GÖTEBORGS UNIVERSITET
Institutionen för Informatik

Utveckling av expertsystem med Java

Stellan Aspenström

EXAMENSARBETE 1
Vårterminen 1997
Handledare: Faramarz Agahi

ABSTRAKT

Java är ett av de mest aktuella ämnena inom IT branschen för tillfället. Åtskilliga hyllmeter böcker har skrivits om det och om man gör en sökning på internet via någon sökmotor får man upp åtskilliga tusen träffar. Ett ämnesområde där det kan vara aktuellt att använda sig av Java vid utveckling är expertsystem. Frågan som jag ställde vid starten av detta arbete var om det gick att kombinera dessa två områden. I detta arbete visades hur man kunde kombinera dessa två ämnesområden, Java och expertsystem, genom utvecklandet av ett larmtolkningssystem. Jag ämnade inte att visa att Java var den ultimata tekniken att utveckla expertsystem med, jag ämnade heller inte att ta fram den bästa tekniken att utveckla expertsystem med. Vad som kom fram var att det på ett enkelt sätt gick att utveckla expertsystem med Java, även om denna utvecklingsmiljö innehöll vissa brister.

INNEHÅLLSFÖRTECKNING

1. INTRODUKTION

'Much ado about nothing.'
- William Shakespear

1.1. BAKGRUND

Ett av de hetare ämnena inom IT branschen under de sista två åren är Java. Det har skrivits åtskilliga hyllmeter om det och om man gör en sökning på internet via någon sökmotor får man upp åtskilliga tusen träffar. Man talar om Java i termer alltifrån bara ännu ett programmeringsspråk till det nya saliggörande universal hjälpmedlet som skall rädda världen. I och med att Java är såpass nytt har det ännu inte hunnit visa upp sina brister och förtjänster. Vilka användningsområden som är mer, eller mindre, lämpliga att använda Java till är fortfarande osäkert.

Ett ämnesområde som Java kan vara aktuellt för att bedriva utveckling i är expertsystem. Expertsystem är intressant ur den synpunkten att det handlar om kunskap. Det talas om att vi lämnar (eller redan har lämnat?) industrisamhället och går in i kunskapssamhället. Information och kunskap är något som blir alltmer viktigt, att då också lämna de grundläggande funktionerna inom informationsbehandling såsom lagring och hämtning av data även för nya tekniker såsom Java och flytta sig högre upp i värdekjedjan, mot analys och kunskapsbehandling av data, torde bli av allt högre vikt.

1.2. UPPGIFTSBESKRIVNING

Jag ämnar att i detta arbete utveckla ett larmtolkningssystem i Java. Ett larmtolkningssystem är ett expertsystem och detta expertsystem skall användas på en övervakningscentral för tele- och datakomnät. Systemet skall där hjälpa operatörerna att tolka larmar som uppkommer på en viss transmissionsplattform - Martis. Larmar uppkommer i nätet på grund av någon typ av fel, dessa fel kan bero på ett flertal olika orsaker, vilken orsak som orsakat felet är det upp till operatören att komma fram till genom tolkning av de tidigare nämnda larmarna. När operatören sedan kommit fram till vilket fel det är skall han eller hon se till att rätt åtgärd vidtages för att rätta till felet. Systemet skall utvecklas genom att hämta kunskaper, dels från manualer och annan dokumentation, dels från erfarna operatörer med deras expertis. Denna kunskap skall sedan finnas tillgänglig för operatörerna genom expertsystemet. Förhoppningen är att detta system skall hjälpa operatörerna att snabbare komma fram till vilket fel det är.

1.3. PERSPEKTIV - AVGRÄNSNING

Denna uppgift är mer av karaktären praktisk - experimentellt än teoretisk, jag ämnar inte visa att Java är den bästa tekniken att utveckla expertsystem med, jag ämnar heller inte att ta fram den bästa tekniken att för utveckla expertsystem med. Det tillvägagångs sätt som jag kommer att använda mig av är att utveckla ett expertsystem med hjälp av Java, utifrån detta kommer jag sedan att kunna se vilka problem som uppkommer, vad som fungerar bra respektive dåligt med att utveckla expertsystem i Java.

1.4. PROBLEMBESKRIVNING

Den hypotes som jag utgår från är att det borde gå att utveckla ett expertsystem med hjälp av Java. Denna hypotes leder fram till två problem som skall besvaras, nedan följer formulering av dessa.

Det första problemet är att hitta en lämplig teknik att kunna koppla ett expertsystem till Java, finns det något gjort tidigare, eller blir man tvungen att skriva ett expertsystem för Java från grunden?

Det andra problemet är beroende av utfallet av det första problemet, om det är så att det finns tidigare arbete inom området utveckling av expertsystem i Java kommer jag att titta på denna / dessa tekniker och se om detta är bra eller dåliga tekniker. Är det så att det inte kommer fram att något tidigare har gjorts kommer jag att titta på huruvida det är möjligt att åstadkomma en koppling mellan ett expertsystem och Java.

1.5. METOD

Den metod jag tänker att använda mig av vid besvarandet av det första problemet är att söka efter information dels i litteratur, dels på internet om utveckling av expertsystem i Java. För de andra två problemen kommer jag, som tidigare nämnts, att använda mig av ett praktiskt arbete för att utveckla ett expertsystem för ett givet ändamål (se under 1.3.). Resultatet kommer inte att vara ett fullfjädrat expertsystem för tidigare nämnda ändamål, utan ett system som skall vara påbyggnadsbart då det gäller att utöka expertsystemets kunskap med nya regler och fakta.

1.6. FÖRETAGSBESKRIVNING

Det företag som kommer att använda sig av det inom ramen för detta arbete utvecklade systemet heter Unisource Business Networks. Unisource Business Networks är en del av Unisource som ägs utav Telia, PTT Netherlands och Swisscom (tidigare PTT Switzerland). Tidigare var Telefónica delägare, men man drog sig ur. På en global nivå agerar man tillsammans med AT&T i ett sammarbetsbolag som heter AT&T Unisource Kommunikation Services (AUCS), och man ingår i World Partners alliansen tillsammans med bland andra KDS och Hong Kong Telecom. Unisource producerar ett stort antal tele- och datakom tjänster baserade på ett antal olika teknik plattformar. Unisource Business Networks driver de tjänster som har med företagsnät att göra. I Göteborgs finns övervakningscentralen för tjänsterna FAM (Företags Anpassade Muxnät) och Unistream, den sistnämnda tjänsten, Unistream, använder sig av teknik plattformarna Martis och Timeplex för att realiseras, och detta arbete görs som tidigare nämnts på plattformen Martis.

2. DEFINITION

'Det första steget till visdom är att känna själva tingen.'
- Carl von Linné

2.1. INLEDNING

För att ge läsaren till detta arbete en större insikt om de begrepp som används kommer de centrala begreppen i denna uppsats här att presenteras närmare. Till att börja med ges en kort förklaring av vad ett expertsystem är och hur man kan klassiciera dessa. Då det har skrivits åtskilliga böcker om expertsystem så görs inte någon djupare diskussion kring expertsystem här, utan läsaren hänvisas till referenserna. Den kunskap som innehas av experter kallas för expertis, en kort beskrivning av detta begrepp görs också. Vidare ges en inledning till de tekniker som används vid utvecklingsarbetet, dessa är Java, Jess och Clips. Denna är inte avsedd att fungera som en fullständig manual till nämnda tekniker, utanv endast just som en inledning. I detta fallet hänvisas också till referenserna om läsaren vill fördjupa sig ytterligare. Det system som kommer att utvecklas skall som tidigare nämnts användas på en specifik teknikplattform som heter Martis, en beskrivning av denna ingår också i detta kapitel. Denna beskrivning fungerar också som en kortfattad introduktion till transmission med PDH teknik, samt multiplexering. Till sist kommer en beskrivning av den situation som den utvecklade systemet kommer att användas i, nämligen larmtolkning.

2.2. EXPERTIS

Expertis definieras i Turban (1995) som väl utvecklad kännedom inom ett visst område, denna kännedom kan ha uppkommit genom antingen studier inom området eller erfarenhet inom området. Expertis kan vidare vara av olika typ. Det kan vara fakta kunskaper inom problem området, det kan vara teori kunskaper inom problem området, det kan vara kunskaper om regler och procedurer inom området, det kan vara tumregler som uppkommit genom erfarenhet inom området, det kan vara kunskap om kunskap inom området dvs meta kunskap. Ovanstående typer av kunskap gör det lättare för en expert än en icke expert att lösa problem inom ett visst område.

2.3. EXPERTSYSTEM

Enligt Turban (1995) är ett expertsystem ett system som tillvaratar mänsklig kunskap inom en specifik domän (expertis) för att lösa problem inom denna, kunskapen är sådan som innehas av experter inom domänen, och därigenom möjliggör systemet för icke experter att lösa problem inom domänen. Luconi, Malone och Morton (1986) definierar expertsystem som dator program som använder symboliskt resonemang för att lösa svåra problem på ett bra sätt, expertsystem använder sig av specifik kunskap inom ett område hellre än generell kunskap, expertsystem använder sig av symboliskt resonemang hellre än numeriskt beräknande, expertsystem har högre kompetens (inom området) än icke experter. Ett tredje sätt att beskriva expertsystem göras av Knight och Rich (1991), expertsystem löser problem som normalt löses av mänskliga experter, för att lösa dessa måste systemet ha tillgång till en domän specifik kunskapsbas. Gemensamt för ovanstående referenser är expertis (specifik kunskap), nedan finns definition av expertis.

Det finns enligt Turban (1995) fem olika teknik nivåer för utveckling av expertsystem, dessa är Specifika expertsystem, Skal, Verktyg, Hybrid System och Programmeringsspråk. *Specifika expertsystem* är system som är gjorda för att stödja en specifik problem situation, exempelvis ett system som löser skatteplanerings problem. *Skal* är i princip ett färdigbyggt expertsystem, med all funktionalitet, förutom kunskapsbasen (expertisen), denna får programmeraren lägga till för att skapa ett komplett system. *Verktyg* skiljer sig från skal så tillvida att man inte bara kan lägga till kunskapsbasen, utan man kan även ändra eller lägga till i funktionaliteten. *Hybrid system* består både av verktyg och programmeringsspråk, detta gör att vissa delar blir enklare att göra med hjälp av verktyg, men att man behåller programmering språkets kraftfullhet. *Programmeringsspråk* gör att man väldigt stor flexibilitet och frihet när man skall göra ett expertsystem, dock kan det kräva mycket arbete i jämförelse med övriga teknik nivåer.

2.4. JAVA

2.4.1. Beskrivning

Gosling, Joy och Steele (1996) beskriver Java som ett klassbaserat, objektorienterat programmeringsspråk avsett för generella ändamål. Det har släktskap med C och C++, det skiljer sig dock på en del punkter, nämnas kan att Java innehåller automatisk minnesallokering, man behöver inte, som i C och C++, allokera och deallokera minnesutrymme. Från det steg där man har sin källkod tills dess att man har ett exekverbart program går till på följande sätt i Java. Källkoden kompileras till maskin oberoende kod representation. Denna kod körs sedan genom en virtuell Java maskin (JVM). JVM finns inkluderad i många webläsare (t.ex. Netscape 3.x och uppåt samt Microsoft Internet Explorer 3.x och uppåt), men finns även tillgänglig för de flesta operativ system. Vid körningen i JVM laddas och länkas de av koden önskade klasserna in, koden optimeras dynamiskt därefter och till sist exekveras programmet. Ovanstående tillvägagångssätt för att köra ett program möjliggör att ingen programvara behövs installeras på användarens maskin (förutom en JVM), allt som behövs för att köra ett program laddas in vid behov.

2.4.2. Skillnader mellan Java applikation och applet

Java program kan skrivas för att antigen exekveras som applikationer eller applets. Skillnaden mellan en Java applikation och en Java applet består i att en applikation exekveras från användarens lokala filsystem, antingen hårddisken eller en server, medan en applet exekveras från en web sidan genom en webläsare. Om programmet körs som en applet läggs en hel del säkerhets restriktioner på programmet, dessa finns ej om programmet körs som en applikation. Enligt Campione & Walrath (1996) är de säkerhets restriktioner som läggs på en Java applet enligt följande. Applets kan inte läsa eller skriva till filer som ligger på det lokala filsystemet, detta göra att man inte direkt kan spara undan resultat från en Java applet till sin lokala hårddisk eller server. Man kan gå runt detta genom att se till att resultatet även kan tas fram genom en web sida som sedan kan sparas undan om användaren vill detta. Applets kan inte starta program på den lokala dator på vilken appleten exekveras. En applet kan endast skapa nätverks kopplingar till den server från vilken appletten laddades ner ifrån. En applet kan endast läsa vissa utvalda system egenskaper. De fönster som en applet skapar på användarens dator får ett något annorlunda utseende än vad motsvarande fönster ser ut som skapas av det lokala operativsystemet. De system egenskaper som kan läsas av en applet är följande:

file.separator	Talar om vilket filavskiljningstecken som används
java.class.version	Vilken klass verision som används i Java programmet
java.vendor	Namnet på tillverkaren av Java klassen
java.vendor.url	URL till Java tillverkarens hemsida
java.version	Vilken Java verision som används
line.separator	Talar om vilket linjeavskiljningstecken som används
os.arch	Vilken operativsystems arkitektur som används
os.name	Namnet på det lokala operativsystemet
path.separator	Talar om vilket sökvägsavskiljningstecken som används

En del av ovanstående begränsningar kan man gå runt genom att använda olika typer av server program, t.ex. för att koppla sig vidare mot andra applets på andra maskiner.

2.5. JESS

Jess står för Java Expert System Shell och är ett paket av ett antal Java klasser vilka gör att man till en Java applikation eller applet kan koppla expertsystem skrivna i språket Clips. Jess implementerar inte samtliga funktioner som finns i Clips, dock de som är nödvändiga för att skriva expertsystem. För att kunna använda Jess funktionaliteten i ett Java program måste man först importera paketet jess till sitt Java program, detta görs i början av Java programmet med satsen:

```
import jess.*;
```

Vidare är det tre delar av Jess som måste implementeras i ens Java program, en syntax genomgångs del, en expertsystem motor och ett interface mellan Java och expertsystemet för att kunna sända data mellan Java delen och expertsystem delen. De olika objekt som behövs skapas för Jess är ett av klassen Rete, ett av klassen Jesp, ett av klassen LostDisplay samt ett av klassen PrintStream (utöver övriga objekt som behövs för andra ändamål såsom in- och utmatning t.ex.). Vid skapandet är PrintStream objektet beroende av existensen av ett standard TextArea, LostDisplay objektet är beroende av tidigare nämnda PrintStream objekt, till sist är Rete objektet beroende av LostDisplay objektet. Nedan följer exempel på definering samt skapade av ovan nämnda objekt:

```
private Rete rete;
private Jesp jesp;
private LostDisplay ld;
private PrintStream out;
private TextArea txtCause=new TextArea(6,80);
...
out=new PrintStream(new
     TextAreaOutputStream(txtCause),true);
ld=new LostDisplay(out,System.in,this);
rete=new Rete(ld);
```

För att kunna läsa in Clips programmet och göra en syntaktisk kontroll används objektet Jesp. Jesp objektet är beroende av att Rete objektet existerar. Nedan visas inläsning och användning av Jesp objektet (själva expertsystemet förutsätts ligga i filen ”rulebase.clp” i samma bibliotek som Java programmet):

```
InputStream is=new
     URL(getDocumentBase(),"rulebase.clp").openStream();
jesp=new Jesp(is,rete);
jesp.parse(false);
is.close();
```

När man läst in sitt expertsystem och skapat de nödvändiga objekten kan man sedan se till att man lägger till lämpliga värden från Java programmet till expertsystemet under exekveringen, varpå man exekverar expertsystemet. Nedan följer exempel på hur detta kan gå till, listan lista töms, varpå värdet Adam läggs in och till sist exekveras systemet:

```
rete.ExecuteCommand("(reset)");
rete.ExecuteCommand("(assert (lista (namn Adam)))");
rete.ExecuteCommand("(run)");
```

2.6. CLIPS

2.6.1. Bakgrund

Clips står för C Language Integrated Production System *(JSC-25012)* och har sitt ursprung hos NASA. Man hade där problem med att expert system som utvecklades sällan togs i drift, undersökningar som då gjordes visade på att ett problem var att man använde sig av Lisp för att utveckla expert system. Problemen med Lisp var tre, begränsad tillgänglighet för olika typer av plattformar, höga kostnader samt svårt att integrera med andra utvecklingsverktyg. För att förändra den befintliga situationen startade man utvecklingen av ett expertsystem verktyg, detta blev så småningom CLIPS.

2.6.2. Översikt

Program skrivna i Clips kan bli exekverade på tre sätt, genom ett kommando baserat text gränssnitt, genom fönsterbaserade gränssnitt eller genom inbäddning i större system. Jess använder sig av den tredje metoden - inbäddning. Clips innehåller alla de element som vanligtvis återfinns i programmeringsspråk, funktioner, variabler, listor och kommentarer. Dessutom finns två element för att utveckla expertsystem, fakta och regler.

2.6.3. Syntax

Det enklaste elementet kallas i Clips för atom, en atom skrivs på följande sätt :

```
(atom)
```

Det finns ett antal atomvärden som är fördefinerade i Clips och fungerar som kommandon till systemet. Två av dessa återfinns i samtliga Clips program, dessa är `(run)` och `(reset)`. `(reset)` har till uppgift att tömma listor på dess värden så att nya värden kan läggas till listorna `(run)` har till uppgift att exekvera expertsystemet. En lista är en samling av atomer, det första elementet i en lista kallas för dess huvud. En lista skrivs på följande sätt:

```
(alfa beta gamma)
```

Variabler i Clips definieras med ett ? framför atomnamnet. Värden läggs till atomen med hjälp av kommandot `(bind)`, nedan läggs värdet 3 in i variabeln ?alfa:

```
(bind ?alfa 3)
```

Kommentarer kan läggas in överallt i ett Clips program, en kommentar skapas med ett ; och fortsätter raden ut, exempel på kommentar är:

```
; Detta är en kommentar
```

Funktioner i Clips skapas med hjälp av funktionen `(deffunction)`, alla variabler som skickas med till funktionen sänder endast sitt värdet, inte sin referens. En funktion returnerar ett värdet med hjälp av kommando `(return)`. Nedan följer ett exempel som skapar en funktion som har till uppgift att hitta det minsta av två medsända värden:

```
(deffunction min(?a ?b)
          if (> ?a ?b) then
               return ?a)
          else
               return ?b)))
```

Den första delen som är speciell just vid expertsystem byggnad är fakta. Fakta är av två typer, ordnade eller oordnade fakta. Ordnade fakta är listor, medan oordnade fakta är mer komplexa. Ett oordnat fakta definieras genom att skapa en vektor som kan innehålla den önskade fakta, detta görs med (`deftemplate`). Till denna vektor kan värden läggas till, detta görs med kommando (`assert`). Nedan är ett exempel på hur en fakta lista för en person skapas, samt att några exempel fakta läggs in faktalistan:

```
(deftemplate person (slot namn) (slot vikt))
(assert (person (namn Adam) (vikt 70)))
(assert (person (namn Eva) (vikt 60)))
```

Regler är själva huvuddelen då det gäller expertsystem, det är reglerna som är själva kunskapsbasen. Regler definieras med funktionen (`defrule`). Regler kan sägas vara avancerade IF - THEN satser. Alla regler skall ha ett namn, ett evaluerings uttryck och en handlingssats. Man kan även lägga in en beskrivning av regeln i regeln själv. Namnet skall vara ett tillåtet atomnamn, evalueringsuttrycken sker på de fakta som finns tillgängliga för systemet. Handlingssatsen kan vara exempelvis funktionsanrop eller utskriftssatser. Nedan följer exempel på regler samt en förklaring till vad de utför:

```
(defrule testregel1 ”Beskrivning av testregel1”
     (fault (alarm 42) (node A))
=>
     (printout t ”Meddelande till användare” crlf))
```

Ovanstående regel kommer när den exekveras att se efter om något element i faktalistan fault har värdet 42 i alarm positionen och värdet A i node positionen. Om det är så så kommer systemet att skriva ut meddelandet ”Meddelande till användare” till användaren samt en radbrytning. Man kan i viss mån styra i vilken ordning som olika regler skall evalueras, detta görs med hjälp av att man lägger till parametern (`salience`). En regler som inte ges ett salience värde får värdet 0, regler med högt värdet exekveras före regler med lågt värde och därigenom ges möjlighet till kontroll av exekverings följd, nedan följer exempel:

```
(defrule testregel2
     (declare (salience -100))
     (command exit-when-idle)
=>
     (printout t "avslutar..." crlf))
```

Ovanstående exempel ges ett lågt salience värdet och kommer därigenom att exekveras sent och skriver ut strängen ”avslutar...”.

2.7. MARTIS

Martis är en leverantör av en specifik transmissionsplattform. Denna plattform bygger på PDH (Plesiochronous Digital Hierarchy) och SDH (Synchronous Digital Hierarchy) teknik. PDH har tidigare varit den dominerande transmissions tekniken, med idag börjar SDH och även ATM (Asynchronous Transfer Mode) ta över som dominerande transmissions teknik.

PDH bygger på en transmissions hastighet av 2 Mbit/s, för att överföra data vid större hastigheter måste man med PDH teknik multiplexera samman 4 st 2 Mbit/s förbindelser till en 8 Mbit/s förbindelse, 4 av dessa kan i sin multiplexeras upp till en 34 Mbit/s förbindelse, vidare kan 4 st 34 Mbit/s förbindelser multiplexeras upp till en 140 Mbit/s förbindelse. I andra ändan av förbindelsen kan man sedan demultiplexera ned 140 Mbit/s förbindelsen först till 4 st 34 Mbit/s förbindelser, därefter varje 34 Mbit/s förbindelse till 4 st 8 Mbit/s förbindelser och till sist varje 8 Mbit/s förbindelse till 4 st 2 Mbit/s förbindelser. I dessa 2 Mbit/s förbindelser transporteras 32 st 64 kbit/s kanaler. Ovanstående beskrivning visar att det krävs ett stort antal multiplexorer och demultiplexorer för att överföra data. En liten notering här, om man räknar samman 4 x 8 Mbit/s så borde man få 32 Mbit/s, och inte 34 Mbit/s. Förklaringen är att det krävs viss kapacitet för kontroll av förbindelsens klockning, likadant i fallet med 4 x 34 Mbit/s blir om man räknar 136 Mbit/s, och inte 140 Mbit/s.

SDH bygger på en transmissions hastighet av 155 Mbit/s (STM-1), det finns även högre transmissionshastighter, 622 Mbit/s (STM-4) och 2488 Mbit/s (STM-16). Den stora fördelen är dock att man ur en STM-1 förbindelse direkt kan plocka ut en 2 Mbit/s förbindelse utan att behöva demultiplexera ned denna först i ett antal steg.

Ovan båda nämnda tekniker används för transmission av data på en specifik plattform, av denna plattform bygger man sedan transmissionsnät. Dessa nät består av ett antal nätelement som har olika uppgifter i nätet. De två grundläggande nätelementen är noder och bärare. En bärare är det fysiska element i vilken själva överföringen sker mellan två platser. En bärare kan vara av olika typ, det kan antingen vara egen eller hyrd av något annat företag (sk leased line). Den kan vara antingen av typen mark- sjökabel eller radio länk, kan vara av typen koppar förbindelse eller optisk kabel. En bärare har alltid två ändar A och B. I varje ända sitter en nod. Noden har till uppgift att antingen dirigera trafiken från en bärare till en annan bärare, eller att koppla in kund utrustning som exempelvis modem.

2.8. LARMTOLKNING

Larmtolkning är den process i vilken en operatör genom sin erfarenhet, kunskap, tillgänglig dokumentation och tillgängliga stödsystem tolkar larm som uppstår på en specifik transmissions plattform. Ett larm uppstår då någonting har blivit fel, genom larmtolkning skall operatören kunna vidtaga rätt åtgärd för att rätta till felet. Larm kan genereras av många olika orsaker, det finns ett stort antal larm som kan uppkomma och likaså kan antalet larm variera beroende av vad det är som är fel.

När ett larm uppstår ser operatören detta på sin NMS (Network Management System) terminal. Varje transmissionsplattform har sitt eget NMS system, vilket gör att det ibland kan vara aktuellt att använda flera NMS system för att skaffa fram information om vilket fel som har uppstått, detta då det gäller förbindelser som använder sig av flera olika teknikplattformar. Operatören försöker med hjälp av dessa system att få fram information om var felet uppstått dels var i vilket land och stad, dels på vilken typ av nätelement som larmet uppstått. Vidare kan sedan operatören plocka fram en lista på vilka larmar som uppkommit på detta nätelement. Det är dessa larmar som sedan skall tolkas i ovan nämnda process. En fullständig lista på de larm som operatören ser på sin terminal återfinns i appendix A. Bilden 2.1. nedan visar huvuddelarna i den process som vidtas då ett fel har uppstått på en förbindelse, vidare följer också en mer utförlig beskrivning av de ingående aktiviteterna.

Bild 2.1. Åtgärd av felsituation

En mer utförlig beskrivning av de i bilden ovan beskrivna aktiviterna följer nedan:

1. Det första som händer är att ett fel uppstår någonstans, antingen upptäcks detta av något NMS system, eller också ringer kunden och anmäler att det är fel på hans förbindelse. Övervakningscentralen öppnar här en felanmälan i ett felanmälninssystem, denna felanmälan innehåller all information som kan tänkas ha med det aktuella felet att göra.

2. Efter det att felanmälan har öppnats startar arbetet med att samla in information kring den aktuella felsituationen. Detta arbete innefattar att ta reda på vilka larm som uppkommit, att göra larmtolkning på dessa, ta reda på om liknande situationer uppkommit förut. Det är i detta steg som det system som utvecklas inom ramen för detta arbete kommer att användas.

3. När all felsökning och informationshantering är avslutad så uppdateras den felanmälan som har med situationen att göra. Man vet här vad för typ av fel man har, samt vilken åtgärd som är lämplig att vidta.

4. I detta steg lämnas felet över till den person, larmcentral eller teleoperatör som har till uppgift att åtgärda felet. Till vem det överlämnas är beroende av vad det är för typ av fel, samt var felet är. Det kan även röra sig om flera instanser. Den aktuella felanmälan sätts hos övervakningscentralen i bevakningsläge, dvs inget arbete utförs av övervakningscentralen men man har fortfarande kontroll på den.

5. I detta steg åtgärdas felet. Detta är något som ligger utanför övervakningscentralens arbetsområde.

6 och 7. När aktuell instans har åtgärdat felet så meddelas detta övervakningscentralen. Man upptar åter arbetet med felanmälan och uppdaterar med den information som man fått från den instans som åtgärdat felet. Vid kontroll att förbindelsen fungerar och då även kunden informerats om att felet är åtgärdat, samt att han eller hon kontrollerat detta så stängs den felanmälan som har med situationen att göra. Man kan senare gå tillbaka och titta i gamla felanmälningar för referens.

8. I detta skede är förbindelsen åter i drift och fungerar.

3. RESULTAT : Utveckling av larmtolkningssystem med Java

'Experience, the name men give to their misstakes.'
- Oscar Wilde

3.1. TIDIGARE ARBETE INOM EXPERTSYSTEM I JAVA

När jag började med detta arbete visste jag inte om det tidigare fanns gjort något inom området expertsystem implementerade i Java. Efter att ha sökt på internet efter arbeten inom området kom jag i kontakt med Jess som var ett koppling mellan applikationer skrivna i Java och Clips som är ett språk avsett att utveckla expertsystem med. Detta var tyvärr också det enda verktyget som jag fann någon information om, så det kunde aldrig bli frågan om att kunna göra någon form av jämförande studie.

3.2. KONSTRUKTION AV LARMTOLKNINGSSYSTEM

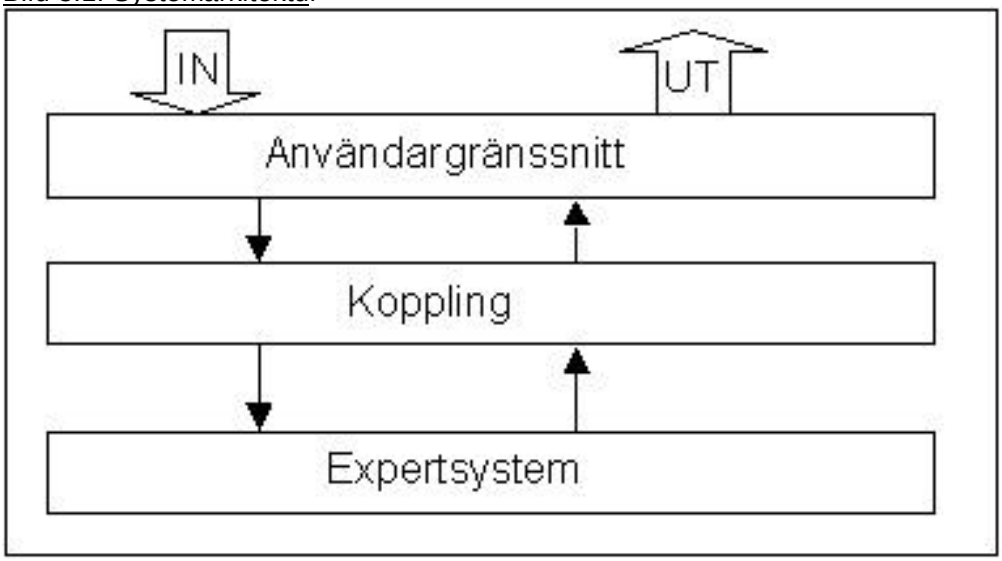
Denna del handlar om hur själva systemet byggs upp, först kommer en genomgång av dessa arkitektur. Därefter följer en detaljerad redogörelse över dess ingående delar. Det körbara systemet med dess regelbas ligger på Unisource intranät och är ej åtkomligt publikt, dock ligger en prototyp körbar på internet, det som skiljer denna från den fulla versionen är att prototypen ej innehåller några regler, användargränssnittet är dock det samma. Dess URL är http://w1.311.telia.com/~u31103017/ExJobb1/ExJobb1.html. Anledningen till att endast en prototyp är tillgänglig för allmänt beskådande på internet istället för det fullständiga programmet är att Unisource ej vill att systemet skall vara tillgängligt för andra teleoperatörer som också använder sig av Martis plattformen för sina tjänster.

3.2.1. Systemarkitektur

Systemet består av tre delar, den första delen är användargränssnittet vilket utformas för bruk över ett intranet, alternativt internet, detta görs i Java. Användargränssnittet skall sköta all interaktion med användaren såsom ut- och inmatning av data. Inmatningen handlar om att tala om för systemet vilka larm som operatören för närvarande har i nätet. Utmatningsdelen skall kunna informera användaren om han eller hon har gjort något fel vid inmatningen, samt att presentera resultatet av expertsystemets larmtolkning för användaren. Nästa del i systemet är kopplingen mellan användargränssnittet och expertsystemet, denna del hanteras av Jess. Syftet med kopplingen är två, dels att bearbeta den i användargränssnittet inmatade informationen och lägga in denna som fakta till expertsystemet, dels att ta emot expertsystemets tolkning av nämnda fakta och sända denna vidare till användargränssnittet. Den sista delen i systemet är själva expertsystemet, detta skrivs i Clips och innehåller de regler som systemet består utav. Reglerna är dock beroende av att ha tillgång till en mängd fakta som reglerna kan tillämpas på. När expertsystemet har tillämpat det befintliga reglerna på den inmatade fakta så sänds information från expertsystemet om resultatet av tolkningen.

På nästa sida återfinns en bild över systemets arkitektur. Pilarna UT respektive IN syftar på in- och utmatning av information från respektive till användaren.

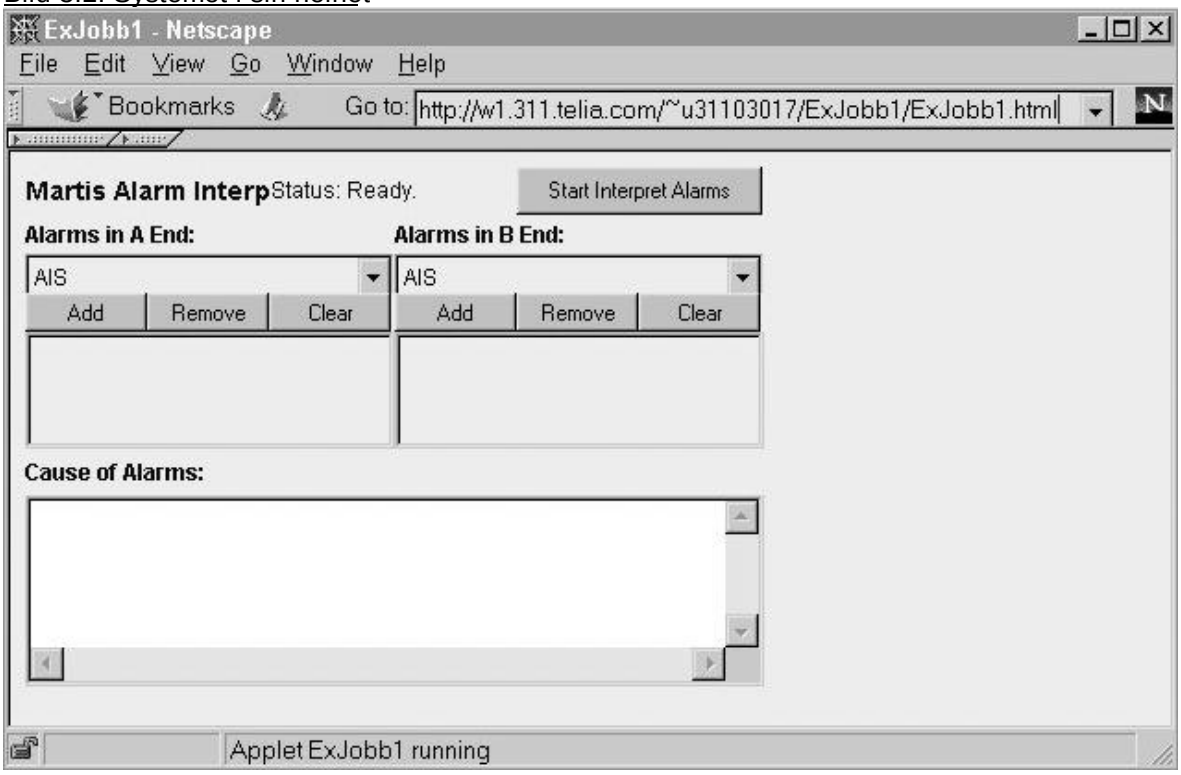
Bild 3.1. Systemarkitektur



3.2.2. Utformning av användargränssnittet

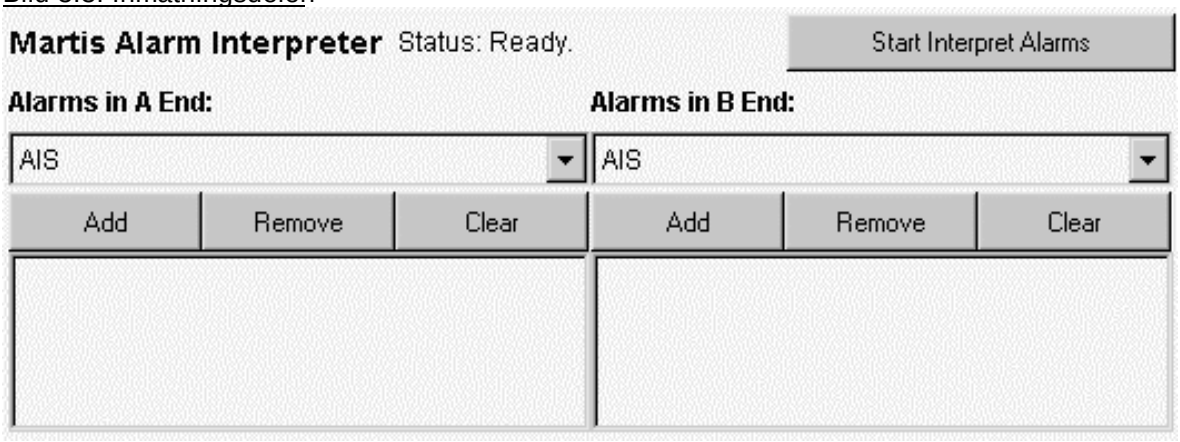
Användargränssnittet byggs upp av två huvuddelar, en del som är avsedd att hantera inmatning från användaren och en del som har till uppgift att presentera uppgifter för användaren. I grunden ligger en web sida (för utformning av sidan se appendix D), denna har en Java applet kopplad till sig (källkoden återfinns i appendix B). Java appleten innehåller de ovan två nämnda delarna. Nedan är en bild som visar hur användargränssnittet ser ut vid exekvering av programmet.

Bild 3.2. Systemet i sin helhet



Föregående bild visar helheten hos användargränssnittet. Nedan följer lite mer detaljerade bilder av de olika i systemet ingående delarna. Inmatningsdelen består av två paneler, en status ruta och en start knapp. Panelerna har till uppgift att lägga till och ta bort uppgifter till en larmlista (för varje panel), det är sedan uppgifterna i denna lista som skall tolkas. Status rutan har till uppgift att informera användaren om systemets nuvarande status. Med start knappen startar man regelbasen, som tolkar uppgifterna i larmlistorna.

Bild 3.3. Inmatningsdelen

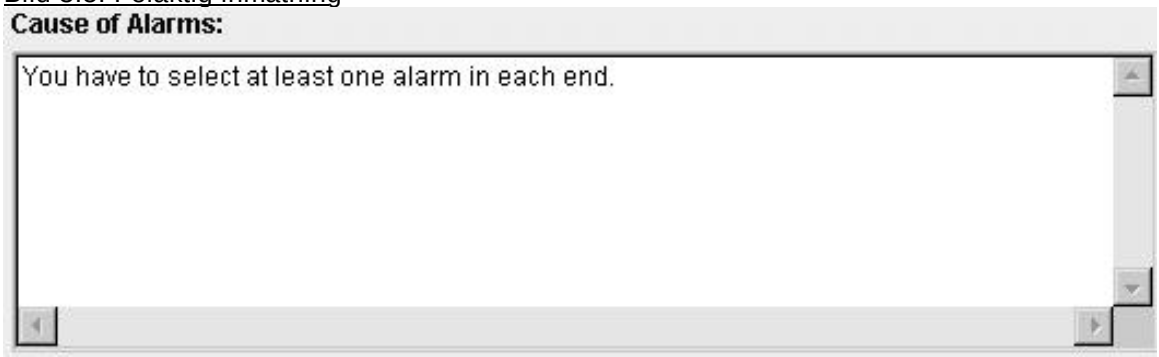


Utmatningsdelen består av en enkel text ruta som talar om för användaren hur regelbasen har tolkat de inmatade uppgifterna. Vid körning av programmet får här användaren all den information som systemet har tagit fram.

Bild 3.4. Utmatningsdelen

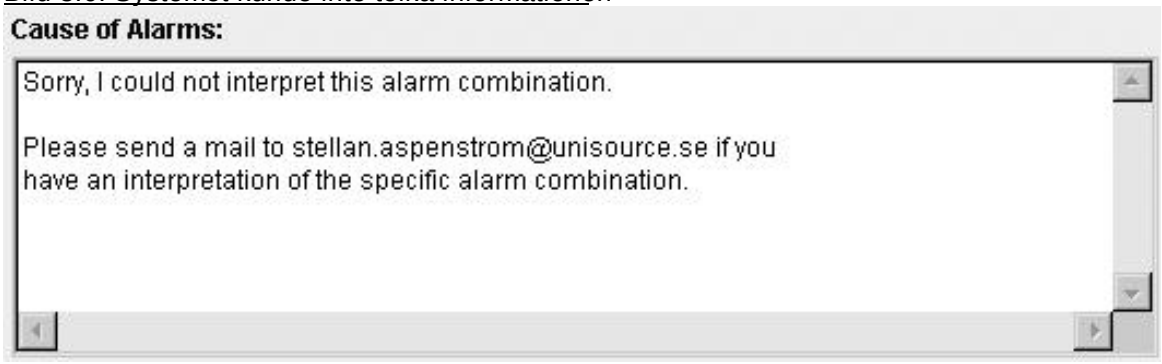
Det som kan hända då användaren har matat in sina uppgifter om aktuella larm och startar tolkningsprocessen är ett av tre alternativ. 1. Användaren kan ha matat inkorrekta uppgifter. 2. Systemet kan inte finna en tolkning av de aktuella larmen. 3. Systemet hittar en tolkning av uppgifterna. I samtliga fall skall användaren få någon information från systemet. I och med att användaren endast kan välja värden från en lista så behövs inte värdenas syntax kontrolleras, utan endast om det existerar eller ej. Om det är så att de ej existerar så har användaren gjort en felaktig inmatning och skall få information om detta enligt nedan:

Bild 3.5. Felaktig inmatning



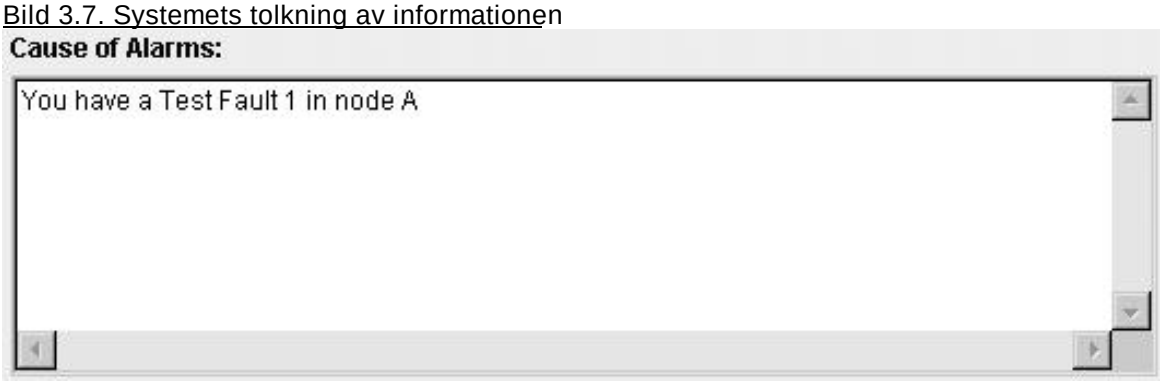
Om det är så att systemet inte kunde göra en tolkning av den information som användaren matat in så skall användaren få ett meddelande om detta samt vad han eller hon kan göra om det är så att han har en tolkning av den sökta situationen. I denna verision har jag valt att lägga in ett meddelande om att användaren kan sända mig ett e-post meddelande om en tolkning för den sökta situationen, därefter kan jag gå in och utöka systemets kunskapsbas efterhand. Denna funktion kan komma att förbättras i senare versioner, se under rubriken 4.4. Utvecklingsmöjligheter för en utförligare diskussion. Meddelande enligt nedan:

Bild 3.6. Systemet kunde inte tolka informationen



När systemet har funnit en tolkning av larmet så skall detta meddelas användaren på ett lämpligt sätt. Detta görs genom att ett meddelande sänds till utmatningsdelen där tolkningen presenteras tillsammans med en förklaring om vad det är som är den bakomliggande orsaken.

Bild 3.7. Systemets tolkning av informationen



3.2.3. Skapande av regelbas

Att skapa en regelbas på vilka fakta skall testas för att kunna ge ett resultat involverar arbete med att inhämta kunskap (expertis) från de experter som finns tillgängliga, samt formalisera denna så att denna kunskap kan användas i ett expertsystem. Hur man skall kunna få fram denna kunskap ur experterna finns det många tillvägagångssätt för, i detta fallet har jag tagit ut gamla larmlistor från övervakningssystemets databas och sedan gått igenom dessa larm tillsammans med en expert för att kunna få fram tolkning av larmen. Nedan följer ett antal olika larmkombinationer, samt den tolkning som gjordes av experten.

Regel 1: Om följande larm finns i nod A respektive nod B:

Nod A	**Nod B**
AIS	AIS
Unavailable state in terms of G.821	Unavailable state in terms of G.821

Så innebär detta ett totalt avbrott på den hyrda förbindelsen mellan noderna.

Regel 2: Om det är så att följande larm återfinns i nod A respektive nod B:

Nod A	**Nod B**
AIS	Frame far end alarm
Unavailable state in terms of G.821	Unavailable state in terms of G.821

Så innebär detta att nod A inte får någon signal från nod B, felet kan antingen vara fel i sändaren i nod B, eller i mottagaren i nod A.

Tyvär har expertens tid varit begränsad under tiden för detta arbete, så delen med utvecklandet av regler har knappt kunnat påbörjas.Ovanstående visar dock principen för hur detta arbete kommer att gå till. Nästa steg blir att skriva om ovanstående regler i Clips, nedan följer hur detta går till med regel 1.

```
(defrule interpretation1 "Total Outage"
  (fault (alarm 23) (node A))
  (fault (alarm 48) (node A))
  (fault (alarm 23) (node B))
  (fault (alarm 48) (node B))
  (counter (alarms 2) (node A))
  (counter (alarms 2) (node B))
  =>
  (printout t "Complete outage on the leased line between
the nodes." crlf))
```

I och ovanstående fel har samma larm så behövs endast regeln läggas in en gång, i de situationer där det är olika larm i nod A respektive nod B så får regeln läggas in två gånger, en för nod A och en för nod B, se appendix C för samtliga reglers utformning.

4. DISKUSSION

'He estimated that he'd covered at least a kilometer before he noticed the light.'
- William Gibson

4.1. FUNKTIONALITETEN HOS JESS OCH CLIPS

Den i detta arbete använda kombinationen Jess - Clips tillsammans med Java är av typen programmeringsspråk, enligt Turbans klassificering av expertsystemstyper. Detta gör att det är inte helt elementärt att skapa ett expertsystem med dessa verktyg, en expert med stor domänkunskap men utan eller med ringa systemutvecklingskunskap skulle inte ensam kunna utveckla ett expertsystem med denna kombination av tekniker. Detta kunde vara möjligt med en annan form av expertsystemtyp såsom expertsystemskal. Vad man dock kan åstadkomma med kombinationen Java-Jess-Clips är att man i inledningsskedet tar hjälp av en systemutvecklare för att utveckla ett expertsystemskal för den specifika uppgiften, med användargränsnitt och koppling till expertsystemet men utan regelbas. Denna regelbas kan sedan utvecklas vidare av experten själv utan hjälp av systemutvecklaren.

Jess är väldigt enkelt att arbeta med som programmerare, det är ett paket bestående av ett antal Java klasser som man importerar till sitt Java program och där kan använda sig av dem, detta på samma sätt som sker med de standard paket som finns i Java.

Clips är ett tämligen enkelt sätt att utveckla expertsystem, och då menar jag enkelt i bemärkelsen att det är lätt att arbeta med. Visserligen arbetar man med att skriva regler direkt till en textfil, och editerar denna direkt. Det finns till exempel ingen debugger som man kan använda sig av för att kontrollera var ett eventuellt fel har uppstått, det finns heller inte något grafiskt användargränssnitt som skulle kunna göra det enkelt för individer med ringa programmerings erfarenhet att utveckla expertsystem, en viss programeringsvana krävs. Vid utveckling av större expertsystem kan det också vara svårt att hålla reda på var man lagt in alla regler i sin Clips källkodsfil, man bör lägga in kommentarer i sin källkod flitigt för att underlätta för framtida utveckling av systemet. En fördel som dock finns med detta utvecklings förfarande är att det lämnar stor frihet åt utvecklaren hur han eller hon vill bygga upp sitt system. Bortsett från att själva kodningen av Clips system är tämligen primitiv, så är det väldigt enkelt att skriva regler och funktioner för sitt system, byggnationen av regler liknar väldigt mycket konstruktionen IF - THEN, en konstruktion som väl alla som sysslat det minsta med programmering kommit i kontakt med, det som skiljer är att flödet av regler inte är sekventiellt, utan om man har en regel som ändrar på värdet av en variabel så kommer de regler som använder sig av denna i IF delen att exekveras. Det finns även nackdelar med Jess-Clips kombinationen. En är att Jess endast kan tolka en delmängd av den totala mängden Clips kommandon och funktioner som finns. För de delar som har med utveckling av expertsystem är dock stödet i Jess fullt. Andra Clips tolkar kan dock använda sig av system skriva i Clips för Jess. Nackdelen med detta är att man inte kan ta ett Clips system som man kanske tidigare utvecklat för en annan miljö och direkt köra detta genom Jessi Java, man får i ett sådant fall först kontrollera att de funktioner man använt sig av stöds av Jess. Problemet är dock inte det omvända, har man skrivit ett Clips system för Jess miljön kan detta direkt köras av tolkar som stöder Clips fullt ut.

4.2. EXPERTSYSTEMSUTVECKLING MED JAVA

Det första problemet var att ta reda på om det fanns något gjort inom området expertsystem och Java, det fanns det i Jess-Clips kombinationen, nackdelen var att jag inte fann något ytterligare verktyg som man kunde göra en jämförelse med. Fördelen var att man slapp arbetet med att utveckla ett expertsystem från grunden med regeltolkar och kunskapstolkar, utan mycket av arbetet var reda gjort i och med detta.

Det andra problemet blev i och med att jag fann ett verktyg att utvärdera detta, en utvärdering av huruvida det är lämpligt att skapa ett expertsystem med Java från intet gjordes därmed ej. En stor del av diskussionen kring detta gjordes under 4.1., sammanfattningsvis kan sägas om detta att det finns en del svagheter med kombinationen Jess-Clips såsom felsökningsfunktioner och att stödet i Jess för Clips är ofullständigt. Dock erbjuder kombinationen ett enkelt sätt att utveckla expertsystem med, kombinerat med Java så ger detta ett väldigt flexibelt utvecklingsverktyg. Svaret på problemet blir att verktyget erbjuder en god utvecklingsmiljö, om än lite stökig att arbeta med.

Om det är lämpligt att använda sig av Java när man skall utveckla ett expertsystem, är lite problematiskt att besvara med ett odelat ja eller nej. Det finns på marknaden idag ett antal andra verktyg att utveckla expertsystem med så det saknas ej alternativ då det gäller val av utvecklingsmiljö, exempel på ett sådant verktyg är Exsys. Det är snarare situationen som får avgöra om det är lämpligt eller ej att använda sig av Java som utvecklingsverktyg. Den första situationen där det kan vara lämpligt att använda sig av Java som utvecklingsverktyg för expertsystem är om stor del av övrig systemutveckling sker med hjälp av Java, i ett sådant fall finns det ingen anledning att byta utvecklingsplattform för just expertsystems ändamålet. Det räcker här med att använda sig av det stöd för expertsystem som kombinationen Jess-Clips ger.

Nästa situation som är det är lämpligt att använda sig av är om man skall utveckla ett expertsystem vilket skall vara åtkomligt över internet, alternativt över ett intranet. I detta fall utgör kombinationen Java-Jess-Clips ett tämligen lättanvänt koncept. Ett alternativ om man vill kunna använda sig av ett expertsystem i internet/intranet miljö skulle vara att använda sig av en Client-Server lösning, där expertsystemet utvecklas för att köras på en server, denna kan sedan klienter utvecklade med t.ex. dynamiska web sidor koppla upp sig mot. Om detta är en bättre-sämre, enklare-svårare lösning än den tidigare nämnda med Java-Jess-Clips avstår jag att svara på då jag ej tittat på en lösning utvecklat med detta alternativ.

Sammanfattningsvis kan man säga att om det är så att man har en miljö där man utvecklar system att köras i en miljö uppbyggd med internet teknik så utgör kombinationen Java-Jess-Clips ett bra alternativ.

4.3. UTVECKLINGSMÖJLIGHETER

Det finns flera möjligheter att utveckla detta system, de främsta möjligheterna är följande: Utveckling av systemets kunskapsbas för att kunna identifiera fler typer av larm, detta är någon som också kommer att ske fortlöpande efter det att systemet har implementerats i användarmiljön. Till en början kommer alla tillägg till systemet att göras av utvecklaren, i ett senare skede skulle det dock vara möjligt att kunna lägga till nya tolknings regler dynamiskt till systemet. Detta skulle exempelvis kunnas göra med en web sida kopplad till ett program som uppdaterar den textfil som innehåller systemets regelbas. Ett mellan steg mellan den nuvarande och den nyss nämnda situationen är att utbilda experterna i Clips så att de enkelt själva kan göra tillägg till regelbasen.

Nästa utvecklingsmöjlighet är utveckling av systemets förmågatt tolka andra typer av felsituationer där förbindelserna inte enbart består utav ett interface som larmar i vardera änden av förbindelsen, utan av tre interface i vardera änden, detta förekommer på sk korskopplings förbindelser, där finns det ett fysiskt interface, ett logiskt interface samt ett korskopplings interface som alla tre kan larma vid fel.

En tredje utvecklingsmöjlighet är att utöka de transmissionsplattformar som systemet kan tolka larm på till de på övervakningscetralen förekommande plattformarna, principen är den samma, dock har larmen som där kan förekomma andra namn, men i grund och botten handlar det om samma sak.

5. REFERENSER

'Allt går igen, allt.'
- August Strindberg

Campione, M., Walrath, K. (1996).*The Java Tutorial*. Addison-Wesley.

Friedman-Hill E. J., (1997).*Jess, The Java Expert System Shell.*
URL: http://herzberg.ca.sandia.gov/jess/README.html (hämtad 19971130).

Gosling, J., Joy, B., Steele, G. (1996).*The Java Language Specification.* Addison-Wesley.

JSC-25012. CLIPS Basic Programming Guide. Lyndon B. Johnson Space Center, Software Technology Branch.

Knight, K., Rich, E. (1991).*Artificial Intelligence.* McGraw-Hill, Inc.

Luconi, F. L., Malone, T. W. & Morton, M. S. (1986). ”Expert Systems: The Next Challange for Managers”.*Sloane Management Review, Summer 1986, 3-14* .

Turban, E. (1995).*Decision Support and Expert Systems*. Prentice-Hall International, Inc.

APPENDIX A

Nedan listade är de larm som kan uppkomma i Martis systemet.

Id	Alarm Namn
1	Unpredicted fault
2	Reset
3	Installation error
4	Power supply faults
5	Memory faults
6	Missing subracks
7	Extra subracks
8	Missing units
9	Extra units
10	Incorrect or missing interface module
11	Fatal problems protected signal(1+1)
12	Missing interface signal
13	BER 10E-3
14	BER 10E-4
15	BER 10E-5
16	BER 10E-6
17	BER 10E-7
18	BER 10E-8
19	BER 10E-9
20	Frame far end alarm
21	Multiframe far end alarm
22	CRC errors from far end
23	AIS
24	Frame alignment lost
25	Multiframe alignment lost
26	CRC faults
27	Loops
28	IA faults
29	Unexpected neighbour node
30	Buffer slips
31	Clock far end alarm
32	ASIC problems
33	X-Connect bus problem
34	Master clock problems
35	Time controlled connections
36	Start permission problems
37	Opto component problems
38	Bus sync fault
39	Clock fault
40	Missing IA activity
41	Configured bit-overlap
42	AIS from X-bus
43	AIS in signaling

44 Local VTP bus problems
45 Cluster VTP bus problems
46 Fatal problems in protected SXUs
47 Fault data base reinitialization
48 Unavailable state in terms of G.821
49 Signal quality alarm
50 Missing carrier
51 Line scanning
52 DTE scanning
53 Protection switch in position 1+1
54 Protection switch in position SXUs
55 Incompatible system and application SW
56 NTU line fault
57 NTU power fault
58 Fault masks
59 Line tests
60 Performance event
61 Fault data base reconfiguration
62 General HW fault
63 IF blocked
64 Input level error
65 Output level error
66 Signalling error
67 Errors in Tx/Rx signal
68 External alarm
69 Setup conflict

APPENDIX B

I detta appendix återfinns källkoden till användargränssnittet som är skrivet i Java. Det finns även en del kommetarer inlaggda. Programmet består av tre klasser, jag vill dock påpeka att jag inte till etthundra procent använt mig av objektorientering här, utan det finns inslag av sekventiellt tänkande. Kommentarer markeras med att raden börjar med //, samt att texten är något större än själva koden.

```
// Import av standard klasser, samt Jess klasserna
import java.awt.*;
import java.applet.Applet;
import java.applet.AppletContext;
import java.io.*;
import java.net.*;
import jess.*;

// Huvudprogrammet
public class ExJobb1 extends Applet implements Runnable {
        private Button btnStart=new Button("Start Interpret Alarms");
        private TextArea txtCause=new TextArea(6,80);
        private alarmPanel aPanelA = new alarmPanel("A");
        private alarmPanel aPanelB = new alarmPanel("B");
        private Label statusLabel=new Label();
        private Rete rete;
        private Jesp jesp;
        private LostDisplay ld;
        private PrintStream out;
        private boolean firstTime=true;
        private String TEXT_READY="Status: Ready.";
        private String TEXT_WORKING="Status: Interpreting Alarms...";
        private String TEXT_LOADING="Status: Loading Rulebase...";

        // Tilldelning av variabler som Jess behöver
        public void init(){
                out=new PrintStream(new TextAreaOutputStream(txtCause),true);
                ld=new LostDisplay(out,System.in,this);
                rete=new Rete(ld);
        }

        public void start() {
                drawScreen();
        }

        // Funktion för att rita upp användargränssnittet
        private void drawScreen(){
                Label lbl=new Label();
                Label lbl1=new Label();
                Panel p1=new Panel();
                Panel p2=new Panel();
                Panel p3=new Panel();
                Panel p=new Panel();
                setLayout(new BorderLayout());
                setBackground(new Color(240,240,208));
                statusLabel.setFont(new Font("Helvetica",Font.PLAIN,12));
                statusLabel.setBackground(new Color(240,240,208));
                statusLabel.setForeground(new Color(0,0,128));
                statusLabel.setText(TEXT_READY);
                lbl.setFont(new Font("Helvetica",Font.BOLD,14));
                lbl.setBackground(new Color(240,240,208));
                lbl.setForeground(new Color(0,0,128));
                lbl.setText("Martis Alarm Interpreter");
                lbl1.setFont(new Font("Helvetica",Font.BOLD,12));
                lbl1.setBackground(new Color(240,240,208));
                lbl1.setForeground(new Color(0,0,128));
                lbl1.setText("Cause of Alarms:");
                txtCause.setFont(new Font("Helvetica",Font.PLAIN,12));
```

```
            txtCause.setBackground(new Color(240,240,208));
            txtCause.setForeground(new Color(0,0,0));
            txtCause.setText("");
            p1.setLayout(new GridLayout(1,3));
            p1.setBackground(new Color(240,240,208));
            p1.add(aPanelA);
            p1.add(aPanelB);
            p2.setLayout(new GridLayout(1,2));
            p2.setBackground(new Color(240,240,208));
            p2.add(lbl);
            p2.add(statusLabel);
            p2.add(btnStart);
            p3.setLayout(new BorderLayout());
            p3.setBackground(new Color(240,240,208));
            p3.add("North",lbl1);
            p3.add("Center",txtCause);
            p.setLayout(new GridLayout(2,1));
            p.setBackground(new Color(240,240,208));
            p.add(p1);
            p.add(p3);
            add("North",p2);
            add("Center",p);
            validate();
}

public void stop() {

}
```

// Funktion för att göra en syntaktisk analys av regelbas filen, samt inläsning av dito

```
private void parseInFile() {
            statusLabel.setText(TEXT_LOADING);
            try {
                        InputStream is=new URL(getDocumentBase(),"rulebase.clp").openStream();
                        jesp=new Jesp(is,rete);
                        jesp.parse(false);
                        is.close();
            } catch (ReteException e) {
                        txtCause.setText("Error while parsing file 'rulebase.clp'.");
            } catch (IOException e) {
                        txtCause.setText("Error while loading file 'rulebase.clp'.");
            }
            statusLabel.setText(TEXT_READY);
}

public void run() {
   txtCause.setText("");
   if (firstTime==true) {
            parseInFile();
            firstTime=false;
   }
   statusLabel.setText(TEXT_WORKING);
   try {
      rete.ExecuteCommand("(reset)");
      for (int i=0; i<aPanelA.alarmList.countItems();i++) {
        rete.ExecuteCommand("(assert (fault (alarm "+getAlarmNumber(aPanelA.alarmList.getItem(i))+") (node A)))");
      }
      for (int i=0; i<aPanelB.alarmList.countItems();i++) {
                rete.ExecuteCommand("(assert (fault (alarm "+getAlarmNumber(aPanelB.alarmList.getItem(i))+") (node B)))");
      }
      rete.ExecuteCommand("(assert (counter (alarms "+aPanelA.alarmList.countItems()+") (node A)))");
      rete.ExecuteCommand("(assert (counter (alarms "+aPanelB.alarmList.countItems()+") (node B)))");
      rete.ExecuteCommand("(run)");
   } catch (Throwable ex) {
      txtCause.setText("Error while running file 'rulebase.clp'.");
   }
   statusLabel.setText(TEXT_READY);
   if (txtCause.getText().length()==0) {
            txtCause.setText("Sorry, I could not interpret this alarm combination.\n\n");
            txtCause.appendText("Please send a mail to stellan.aspenstrom@unisource.se if you\n");
            txtCause.appendText("have an interpretation of the specific alarm combination.");
   }
```

```
}

// Hantering vid tryck på start knappen
public boolean action(Event e,Object arg) {
	if (e.target==btnStart && aPanelA.alarmList.countItems()>0 && aPanelB.alarmList.countItems()>0) {
		new Thread(this).start();
		return true;
	}
	if (e.target==btnStart || aPanelA.alarmList.countItems()==0 && aPanelB.alarmList.countItems()==0) {
		txtCause.setText("You have to select at least one alarm in each end.");
	}
	return false;
}

// Funktion för att översätta en sträng med felbeskrivning till alarmid
private int getAlarmNumber(String inString) {
	int tmpNumber=0;
	if (inString=="AIS") {tmpNumber=23;}
	if (inString=="AIS from X-bus") {tmpNumber=42;}
	if (inString=="AIS in signaling") {tmpNumber=43;}
	if (inString=="ASIC problems") {tmpNumber=32;}
	if (inString=="BER 10E-3") {tmpNumber=13;}
	if (inString=="BER 10E-4") {tmpNumber=14;}
	if (inString=="BER 10E-5") {tmpNumber=15;}
	if (inString=="BER 10E-6") {tmpNumber=16;}
	if (inString=="BER 10E-7") {tmpNumber=17;}
	if (inString=="BER 10E-8") {tmpNumber=18;}
	if (inString=="BER 10E-9") {tmpNumber=19;}
	if (inString=="Buffer slips") {tmpNumber=30;}
	if (inString=="Bus sync fault") {tmpNumber=38;}
	if (inString=="Clock far end alarm") {tmpNumber=31;}
	if (inString=="Clock fault") {tmpNumber=39;}
	if (inString=="Cluster VTP bus problems") {tmpNumber=45;}
	if (inString=="Configured bit-overlap") {tmpNumber=41;}
	if (inString=="CRC errors from far end") {tmpNumber=22;}
	if (inString=="CRC faults") {tmpNumber=26;}
	if (inString=="DTE scanning") {tmpNumber=52;}
	if (inString=="Errors in Tx/Rx signal") {tmpNumber=67;}
	if (inString=="External alarm") {tmpNumber=68;}
	if (inString=="Extra subracks") {tmpNumber=7;}
	if (inString=="Extra units") {tmpNumber=9;}
	if (inString=="Fatal problems in protected SXUs") {tmpNumber=46;}
	if (inString=="Fatal problems protected signal(1+1)") {tmpNumber=11;}
	if (inString=="Fault data base reconfiguration") {tmpNumber=61;}
	if (inString=="Fault data base reinitialization") {tmpNumber=47;}
	if (inString=="Fault masks") {tmpNumber=58;}
	if (inString=="Frame alignment lost") {tmpNumber=24;}
	if (inString=="Frame far end alarm") {tmpNumber=20;}
	if (inString=="General HW fault") {tmpNumber=62;}
	if (inString=="IA faults") {tmpNumber=28;}
	if (inString=="IF blocked") {tmpNumber=63;}
	if (inString=="Incompatible system and application SW") {tmpNumber=55;}
	if (inString=="Incorrect or missing interface module") {tmpNumber=10;}
	if (inString=="Input level error") {tmpNumber=64;}
	if (inString=="Installation error") {tmpNumber=3;}
	if (inString=="Line scanning") {tmpNumber=51;}
	if (inString=="Line tests") {tmpNumber=59;}
	if (inString=="Local VTP bus problems") {tmpNumber=44;}
	if (inString=="Loops") {tmpNumber=27;}
	if (inString=="Master clock problems") {tmpNumber=34;}
	if (inString=="Memory faults") {tmpNumber=5;}
	if (inString=="Missing carrier") {tmpNumber=50;}
	if (inString=="Missing IA activity") {tmpNumber=40;}
	if (inString=="Missing interface signal") {tmpNumber=12;}
	if (inString=="Missing subracks") {tmpNumber=6;}
	if (inString=="Missing units") {tmpNumber=8;}
	if (inString=="Multiframe alignment lost") {tmpNumber=25;}
	if (inString=="Multiframe far end alarm") {tmpNumber=21;}
	if (inString=="NTU line fault") {tmpNumber=56;}
	if (inString=="NTU power fault") {tmpNumber=57;}
	if (inString=="Opto component problems") {tmpNumber=37;}
	if (inString=="Output level error") {tmpNumber=65;}
```

```
                if (inString=="Performance event") {tmpNumber=60;}
                if (inString=="Power supply faults") {tmpNumber=4;}
                if (inString=="Protection switch in position 1+1") {tmpNumber=53;}
                if (inString=="Protection switch in position SXUs") {tmpNumber=54;}
                if (inString=="Reset") {tmpNumber=2;}
                if (inString=="Setup conflict") {tmpNumber=69;}
                if (inString=="Signal quality alarm") {tmpNumber=49;}
                if (inString=="Signalling error") {tmpNumber=66;}
                if (inString=="Start permission problems") {tmpNumber=36;}
                if (inString=="Time controlled connections") {tmpNumber=35;}
                if (inString=="Unavailable state in terms of G.821") {tmpNumber=48;}
                if (inString=="Unexpected neighbour node") {tmpNumber=29;}
                if (inString=="Unpredicted fault") {tmpNumber=1;}
                if (inString=="X-Connect bus problem") {tmpNumber=33;}
                return tmpNumber;
        }
}
```

// Klass för att hantera alarm panelerna

```
class alarmPanel extends Panel {
        private Label lbl=new Label();
        private Button btnAdd=new Button("Add");
        private Button btnRemove=new Button("Remove");
        private Button btnClear=new Button("Clear");
        private Choice chList=new Choice();
        public List alarmList=new List();
        public martisAlarm alarm[]=new martisAlarm[100];
        public int alarms=0;

        public alarmPanel(String lblText) {
                Panel p1=new Panel();
                Panel p2=new Panel();
                Panel p3=new Panel();
                setLayout(new GridLayout(2,1));
                setBackground(new Color(240,240,208));
                lbl.setBackground(new Color(240,240,208));
                lbl.setFont(new Font("Helvetica",Font.BOLD,12));
                lbl.setForeground(new Color(0,0,128));
                lbl.setText("Alarms in "+lblText+" End:");
                chList.setBackground(new Color(240,240,208));
                chList.setFont(new Font("Helvetica",Font.PLAIN,12));
                chList.setForeground(new Color(0,0,0));
                alarmList.setBackground(new Color(240,240,208));
                alarmList.setFont(new Font("Helvetica",Font.PLAIN,12));
                alarmList.setForeground(new Color(0,0,0));
                p3.setLayout(new GridLayout(1,3));
                p3.setBackground(new Color(240,240,208));
                p3.add(btnAdd);
                p3.add(btnRemove);
                p3.add(btnClear);
                p1.setLayout(new GridLayout(3,1));
                p1.setBackground(new Color(240,240,208));
                p1.add(lbl);
                p1.add(chList);
                p1.add(p3);
                p2.setLayout(new BorderLayout());
                p2.setBackground(new Color(240,240,208));
                p2.add("Center",alarmList);
                add(p1);
                add(p2);
                createChoiceList();
                validate();
        }

        public boolean action(Event e,Object arg) {
                if (e.target==btnClear) {
                        alarmList.clear();
                        return true;}
                if (e.target==btnRemove) {
                        if (alarmList.getSelectedIndex()>-1) {
                                alarmList.delItem(alarmList.getSelectedIndex());
                        }
                        return true;
                }
```

```
		if (e.target==btnAdd && exists(chList.getSelectedItem())==false) {
			alarmList.addItem(chList.getSelectedItem());
			return true;}
		return false;
	}

	private boolean exists(String item) {
		boolean tmpValue=false;
		for (int i=0;i<alarmList.countItems();i++) {
			if (alarmList.getItem(i).compareTo(item)==0) {tmpValue=true;}
		}
		return tmpValue;
	}

	private void createChoiceList() {
		alarms=0;
		alarm[alarms]=new martisAlarm(23,"AIS");
		alarms++;
		alarm[alarms]=new martisAlarm(42,"AIS from X-bus");
		alarms++;
		alarm[alarms]=new martisAlarm(43,"AIS in signaling");
		alarms++;
		alarm[alarms]=new martisAlarm(32,"ASIC problems");
		alarms++;
		alarm[alarms]=new martisAlarm(13,"BER 10E-3");
		alarms++;
		alarm[alarms]=new martisAlarm(14,"BER 10E-4");
		alarms++;
		alarm[alarms]=new martisAlarm(15,"BER 10E-5");
		alarms++;
		alarm[alarms]=new martisAlarm(16,"BER 10E-6");
		alarms++;
		alarm[alarms]=new martisAlarm(17,"BER 10E-7");
		alarms++;
		alarm[alarms]=new martisAlarm(18,"BER 10E-8");
		alarms++;
		alarm[alarms]=new martisAlarm(19,"BER 10E-9");
		alarms++;
		alarm[alarms]=new martisAlarm(30,"Buffer slips");
		alarms++;
		alarm[alarms]=new martisAlarm(38,"Bus sync fault");
		alarms++;
		alarm[alarms]=new martisAlarm(31,"Clock far end alarm");
		alarms++;
		alarm[alarms]=new martisAlarm(39,"Clock fault");
		alarms++;
		alarm[alarms]=new martisAlarm(45,"Cluster VTP bus problems");
		alarms++;
		alarm[alarms]=new martisAlarm(41,"Configured bit-overlap");
		alarms++;
		alarm[alarms]=new martisAlarm(22,"CRC errors from far end");
		alarms++;
		alarm[alarms]=new martisAlarm(26,"CRC faults");
		alarms++;
		alarm[alarms]=new martisAlarm(52,"DTE scanning");
		alarms++;
		alarm[alarms]=new martisAlarm(67,"Errors in Tx/Rx signal");
		alarms++;
		alarm[alarms]=new martisAlarm(68,"External alarm");
		alarms++;
		alarm[alarms]=new martisAlarm(7,"Extra subracks");
		alarms++;
		alarm[alarms]=new martisAlarm(9,"Extra units");
		alarms++;
		alarm[alarms]=new martisAlarm(46,"Fatal problems in protected SXUs");
		alarms++;
		alarm[alarms]=new martisAlarm(11,"Fatal problems protected signal(1+1)");
		alarms++;
		alarm[alarms]=new martisAlarm(61,"Fault data base reconfiguration");
		alarms++;
		alarm[alarms]=new martisAlarm(47,"Fault data base reinitialization");
		alarms++;
		alarm[alarms]=new martisAlarm(58,"Fault masks");
		alarms++;
```

```
alarm[alarms]=new martisAlarm(24,"Frame alignment lost");
alarms++;
alarm[alarms]=new martisAlarm(20,"Frame far end alarm");
alarms++;
alarm[alarms]=new martisAlarm(62,"General HW fault");
alarms++;
alarm[alarms]=new martisAlarm(28,"IA faults");
alarms++;
alarm[alarms]=new martisAlarm(63,"IF blocked");
alarms++;
alarm[alarms]=new martisAlarm(55,"Incompatible system and application SW");
alarms++;
alarm[alarms]=new martisAlarm(10,"Incorrect or missing interface module");
alarms++;
alarm[alarms]=new martisAlarm(64,"Input level error");
alarms++;
alarm[alarms]=new martisAlarm(3,"Installation error");
alarms++;
alarm[alarms]=new martisAlarm(51,"Line scanning");
alarms++;
alarm[alarms]=new martisAlarm(59,"Line tests");
alarms++;
alarm[alarms]=new martisAlarm(44,"Local VTP bus problems");
alarms++;
alarm[alarms]=new martisAlarm(27,"Loops");
alarms++;
alarm[alarms]=new martisAlarm(34,"Master clock problems");
alarms++;
alarm[alarms]=new martisAlarm(5,"Memory faults");
alarms++;
alarm[alarms]=new martisAlarm(50,"Missing carrier");
alarms++;
alarm[alarms]=new martisAlarm(40,"Missing IA activity");
alarms++;
alarm[alarms]=new martisAlarm(12,"Missing interface signal");
alarms++;
alarm[alarms]=new martisAlarm(6,"Missing subracks");
alarms++;
alarm[alarms]=new martisAlarm(8,"Missing units");
alarms++;
alarm[alarms]=new martisAlarm(25,"Multiframe alignment lost");
alarms++;
alarm[alarms]=new martisAlarm(21,"Multiframe far end alarm");
alarms++;
alarm[alarms]=new martisAlarm(56,"NTU line fault");
alarms++;
alarm[alarms]=new martisAlarm(57,"NTU power fault");
alarms++;
alarm[alarms]=new martisAlarm(37,"Opto component problems");
alarms++;
alarm[alarms]=new martisAlarm(65,"Output level error");
alarms++;
alarm[alarms]=new martisAlarm(60,"Performance event");
alarms++;
alarm[alarms]=new martisAlarm(4,"Power supply faults");
alarms++;
alarm[alarms]=new martisAlarm(53,"Protection switch in position 1+1");
alarms++;
alarm[alarms]=new martisAlarm(54,"Protection switch in position SXUs");
alarms++;
alarm[alarms]=new martisAlarm(2,"Reset");
alarms++;
alarm[alarms]=new martisAlarm(69,"Setup conflict");
alarms++;
alarm[alarms]=new martisAlarm(49,"Signal quality alarm");
alarms++;
alarm[alarms]=new martisAlarm(66,"Signalling error");
alarms++;
alarm[alarms]=new martisAlarm(36,"Start permission problems");
alarms++;
alarm[alarms]=new martisAlarm(35,"Time controlled connection");
alarms++;
alarm[alarms]=new martisAlarm(48,"Unavailable state in terms of G.821");
alarms++;
```

```
                alarm[alarms]=new martisAlarm(29,"Unexpected neighbour node");
                alarms++;
                alarm[alarms]=new martisAlarm(1,"Unpredicted fault");
                alarms++;
                alarm[alarms]=new martisAlarm(33,"X-Connect bus problem");
                alarms++;
                for (int i=0;i<alarms;i++) {
                        chList.addItem(alarm[i].getName());
                }
        }
}
```

// Klass för att beskriva ett alarm

```
class martisAlarm {
        private String alarmName="";
        private int alarmId=0;

        public martisAlarm(int alarmId,String alarmName) {
                this.alarmName=alarmName;
                this.alarmId=alarmId;
        }

        public String getName() {return alarmName;}

        public int getId() {return alarmId;}

        public boolean equals(int alarmId) {
                if (alarmId==thisalarmId) {
                        return true;
                } else {
                        return false;
                }
        }

        public boolean equals(String alarmName) {
                if (alarmName.trim().compareTo(this.alarmName.trim())==0) {
                        return true;
                } else {
                        return false;
                }
        }
}
```

APPENDIX C

Nedan följer hur själva Expertsystemet är utformat.

```
;This is the rulebase for the Martis Alarm Interpreter

;Template for the fault object
(deftemplate fault "Current faults."
     (slot alarm)
     (slot node))

;Template for holding the number of alarms in each node
(deftemplate counter "Number of alarms."
     (slot alarms)
     (slot node))

;Interpretation rules
(defrule interpretation1 "Total Outage"
  (fault (alarm 23) (node A))
  (fault (alarm 48) (node A))
  (fault (alarm 23) (node B))
  (fault (alarm 48) (node B))
  (counter (alarms 2) (node A))
  (counter (alarms 2) (node B))
  =>
  (printout t "Complete outage on the leased line between the
nodes." crlf))

(defrule interpretation2 "Missing Signal in A"
  (fault (alarm 23) (node A))
  (fault (alarm 48) (node A))
  (fault (alarm 20) (node B))
  (fault (alarm 48) (node B))
  (counter (alarms 2) (node A))
  (counter (alarms 2) (node B))
  =>
  (printout t "Missing signal in node A from node B." crlf))

(defrule interpretation3 "Missing Signal in B"
  (fault (alarm 20) (node A))
  (fault (alarm 48) (node A))
  (fault (alarm 23) (node B))
  (fault (alarm 48) (node B))
  (counter (alarms 2) (node A))
  (counter (alarms 2) (node B))
  =>
  (printout t "Missing signal in node B from node A." crlf))
```

APPENDIX D

Nedan följer hur html koden för själva web sidan som innehåller Java appleten ser ut.

```
<html>
<head>
<title>ExJobb1</title>
</head>
<body bgcolor="#F0F0D0" text="#000080">
<applet
code=ExJobb1
width=525
height=350>
</applet>
</body>
</html>
```