▷ **ServerStatus** : Se avsnitt 6.4.8
- ▷ **SmsNotDispatcherApiAlarm** : Alarmidentiteter som bl.a används i sändenheten. Används ej.
- ▷ **SmsNotDispatcherApiProperties** : Implementerar ett konfigurationsobjekt för sändenheten. Används ej.

- ▷ **SmsNotDispatcherQueInterface** : Gränssnitt för köhantering.
- ▷ **BooleanSyncObject**: En klass där en flagga beskriver ett av två lägen, sant eller falskt.
- ▷ **SmsNotDispatcherApiInterface** : Gränssnitt för sändenhetens API.
- ▷ **Status**: Gränssnitt med konstanter för olika servertillstånd t.ex. “running” eller “connecting”.
- ▷ **InternalErrorNotDispSmsException**: Undantag
- ▷ **IllegalProtocolSmsException** : Undantag

△ **que** : Huvudnamnrymd, se avsnitt 6.4.9

- ▷ **Que** : Se avsnitt 6.4.9
- ▷ **CompoundQueObject** : Se avsnitt 6.4.9
- ▷ **QueException** : Ett undantag för en kö.

△ **supervisory** : Huvudnamnrymd, se avsnitt 6.4.10

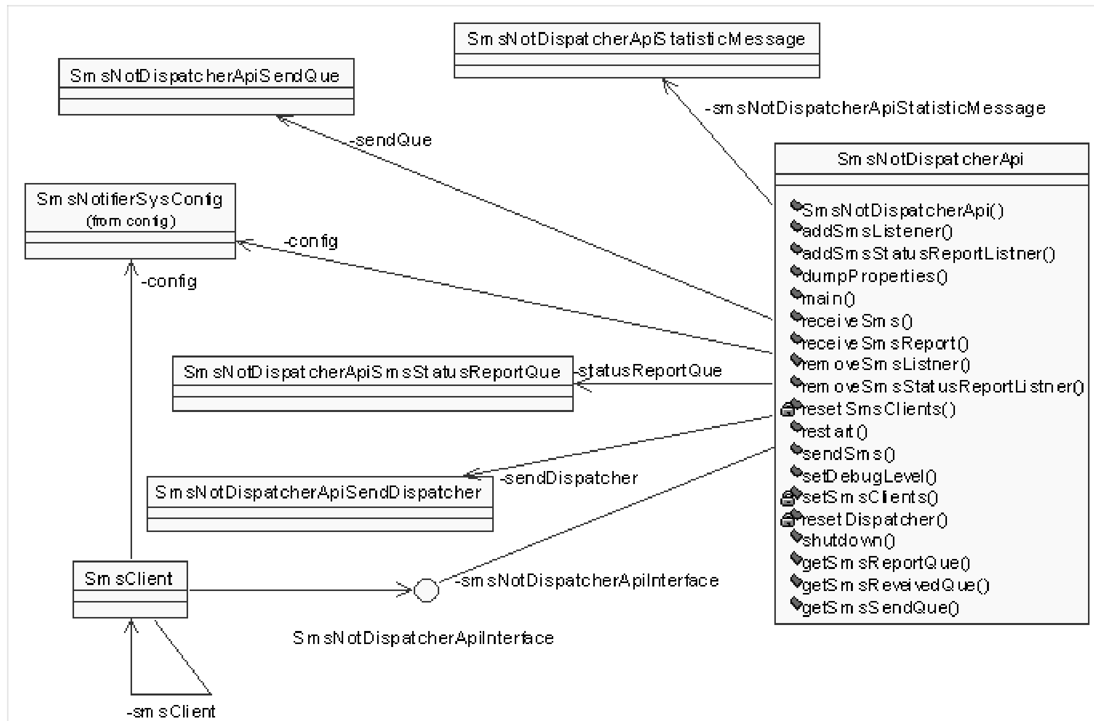
- ▷ **Client** : Se avsnitt 6.4.10
- ▷ **SupervisoryClient** : Se avsnitt 6.4.10
- ▷ **SupervisoryCommand** : Se avsnitt 6.4.10
- ▷ **SupervisoryCommandListener** : Ett gränssnitt.

△ **thread** : Huvudnamnrymd, se avsnitt 6.4.11

- ▷ **ThreadParameters** : Se avsnitt 6.4.11
- ▷ **TimeStampedThread** : Se avsnitt 6.4.11

6.4.1 SmsNotDispatcherApi

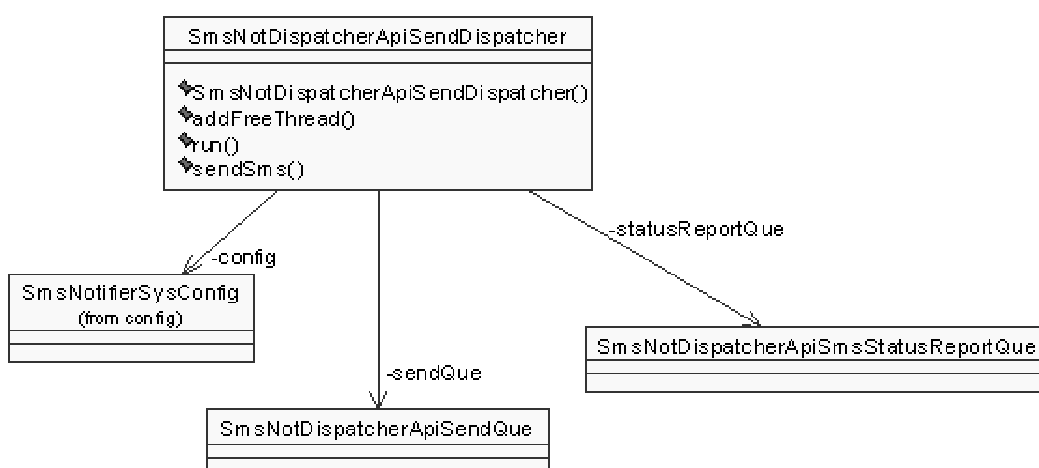
SmsNotDispatcherApi(förkort: SNDApi) startar en tjänst som sänder och tar emot sms. SNDApi registrerar klienter (användare) som skickar ett meddelande till SNDApi. Dessa klienter kan även vänta på och ta emot sms och leveransrapporter. Efter att meddelandet har inkommit till SNDSApi sänder den meddelandet vidare till en instans av klassen SmsNotDispatcherSendDispatcher. Meddelandet sänds m.h.a instansen av klassen SmsClient som den får från konfigurationsinstansen SmsNotifierSysConfig. SmsNotifierSysConfig används för att läsa in inställningar till SNDApi. Slutligen erhålls resultatet av sändningen m.h.a instanserna av klasserna SmsNotDispatcherStatisticMessage för att sedan sparas i instansen av SmsNotDispatcherApiSmsStatusReportQue. Klassen implementerar gränssnittet SmsNotDispatcherApiInterface och instansierar de övriga klasserna i diagrammet.



Figur 6.7: Klassberoenden för smsnotdispatcherapi.

6.4.2 SmsNotDispatcherApiSendDispatcher

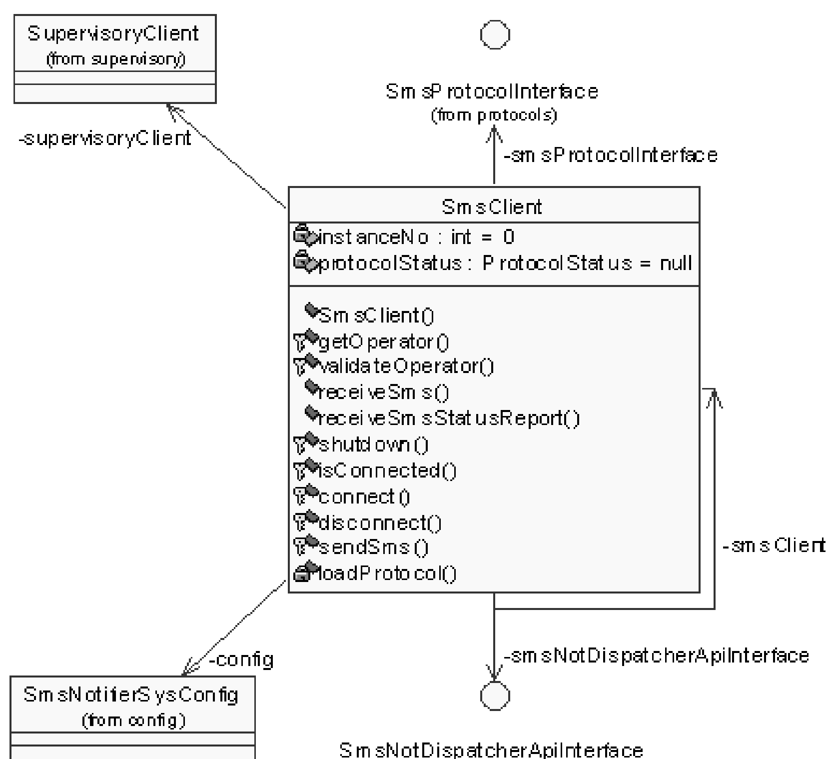
SmsNotDispatcherApiSendDispatcher(förkort: SNDASDispatcher) implementerar en tråd som också använder SmsNotifierSysConfig-instansen för att hämta inställningar. Då ett meddelande inkommer från SmsNotDispatcherApi lagrar SmsNotDispatcherSendDispatcher meddelandet i en instans av köklassen SmsNotDispatcherSendQue. Därefter sänds meddelandet med en instans av klassen SmsClient. Till slut lagras resultaten av sändningarna m.h.a SmsNotDispatcherApiSmsStatusReportQue-instansen.



Figur 6.8: Klassberoenden för smsnotdispatcherapisenddispatcher.

6.4.3 SmsClient

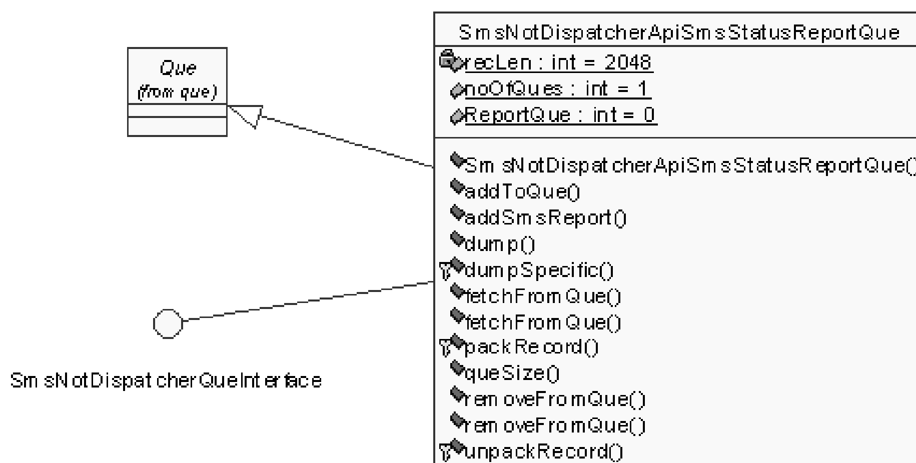
SmsClient är en klient som använder en instans av gränssnittet SmsProtocolInterface för att sända sms. Den hämtar sin konfiguration från SmsNotifierSysConfiginstansen. Den skapar även en instans av gränssnittet SmsNotDispatcherApi som används för uppkoppling mot sms-centralen samt att skicka och hämta meddelanden och leveransrapporter med. En instans av SupervisoryClient används för att kontrollera statusen på uppkopplingen.



Figur 6.9: Klassberoenden för smsclient.

6.4.4 SmsNotDispatcherApiSmsStatusReportQue

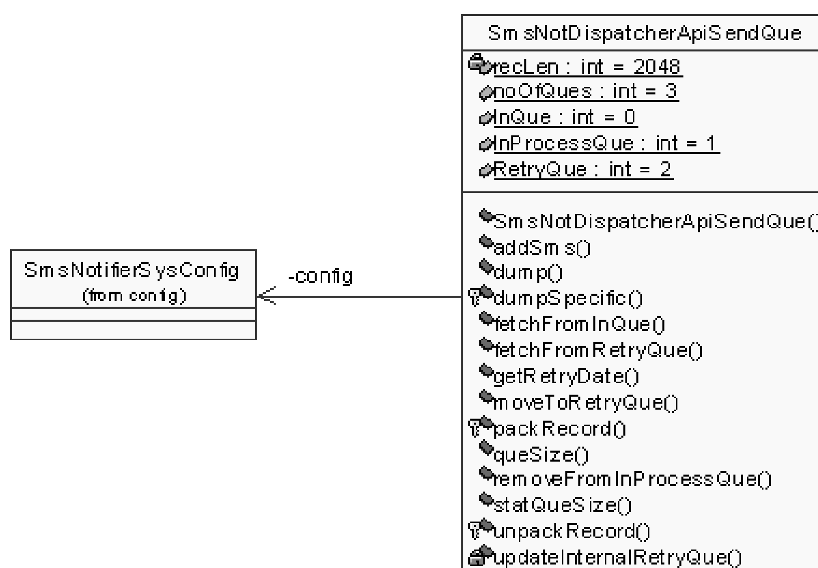
I klassdiagrammet nedan ser vi att SmsNotDispatcherApiSmsStatusReportQue ärver från klassen Que och implementerar gränssnittet SmsNotDispatcherQueInterface. Klassen hanterar en kö med statusmeddelanden, s.k leveransrapporter. Den innehåller metoder för att lägga till, ta bort, hämta och skriva ut data ur kön.



Figur 6.10: Klassberoenden för smsnotdispatcherapismsstatusreportque.

6.4.5 SmsNotDispatcherApiSendQue

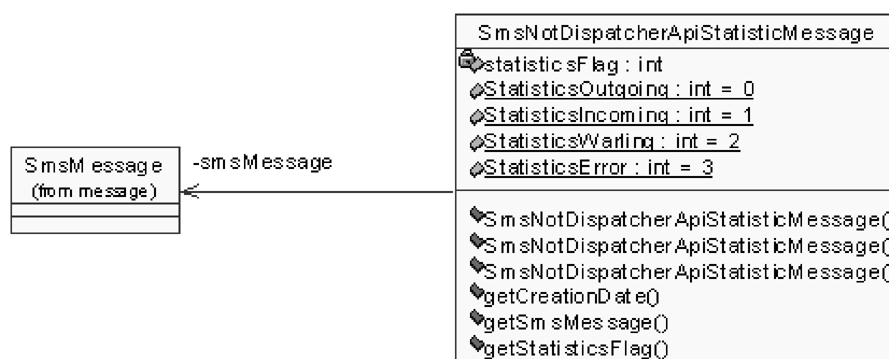
SmsNotDispatcherApiSendQue implementerar en sändkö delad i tre olika delar. En kö för inkommande sms från användarna, den s.k InQue. En kö för meddelanden som håller på att sändas, dessa återfinns i kön InProcessQue. Den sista kön är RetryQue, en kö för meddelanden som skall sändas om. Klassen ärver från hjälpklassen Que. Den använder också instansen av SmsNotifierSysConfig. Klassen använder inga andra objekt som inte härrör från den inbyggda namnrymden *java*.



Figur 6.11: Klassberoenden för smsnotdispatcherapisendque.

6.4.6 SmsNotDispatcherApiStatisticMessage

SmsNotDispatcherApiStatisticMessage(förkort: SNDASMessage) skapar en instans av hjälpklassen SmsMessage. Tillsammans med objektet knyter SNDASMessage samman information som skapandedatum samt information om i vilken riktning meddelandet sändes, till eller från SMS Notifier, men även information som om sändningen av meddelandet misslyckades sparas.



Figur 6.12: Klassberoenden för smsnotdispatcherapistatisticmessage.

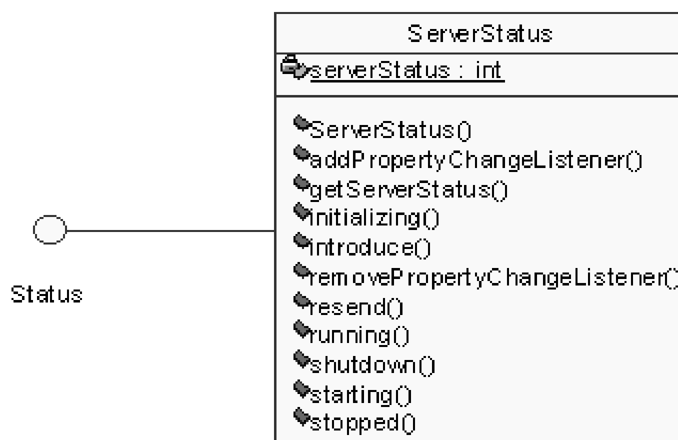
6.4.7 SmsNotDispatcherApiStatisticWriter

SmsNotDispatcherApiStatisticWriter använder endast inbyggda klasser från C#-namnrymden *java*. Den skriver statistikinstanser av SmsNotDispatcherApiStatisticMessage till en fil.

Figur 6.13: Klassberoenden för smsnotdispatcherapistatisticwriter.

6.4.8 ServerStatus

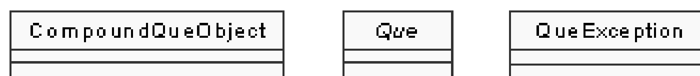
Utlöser en händelse när SMS Notifiers serverdel byter tillstånd från t.ex. “running” till “stopped”. När ServerStatus utlöser händelserna blir även andra klasser varskodda om att ett tillstånd har bytts vilket t.ex. i det här fallet skulle stoppa en sms-sändande tråd.



Figur 6.14: Klassberoenden för serverstatus.

6.4.9 Namnrymden que

que är en namnrymd som består av två hjälpklasser och ett gränssnitt för köhantering. Dessa beskrivs kort nedan.



Figur 6.15: Klassberoenden för namnrymden que.

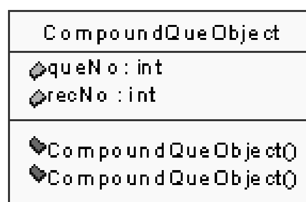
Que

Que är en köimplementation. Instansierar ett objekt av DaFile från namnrymden *smsnotifier.util.file* och skriver på så sätt kön på hårddisken.

Figur 6.16: Klassberoenden för queklassen.

CompoundQueObject

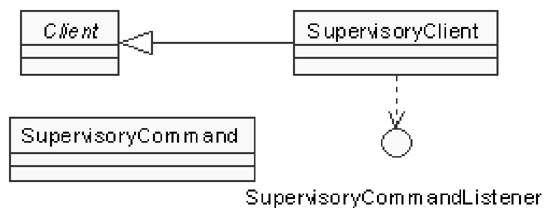
CompundQueObject är helt uppbyggt av funktioner från *java*-namnrymden. CompoundQueObject är ett köobjekt som lagras på fil. Alla objekt som sparas på hårddisken är av denna typ. Klassen innehåller information som t.ex. vilken kö den ska lagras i och vilket sessionsnummer den tillhör.



Figur 6.17: Klassberoenden för compundqueobjectklassen.

6.4.10 Namnrymden supervisory

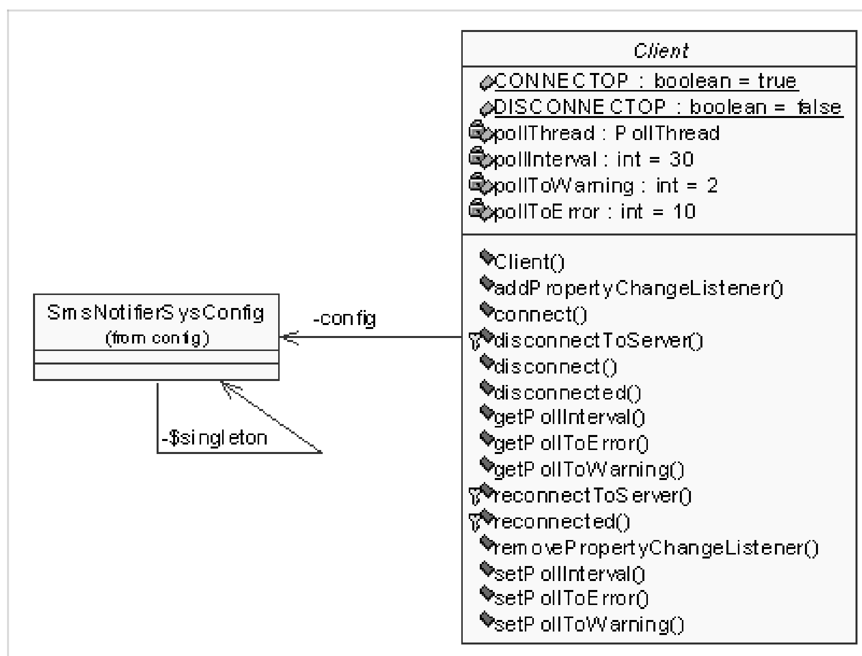
Supervisory är en namnrymd innehållande fyra stycken hjälpklasser till sändenheten. Dessa hjälpklasser handhar serverdelen av SMS Notifier, d.v.s den sms-sändande delen.



Figur 6.18: Klassberoenden för namnrymden supervisory.

Client

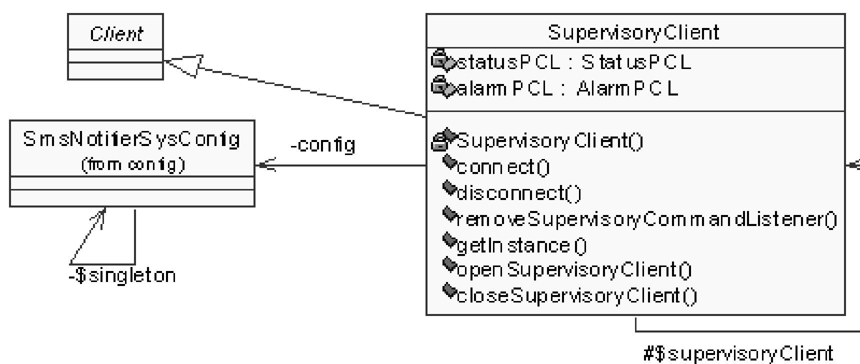
Client implementerar en tråd som handhar uppkopplingen mot en server. M.h.a händelser vet tråden när något skall göras. Client använder den statiska instansen av SmsNotifier-SysConfig för att hämta inställningar.



Figur 6.19: Klassberoenden för clientklassen.

SupervisoryClient

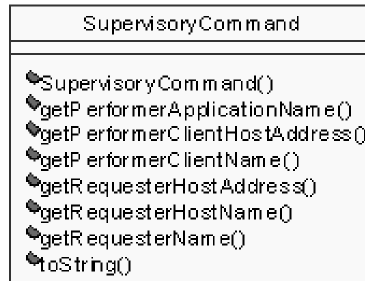
SupervisoryClient är en centraliserad servertillståndshanterare. Denna klass lyssnar efter alla typer av händelser som genereras i SMS Notifier och sparar den informationen till andra klasser. SupervisoryClient ärver klassen Client och instansierar ett objekt av klassen SmsNotifierSysConfig för att hämta inställningar. Dessutom instansierar den sig själv för att kunna skicka just den instansen till andra klasser. Instansen är av singleton-typ.



Figur 6.20: Klassberoenden för supervisoryclientklassen.

SupervisoryCommand

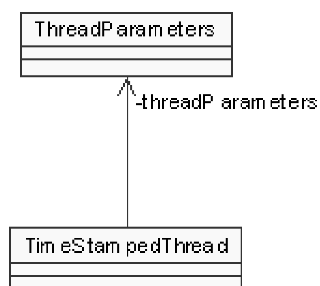
SupervisoryCommand är en klass som är tänkt att användas för lastbalansering. Lastbalansering innebär att flera separata SMS Notifier-servrar exekverar parallellt för att sända sms. SupervisoryCommand används då för att hantera alla servrar och identifiera dessa.



Figur 6.21: Klassberoenden för supervisorycommandklassen.

6.4.11 Namnrymden thread

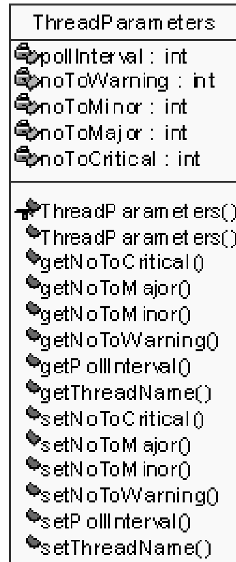
Thread består av två klasser med specialiserade trådar. Klasserna utökar den vanliga funktionaliteten hos trådar med egenskaper som t.ex. namn, senaste exekveringsdatum och funktioner som sätter dessa egenskaper. Detta för att ge så mycket information som möjligt i loggen.



Figur 6.22: Klassberoenden för namnrymden thread.

ThreadParameters

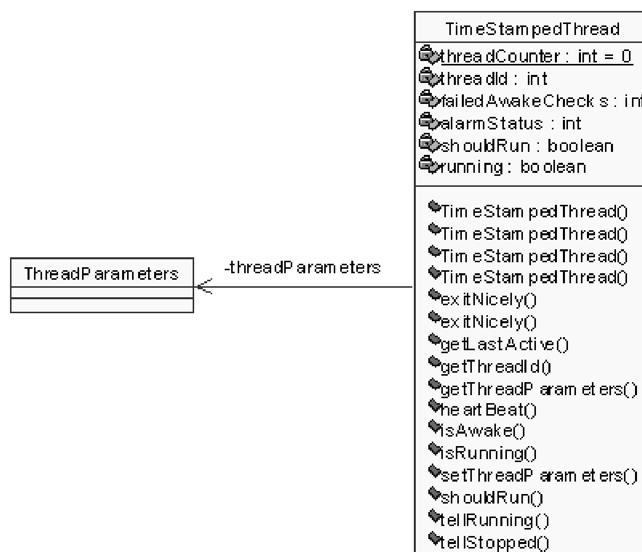
Threadparameters är en klass för att lagra parametrar till en generell tråd. Dessa parametrar kan t.ex. vara vilket namn tråden har (d.v.s. vilken klass som exekverade tråden).



Figur 6.23: Klassberoenden för threadparametersklassen.

TimeStampedThread

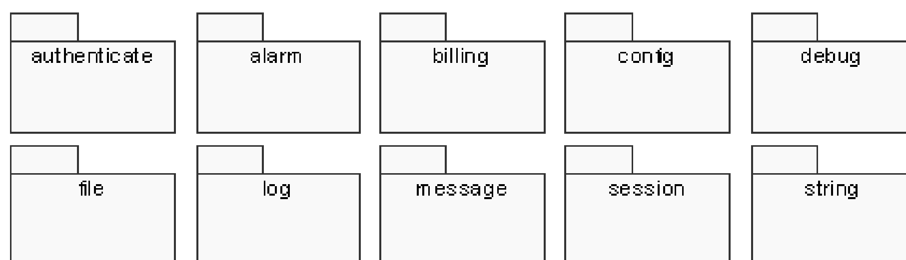
TimeStampedThread instansierar ThreadParameters och implementerar en tråd som bl.a. lagrar när den sist exekverade, om den exekverar just nu o.s.v.



Figur 6.24: Klassberoenden för timestampedthreadklassen.

6.5 Hjälpklasserna i util

Namnrymden *util* innehåller en mängd små verktyg och hjälpmedel som bistår resten av applikationen. Här finns även hjälpklasserna för billing som är den enda delen i kärnan utanför namnrymden *smsnotdispatcher*.



Figur 6.25: De olika hjälpklasserna.

△ ***alarm*** : Huvudnamnrymd

- ▷ **Severities** : Ett gränssnitt där olika grader finns definierade för hur allvarligt ett larm är. Används ej.
- ▷ **Alarm** : Implementerar severities. Hanterar alarm, lägger till alarm i en lista o.s.v. Används ej.

△ ***authenticate*** : Huvudnamnrymd

- ▷ **Authenticate** : Verifierar användare.
- ▷ **AuthStorage** : Visar, skapar eller tar bort användare.
- ▷ **StorageInterface** : Gränssnitt för Authenticateklassen.
- ▷ **Manage** : Textbaserat gränssnitt för AuthStorage.

△ ***billing*** : Huvudnamnrymd

- ▷ **BillingInfo** : Objekt som innehåller all kundinformation för skrivning till en databas. Används ej.
- ▷ **Billing** : Skriver kostnadsstatistikobjekt i en databas. Används ej.
- ▷ **BillingQue** : En kö av BillingInfoobjekt som skall skrivas till databasen. Används ej.

△ ***config*** : Huvudnamnrymd för SmsNotifierSysConfigklassen

- ▷ **SmsNotifierSysConfig** : Läser konfigurationsfiler i XML-format. Instansierar sig själv som singleton vilket gör att endast en instans av klassen kan hämtas. Det är denna instans som alla andra klasser hämtar inställningar av genom att begära den från SmsNotifierSysConfig.

△ ***debug*** : Huvudnamnrymd för Debugklassen

- ▷ **Debug** : Skriver meddelanden till systemkonsollen som t.ex. "Sänder meddelande.." etc.

△ ***file*** : Huvudnamnrymd för DaFileklassen

- ▷ **DaFile** : En generell datafil som används av klassen Que.

△ ***log*** : Huvudnamnrymd

- ▷ **Log** : Implementerar en loggfil.
- ▷ **LogFileWriter** : Skriver loggfilsobjektet på disk.

△ ***message*** : Huvudnamnrymd

- ▷ **Address** : Gränsssnitt med funktioner för att returnera telefonnummerdata som landsprefixet eller hela telefonnumret.
- ▷ **BinaryMessage** : Skapar ett binärt meddelande av meddelandets kropp etc.

- ▷ **Message** : Representerar ett generellt meddelande med fix maxlängd. Lagrar egenskaper som avsändare, mottagare, meddelande etc.
- ▷ **SmsMessage** : Implementerar ett smsmeddelande baserat på den mer generella Messageklassen.
- ▷ **MessagingException** : Undantag för alla meddelandefel som kan uppstå.
- ▷ **SmsReport** : Den typ av resultatobjekt som kommer in till SMS Notifier, fungerar alltså som leveransrapport.
- ▷ **SmscStatusReport** : Basklass för statusrapporter från smscentralen.
- ▷ **SmsStatusReport** : Basklass för leveransrapporter från smscentralen.
- ▷ **MsisdnAddess** : Implementerar funktioner för att formatera telefonnummer på s.k MSISDN-standard[2], t.ex. måste ett telefonnummer till Sverige börja med +46 eller 046.

△ ***session*** : Huvudnamnrymd

- ▷ **SessionIdCounter** : Räknar ut ett sessionsnummer.
- ▷ **SessionId** : “Delar ut” sessionsnummer till nya användare för varje gång en sändklient av SMS Notifier startas.

△ ***stringutil*** : Huvudnamnrymd

- ▷ **StringUtil** : Små stränghjälpmedel som att ta bort mellanslag t.ex.
- ▷ **ZeroPadder** : Fyller ut strängar som är för korta till en specificerad längd med nollor.

[2] Mobile Station International Integrated Services Digital Network

7 Porteringsprocessen

I detta kapitel kommer jag att beskriva porteringsprocessen samt de ändringar och uppdateringar jag gjort i SMS Notifier. Jag kommer även att rekommendera ändringar som kan göras för att ytterligare förbättra applikationen.

För att utvecklingen av applikationen skulle vara så lätt som möjligt för personalen på Devo IT att fortsätta, bestämde jag mig för att, i den utsträckning jag fann det lämpligt, använda strukturen från den gamla SMS Notifier. Skillnaderna i den nya designen och den ursprungliga versionen är därför inte stor.

Jag började med att försöka sätta mig in i hur produkten fungerade d.v.s vilka klasser som gjorde vad. Utöver sparsamt kommenterad kod fanns ingen dokumentation av SMS Notifier som t.ex. klassdiagram eller metodbeskrivningar. Bl.a. av denna anledning blev det ett mål att dokumentera produkten i sig för att porteringen skulle bli bra.

Porteringsarbetet började med att sätta upp arbetsstationen jag skulle använda under arbetet. Detta innebar bl.a. att hämta den kod Javaversionen av SMS Notifier bestod av till arbetsstationen. Detta skedde via kodrevisionshanteringsprogrammet WinCVS[4]. WinCVS är ett opensource-program som hanterar olika revisionsupplagor av kod. WinCVS ser till att utvecklare alltid har den senast uppdaterade koden i ett projekt och programmet sparar alla kodändringar som varje utvecklare gjort i projektet under utvecklingen.

Själva porteringen gick till så att jag lade in Javakoden i ett nytt projekt i Microsoft Visual Studio.Net 2003. Därefter kompilerade jag för att se vilka fel jag fick och analyserade felen genom att titta på Javas API[2]. Här såg jag hur de felaktiga metoderna eller klasserna fungerade och kunde då avgöra vad den felaktiga koden var menad att göra. Därefter letade jag upp en motsvarande metod i C#[3] och ersatte den felaktiga Javakoden med motsvarande C#-kod.

Efter att porteringsprocessen av protokollkoden i namnrymden *vodafone* pågått i ungefär två veckor fick jag hjälp av Anders Bergkvist på Devo IT. Anders skulle hjälpa mig att konfigurera utvecklingsmiljön Borland JBuilder X. Miljön används för att utveckla Javaap-

plikationer och här analyserade jag bl.a Javakoden. Det var i utvecklingsmiljön Microsoft Visual Studio.NET 2003 jag utförde själva implementationen. Anders hjälpte mig att ställa in miljön så att jag kunde kompilera och exekvera applikationen. Under den första halvan av tiden för examensarbetet var Anders uthyrd till ett externt företag och var således sällan i Devo IT's lokaler. Då Anders hjälpte mig såg han att den kod jag porterade ej var den aktuella. Det protokoll som faktiskt användes var ju det i namnrymden *vodafonetest*. Han såg då till att all kod och alla inställningar var korrekta för det fortsatta arbetet.

Efter detta gick porteringsarbetet relativt bra men det tog något längre tid än jag räknat med p.g.a att koden i projektet var oerhört omfattande. Dessutom berodde klasserna mer av varandra än vad jag uppfattat när jag studerade koden i början av porteringen.

Då jag hade suttit med porteringsarbetet i ca sju veckor fick jag ett råd av en anställd på Devo IT att leta efter ett verktyg att hjälpa mig portera koden med. Då jag började få ont om tid för att hinna slutföra projektet följde jag hans råd och hittade snabbt en produkt som hette *Microsoft Java Language Conversion Assistant 2.0*[5](förkort:JLCA) att portera kod från Java till C# med. Verktyget var endast integrerbart med Microsofts utvecklingsmiljö Visual Studio.NET 2003.

JLCA startades i form av en s.k *wizard* som först frågade efter vilken typ av projekt som skulle porteras. De tre olika alternativen var ett Visual Studio J#-projekt, en enkel mapp med filer eller ett webprojekt som inkluderar JSP/servlets. Servlets är ett sätt att med Java implementera webtjänster. JSP (Java Server Pages) är ett sätt att bygga upp dynamiska websidor i programspråket Java. Då SMS Notifier är just ett webprojekt med JSP och servlets valde jag denna. Därefter frågade JLCA var all nödvändig källkod fanns. Detta inkluderar Suns Javaapi och de tilläggsfiler Devo IT använt. När jag angett sökvägarna till dessa konverterade JLCA det mesta av koden till C# på knappt 5 minuter.

7.1 Strukturen hos den porterade SMS Notifier

I detta avsnitt börjar jag med att ge läsaren en kort inblick i struktur och modularisering. Jag vill med detta få läsaren att förstå hur jag tänkt då jag porterat koden.

7.1.1 Kort om modularisering

När ett projekt där flera programmerare deltar utvecklas och då ett program blir väldigt stort, är modularisering viktig. Olika hjälpmedel för att skapa en bra struktur finns i många moderna programspråk. I programspråket Java t.ex. består program ofta av paket. Dessa paket utgörs av klasser och/eller gränssnitt. De kan ha definierats av utvecklaren själv eller någon annan. Paketen fungerar ungefär på samma sätt som de namnrymder en programmerare definierar i C++. T.ex. så kan det i Java se ut så här:

```
import java.io.*;
import java.lang.math;
```

Detta informerar kompilatorn om vilka klasser man kommer att använda i programmet. Kompilatorn vet då var den ska leta efter de klasser som den ej känner igen. Vilken klass som ska användas blir lätt för kompilatorn att avgöra då klasserna finns i olika paket.

Javas paketstruktur är ett sätt att abstrahera klasser, ett sätt att sortera dem i en enhetlig och sammanhängande struktur. Med detta menas att de klasser som har ett samband, t.ex. aritmetiska funktioner, samlas i ett paket t.ex. *java.lang.math*. Alla av Sun utvecklade paket hamnar i en hierarkisk ordning under roten *java.**. Om man samlar egna klasser till ett paket kan dessa läggas i en valfri struktur t.ex. *minapaket.matematik.trigonometri*.

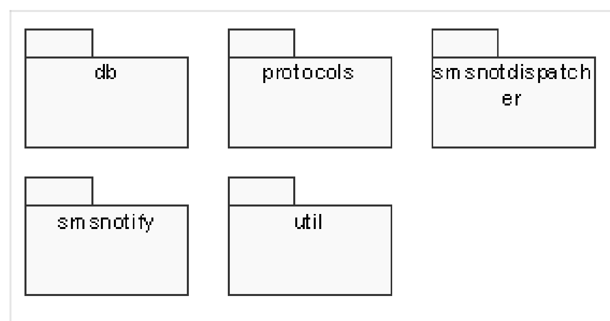
C# har ett liknande sätt att hantera modularisering kallat *namespaces*(namnrymder). En namnrymd är dels en samling klasser och dels det scope som klasserna och metodernas namn är giltiga i. En namnrymd kan även innehålla andra namnrymder. Den namnrymd som alla av Microsoft utvecklade klasser hamnar under är *System*. Även här hamnar klasserna i en hierarkisk struktur där *System* ligger högst upp. Då man refererar en klass eller

en namnrymd är det den sista man skriver som man avser att använda, se exemplet nedan.

```
using System.Collections.Hashtable;
```

I C#-exemplet ovan ser vi att klassen *Hashtable* finns i namnrymden *System.Collections*. Denna metod för att hantera namnrymder är samma som i C++.

7.1.2 Huvudnamnrymden *smsnotifier*



Figur 7.1: Namnrymdsstrukturen hos den porterade SMS Notifier.

SMS Notifiers struktur lät jag vara relativt intakt. Genom att jämföra figur 7.1 med figur 6.1 på sidan 21, ser man att den enda förändring jag gjort på SMS Notifiers huvudnamnrymd är att jag tagit bort namnrymden *smsnotifier.smsnotservlets*. Servlets är Javas sätt att i Java bygga ut funktionaliteten hos en webserver. I Visual Studio och C# finns andra lösningar för detta t.ex. Active Server Pages. I listan nedan har jag sammanfattat namnrymdsstrukturen hos den porterade SMS Notifier. De namnrymder jag ändrat kommer ändringarna att beskrivas.

- △ **SMS Notifier** : Huvudnamnrymd, namnrymden *smsnotservlets* har tagits bort.
  - △ **db**
    - △ **failed**
    - △ **history**

- △ **msgtemplates**

△ **protocols** - Alla ickefungerande och icke-använda protokoll har tagits bort.

- △ **cimd2** - Namnrymden har slagits samman från namnrymderna *vodafonetest* samt *vodafone* i den ursprungliga SMS Notifier.
  - △ **utils** - Namnrymden har lagts till för att separera hjälpklasser och konstanter från protokollets huvudklasser.

△ **smsnotdispatcher** - Namnrymden *events* har lagts till.

- △ **events** - Namnrymd som innehåller de händelser med vilka klasserna i SMS Notifier kommunicerar med varandra.
- △ **que**
- △ **supervisory**
- △ **thread**

△ **smsnotify**

△ **util** - Namnrymden *billing* borttagen p.g.a att den ej används.

- △ **alarm**
- △ **authenticate**
- △ **config**
- △ **debug**
- △ **file**
- △ **log**
- △ **message**
- △ **session**
- △ **string**

I tabellen ovan ser vi den nya namnrymdsstrukturen. De förändringar jag gjort åskådliggörs tydligast vid jämförande med tabell 6.1 på sidan 21. Förändringarna beskrivs närmare

senare i kapitlet då jag bl.a. beskriver delnamnrymderna *protocols* och *smsnotdispatcher* närmare. Dessa är nämligen de namnrymder där jag gjort de största förändringarna.

7.1.3 Protokollnamnrymden *protocols.cimd2*

I tabell 6.3 på sidan 26 ser man de olika protokoll som tidigare fanns i namnrymden *protocols*. Dessa var bl.a. *vodafonetest*, *telia* och *serialphone* o.s.v. Efter porteringen skall SMS Notifier endast använda sig av operatören Vodafone. P.g.a detta har jag tagit bort de protokoll som ej stöds av Vodafone. Dessa var protokollen i namnrymderna *serialphone* och *telia*.

De två kvarvarande protokollen var det fungerande CIMD-2 protokollet i namnrymden *vodafonetest* och det som inte fungerade i namnrymden *vodafone*. I namnrymden *protocols.vodafone* fanns det en mängd små klasser som båda protokollen utnyttjade. Eftersom protokollet i namnrymden *vodafone* ej fungerade flyttade jag helt enkelt de klasser som cimd-2protokollet behövde till namnrymden *vodafonetest*. I namnrymde *vodafonetest* tog jag också bort klassen CIMD2Auto som var en implementation för att automatiskt ta emot sms. Anledningen till att denna togs bort var att dess funktionalitet ej var testad tidigare.

Slutligen döpte jag om namnrymden *vodafonetest* till *cimd2* vilket jag upplevde som ett mer semantiskt korrekt namn. Anledningen till namnbytet är att protokollet i sig inte är specifikt för operatören Vodafone utan generellt för alla operatörer som använder CIMD-2 protokollet för att hantera smstrafik. I Sverige är det dock endast Vodafone som använder detta protokoll.

Slutligen lade jag till en undernamnrymd i *cimd2* med namnet *utils*. Här lade jag klassen Utils och ett antal klasser som endast implementerade konstanter och makron. Strukturen hos den porterade SMS Notifier ses i punktlistan 7.1.3 på nästa sida där en pil upp indikerar namnrymder och en högerpil indikerar en klass.

△ ***protocols***

- ▷ **BooleanMutex**
- ▷ **SmsException**
- ▷ **SmsProtocolInterface**
- ▷ **SmsProtocolClientInterface**

△ ***cimd2*** - Namnrymderna *vodafone* och *vodafonetest* ihopslagna till en namnrymd. Klassen CIMD2Auto har tagits bort p.g.a att den ej fungerar.

- ▷ **CIMD2**
- ▷ **CIMD2Functions**
- ▷ **CIMDMsg**
- ▷ **Parameter**
- ▷ **CIM2DSmscStatusReport** - Flyttades från namnrymden *vodafone*.
- ▷ **CIMDHeader** - Flyttades från namnrymden *vodafone*.
- ▷ **CIMDData** - Flyttades från namnrymden *vodafone*.
- ▷ **CIMDTrailer** - Flyttades från namnrymden *vodafone*.

△ ***utils*** - Ny namnrymd.

- ▷ **CIMD20Constants** - Flyttades från namnrymden *vodafonetest*.
- ▷ **ErrorCodes** - Flyttades från namnrymden *vodafone*.
- ▷ **OperationNumbers** - Flyttades från namnrymden *vodafone*.
- ▷ **Utils** - Flyttades från namnrymden *vodafone*.

7.1.4 Dispatchernamnrymden *smsnotdispatcher*

I namnrymden *smsnotdispatcher* fann jag inga namnrymder som var onödiga. Istället lade jag till en egen namnrymd *events*. Detta beror på att då jag porterat koden blev jag tvungen att implementera händelsehanteringen i C# manuellt. Detta beskrivs närmare i nästkommande kapitel.

I namnrymden *smsnotdispatcher* fanns det en klass som inte användes. Detta var SmsNotDispatcherApiProperties. Denna klass hämtade inställningar åt klassen SmsNot-DispatcherApi. Men denna klass har inte använts på länge utan alla klasser hämtar nu inställningar från instansen av klassen SmsNotifierSysConfig. Av denna anledning tog jag bort klassen SmsNotDispatcherApiProperties ur namnrymden.

△ ***smsnotdispatcher*** : Huvudnamnrymd för sändenheten

- ▷ **SmsNotDispatcherApi**
- ▷ **SmsNotDispatcherApiSendDispatcher**
- ▷ **SmsClient**
- ▷ **SmsNotDispatcherApiSendQue**
- ▷ **SmsNotDispatcherApiStatisticMessage**
- ▷ **SmsNotDispatcherApiSmsStatusReportQue**
- ▷ **SmsNotDispatcherApiStatisticWriter**
- ▷ **SmsNotDispatcherApiAlarm**
- ▷ **SmsNotDispatcherApiProperties**den nya versionen av SMS Notifier.
- ▷ **ServerStatus**
- ▷ **SmsNotDispatcherQueInterface**
- ▷ **BooleanSyncObject**
- ▷ **SmsNotDispatcherApiInterface**

- ▷ **Status**
- ▷ **InternalErrorNotDispSmsException**
- ▷ **IllegalProtocolSmsException**

△ ***events*** - Huvudnamnrymd för händelserna i SMS Notifier.

- ▷ **AlarmChanged** - Händelsen av att ett alarm ändras, läggs till eller tas bort.
- ▷ **SmsReceived** - Händelsen att ett sms har mottagits.
- ▷ **StatusReportReceived** - Händelsen att en statusrapport har mottagits.
- ▷ **ClientStatusChanged** - Händelsen att klienttråden som använder protokollet har ändrat status, t.ex. från uppkopplad till nedkopplad etc.
- ▷ **ServerStatusChanged** - Händelsen att servern ändrat status från t.ex. nedkopplad till uppkopplad.

△ ***que***

- ▷ **Que**
- ▷ **CompoundQueObject**
- ▷ **QueException**

△ ***supervisory***

- ▷ **Client**
- ▷ **SupervisoryClient**
- ▷ **SupervisoryCommand**
- ▷ **SupervisoryCommandListener**

△ ***thread***

- ▷ **ThreadParameters**
- ▷ **TimeStampedThread**

7.2 Förslag till ytterligare ändringar

För att göra strukturen mer logisk och modulariserad rekommenderar jag att moduluppdelningen från figur 5.1 på sidan 17 avspeglas i SMS Notifiers namnrymdsstruktur. Med detta menar jag att de klasser som hanterar t.ex. sändenheten lyfts ut för att utgöra sin egen namnrymd, se figur 6.4 på sidan 30. Detta skulle göra det lättare att förstå SMS Notifiers moduluppbyggnad.

Ett annat förslag på en förbättring är att API-modulen mellan SMS Notifiers kärnfunktioner och utomstående applikationer implementeras som en separat klass. Detta skulle underlätta för SMS Notifiers utvecklare att lättare och säkrare öppna rätt funktioner in till SMS Notifier. På detta vis skulle en stark restriktion finnas mot att utvecklare av andra produkter skall använda metoder hos SMS Notifier som de ej behöver eller skall ha tillgång till.

8 Manuell implementation vid porteringen

I detta kapitel nämns de problem som uppstod vid den automatiska porteringen av SMS Notifier. Dessutom beskrivs de lösningar som togs fram för att lösa problemen.

När Microsoft Java Language Conversion Assistant hade konverterat koden visades ett dokument som beskrev hur många filer som hade konverterats från Javakod till motsvarande C#-kod, vilka fel som uppstått o.s.v. Dokumentet beskrev också vad som *inte* kunde porteras. JLCA hade inte hittat några motsvarigheter hos C# till Javas gränssnitt PropertyChangeListener eller klasserna PropertyChangeSupport, PropertyChangeEvent, SimpleDateFormat eller XML-hanteraren jdom[6]. Alla klasser utom jdom ingår i Javas standardklassAPI. Nedan beskrivs klasserna och problemen mer ingående.

8.1 PropertyChange

Klasserna PropertyChangeSupport och PropertyChangeEvent samt gränssnittet PropertyChangeListener är en del av Javas händelsehanteringssystem. Händelsestödet fungerar så att då tillståndet hos ett objekt ändras kan objektet sända ut en s.k. händelse. De klasser som lyssnar efter händelsen får information om att tillståndet förändrats och kan på detta sätt hålla sig uppdaterade. Detta är ett mycket smidigare system än att t.ex. ha en meddelandekö där varje tråd som ändrar tillstånd lämnar ett meddelande om detta i kön och övriga trådar kontinuerligt måste undersöka meddelandekön om något har hänt.

I Java tar alla klasser som lyssnar efter en händelse emot *alla* händelser. En händelse kan t.ex. vara att ett objekt har ändrat en boolesk variabel *isConnected* från *true* till *false*. Därefter jämförs händelsen(d.v.s variabelnamnet isConnected) med händelsen som klassen i själva verket är intresserad av. Om händelserna stämmer överens använder klassen den mottagna händelsen och tar emot det nya värdet på händelsen. I C# lyssnar inte objekt efter alla händelser. Programspråket kräver istället att man måste specificera *vilken* händelse ett objekt skall sända och/eller ta emot. Alltså var jag tvungen att själv implementera de

händelser som skulle användas i form av klasser. Detta var anledningen till att jag valde just namnet events för namnrymden, här sparade jag de händelser som jag implementerade i form av klasser till SMS Notifier.

```
// The event to be published by
// class ServerStatus and subscribed to by other classes
public static event ServerStatusChangedEvent onServerStatusChange;

// The delegate to pass the event
public delegate void ServerStatusChangedEvent(
                object getInstance, ServerStatusChanged newServerStatus);
```

I C# specificerar programmeraren en händelse själv. I kodexemplet ovan ser vi att händelsen onServerStatusChanged skapas av nyckelordet *event*. onServerStatusChanged är den händelse att serverdelen, den del av SMS Notifier som tar emot sms från användaren och sänder dessa till operatören, byter tillstånd. Tillståndsbytet kan vara från t.ex. "nedkopplad" till "uppkopplad". Händelsen onServerStatusChanged tilldelade jag typen ServerStatusChangedEvent som jag själv definierat i samma uttryck som jag skapat händelsen onServerStatusChange, se ovan. Efter att ha definierat händelsen måste en s.k. *delegate* skapas. Delegaten är den metod som måste implementeras av alla objekt som lyssnar efter den specifika händelse som delegaten refererar till. Delegaten döps till samma typ händelsen är av. Dessutom är delegatens parametrar de parametrar som följer med händelsen. Den första parametern är objektet som sänder händelsen och den andra är den klass som representerar händelsen som inträffat. Händelseklasserna är de klasser jag placerat i namnrymden *smsnotifier.smsnotdispatcher.events*. Delegatens signatur är ej fast utan godtycklig. Dock säger Microsofts kodningskonvention att delegaten *bör* se ut som ovan.

```
// Tell subscribers that the server is initializing
```

```
public static void  initializing()
{
  // Set the server status
  serverStatus = 3;

  // Check that there are subscribers (listeners) and
  // fire the event notifying other classes
  if(onServerStatusChange != null)
  {
    onServerStatusChange(null, new ServerStatusChanged(serverStatus));
  }
}
```

Efter att händelsen och delegaten definierats var jag tvungen att sända händelsen där den behövde skickas. I händelsen ovan ser vi att servern är i initieringsläge. Först sätts den lokala variabeln *serverStatus* till det nya värdet. Därefter undersökte jag om det fanns några objekt som lyssnar efter denna typ av händelse med villlkoret *onServerStatusChange != null*. Därefter sänds händelsen genom att händelsens namn skrivs följt av de parametrar som definierats i delegaten, som i ett funktionsanrop. I de flesta fall sände jag ej med det objekt som sänt händelse. Detta helt enkelt för att objektet användes inte av mottagarobjektet.

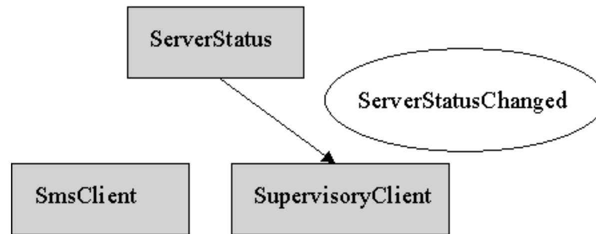
```
// Subscribes this class for the onServerStatusChange-event
public void Subscribe(ServerStatus newServerStatus)
{
  // Store event in queue
  ServerStatus.onServerStatusChange += new
  // Invoke method ServerStatusChangedHandler through the delegate
  ServerStatus.ServerStatusChangedEvent(ServerStatusChangedHandler);
```

```
}

// Handles the event onServerStatusChange
public void ServerStatusChangedHandler(object server,
                              ServerStatusChanged newServerStatus)
{
  // Set the serverStatus of 'this' object
  this.serverStatus = ServerStatusChanged.serverStatus;
}
```

Efter att händelsen har sänts så måste händelsen tas emot. Detta gjorde jag genom att i mottagarklassen definiera en "prenumeration" på händelser från den klass jag förväntade mig skulle sända den typ av händelser jag är intresserad av. Att starta en prenumeration på en händelse gjorde jag genom att implementera metoden *Subscribe*. I metodens signatur angavs som formell parameter ett objekt av den klass som förväntades sända händelsen, i detta fallet ServerStatus-klassen. Om en sådan händelse inkommer så läggs händelsen till en kö för att bearbetas av metoden ServerStatusChangedHandler. Denna metod har ett godtyckligt namn och bearbetar de data som händelsen genererat. Här vidtas alltså de åtgärder som tar hand om resultaten då denna händelse uppstått. I detta fall är det klientdelen av SMS Notifier som lyssnar efter händelsen att servern bytt tillstånd.

I figur 8.1 nedan illustreras hur ett tillstånd ServerStatusChanged har ändrats i objektet av klassen ServerStatus. Instansen av klassen SupervisoryClient är en registrerad lyssnare efter händelser från klassen ServerStatus och tar då emot händelsen när den sänds från avsändarklassen. Däremot lyssnar inte instansen av klassen SmsClient efter händelser av denna typ och känner således inte ens till att händelsen kastats. I Java hade bägge objekten i figur 8.1 tagit emot data, undersökt vilken händelse det är som kastats för att därefter bestämma om objektet skall bearbeta händelsen eller ej.



Figur 8.1: En händelseavsändare och lyssnare.

De klasser som implementerade händelsehanteringssystemet var ServerStatus, Client, SupervisoryClient, SmsNotDispatcherApi samt Alarm.

8.2 SimpleDateFormat

SimpleDateFormat i Java skapar m.h.a en textsträng en mall för hur ett datum skall formateras. T.ex. ger strängen “YY:mm:dd” ett tvåsiffrigt årtal och datum. Denna klass hade kodporteringsverktygen JLCA inte någon motsvarighet till, men detta problem fann jag snart en lösning på med klassen DateTime.

```
// Get cultureinfo for sweden
CultureInfo sdf = new CultureInfo(‘‘sv-SE’’);
// Get current time and date
System.DateTime date = System.DateTime.Now;

try
{
// Set the formattingpattern for the datetimestructure
sdf.DateTimeFormat.FullDateTimePattern = "yyMMddHHmmss";
// Format the current date accordingly
date = System.DateTime.ParseExact(System.DateTime.Now.ToString(),
                             "yyMMddHHmmss", sdf.DateTimeFormat);
```

```
}
```

Jag skapade ett objekt av klassen *System.Globalization.CultureInfo*. Klassen innehåller bl.a. fält där information som t.ex. vilken kalender eller vilket nummersystem som används i en viss kultur sparas. Vid instansieringen hämtades de kulturinställningar som finns förinställda för Sverige i Microsofts .NET-ramverk. Därefter instansierade jag ett objekt av klassen DateTime där datum och tid lagras och hämtas. I klassen CultureInfo använde jag ett fält kallat DateTimeFormat som beskriver hur tiden presenteras i den aktuella kulturen. Jag använde formateringssträngen "yyMMddHHmmss" som beskrev den formatering jag behövde på tiden. Strängen lagrades i fältet DateTimeFormat.FullDateTimePattern i CultureInfo-objektet. Till slut använde jag metoden *ParseExact* i klassen DateTime för att formatera ett inbyggt fält i C#, nämligen tidsfältet*System.DateTime.Now*. Detta fält ger kontinuerligt den aktuella tiden i upplösningen tusendelar av en sekund(i C# kallade ticks). Tillsammans gav den aktuella tiden, fältet sdf.DateTimeFormat samt strängen "yyMMddHHmmss" den tid jag behövde representerat på det sätt jag behövde.

För att formatera tiden på rätt sätt använde jag nu metoden ParseExact i DateTimeklassen. Som parametrar angav jag den exakta tiden för tillfället i strängformat, strängen med vilken tiden skulle formateras samt det objekt av CultureInfo jag skapat.

Den klass som hade detta problem var SmsNotDisptatcherStatisticWriter.

8.3 jdom

I den nuvarande versionen av SMS Notifier används XML-filer för konfiguration. I den färdiga produkten skall en databas användas för att lagra inställningar. XML-hanteringen skall ändå finnas kvar för att SMS Notifier skall kunna lagra inkomna smsmeddelanden som externa applikationer kan läsa. För att hantera XML-filer användes i C# istället för jdom en intern XML-hanterare. De felaktigheter som rättades till från porteringen var bl.a. att objekten av klassen SAXBuilder i Java byttes ut mot XmlValidatingReader i C#. Dessa två klasser implementerade XML-läsarobjekt som var lika bortsett från vissa metodnamn.

Efter att ha ändrat dessa metod- och konstruktornamn kompilerades klassen utan fel.

Den klass som hade detta problem var SmsNotifierSysConfig.

9 Resultat och rekommendationer

Nedan följer en redogörelse för de resultat som nåddes i detta examensarbete. Det primära målet att portera SMS Notifier uppfylldes. Resultatredogörelsen följs av rekommendationer menat till de som skall utföra en liknande uppgift.

9.1 Huvudresultat

9.1.1 Kravspecifikation

Att ta fram en kravspecifikation gick bra. Detta tog dock något längre tid än jag trodde p.g.a att varken jag eller Lars Öhrwall på Devo IT kunde särskilt mycket om SMS Notifier. Vi utgick från den ursprungliga SMS Notifier och analyserade egenskaperna hos denna version. Därefter diskuterade vi om de lösningar som fanns i den nuvarande SMS Notifier var bra eller ej. Om inte, vilka krav behövde isåfall ändras, tas bort eller läggas till? Om någon av oss hade haft mer erfarenhet av SMS Notifier hade frågorna kring dessa detaljer kunnat bli besvarade snabbare.

9.1.2 Dokumentation

För att förstå och kunna dokumentera produkten använde jag Rational Rose Enterprise Edition samt den Java-kod som fanns tillgänglig. Det var väldigt svårt att förstå kodens innebörd då det fanns väldigt lite kommentarer i koden. Dokumentationen gick ändå ganska bra och förhållandevis snabbt att göra. Speciellt med lite hjälp från personalen på Devo IT.

9.1.3 Porteringen

Porteringen gick sämre än jag väntat mig. Detta på grund av att jag, som tidigare nämnt, porterade fel kod i början av examensarbetet. Dessutom tog det lång tid innan jag hittade

porteringsverktyget att portera kod med. Avslutningsvis lyckades jag ändå att portera all kod. Koden har dock ej testats utan endast kompilerats utan felmeddelanden.

9.1.4 Webgränssnitt för test

Detta mål hann jag ej med. Den tid det tagit att portera koden gjorde att jag ej hann börja skapa ett gränssnitt för att testa den porterade koden.

9.2 Rekommendationer

9.2.1 Förstå produkten

För de personer som skall utföra en liknande uppgift rekommenderar jag att, om möjlighet finnes, läsa dokumentationen. Försök att förstå vad koden gör och vilka klasser/filer som hör ihop för att utföra en uppgift. Detta gör det framförallt lättare om problem uppstår.

9.2.2 Undersökning

Undersök om det finns någon som gjort något liknande tidigare. Vad har isåfall denne haft för förutsättningar jämfört med dig som nu skall göra detta? Finns det några verktyg för att göra arbetet lättare? Kan verktyget göra en liten del, en stor del eller hela jobbet? När verktyget är klart, kan några förbättringar göras?

10 Summering av projektet

Av projektet har jag framförallt lärt mig meningen av att utarbeta en arbetsplan samt värdet av en bra dokumentation. Speciellt då en uppgift gäller ett stort projekt som detta. Välkommenterad kod och en välskriven dokumentation kan göra att instuderingstiden väsentligt förkortas och mer tid kan då istället läggas på att nå övriga mål.

Jag har även fått goda kunskaper kring projekthantering. Hur en idé blir realiserad på en verklig arbetsplats. Med detta avser jag moment som att hur företag hanterar revisioner av kod och dokumentation t.ex. Även saker som att behandla produkten ur t.ex. användarperspektiv och utvecklarperspektiv har jag förstått nyttan av. Dessutom har jag förutom att ha vant mig vid utvecklingsmiljön Microsoft Visual Studio.Net och programspråket C# även fått mer avancerade kunskaper i programspråket Java. Jag har fått en djupare förståelse för t.ex. programmeringsmässiga termer och avancerad programmering, både i språken Java och C# men även generellt.

Det svåraste med projektet var att sätta sig in i den, i princip, odokumenterade koden. Detta i sin tur försvårade det manuella porteringsarbetet. Det lättaste med arbetet var att förstå C#. Då det liknar Java i mångt och mycket var det inte så svårt att förstå hur språket var uppbyggt.

Arbetet har, tillsammans med rapportskrivningen, ganska precis tagit de 10 veckor som har varit den tid som funnits att tillgå.

Om jag fick göra om arbetet skulle jag framförallt leta efter ett porteringsverktyg från början. Verktyget klarar inte riktigt allt, men kan ändå ge en mycket bra grund att fortsätta porteringen ifrån. Jag skulle även undersöka om det fanns någon som gjort något liknande och publicerat detta. De saker som hjälpt mest under porteringens gång har varit de dokumentationer för programspråken Java[2] och C#[3] som finns tillgängliga. Dessa ger tillsammans med den gamla koden en bra hjälp vid portering.

Referenser

[1] CIMD_Interface_Specification.pdf, 20040420, http://nds1.forum.nokia.com/nnds/ForumDownloadServlet?id=4418&name=CIMD%5FInterface%5FSpecification%2Epdf

[2] Java JDK 1.4.2 API, 20040510, http://java.sun.com/j2se/1.4.2/docs/api

[3] Microsoft.NET Framework v.1.1, 20040510, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp

[4] WinCVS, 20040215, http://www.wincvs.org

[5] Microsoft Java Language Conversion Assistant, 20040325, http://www.msdn.microsoft.com/vstudio/downloads/tools/jlca

[6] XML-API, 20040430, http://www.jdom.org/docs/apidocs/index.html

A Delmål till den färdiga SMS Notifier

Nedan är en kompletterande lista över de delmål som måste nås för den färdiga C#-applikationen av SMS Notifier. Dessa delmål kommer inte att behandlas i denna rapport.

1. **Utveckla ett API för databashantering och åtkomst.**
   API(Application Programmers Interface) för databasen kommer att tillhandahålla de metoder som SMS Notifier använder för att hantera och bearbeta de data som finns i databasen. Denna del behövs för att SMS Notifier skall kunna lagra telefonnummer, namn, inställningar etc för användaren.

2. **Skapa ett API mot SMS Notifiers funktioner.**
   API är gränssnittet utåt från SMS Notifier. Det tillhandahåller de funktioner som utvecklare och användargränssnitt behöver för att använda SMS Notifiers funktioner. API hanterar autentisering, sändning av sms och återkoppling av sändresultat och inkommande sms.

3. **Konstruera en management-del.**
   I SMS Notifier är det tänkt att varje kund har ett konto och varje konto kan ha flera användare. Vad managementdelen gör är att den ställer in SMS Notifier på det sätt som kunden vill ha den. Det kan handla om maxgräns för sms, vad som kunden vill spara(logga) i databasen etc. Detta är den primära administratörsfunktionen hos SMS Notifier. Management-delen ger även användaren möjlighet att konfigurera sitt konto och göra egna inställningar. Här anger användaren sina kontakter o.s.v.

4. **Skapa användargränssnitt.**
   Användargränssnittet kommer att vara ett webgränssnitt. Detta användargränssnitt kommer att vara mer inriktat på användarvänlighet än det webgränssnitt som används under testfasen.