2006:045 HIP

EXAMENSARBETE

GSM-R Probe

En flexibel lösning för övervakning till GSM-R

Fredrik Ahlbäck
Edward Domeij

Luleå tekniska universitet

Högskoleingenjörsprogrammet
Datateknik
Institutionen för LTU Skellefteå
Avdelningen för Mobila nätverk och programutveckling

2006:045 HIP - ISSN: 1404-5494 - ISRN: LTU-HIP-EX--06/045--SE

GSM-R Probe

En flexibel lösning för övervakning till GSM-R

Edward Domeij & Fredrik Ahlbäck

10p Examensarbete i Datateknik 2006

Förord

Syfte med rapporten

Med denna rapport ämnar vi redovisa våra framsteg inom ämnet övervakningsprobe för GSM-R [1]nätet. I rapporten kommer vi att beskriva vissa grunder inom GSM-R och andra teknologier som vi använt oss av samt redovisa en möjlig design på en övervakningsprobe för sagda nät. Proben har vi designat utifrån de kunskaper vi samlat på oss inom ämnet.

Erkännanden, handledare

Vi vill framföra ett tack till följande personer som hjälpt oss genom vår utbildning och vårt examensarbete.

Tack till alla på Data Ductus och Pertexo Systems, framförallt vår handledare Stefan Lundström, Magnus Karlsson och Jesper Berglund på Pertexo. Vi vill även tacka Jonas Umander och Mathias Lundgren, Data Ductus. Era synpunkter har hjälpt oss mycket och resulterat i många bra idéer.

Vi vill även passa på att tacka vår examinator Staffan Nilsson och all annan personal på Luleå Tekniska universitet i Skellefteå.

Tack även till Robert Wedin och Mats Andersson, för de gånger ni hjälpt oss och den konstruktiva kritik ni givit oss under arbetes gång.

Sist men inte minst ett stort tack till våra nära och kära.

Abstract

This report is carried out to bring some light in surveillance of GSM-R technology. The report will introduce some of the basics in GSM-R and other technologies that we have used. Later on there will be a description of our Surveillance Probe and our conclusions.

The Primary purpose is to find a way to monitor how the net behaves under load and in real scenarios, can trains that communicate through GSM-R always be reached, and does the network work within its expected boundaries? The surveillance Probe is a way to verify that the security standards have been met.

Our working progress has been to study the GSM-R specification, a test phone from Triorail and with the information gathered bring up a design proposal to a test-probe. The conclusion we drew were that a dynamical probe that gives Test cases access to all the probe resources GPS, AT commands and Trace messages with a possibility to load test cases over internet is the solution to a versatile probe that can monitor the net from many different perspectives.

The server should be able to monitor, control the probe and display geographic information from the test cases. A tool for displaying information from the logs will also be needed. We have chosen to implement the server as a module in Fenix, Fenix is a framework written by Pertexo System. We have also designed two test cases to demonstrate the probes capabilities and the way the probe should be operated.

Sammanfattning

Många av de europeiska järnvägsbolagen håller på eller har gått över till att använda GSM-R teknologi för kontakt med sina tåg och personal. Förut användes enklare sätt för att hålla kontakt med tåg, till exempel sändare nedgrävda längs rälsen och visuella tavlor. På det viset kunde tågen positionsbestämmas och föraren kunde kontaktas.

När ett modernare system skulle byggas kom man fram till att GSM-R standarden skulle konstrueras. Vad innebär egentligen GSM-R standarden? Jo, det är en utbyggnad av GSM-nätet och opererar på frekvenserna 876 - 880 MHz och 921 - 925 MHz. Säkerhet har varit en viktig hörnsten när nätet konstruerats, täckningen ska ligga på hela 99.5 %. Dessutom finns det väldigt många krav på utrustningen, ett SMS måste exempelvis vara framme på 30 sekunder. Utöver detta har fler funktioner lagts till, bland annat flerpartssamtal och möjligheten att ringa upp olika funktioner istället för olika telefoner så att man kan ringa alla banarbetare längs en specifik räls eller all tågpersonal på ett specifikt tåg.

All denna säkerhet är bra, men hur kan man förebygga risken att ett tåg tappar täckningen? Hur ska testen gå till väga för att man ska kunna säkerställa möjligheten att ringa från tågen i en verklig situation? Detta är den frågeställning och de problemområden som vi fick på vårt bord att designa en lösning till.

GSM-nätet har många olika testbäddar för att undersöka sändningsstyrka, QoS-meddelanden mm. Däremot anses GSM-R marknaden vara för liten för att de större operatörerna ska ta fram en specifik testbädd som kan kontrollera alla extra funktioner och säkerhetsparametrar specifikt för GSM-R.

Vår design är en mångsidig testprobe, med en GSM-R mobil och en GPS inkopplad, vilken kan övervaka proben och testa parametrar i det luftburna interfacet samt testa nätets stabilitet. För att implementera mångsidighet har vi valt att dra ned intelligensen i proben och lägga över den till olika testfall. Dessa kan i sig få tillgång till stora delar av probens resurser och styras över en vanlig internetuppkoppling.

Vi kommer att presentera testfallsdesign, probens design, serverns användning samt presentera resultaten från våra testkörningar. Förbättringar och förändringar är en viktig del i vår rapport eftersom vi med dessa vill hjälpa våra efterföljare att undvika fallgropar.

Innehållsförteckning

Introduktion

Som examensarbete i vår utbildning Datateknik i Skellefteå har vi valt att göra en övervaknings-probe i samarbete med Data Ductus.

Proben är ett sätt att övervaka och kontrollera att GSM-R nätet fungerar som det ska samt lokalisera problemkällor som exempelvis där täckningen är dålig.

Problemställning

Hur kan man veta att ett tåg både kan kontakta och kontaktas av andra längs hela rälssträckningen? Hur kan vissa kritiska delar i GSM-R nätet kontrolleras som exempelvis vid handover? Detta är den frågeställning och de problem som vårt examensarbete ska lösa.

Bakgrund

Bakgrunden till arbetet är att inget befintligt testverktyg för GSM-R näten existerar. För det traditionella GSM-nätet finns det många olika testplattformer, men som marknaden ser ut idag anser många av de större företagen, bl.a. Ericsson och Siemens, att GSM-R marknaden är för liten för att utveckla ett separat verktyg för att testa nätet. Däremot kan det vara en attraktiv marknad att ta sig in på för en mindre aktör som Data Ductus. Därefter behövs det ett designförslag på en testprobe som är tillräckligt mångsidig för att kunna övervaka hela kedjan. Det räcker inte med att bara kunna ta ut QoS-meddelanden, utan riktiga tester bör utföras i en verklighetstrogen miljö. Mångsidighet är av stor vikt för att samma probe ska kunna utföra många olika tester oberoende av vem användaren är. Nätoperatören vill kontrollera QoS-meddelanden, säkerhetsansvarige vill kontrollera att tågen verkligen kan nå varandra.

Syfte med arbetet

Syftet med arbetet är att kontrollera och testa funktionaliteten i GSM-R nätet. Funktioner som borde testas är att handover går bra, hur täckningen är och loggning av positioner så man kan få reda på var täckning är dålig. Det ska också erbjudas ett sätt att konstruera testfall så att probens beteende kan modifieras efter behov. På så sätt ska en tågresa kunna göras ännu säkrare genom att verifiera att ett tåg kan nås längs hela järnvägssträckan.

Proben ska vara placerad på ett tåg där den ska mäta och testa nätet under färd. Vid stationer skickas data till huvudservern via WLAN.

Förkortningar

API:	Application programming interface.
AT:	Attention.
CVS:	Concurrent Versions System.
ETSI:	European Telecommunications Standards Institute
DOM:	Document Object Model.
GMT:	Greenwich Mean Time.
GPS:	Global Positioning System.
GSM-R:	GSM-Railway.
GSM:	Global System for Mobile Communications.
NMEA:	National Marine Electronics Association.
JDK:	Java Development Kit.
QoS:	Quality of service.
SAX:	Simple API for XML.
UID:	Unique Identifier.
WLAN:	wireless local area network.
XML:	Extensible Markup Language.

Genomförande

Metoder arbetssätt

Programmeringsspråk

All utveckling har skett i Java[2]. Proben är skriven i jdk 1.4 och servern i jdk 1.5. Till en början var vi osäkra på om Java eller C++ skulle användas som primärt språk. Det som gjorde oss osäkra var integration med hårdvara eftersom det finns betydligt bättre stöd för det i C++. Java hade smidigare stöd för dynamisk laddning av data och Fenixramverket är skrivet i Java vilket resulterade i att vi beslutade oss för att skriva proben i Java. Vi undersökte även om serieportskommunikation skulle skrivas i C++, efter lite huvudbry hittade vi ett bibliotek i java för serieportskommunikation som vi beslöt oss för att använda.

Använda bibliotek

Alla bibliotek vi använt under utvecklingen presenteras här. De kriterier vi ansåg viktiga var att det skulle vara open source samt att det gärna skulle vara från kända tillverkare. Detta på grund av vi ville, i den mån det är möjligt, garantera att biblioteken inte läggs ner, åtminstone inte i en närstående framtid.

Seriekommunikation

All kommunikation med hårdvara sker i Java med biblioteket *Java Communications API*[3]*,* detta API är sedan jdk 1.3 bortaget från Java-standarden eftersom det gör koden mindre plattformsoberoende. Däremot finns det både till Linuxplattformer och till Windowsplattformer, proben körs i detta stadium av utvecklingen i Windows eller i en Linuxmiljö. Med det i bagaget beslutade vi oss för att använda detta bibliotek trots de klara nackdelarna av minskat plattformsoberoende.

Loggning

Loggningen sker genom *Log4J*[4] vilket är ett open source loggningbibliotek från Apache. Orsaken till att vi valde detta bibliotek för loggning är att det är ett trådsäkert och väl inarbetat bibliotek. Trådsäkringen gör även att det går bra att använda inne i testfallen, på detta vis blir testning och uppföljning av testfallen enklare.

XML

För all hantering av XML[5] används biblioteket Dom4J[6], även detta från Apache. Orsaken var att det är ett enkelt och reducerat bibliotek för XML-hantering. Dom4J är en så kallad DOM[7] parser vilket innebär att den lagrar hela XML-dokumentet som en trädstruktur i minnet.

Databas

För all databashantering i servern används *Hibernate* [8]och databashanteraren är MySQL[9]. Hibernate är ett Objekt relationsbibliotek vilket innebär att en klass som mappats mot hibernate indirekt får en koppling mot en tabell i en databas. Detta betyder att om man förändrar medlemsvariabler i en klass så sparas dom automatiskt in i databasen.

Fenix

Fenix är ett applikationsramverk från Pertexo Systems. Grundtanken med Fenix är att många av grundfunktionaliteterna redan finns i ramverket så som karthantering, larmhantering, användarhantering och domänhantering. Moduler som skrivs till Fenix kan därför tillgodoräkna sig viss funktionalitet som är liknande för många olika övervakningssystem. Orsaken till att vi skrev servern som en modul i Fenix är dels för att spara tid, eftersom mycket av grundfunktionaliteten som redan finns i Fenix är ytterst tidskrävande, samt p.g.a. att det var ett önskemål från vara uppdragsgivare på Data Ductus.

Testtelefon

Mobiltelefonen vi använt är en Testtelefon från TrioRail[10], telefonen är en ombyggd Siemenstelefon och fungerar som GSM-R telefon. Den är även utbyggd med ett speciellt Trace-interface, det vill säga att vi kan hämta ut spårmeddelanden från GSM-R telefonen. Genom detta kan vi avkoda QoS-meddelanden, Layer 1 informationsmeddelanden mm.

GPS

Den GPS vi använt oss av är en RM-500, en standard-GPS som sänder ut NMEA-meddelanden varje sekund.

Kodhantering

För att undvika missöden med kodförlust och för att ha möjligheten att göra en rollback vid behov har vi använt CVS.

GSM-R

GSM-R är baserat på GSM-standaren men har utvecklas för att passa järnvägsnätets speciella behov.

I Europa har ETSI[11] reserverat frekvenserna 876 – 880 och 921 – 925 för GSM-R, men alla mobiler för GSM-R nätet ska kunna användas i frekvenserna 876-915 och 921-960. Det inkluderar de frekvenser som är reserverade för GSM-R och de som är reserverade för GSM-nätet. GSM-R ska fungera på tåg som kör 0-500 km/h.

GSM-R har prioritet på samtal där högre prioritet har rätt att bryta samtal med lägre prioritet. Ett samtal som automatiskt får hög prioritet är ett nödsamtal. Om ett sådant samtal sker ska en färddator, liknande de svarta lådorna som flygplan är utrustade

med, börja spela in samtalet. Det enda sättet att bryta ett nödsamtal är om källan väljer att avbryta, om övervakningscentralen avbryter samtalet eller om nätet konstaterat att ingen har talat under en viss tid.
Ett GSM-R nödsamtal ska ta mindre än 2s att koppla upp, den tiden ska hållas i 95 % av fallen och får inte ta längre än 1½ ggr av den tiden i 99 % av fallen.

GSM-R använder sig av funktionell adressering samt via ”location dependent adressing” vilket innebär att man kan ringa till vissa platser. Funktionell adressering innebär adressering till, ex tåg, motorer eller vagnsnummer. När någon ringer inom GSM-R nätet ska den funktionella adresseringen presenteras på ett lättförståligt sätt t.ex. ”Förare av tåg 3241” i stället för bara ”3241”.

Arbetssätt

Den första tiden ägnade vi åt att inhämta kunskap om GSM-R teknologin och att bearbeta problemområdet och frågeställningen. Vi testade även olika moment som vi visste skulle ingå i designen så som serieportsläsning, klassladdning och laddning av dll:er. Därefter påbörjade vi vår design och presenterade den för vår handledare. När alla parter var nöjda med designen skrev vi vår probe och server och utförde kontinuerliga tester för att befästa designen och om möjligt kunna förutse olika problem och svårigheter.

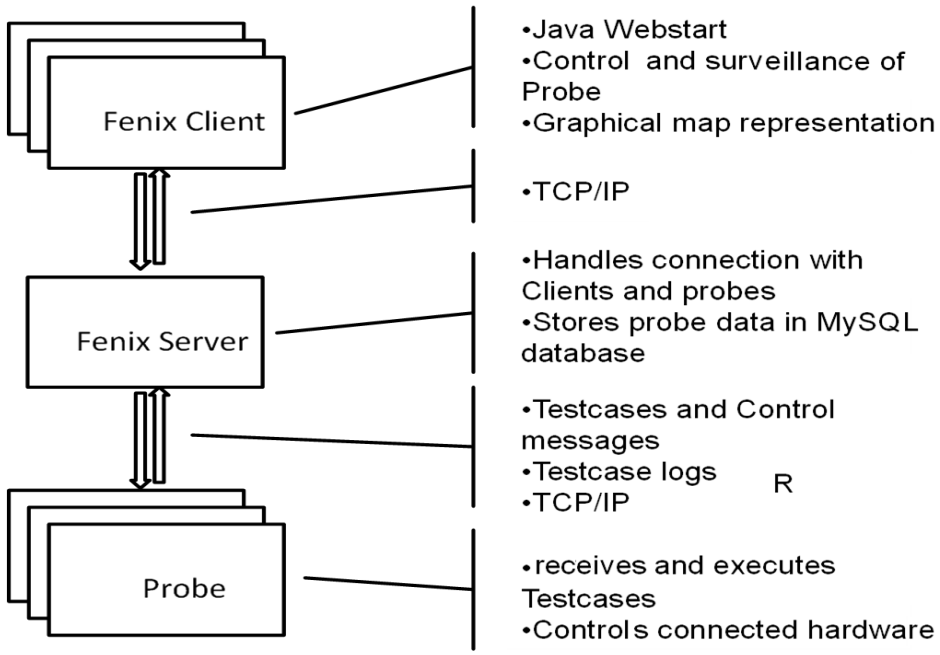
Efter att ha skrivit proben designade vi två exempeltestfall, med hjälp av dom så utfördes mer riktiga tester för att testa probens hela funktionalitet och inte bara de olika byggstenarna var för sig. Däremot utfördes alla våra tester i det vanliga GSM-nätet så vi kunde inte undersöka specifika GSM-R funktioner. Men vi konstaterade att proben beter sig som vi förväntade oss.

Därefter sammanställde vi vår design och diskuterade vad vi hade kunnat göra annorlunda.

Resultat

Proben vi skrivit körs som en java-applikation på en dator med en inkopplad GSM-R telefon och en GPS-mottagare. Fenix delas upp i två delar, en server och en klient, vilka kan köras på olika datorer. Under våra tester har proben körts på en laptop som förutom GSM-R telefon och GPS har haft en WLAN förbindelse. Fenixklienten och servern har körts på en stationär dator med fast uppkoppling. Proben har sedan kommunicerat via WLAN till Fenixservern.

Översiktsdesign



Figur 1, En enkel skiss över systemet och dess dataflöden.

Designen består av tre delar; proben, servern och klienten. Servern och klienten är skrivna som en modul i Fenix, därigenom finns det stöd för java Webstart, domänhantering, stöd för multipla användare samt en kartmotor.

Servern sköter all kommunikation med proben och lagrar all resultatdata från probernas testfall. Servern hanterar även alla Fenixklienter.

Probens huvudsakliga uppgift är att hantera alla inkopplade resurser så som GPS och GSM-R telefon samt hantera alla testfall som är inladdade i minnet.

Probe

Probens huvudklasser är ResourceManager-klasserna, det är dessa som hanterar testfall och inkopplad hårdvara.

När ett testfall skickas till proben tas det emot av en SocketControler och därefter postas det till en ClassLoader som i sin tur laddar in den i proben och placerar testfallet i RMTestCases testfallshantering.

När ett testfall vill använda sig av probens resurser anropas funktioner i ProbeControl. Där kommer begäran att skickas vidare och läggas på kö i RMDevices för att sedan distribueras till motsvarande kontrollerklass som i sin tur skickar kommandon till hårdvara.

När data kommer från hårdvaran kommer ett event att genereras i motsvarande kontrollerklass som sedan postar det till RMTestCases via RMDevices. För att i prioritetsordning dela ut eventen till de testfall som angett intresse för den eventtypen.

För ett överskådligt klassdiagram se bilaga 1.

RMDevices

ResourceManageren hanterar alla kontrollerklasser för inkopplad hårdvara. Alla event som genereras i kontrollerklasserna vidarebefordras till RMTestCases.
Alla ControlEvent hanteras i RMDevices och baseras på informationen i ControlEventet så svarar proben genom att skicka en ping, ladda/ta bort testfall eller skicka resultatdata.

RMTestCases

Hanteringen av alla testfall sköts av RMTestCases. Där säkerställs att alla testfall får alla eventtyper som har begärts. Om mer än ett testfall har begärt samma eventtyp kommer eventet att delas ut till testfallen i prioritetsordning. Upptäcks en senare version av ett testfall hanteras uppdaterandet till den nya versionen här.

Klassladdning

Vid uppstart av proben laddas testfall i form av javaklasser från en katalog i probens lagringsmedia. Finns inga testfall väntar klassladdaren på att Fenix ska ansluta mot proben och skicka testfall. När ett testfall mottagits försöker klassladdaren att ladda detta, först kontrolleras däremot att det verkligen är av rätt typ. Skulle det visa sig vara felaktigt kommer klassladdaren inte att ladda något och ett felmeddelande kommer att loggas till probens loggfil.

Kontroll

Proben svarar på vissa kontrollmeddelanden, dessa meddelanden ska vara av typen ControllerEv. Där controlStr är vilken typ av meddelande det är och rawData innehåller eventuell rådata så som testfall och IP-adress. I skrivandets stund finns det stöd för fyra olika kommandon.

ADD_FILE - Lägger till/laddar om ett testfall bifogat i rawData.
REMOVE_FILE - Tar bort ett testfall från proben.
PROBE_PING - Proben Svarar till ”IP-address:Port” taget från rawData
REQ_RESULT - Proben skickar sin XML-logg från testfallen till ”IP-address:Port” taget från rawData.

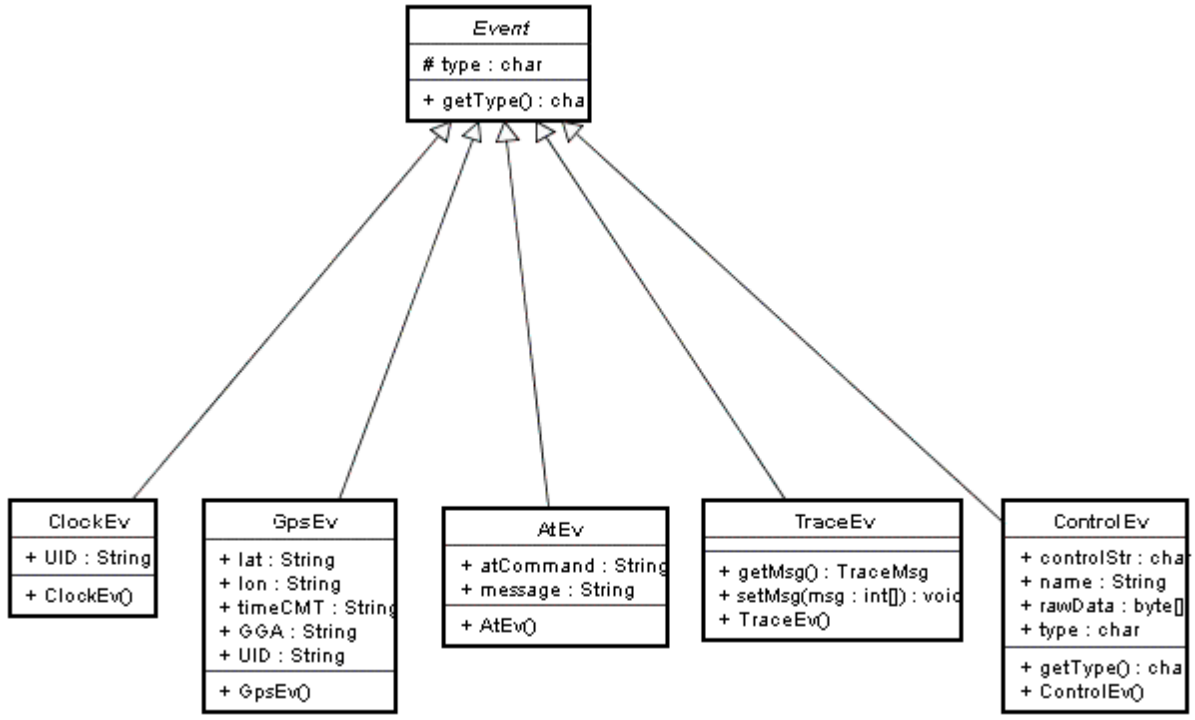
Proben lyssnar på port 4545 efter ControllerEvents och svarar med en PROB_PING efter att ett ControllerEv blivigt mottaget och behandlat.

Eventhantering

När en händelse inträffar kommer kontrollerklassen, som hanterar den hårdvara som händelsen kommer ifrån, att skapa ett event och posta det till RMTestCases via RMDevices. Testfallshanteraren kommer att distribuera eventet till de testfall som angett intresse av eventtypen i prioritetsordning.

Eventtyper

Det finns fem olika typer av event som proben hanterar. Alla dessa event ärver från den abstrakta klassen Event.



Figur 2 Diagram över Eventstrukturen.

AT-Event

Detta event inträffar när telefonen har mottagit och svarat på ett AT-kommando. Attributerna message och atCommand kommer att vara tilldelade data från telefonen och vilket AT- kommando som utförts. Är atCommand blankt innebär det att meddelandet är på telefonens eget initiativ.

GPS-Event

GPS-eventet skickas med en period av 1 s, eventen innehåller data tolkat från NMEA-meddelandet som mottagits av GPS:en. Data som går att få ut från eventet är longitud och latitud i formatet WGS84 samt en tidstämpel i GMT.

CLOCK-Event

Clock-eventet innehåller i sig inte någon mer data än en UID på det testfall som beställde den och fungerar bara som en timer-signal om ett testfall vill få en påminnelse vid en specifik tidpunkt eller i en viss period.

Control-Event

Kontrolleventet skickas med kommandon som styr proben, exempelvis för att lägga till testfall eller att skicka loggar från testfallen till servern.

Trace-Event

Trace-eventet innehåller data från Trace-interfacet på TrioRail-telefonen.

Properties

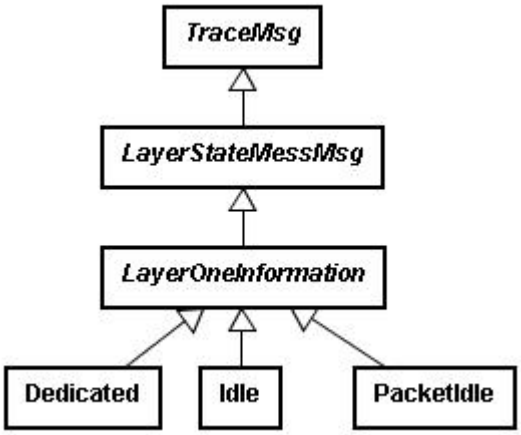
Probens konfiguration finns i en property-fil. Alla nycklar och värden som saknas kommer att skrivas in i MissingProperties.txt och den läggs i Classpathen för proben. Ett exempel på properties-fil finns i bilaga 2.

Trace-avkodning

När man ska avkoda ett trace-meddelande görs detta genom att hela meddelandet skickas till den abstrakta basklassen TraceMsgs *evaluate(int[] msg)* funktion. Evaluate-funktionen kommer att bestämma vilken typ av meddelande det är och anropa den korrekta meddelandetypens funktion *getFormat(int[] msg).* Denna funktion kommer att returnera den typ av meddelande bytearrayen msg beskriver med alla parametrar avkodade.

Exempel

Avkodning av LayerStateMesurment Message. Strukturen för avkodning av detta meddelande ser ut på följande sätt:



Figur 3 Strukturen för avkodning av LayerOneInformation meddelanden. Se Bilaga 3 för komplett struktur av Traceavkodning.

Ett Trace-meddelande kommer från COM-porten, detta matas in i TraceMsg evaluate funktion, där sker en bedömning om vilken typ av meddelande det är. Om det skulle visa sig vara ett ”LayerStateMessurment” meddelande skickas detta in i LayerStateMessMsg, där ytterligare en bedömning av meddelandetypen görs. Om det visar sig vara ett ”LayerOneInformation” skickas meddelandet in motsvarande klass. Där sker en bedömning om det är ett ”Dedicated”, ”Idle” eller ”Packet Idle” meddelande. I dessa klasser sker den riktiga avkodningen av meddelandet. Det bör nämnas att både i TraceMsg och i LayerStateMessMsg finns det flera typer av meddelande som kan avkodas. Om det skulle visa sig vara en annan typ av medlande skickas det in i respektive klass, i dessa sker antingen en ny bedömning eller så avkodas meddelandet direkt. När avkodningen är klar returnerar den klassen som avkodat meddelandet sig själv omkastat till TraceMsg, som returneras hela vägen till TraceMsgs evaluate som sedan returnerar detta. Det är upp till mottagande klass att kontrollera typen och kasta om den.

Exempel på omkastning:

```
try
{
TraceMsg traceMsg = TraceMsg.evaluate(msg);

  if(traceMsg.equals(Idle.class))
  {
    Idle idle = (Idle) traceMsg;
    //Handle Idle message
  }
  else if(traceMsg.equals(LLCInformation.class))
  {
    LLCInformation lLCInformation = (LLCInformation) traceMsg;
    //handle LLCInfornmation message
  }
}
catch (UnknownMsgException e)
```

```
{
e.printStackTrace();
}
```

För en överblick av klass-strukturen vid avkodning se Bilaga 3.

Fenix

Servern till Proben är skriven som en modul i Fenix. Fenix är uppdelat i två delar, en klient-del och en server-del. Servern hanterar all data, alla Fenixklienter samt alla probar som är inkopplade.

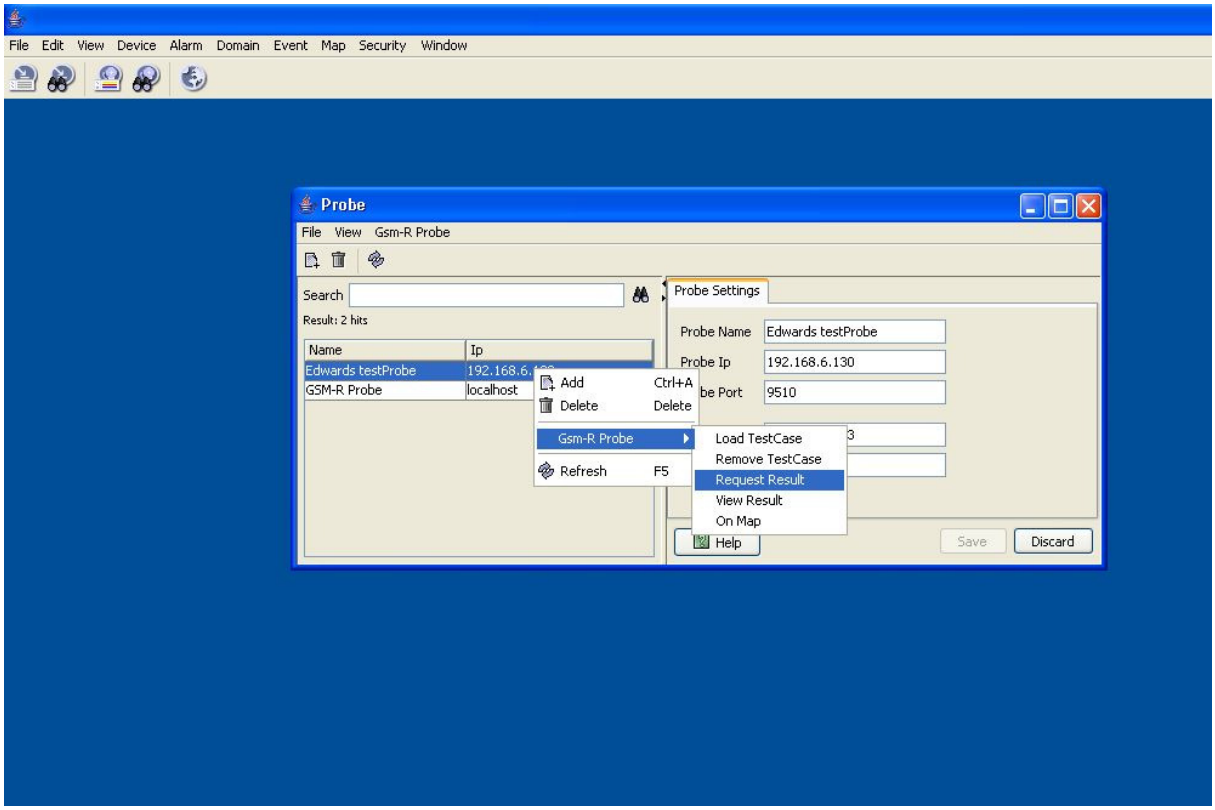
Kommunikation med probe

Kommunikationen med proben sker med sockets. Servern är initiativtagare till all kommunikation. När ett kommando ska skickas byggs först ett ControllerEv och eventuell data tilldelas till rawData. Därefter skickas klassen till proben med hjälp av en ObjectOutputStream.

Lagring av testfallsdata

När servern har hämtat testfallsdata från en probe lagras all XML som binärdata i en databasen. Därefter kan klienterna komma åt all data och presentera det för användaren.

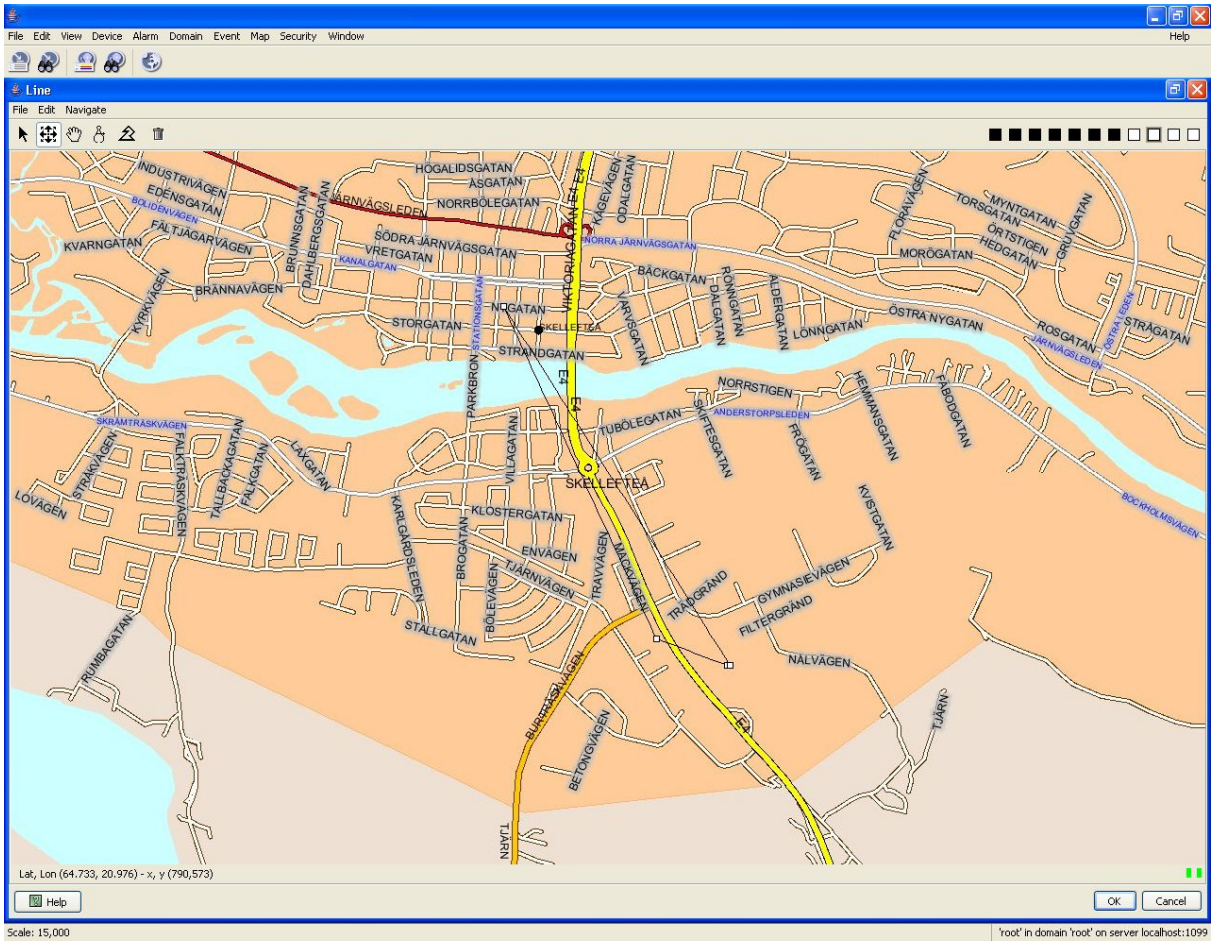
Kontrollering av Probe



Figur 4. Från en enkel meny kan proben styras.

För att kontrollera en probe krävs det först att servern är konfigurerad till proben, IP-adress och portnummer måste ställas in. Genom en enkel meny kan man därefter välja att skicka eller ta bort testfall till proben samt samla in loggdata och slutligen visa probens positioner på en karta.

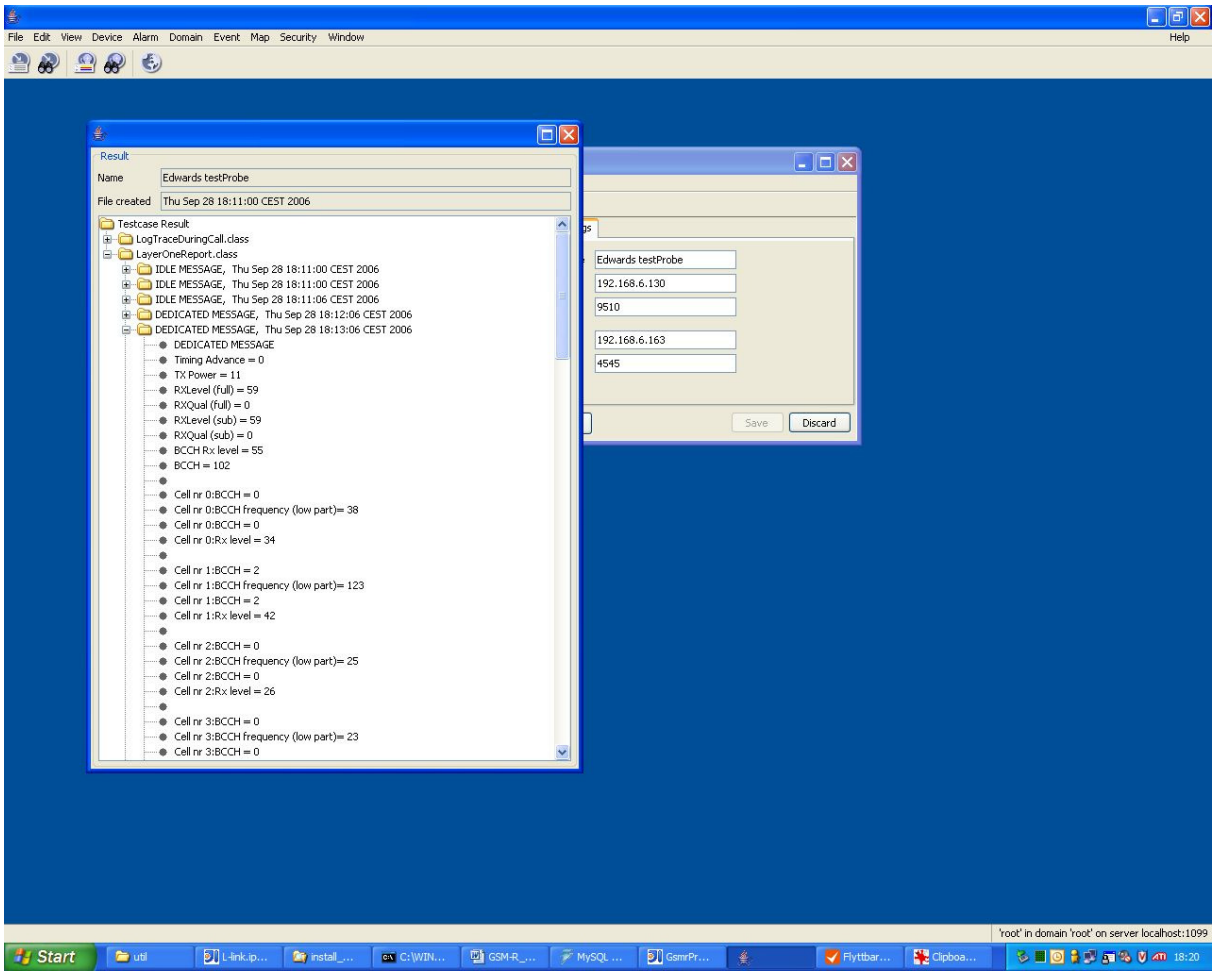
Kartpresentation



Figur 5. Fenix kartvy, probens position visas på karta.

Med kartvyn i Fenix syns probens rörelser, kartvyn kan lätt zoomas eller panoreras.

Loggvisning



Figur 6. Resultaten från testfall visas i en trädstruktur.

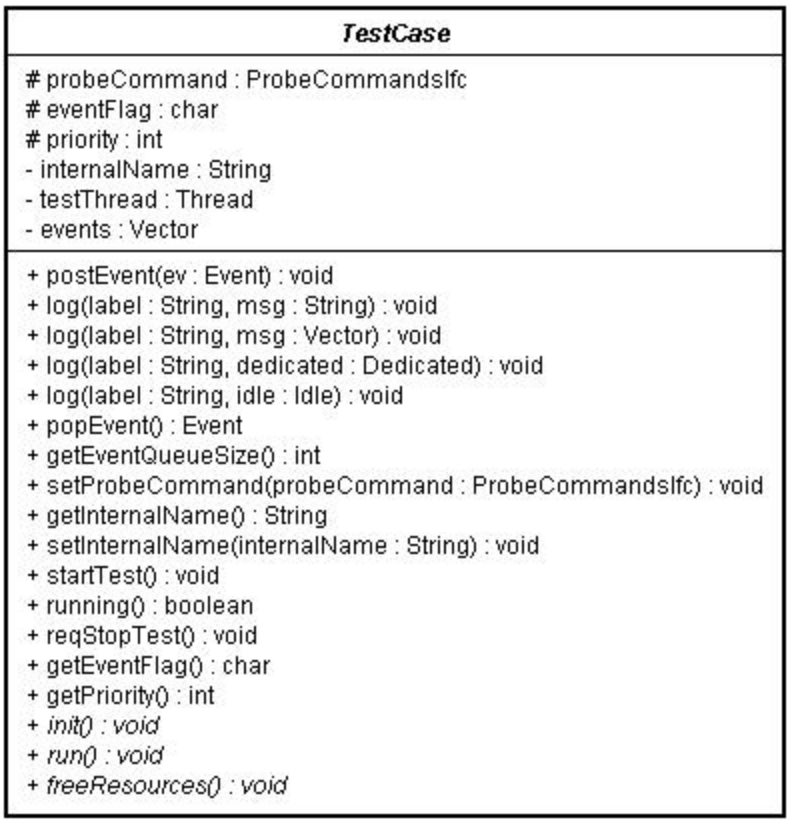
Inhämtade logg-filer presenteras i en trädstruktur. Trädet struktureras efter testfall och loggtillfälle.

Testfall

För att kunna använda GSM-R probens möjligheter att övervaka/testa GSM-R nätet måste man skapa ett testfall. Med ett testfall kan du ställa in tid, specifik tidpunkt eller återkommande, och/eller när probens hårdvara genererar ett event. Du kan även ställa in när ett trace-meddelande kommer eller att ett AT-kommando har utförts. När ett testfall körs är det upp till skaparen av testfallet att bestämma vad det ska göra. Man använder probeCommand för att komma åt de olika resurser som proben har.

Design av testfall

För att kunna skapa ett testfall måste man ärva från abstrakta klassen TestCase.



Dessa funktioner kan användas vid design av testfall, övriga funktioner och medlemsvariabler är interna och bör ej användas i testfallen eftersom att funktionaliteten för proben och dess hantering av testfall då kan ändras.

Variabler:
eventFlag: Bestämmer vilka event som är intressant för testfallet.
priority: Anger testfallets prioritet.
probeCommand: En instans av ProbeCommand används för att få tillgång till probens resurser

Funktioner:
Log(..): Används för att skriva data till probens resultatfil.
getEvetQueueSize(): Antalet element i event-kön.
popEvent(): tar ut det äldsta eventet ur event-kön.
getInternalName(): returnerar testfallets interna namn.

I klassen EventFlags finns de olika eventflaggor som kan användas. Är flera event av intresse så används ”ELLER” dvs. ”|” tecknet.
Exempel:

```
this.eventFlag = EventFlags.TRACE | this.eventFlag = EventFlags.AT | EventFlags.GPS;
```

Detta testfall kommer att mottaga event av typen atEv, traceEv och gpsEv.

De funktioner användaren själv måste skapa är:

```
void init()
```

Vilket är en funktion som kör vid inladdning av ett testfall där anger man prioriteten och event-flaggorna som är intressanta.

```
void run()
```

Här ska själva koden för testfallet ligga, den metoden körs varje gång som ett av de intressanta eventen kommer.

```
void freeResorces()
```

Är destruktorn på testfallet, här skrivs kod som körs när testfallet laddas ur proben t.ex. när en uppdatering av samma testfall kommer eller när proben får ett kommando från Fenix som säger att den ska ta bort testfallet.

Loggning från testfall till XML

Eftersom all loggning från testfallen borde vara enhetlig skapades en loggklass som medför att testfallen kan skriva all data till en XML-fil via ett enkelt interface. Sagda interface fås tillgång till från den abstrakta basklassen TestCase. Vid varje anrop till log kommer ett element för det anropade testfallet läggas in i logg-filen. Varje log kommer att få använda inparametern *label* som överskrift tillsammans med en positionsstämpel från GPS samt tid. Loggningen har i nuläget tre stycken loggningsmöjligheter.

```
void log(String label, String msg);
```

Msg loggas med label som överskrift.

```
void log(String label, Idle idleMsg);
```

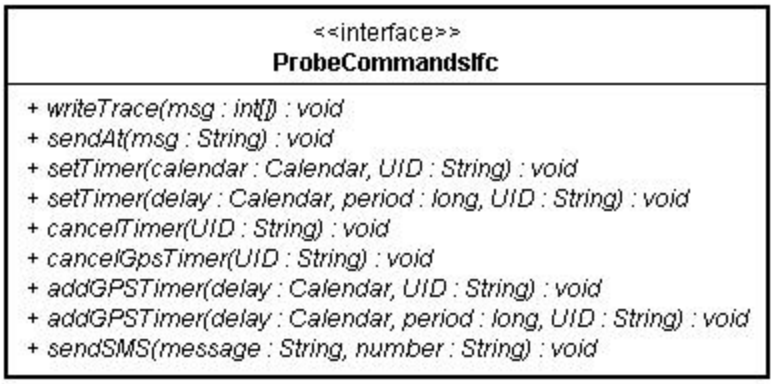
En överlagring för att logga Idle meddelanden från trace-interfacet på telefonen. All data från Idle meddelandet kommer att skrivas ut till XML-filen.

```
void log(String label, Dedicated dedicatedMsg);
```

En överlagring för att logga Dedicated meddelanden från trace-interfacet på telefonen. All data från Dedicated-meddelandet kommer att skrivas ut till XML-filen.
För att se ett exempel av en resultatfil se Bilaga 4.

Resurser från probe

I ett testfall finns en medlemsvariabel som heter probeCommand, denna variabel används i ett testfall för att komma åt resurserna i proben så som GSM-R mobilen och GPS. ProbeCommand är av typen ProbeCommandsIfc vilket innehåller följande:



Via *WriteTrace* aktiveras olika Trace-meddelanden, dessa kommer sedan till testfallet om den har valt att aktivera det. Om man aktiverar ett trace-meddelande måste man avaktivera det när man inte längre vill ha det.

Via *SendAT* skickar man AT-kommandon till GSM-R mobilen, exempelvis om man vill ringa ett specifikt nummer.

Om man vill ställa en viss tid då ett testfall ska få ett timer-meddelande använder man *SetTimer*, *addGPSTimer* och *cancelTimer* för att avbryta timern.

Om man vill skicka ett SMS använder man kommandot *sendSMS*.

För exempel på testfall se bilaga 5 och 6.

Diskussion

Våra resultat visar klart och tydligt att proben kan hantera de situationer som vi hade tänkt ut från början. Huruvida dessa situationer täcker upp allting som en mångsidig probe borde klara av, och möjliga förändringar ska vi ta upp på följande sidor.

Vad innebär/betyder resultaten?

Resultatet av vårt arbete är att Data Ductus kan presentera en lösning för övervakning och kontrollering av GSM-R nät till operatörer. Eftersom designen redan är realiserad i liten skala kan även en enklare presentation av proben genomföras, så att slutprodukten kan skräddarsys till de olika behoven operatörerna kan ha.

Slutsatser, vad stödjer dessa slutsatser?

De slutsatser vi dragit är att en probe måste vara mångsidig för att kunna användas på ett smidigt sätt. Hårdvaran till proben måste kunna klara av de miljöer som finns i ett tåg. Eftersom ett tåg skakar måste utrustningen kunna klara av den påfrestning det medför under en lång tid utan att funktionaliteten påverkas. Proben ska med stor sannolikhet placeras framme hos föraren varför en laptop med inkopplad telefon och GPS blir skrymmande. Då installationen ej får vara skrymmande är därför en inbyggd lösning att föredra. Vi kan även dra slutsatsen att järnvägsbolag har extrema krav på driftsäkerhet och därigenom borde en spegling av probens funktionalitet och data implementeras. Detta skulle innebära att en om en probe slutar att fungera skulle backup genast ta över probens uppgifter.

Vad kunde ha gjorts bättre, fördelar och nackdelar

Eftersom vår kunskap inom området växt under utvecklingstiden så finns det vissa saker som vi i nuläget skulle ha gjort annorlunda.

Jboss

Proben skulle vi ha baserat på Jboss eller liknande applikationsserver eftersom vi genom det skulle få en stabil och vältestad grund med stöd av många av de saker vi själva skrivit så som klassladdning.

Quartz

Istället för att använda JavaTimer borde vi ha hanterat alla timers med QuartzTimers. Eftersom quartz har ett smidigare system där cron trigger används. Cron trigger innebär att man kan ställa in en trigger exempel den tredje fredagen varje jämn månad. Detta skulle i sin tur underlätta mycket för testfallsdesignen eftersom tester under en längre tid skulle kunna utföras, där exempelvis tidtabeller kan tas med i beräkningarna.

Quartz skiljer även på *triggers* och *job* det vill säga att en *trigger* kan utlösa flera olika testfall(*job*) och ett *job* kan utlösas av flera olika timers (triggers). Med den möjligheten innebär det att ett testfall nästan kan schemaläggas när som helst.

SAX

Om det skulle visa sig att Proben skulle få prestandaproblem/minnesproblem kan man förbättra det en aning igenom att ersätta Dom4J parsern med en SAX parser, som är en event baserad parser. Detta innebär att den rapporterar läst data genom event till programmet, eftersom man inte sparar hela dokumentet i minnet så som en DOM parser gör kan man parsa dokument som är större än det tillgängliga minnet.

Framtida förändringar och vidareutveckling

Framtida förändringar i systemet som vi skulle föreslå är att utveckla en inbyggd lösning av proben vilket skulle medföra att användningen skulle bli smidigare och ta mindre plats.

Ett webbinterface för inställningar av proben skulle också underlätta inställningen av probens parametrar.

Proben borde utvidgas så att flera olika mobiler kan användas till den. Detta skulle i sin tur innebära att testning och loggning av flera olika nät möjliggörs.

Proben borde aktivt känna av när nätuppkoppling finns tillgänglig och då skicka resultatet till Fenix. Det skulle göra systemet mer självgående. Som systemet ser ut idag måste användaren aktivt hämta resultat-data från proben.

Event-systemet som finns tillgängligt i Fenix borde implementeras. Innebörden av det blir att alarm kan konfigureras till olika event och genom detta skulle proben kunna generera alarm utifrån sina testfall.

GPRS-stöd borde läggas till så att man kan skicka rapporter utan WLAN eller annan nätuppkoppling alternativt att lägga in stöd för att skicka rapporter via SMS eller ett datasamtal. Med det skulle man kunna använda proben till andra saker än att övervaka ett GSM-R nät, t.ex. att övervaka något annat instrument som har en datakoppling och sedan skicka rapporter periodiskt via mobilen.

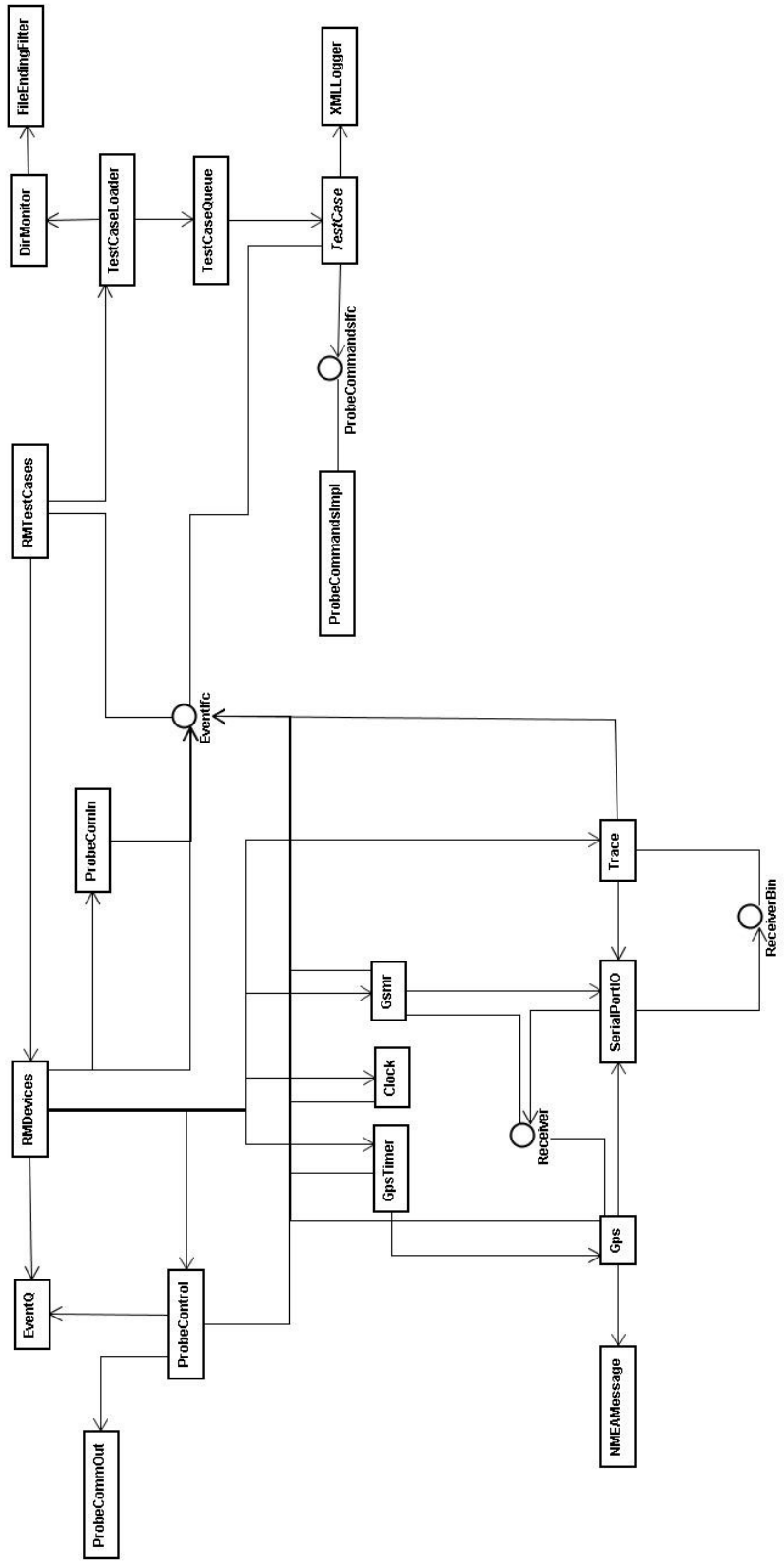
Proben behöver inte användas som ett övervakningsinstrument. Den kan även användas för att fjärrstyra maskiner och använda mobilen för rapportering och styrning av dessa maskiner. Det som krävs för detta är implementering av maskinen som en Kontrollerklass, så att probeCommands kan styra vitala funktioner av sagda maskin.

Summering av betydelsen av arbetet som utförts

Betydelsen av det arbete vi utfört är att Data Ductus har vidareutvecklat kunskaperna i ämnet och numera har ett förslag som kan läggas fram till kunder. Eftersom proben i sig är en enkel men mångsidig lösning finns det förhoppningar att operatörerna över GSM-R näten ska vara villiga att gå vidare till nästa fas i utvecklingsarbetet så att proben kan realiseras.

Bilagor

Bilaga 1



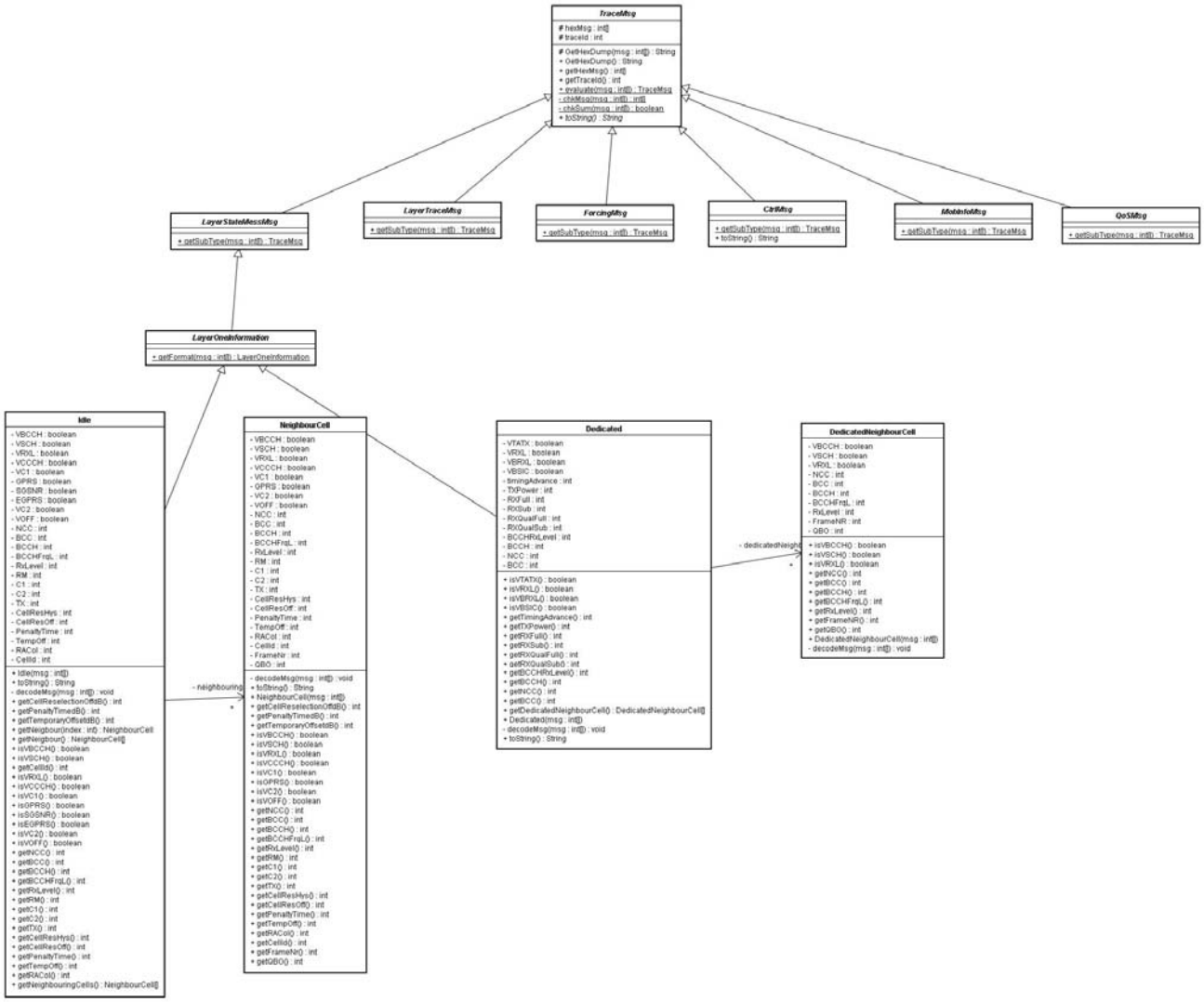
Figur 7 Ett klassdiagram över Proben, endast de mest väsentliga klasserna.

Bilaga 2

Ett exempel på hur property-filen för vår testprobe ser ut.

```
#Thu May 11 22:24:36 CEST 2006
gsmrprobe.resourcemanagers.testcases.RMTestCases.log4j.propertys=\\util\\properties
\\log4j.propertys
gsmrprobe.resourcemanagers.testcases.RMTestCases.TestcaseDir=\\util\\testcases

#XML info
gsmrprobe.resourcemanagers.testcases.XMLLogger.testcase.result=\\util\\result.xml
gsmrprobe.resourcemanagers.testcases.XMLLogger.name=test2


gsmrprobe.classloader.TestCaseLoader.FileEnding=.class
gsmrprobe.classloader.TestCaseLoader.SleepTime=2000

gsmrprobe.resourcemanagers.devices.socket.ProbeComIn.ServerPort=9510

#SerialPort Setup
# DATABITS 8 = 8,  DATABITS 7 = 7, DATABITS 6 = 6, DATABITS = 5
# STOPBITS 1 =  1, STOPBITS 1,5 = 3, STOPBITS 2 = 2
# PARITY_EVEN= 2, PARITY_MARK= 3, PARITY_NONE= 0, PARITY_ODD= 1, PARITY_SPACE= 4
# FLOWCONTROL_NONE= 0, FLOWCONTROL_RTSCTS_IN= 1, FLOWCONTROL_RTSCTS_OUT= 2,
#FLOWCONTROL_XONXOFF_IN= 4, FLOWCONTROL_XONXOFF_OUT= 8


#4800, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE,
SerialPort.FLOWCONTROL_NONE
gsmrprobe.resourcemanagers.devices.gps.Gps.Connected=1
gsmrprobe.resourcemanagers.devices.gps.Gps.Port=COM3
gsmrprobe.resourcemanagers.devices.gps.Gps.BaudRate=4800
gsmrprobe.resourcemanagers.devices.gps.Gps.DataBits=8
gsmrprobe.resourcemanagers.devices.gps.Gps.StopBits=1
gsmrprobe.resourcemanagers.devices.gps.Gps.Parity=0
gsmrprobe.resourcemanagers.devices.gps.Gps.FlowControl=0

#57600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE,
SerialPort.FLOWCONTROL_NONE
gsmrprobe.resourcemanagers.devices.gsmr.Gsmr.Connected=1
gsmrprobe.resourcemanagers.devices.gsmr.Gsmr.Port=COM4
gsmrprobe.resourcemanagers.devices.gsmr.Gsmr.BaudRate=57600
gsmrprobe.resourcemanagers.devices.gsmr.Gsmr.DataBits=8
gsmrprobe.resourcemanagers.devices.gsmr.Gsmr.StopBits=1
gsmrprobe.resourcemanagers.devices.gsmr.Gsmr.Parity=0
gsmrprobe.resourcemanagers.devices.gsmr.Gsmr.FlowControl=0

#57600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE,
SerialPort.FLOWCONTROL_RTSCTS_IN
gsmrprobe.resourcemanagers.devices.gsmr.Trace.Connected=1
gsmrprobe.resourcemanagers.devices.gsmr.Trace.Port=COM1
gsmrprobe.resourcemanagers.devices.gsmr.Trace.BaudRate=57600
gsmrprobe.resourcemanagers.devices.gsmr.Trace.DataBits=8
gsmrprobe.resourcemanagers.devices.gsmr.Trace.StopBits=1
gsmrprobe.resourcemanagers.devices.gsmr.Trace.Parity=0
gsmrprobe.resourcemanagers.devices.gsmr.Trace.FlowControl=1
```

Bilaga 3



Figur 8 Övergripande bild av trace-avkodning.

Bilaga 4

```
<?xml version="1.0" encoding="utf-8" ?>
<priobeID name="GSM-R Probe" fileCreated="Thu Sep 28 18:44:49 CEST 2006">
  <LayerOneReport.class>
    <entry time="Wed Oct 18 08:05:39 CEST 2006" label="DEDICATED MESSAGE"
longitude="21.618055555555557" latitude="64.96444444444444">
      <data>DEDICATED MESSAGE</data>
      <data>Timing Advance = 12</data>
      <data>TX Power = 5</data>
      <data>RXLevel (full) = 37</data>
      <data>RXQual (full) = 2</data>
      <data>RXLevel (sub) = 37</data>
      <data>RXQual (sub) = 2</data>
      <data>BCCH Rx level = 72</data>
      <data>BCCH = 40</data>
      <data />
      <data>Cell nr 0:BCCH = 0</data>
      <data>Cell nr 0:BCCH frequency (low part)= 27</data>
      <data>Cell nr 0:BCCH = 0</data>
      <data>Cell nr 0:Rx level = 28</data>
      <data />
      <data>Cell nr 1:BCCH = 0</data>
      <data>Cell nr 1:BCCH frequency (low part)= 25</data>
      <data>Cell nr 1:BCCH = 0</data>
      <data>Cell nr 1:Rx level = 25</data>
      <data />
      <data>Cell nr 2:BCCH = 0</data>
      <data>Cell nr 2:BCCH frequency (low part)= 28</data>
      <data>Cell nr 2:BCCH = 0</data>
      <data>Cell nr 2:Rx level = 22</data>
      <data />
      <data>Cell nr 3:BCCH = 0</data>
      <data>Cell nr 3:BCCH frequency (low part)= 20</data>
      <data>Cell nr 3:BCCH = 0</data>
      <data>Cell nr 3:Rx level = 19</data>
      <data />
      <data>Cell nr 4:BCCH = 0</data>
      <data>Cell nr 4:BCCH frequency (low part)= 45</data>
      <data>Cell nr 4:BCCH = 0</data>
      <data>Cell nr 4:Rx level = 18</data>
      <data />
      <data>Cell nr 5:BCCH = 0</data>
      <data>Cell nr 5:BCCH frequency (low part)= 23</data>
      <data>Cell nr 5:BCCH = 0</data>
      <data>Cell nr 5:Rx level = 18</data>
    </entry>
  </LayerOneReport.class>
</priobeID>
```

Ett exempel på hur ett logg-meddelande från testfallet LayerOneReport kan se ut.

Bilaga 5

Ett testfall som I ett intervall av en minut hämtar ut LayerStateMesurement meddelanden från traceinterfacet på GSM-R telefonen.

```
public class LayerOneReport extends TestCase
{
  private static Logger logger = (Logger) Logger.getInstance(LayerOneReport.class);

  public void init()
  {
    this.priority = 2;
    this.eventFlag = (char) (EventFlags.TRACE | EventFlags.CLOCK);
    Calendar c = Calendar.getInstance();
    c.set(Calendar.SECOND, c.get(Calendar.SECOND) + 10);

    this.probeCommand.setTimer(c, 60000, this.getInternalName());
  }

  public void run()
  {

    while(this.getEventQueueSize() != 0)
    {
      Event ev = this.popEvent();

      if(ev.getType() == EventFlags.CLOCK)
      {
        logger.info("Requested  trace msg");
        this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
                        GetStateMeasurementInfo(LayerStateMeasurementInfo.
                        LAYER_1_INFOMATION));
      }

      if(ev.getType() == EventFlags.TRACE)
      {
        TraceEv Tev = (TraceEv) ev;
        if(Tev.getMsg().getClass().equals(Idle.class))
        {
          this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
                        GetStateMeasurementInfo(LayerStateMeasurementInfo.OFF));

          logger.info("Logging Idle message");
          this.log("IDLE MESSAGE", (Idle) Tev.getMsg());
        }
        else if(Tev.getMsg().getClass().equals(Dedicated.class))
        {
          this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
                        GetStateMeasurementInfo(LayerStateMeasurementInfo.OFF));

          logger.info("Logging Dedicated message");
          this.log("DEDICATED MESSAGE", (Dedicated) Tev.getMsg());
        }
      }
    }
  }

  public void freeResources()
  {
    this.probeCommand.cancelTimer(this.getInternalName());
    this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
                        GetStateMeasurementInfo(LayerStateMeasurementInfo.OFF));

  }
}
```

Bilaga 6

Ett testfall som med ett intervall av en min ser till att proben ringer upp ett telefonnummer. Medan ett samtal är uppkopplat loggas LayerOneInformation Idle och Dedicated meddelanden under 10 sekunder. Efter 10 sekunder skickas ett kommando till telefonen att koppla ned samtalet.

```
public class LogTraceDuringCall extends TestCase
{
private static Logger logger = (Logger)
Logger.getInstance(LogTraceDuringCall.class);

  public void init()
  {
    this.priority = 2;
    this.eventFlag = (char) (EventFlags.TRACE | EventFlags.CLOCK | EventFlags.AT);
    Calendar c = Calendar.getInstance();
    c.set(Calendar.SECOND, c.get(Calendar.SECOND) + 10);

    this.probeCommand.setTimer(c, 60000, this.getInternalName());
  }

  public void run()
  {
    while(this.getEventQueueSize() != 0)
    {
      Event ev = this.popEvent();

      if(ev.getType() == EventFlags.CLOCK)
      {
        logger.info("Requested  trace msg");
        this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
GetStateMeasurementInfo(LayerStateMeasurementInfo.LAYER_1_INFOMATION));

        this.probeCommand.sendAt("ATD0702506539");
      }
      if(ev.getType() == EventFlags.AT)
      {
        AtEv atEv = (AtEv) ev;
        this.log("AT Command: ", atEv.atCommand + "  Reply: " + atEv.message);
      }

      if(ev.getType() == EventFlags.TRACE)
      {
        TraceEv Tev = (TraceEv) ev;

        if(Tev.getMsg().getClass().equals(Idle.class))
        {
          this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
                  GetStateMeasurementInfo(LayerStateMeasurementInfo.OFF));

          logger.info("Logging Idle message");
          this.log("IDLE MESSAGE", (Idle) Tev.getMsg());
        }
        else if(Tev.getMsg().getClass().equals(Dedicated.class))
        {
          this.probeCommand.sendAt("ATH");
          this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
                  GetStateMeasurementInfo(LayerStateMeasurementInfo.OFF));

          logger.info("Logging Dedicated message");
          this.log("DEDICATED MESSAGE", (Dedicated) Tev.getMsg());
        }
      }
    }
  }
```

```
  public void freeResources()
  {
    this.probeCommand.cancelTimer(this.getInternalName());
    this.probeCommand.writeTrace(new LayerStateMeasurementInfo().
            GetStateMeasurementInfo(LayerStateMeasurementInfo.OFF));
  }
}
```

Referenser

[1] GSM-R website URL: http://gsm-r.uic.asso.fr/ (2006-10-18)

[2] Java website URL: http://java.sun.com/ (2006-10-18)

[3] Java Communications website URL: http://java.sun.com/products/javacomm/ (2006-10-18)

[4] Log4j website URL: http://logging.apache.org/log4j/docs/ (2006-10-18)

[5] W3c page for DOM URL: http://www.w3.org/XML/ (2006-10-18)

[6] DOM4j website URL: http://www.dom4j.org/ (2006-10-18)

[7] W3c page for DOM URL: http://www.w3.org/DOM/ (2006-10-18)

[8] Hibernate website URL: http://www.hibernate.org/ (2006-10-18)

[9] MySQL website URL: http://www.mysql.com/ (2006-10-18)

[10] Triorail website URL: http://www.triorail.com/ (2006-10-18)

[11] ETSI website URL: http://www.etsi.org/ (2006-10-18)