Datavetenskap

Lars-Olof Leander & Daniel Törnqvist

Matrix - ett loggadministreringssystem

Examensarbete, C-nivå

2003-03

Matrix - ett loggadministreringssystem

Lars-Olof Leander & Daniel Törnqvist

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

Lars-Olof Leander

Daniel Törnqvist

Godkänd, 2003-01-15

Handledare: Eivind J. Nordby

Examinator: Martin Blom

Sammanfattning

Den här rapporten är skriven som en del av ett 10 poängs examensarbete på C-nivå. Examensarbetet har utförts av två studenter vid Dataingenjörsprogrammet och genomförts vid Institutionen för datavetenskap vid Karlstads universitet. Examensarbetet bestod av en uppgift given av företaget Veriscan Security AB.

Uppgiften grundas i problemen med att hantera och överblicka den stora mängd loggar som produceras i ett företags datorpark. I uppgiften ingick att undersöka om det gick att konstruera en plattformsoberoende och resurssnål programvara som exporterar information från loggfiler från en datorutrustning till en server. Programvaran är en del i ett större system som går under arbetsnamnet Matrix. I systemet ingår även en server som sparar logginformationen som exporteras. I vår uppgift ingick även att konstruera en prototyp av systemet.

Vi har genom att konstruera en prototyp visat att det är möjligt att konstruera ett plattformsoberoende program som exporterar loggposter till en server. För att åstadkomma plattformsoberoende är vår design baserad på utbytbara delar. Med hjälp av en konfigurationsfil kan programmet anpassas till olika loggsystem.

Matrix – a log administration system

Abstract

Two students attending Karlstad University wrote this report as a part of a bachelor's project. The project was carried out at the Department of computer science. The bachelor project consisted of an assignment given by the company Veriscan Security AB.

The assignment originated in the problem of dealing with the large amount of log files created in a company's computer network. A part of the assignment was to investigate if it was possible to design a platform independent and resource effective software that exports log entries from a computer to a server. The software is part of a larger system called The Matrix system. We were also to implement a prototype of the system. By implementing the prototype we have proven that it's possible to implement platform independent software that exports log entries. To achieve platform independence our design is based on interchangeable parts. The program can be adapted to different log systems with a configuration file.

Innehållsförteckning

Figurförteckning

1 Introduktion

Denna rapport är en del av ett 10 poängs examensarbete på C-nivå som genomförts vid Institutionen för datavetenskap vid Karlstads universitet. Examensarbetet bestod av en uppgift given av företaget Veriscan Security AB.

Detta kapitel syftar till att ge en introduktion till rapporten. Kapitlet börjar med att kort presentera företaget Veriscan och bakgrunden för examensarbetet. Därefter presenteras mål och avgränsningar för uppgiften. Kapitlet avslutas med en genomgång av rapportens struktur.

1.1 Kort bakgrund

Veriscan är ett företag som verkar inom IT-säkerhetsområdet där de huvudsakligen utvärderar andra företags IT-säkerhet. Veriscan har i sitt arbete upptäckt problem i hanteringen av informationen som finns i loggfiler. Dessa loggfiler produceras av de flesta typer av datautrustning som till exempel brandväggar, webbservrar och persondatorer. Loggfilerna innehåller information om händelser och operationer som har utförts på utrustningen.

Matrix-systemet är tänkt att samla all information från utrustningen i nätverket på en plats för att underlätta olika typer av övervakning av utrustningen i nätverket. Systemet består av två beståndsdelar, server och agent. Servern är platsen där all information från loggar samlas. Agenten är ett program som exekveras på varje loggproducerande utrustning i ett nätverk och skickar logginformation från utrustningen till servern.

1.2 Uppgiftens syfte

Syftet med Matrix-systemet är att kunna sammanställa loggfiler från olika datorutrustningar för att få en bättre översikt över informationen i loggfilerna. För att minska risken att en eventuell inkräktare kan radera loggfilen innan informationen i den har blivit exporterad ska exporten av loggposterna ha en så kort fördröjning som möjligt.

Målet för vår uppgift är att undersöka i vilken grad det är möjligt att konstruera en plattformsoberoende Matrix-agent. Med plattformsoberoende menar vi att programmet kan flyttas mellan olika operativsystem utan att behöva kompileras om. För att visa vad vi kommit fram till ska vi konstruera en prototyp till en agent som kan exekveras på flera olika plattformar. Prototypen behöver endast kunna läsa ASCII-filer och Windows 2000 loggfiler

samt kunna exekvera i Windows 2000 och i Redhat Linux. Prototypen bör också vara utbyggbar för att förenkla en vidare utveckling av programmet. För att kunna testa prototypen på ett tillfredställande sätt ska vi också konstruera en enkel server.

1.3 Rapportens struktur

För att ge läsaren underlag för att förstå rapporten börjar den med en bakgrund om loggar, loggproducenter samt de problem som uppstår i samband med dessa. För att förklara vad vår uppgift består av beskrivs sedan uppgiftens uppkomst och den givna uppgiftens mål och förutsättningar. Rapporten fortsätter sedan med att redogöra för lösningens konstruktion och dess olika funktioner. Vidare diskuteras de val och svårigheter vi har konfronterats med under arbetet. För att ge underlag för vidare utveckling av systemet diskuteras avslutningsvis de erfarenheter och slutsatser som arbetet resulterat i.

2 Bakgrund

För att ge en ökad förståelse för resterande delar av rapporten ger detta kapitel en bakgrundsinformation om loggar och vilka problem som uppstår i samband med hantering av dem. Vi kommer att beskriva vad en logg är, vad den innehåller och vad som producerar loggar. De problem som finns när det gäller en effektiv hantering av loggar kommer vi att belysa i slutet av kapitlet.

2.1 Loggar och loggproducenter

Begreppet logg har sitt ursprung i sjöfarten där varje fartyg för en loggbok över viktiga händelser som har inträffat ombord. En loggfil är en fil som liksom loggboken innehåller information om inträffade händelser. Till skillnad från den manuella inskrivningen som krävs i loggböcker genereras loggfiler automatiskt på många datorutrustningar. Det kan vara information om vilka användare som varit inloggade, vilka program som har installerats eller vilka webbsidor som har besökts. Loggfiler är vanligtvis uppbyggda som en textfil där varje post representeras av en rad. Varje post innehåller en tidsangivelse och vilken händelse som har inträffat. Loggfiler genereras av i stort sett all datautrustning som t.ex. brandväggar, persondatorer, epostservrar, webbservrar, routrar och databaser. Även olika mjukvaruapplikationer genererar egna loggfiler.

Syftet med loggfilerna är att förenkla felsökning och systemövervakning. Ett exempel på felsökning är när din PC ”hänger sig”. Då kan du efter omstart läsa i systemets loggfil för att hitta vilket program som orsakat felet.

En nätverksadministratör kan övervaka användandet av nätverket med hjälp av loggfiler i t.ex. brandväggar, routrar och servrar. I brandväggens loggfil kan administratören se exakt vilka platser som har besökts utanför det lokala nätverket och från vilka datorer dessa besök har genomförts. Loggfiler kommer även till användning då det har skett ett dataintrång i nätverket. Nätverksadministratören kan då i loggfilerna se vad inkräktaren har gjort och kanske även identifiera inkräktaren. En inkräktare behöver inte vara en utomstående person som tar sig in i systemet utan kan också vara en person med tillgång till systemet. Då kan intrånget bestå i att använda filer eller andra resurser man inte har befogenhet att använda.

2.2 Problem med loggar

De problem som diskuteras i detta kapitel är sådana problem som personal på Veriscan har upptäckt samt problem vi har kommit i kontakt med under arbetets gång. Vi har namngivit de problem vi diskuterar för att lättare kunna hänvisa till dem senare i rapporten. Namnen på problemen är alltså inte vedertagna begrepp.

Eftersom det finns så många loggproducenter i ett nätverk blir mängden logg snabbt mycket stor. Detta gör det svårt att få en bra översikt över alla dessa loggar. Platsproblemet innebär att datorutrustningen och därmed också loggfilerna finns på olika platser. Det innebär att en administratör måste flytta på sig för att läsa loggfiler från olika loggproducenter. Ett annat problem är formatproblemet. Det innebär att loggar från olika loggproducenter kan ha olika filformat (t.ex. ASCII och binärt), utseende och informationsinnehåll. Platsproblemet och formatproblemet gör det svårt att jämföra loggfiler på olika datorutrustningar. Det är även svårt att se samband mellan samtidiga händelser i olika system eller i olika delar av systemet. Ett exempel som illustrerar dessa problem är då ett intrång sker som genererar logghändelser på flera platser samtidigt. Var för sig kan dessa loggposter vara intetsägande men tillsammans kan de vara tydliga bevis på att ett intrång har skett. Ett annat problem är tidsproblemet. Det innebär att loggproducenterna inte är tidssynkroniserade (d.v.s. samtidiga logghändelser får olika tidsangivelser). Tidsproblemet minskar ytterligare nätverksadministratörens möjligheter att se samband mellan samtidiga händelser. Problemet med raderade loggfiler innebär att en eventuell inkräktare kan radera loggfiler för att på så sätt dölja spåren efter sig.

Ett system där alla loggar samlas in i någon form av databas och konverteras till samma utseende kan lösa många av de ovanstående problemen. En systemadministratör kan då granska alla loggposter som systemet genererat på samma plats och göra jämförelser mellan loggposter. Med loggarna i en databas kan man också välja ut vilka loggar man vill granska grundat på olika kriterier som till exempel tidpunkt för händelsen, plats för händelsen eller typ av händelse. Om loggarna dessutom snabbt sparas i databasen minskar man möjligheterna för inkräktare att dölja sina spår.

3 Uppgift och mål

I detta kapitel kommer vi att ta upp hur uppgiften uppstod. Kapitlet innehåller också en beskrivning av hur det tänkta Matrix-systemet ska se ut och fungera. I kapitel 2.2 behandlas ett antal problem som uppstår vid användning av loggar. Dessa problem diskuteras i detta kapitels avsnitt om lösningen av loggproblemen. Diskussionen leder fram till de krav och avgränsningar som sedan presenteras.

3.1 Uppgiftens uppkomst

Veriscan är ett företag som verkar inom IT-säkerhetsområdet. De utvärderar företags IT-säkerhet med en egenutvecklad metodik och föreslår därefter åtgärder. Veriscan delar in sin utvärdering i tre områden; fysisk säkerhet, logisk säkerhet och systemsäkerhet. Fysisk säkerhet innebär skyddet mot fysisk åverkan på datorutrustningen, så som brand och stöld. Med logisk säkerhet menar Veriscan t.ex. hur användare handhar lösenord. Systemsäkerhet avser säkerheten hos de system och mjukvaror som används.

Under sitt arbete ute hos olika företag insåg de anställda vid Veriscan att den information som fanns i loggfilerna inte utnyttjades till fullo. Ett företags systemsäkerhet skulle kunna förbättras genom att samla alla loggar så att systemadministratören ges större möjlighet att utnyttja loggarnas potential. Loggarnas användningsområden och potential beskrivs i den senare delen av kapitel 2.1.

3.2 En beskrivning av det färdiga Matrix-systemet

Huvudkomponenterna i Matrix-systemet är de så kallade agenterna. En agent är ett program som körs på en loggproducerande enhet i nätverket. Dess uppgift är att så snabbt som möjligt

upptäcka nya loggposter i en eller flera loggfiler och därefter skicka de nya loggposterna vidare till en server. All överföring från agenten till servern ska ske på ett säkert sätt i nätverket. Alla loggposter som servern tar emot ska sparas i en databas. Det ska även finnas möjligheter att genom ett grafiskt gränssnitt presentera statistik över innehållet i serverns databas. I databasen ska loggarna ha ett enhetligt utseende. Det är också viktigt att samtidiga händelser får samma tidsangivelse.

3.3 Lösningen av loggproblemen

I avsnitt 2.2 presenterades ett antal problem, möjliga lösningar till dessa kommer att presenteras i detta avsnitt.

Platsproblemet kan lösas genom att man samlar in all information från de olika datorutrustningarna i nätverket till en central server. Då kan administratören sitta vid en dator och läsa loggposter från loggproducenter i hela nätverket.

Ett annat problem som presenterades i 2.2 var formatproblemet. För att lösa detta problem måste agenten kunna läsa flera olika typer av loggfiler. Binärt och ASCII-format är exempel på olika format. Om man dessutom inför möjligheten att välja struktur på loggposterna som skickas från agenten till servern underlättas behandlingen av logginformationen i servern. Dessutom gör det att man i agenten kan välja bort ointressant information i de loggposter man skickar till servern.

Problemet med raderade loggfiler kan minskas med en så liten fördröjning som möjligt mellan en uppdatering av loggfilen och export av den nya loggposten.

Tidsproblemet kan lösas på flera olika sätt. En lösning vore att före en installation av Matrix-systemet säkerställa att alla klockor i nätverkets datorutrustning går rätt. För detta finns färdiga mjukarvarulösningar. En annan lösning vore att agenterna synkroniserade sina klockor med serverns klocka.

En lösning på de tidigare hämnda problemen är att använda Matrix-systemet. För att åstadkomma ett användbart system måste även följande problem beaktas. De flesta nätverk innehåller utrustning med olika typer av operativsystem. Det är därför önskvärt att agenten kan köras på alla dessa operativsystem utan att behöva kompileras om. Att konstruera en agent som kan läsa alla typer av loggfiler och är helt plattformsoberoende är en för stor uppgift för ett 10 poängs examensarbete. Om vi istället konstruerar en prototyp agent som kan läsa loggfiler i ett fåtal format och som kan exekveras på minst två vanliga operativsystem så har vi visat att det är möjligt att konstruera en plattformsoberoende agent. Eftersom vissa

system använder flera loggfiler för att spara sina loggposter bör agenten kunna bevaka flera loggfiler samtidigt. Agenten ska också fungera på äldre datorutrustning utan att allt för mycket försämra prestanda på utrustningen. Därför behöver agenten vara så resurssnål som möjligt.

3.4 Krav på agenten

Diskussionen i avsnitt 3.3 har lett fram till följande krav på agenten:

- Agenten ska kunna exekveras på minst 2 olika plattformar (Windows NT/2000 och Linux) utan att behöva kompileras om.
- Agenten ska kunna exportera loggposter till en server.
- Agenten ska kunna läsa loggfiler av olika typer (olika format).
- Agenten ska kunna konvertera loggposter till en vald struktur.
- Agenten ska ha så liten fördröjning som möjligt mellan uppdatering av loggfilen och export av den nya loggposten.
- Varje agent ska kunna bevaka flera loggfiler samtidigt.

Mindre viktiga krav är:

- Tidsangivelsen på loggposter som exporteras ska vara synkroniserade i hela systemet.
- Agenten ska vara så resurssnål som möjligt.

Vårt arbete kom till stor del bestå av att göra en design som kan flyttas mellan olika system. En viktig aspekt i designen var att definiera gränssnitten mellan de olika delarna i agenten så att den på ett enkelt sätt kan utökas med nya moduler för andra system. Vår uppgift var också att konstruera en prototyp för att visa att vår design fungerar. För att kunna testa prototypen behövde vi också implementera en enkel server.

3.5 Avgränsningar

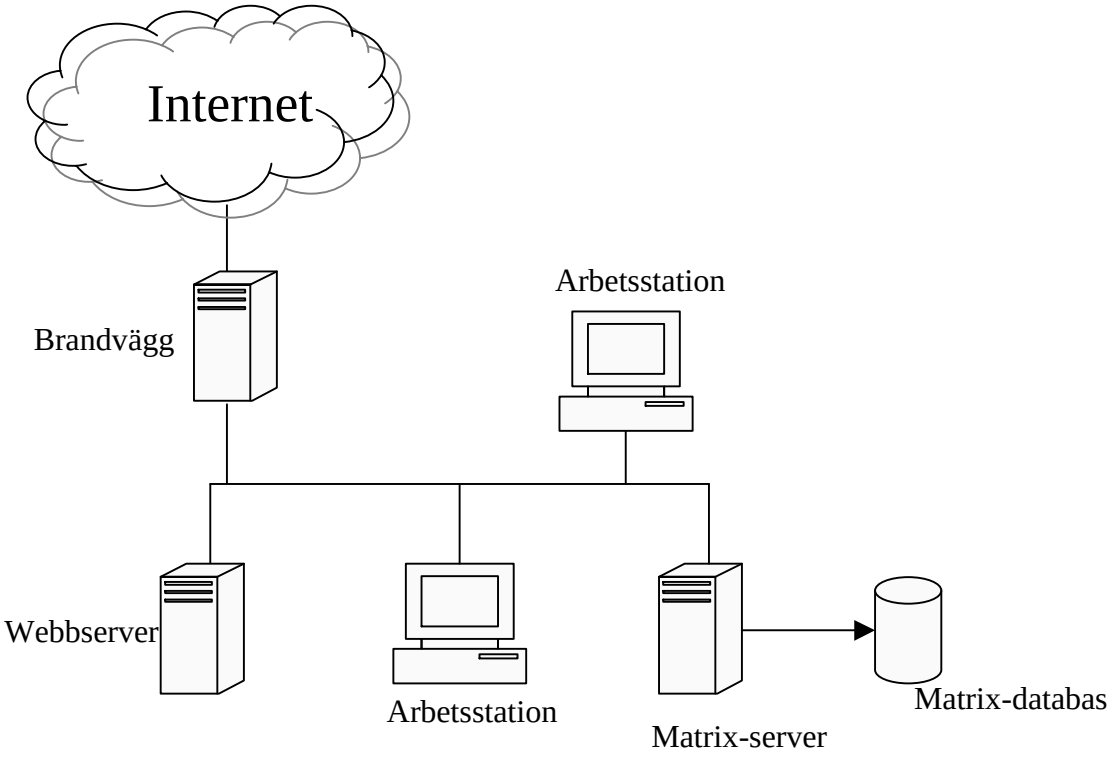
För att inte uppgiften ska bli för stor har servern och kommunikationen endast implementerats för att kunna testa agenten. Vi kommer därför att välja enklast möjliga implementation av server och kommunikation. Servern behöver inte att spara de mottagna loggposterna utan bara skriva ut dem på skärmen.

4 Beskrivning av konstruktionslösning

I följande kapitel kommer vi att beskriva en övergripande arkitektur av hela systemet samt mera ingående beskriva de delar av systemet vi har designat. Huvuddelen av vårt arbete har syftat till att designa en agent och därför kommer detta kapitel att till stor del att behandla dess design. Vi kommer även att översiktligt beskriva hur kommunikationen är uppbyggd samt beskriva servern vi har implementerat för att testa agenten.

4.1 Systemarkitektur

Det färdiga Matrix-systemet kommer att bestå av tre komponenter; agenterna, servern och databasen.

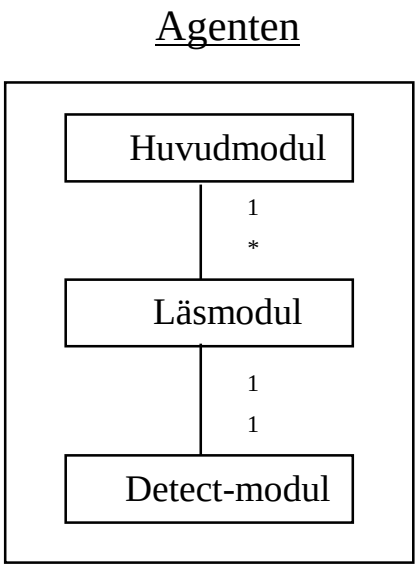


Figur 1: Ett datornätverk med Matrix-server.

I exemplet i Figur 1 är brandväggen, webbservern, arbetsstationerna och Matrix-servern exempel på loggproducerande enheter. En agent ska exekveras på varje loggproducerande enhet i nätverket. Den logg som produceras av operativsystem och applikationer ska skickas till Matrix-servern. Loggarna ska sedan sparas i en databas. Till databasen kommer det att finnas ett verktyg för att presentera olika typer av statistik över logghändelserna.

4.2 Agentens arkitektur

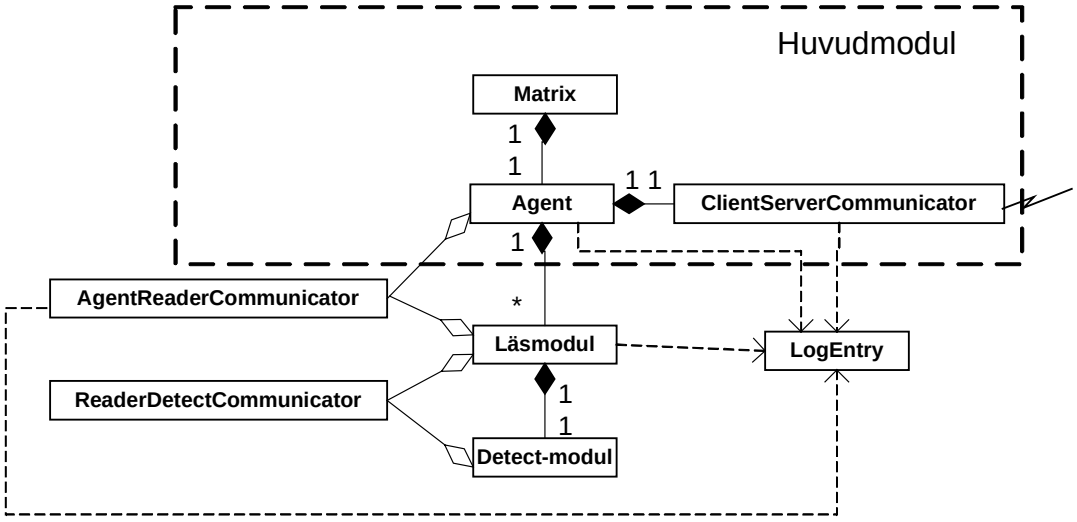
Agenten ska vara så plattformsoberoende som möjligt. Därför har vi valt en lösning uppbyggd av utbytbara moduler, se Figur 2. Huvudmodulen är inte beroende av plattform eller filtyp och behöver därför inte bytas ut. Huvudmodulen väljer med hjälp av en konfigurationsfil vilka typer av läsmoduler och detect-moduler som ska skapas. Huvudmodulen har även till uppgift att sköta kommunikationen med servern. Läs- och detect-modulerna arbetar alltid i par om en läsmodul och en detect-modul. Läsmodulerna har till uppgift att läsa in loggposter och detect-modulerna bevakar en loggfil för att upptäcka förändringar. När detect-modulen upptäcker en förändring i loggfilen skickas ett meddelande till läsmodulen som ingår i paret.



Figur 2: En övergripande skiss av agentens uppbyggnad.

Det är möjligt att byta ut en läsmodul mot en annan typ av läsmodul. Även detect-modulen kan bytas ut mot en annan typ av detect-modul. Man kan efter behov kombinera olika typer av läs- och detect-moduler i par.

För att en agent ska kunna bevaka flera loggfiler på varje dator exekveras varje modul som en egen process. Därför kan en huvudmodul exekvera flera par av läs- och detect-moduler samtidigt. Figur 3 visar en mer utförlig översiktsbild av agentens moduler och deras beståndsdelar.



Figur 3: Översiktsbild över agentens moduler.

4.3 Representation av loggposter

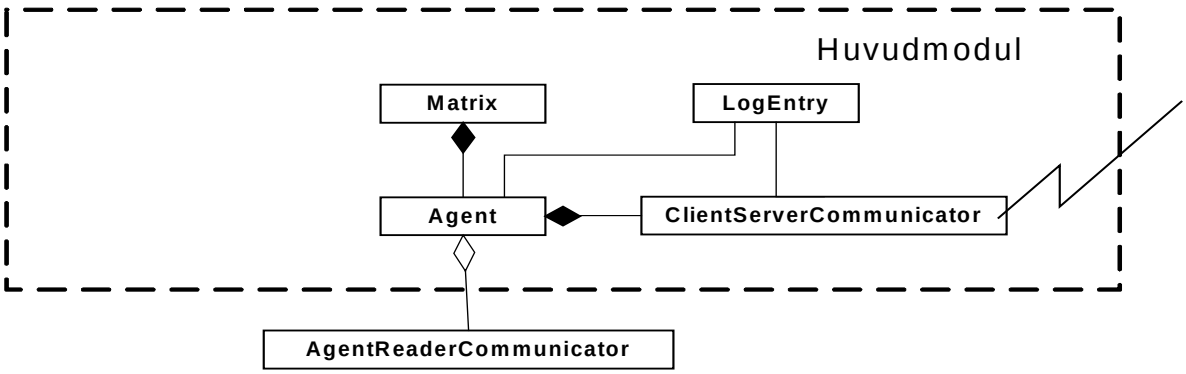
Vi har valt att representera varje loggpost som ett objekt av klassen LogEntry. Att representera loggposter med klassen LogEntry förenklar omvandlingen mellan olika instrukturer på loggposterna till en gemensam ut-struktur. Klassen har bara två operationer: hämta data och spara data. När man sparar data anger man data och en valfri etikett. För att sedan hämta data man sparat anger man etiketten. Oavsett i vilken ordning man har sparat datan kan man hämta den data man söker genom att ange rätt etikett.

4.4 Huvudmodul

Huvudmodulen är den del av programmet som har det huvudsakliga ansvaret för att loggar läses och exporteras till Matrix-servern. Huvudmodulen är den enda modul som inte byts ut, oavsett mot vilken typ av applikation eller i vilket operativsystem agenten arbetar.

Huvudmodulen använder information från en konfigurationsfil för att avgöra vilken eller vilka typer av läsmoduler som behövs (se kap. 4.8) samt hur de ska initieras. Huvudmodulen är också ansvarig för kommunikationen med servern.

I huvudmodulen finns en oändlig loop som väntar på ett meddelande från AgentReaderCommunicator (se Figur 4).



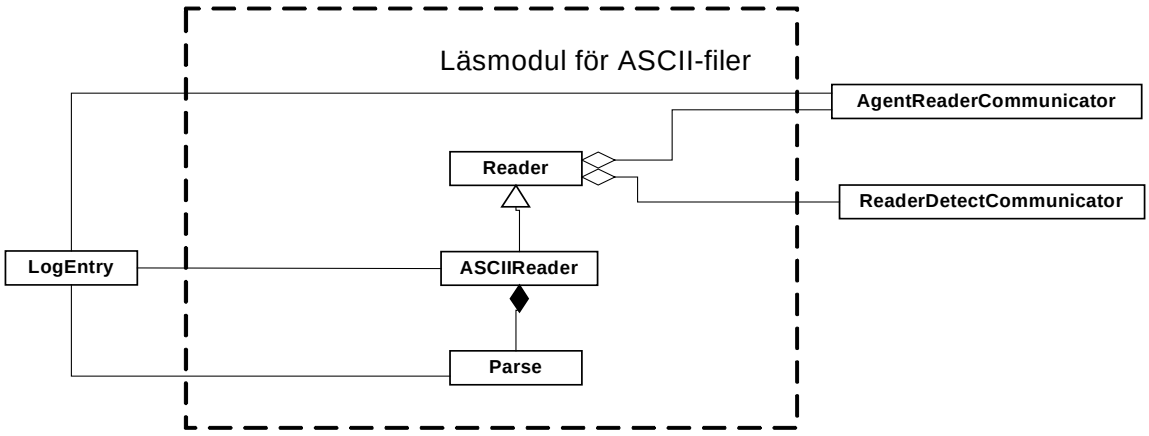
Figur 4: Huvudmodulens uppbyggnad.

Meddelandet informerar huvudmodulen att en av läsmodulerna har läst in en ny loggpost och att denna finns att hämta i AgentReaderCommunicator. Huvudmodulen hämtar då loggposten och skickar den sedan till en Matrix-server (se kap. 4.10).

4.5 Läsmoduler

En läsmoduls uppgift är att läsa loggposter från en fil och att konvertera loggposten till en instans av klassen LogEntry (för beskrivning av LogEntry se kap. 4.3).

En agent har flera olika typer av läsmoduler. De olika läsmodulerna klarar olika format på loggfilerna. Som exempel kan man tänka sig att det finns läsmoduler för filer i ASCII-format, hexadecimalt format, binärt format osv. Figur 5 visar hur en läsmodul för ASCII-filer är uppbyggd.



Figur 5: Uppbyggnad av en läsmodul för ASCII-filer.

Varje aktiv läsmodul läser från en loggfil. Detta gör att agenter som ska läsa från flera loggfiler samtidigt behöver lika många aktiva läsmoduler som antalet loggfiler som ska läsas.

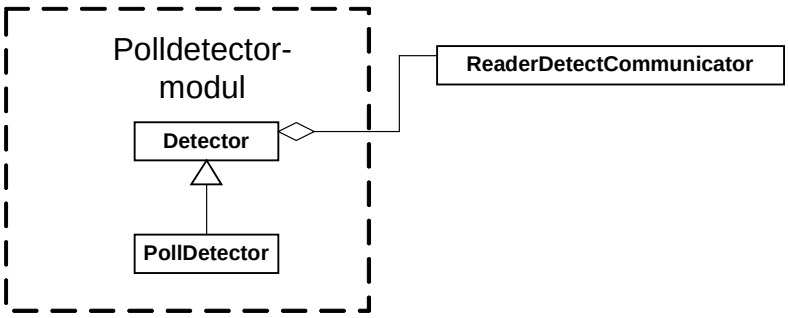
De aktiva läsmodulerna kan vara av olika typer men det kan även finnas flera aktiva läsmoduler av samma typ.

När en läsmodul får ett meddelande från en detect-modul att loggfilen har uppdaterats, läser den in alla nya poster i loggfilen. Efter att en loggpost lästs in konverterar läsmodulen den till ett objekt av typen LogEntry. När loggposten konverterats skickas ett meddelande till huvudmodulen att det finns en ny loggpost att skicka till servern.

4.6 Detect-moduler

Detect-modulens uppgift är att upptäcka förändringar i loggfilen. Det finns två huvudsakliga metoder för att lösa detta. Vi har implementerat en metod som använder sig av pollning för att upptäcka förändringar. Pollning innebär att man med jämna mellanrum (t.ex. varje sekund) kontrollerar om en fil har förändrats. Den andra metoden innebär att man väntar på en signal från operativsystemet eller en applikation. Signalen skickas varje gång operativsystemet eller applikationen skriver till loggfilen.

Figur 6 innehåller ett exempel på hur en polldetector-modul är uppbyggd.



Figur 6: Uppbyggnad av en polldetector.

4.7 Kommunikation mellan modulerna

Kommunikationen mellan de olika modulerna sker via mellanhänder. Detta är nödvändigt eftersom varje modul är en egen process (Javatråd). Därför har vi skapat två kommunikationsklasser som sköter all kommunikation mellan modulerna, se Figur 3 .

Det finns en instans av klassen ReaderDetectCommunicator för varje par av läs- och detect-moduler. Däremot finns det bara en instans av klassen AgentReaderCommunicator. Klassen AgentReaderCommunicator kan alltså sköta kommunikationen mellan en huvudmodul och flera läsmoduler. Den fungerar dessutom som en buffert för loggposter. Loggposterna sparas i bufferten tills de hämtas av huvudmodulen för att skickas till servern.

4.8 Konfigurationsfilens uppbyggnad

Konfigurationsfilen består av tre delar; system, output och information om loggfiler. I exemplet (Figur 7) finns det under rubriken system två poster; ”Linux” och ”Brandvägg”. Detta betyder att agenten i det här fallet ska läsa från två olika loggfiler. Under rubriken output specificeras vilken struktur loggposten har när den skickas till en server. Här finns också IP-adressen till en Matrix-server angiven samt vilken TCP-port som ska användas.

För varje post under rubriken system skapar huvudmodulen en uppsättning läs- och detect-moduler. För att veta vilken typ av moduler som ska skapas för varje post läser huvudmodulen under rubriken som motsvarar posten. I avsnittet Linux i Figur 7 anger första raden att en läsmodul för ASCII-filer ska användas. Raden under talar om vilken typ av detect-modul som ska användas. På tredje raden anges sökvägen till loggfilen som ska läsas och på sista raden specificeras hur strukturen ser ut i loggfilen.

Genom att i specificeringen av loggposten ut-struktur utelämna fält som finns i in-strukturen kan man utesluta ointressant information. I Figur 7 skickas inte fältet ”user” i Linuxloggen vidare till servern.

```
[system]
Linux
Brandvägg
;----------
[Linux]
ascii
poll
/etc/var/syslog.log
date time user message
;----------
[Brandvägg]
binary
poll
c:\brandvägg\log.txt
date time message
;----------
[output]
time date message
192.168.213.100
3729
```

Figur 7: Ett exempel på en konfigurationsfil.

4.9 Server

Vår design av servern har endast gjorts för att kunna testa agenten. Dess enda uppgift är att ta emot loggposter och skriva ut dem på skärmen. Servern startar en ny process (Javatråd) för varje uppkopplingsbegäran den får från en agent. Detta gör att servern kan ta emot loggposter från flera olika agenter samtidigt.

4.10 Kommunikation med servern

Vi valde enklast möjliga lösning för kommunikationen mellan agenten och servern. Vi har därför inte någon krypterad överföring, vilket ska finnas i ett färdigt system. Därför har vi gjort designen på ett sådant sätt att kommunikationsklassen enkelt ska kunna bytas ut. Vi valde att använda oss av socketprogrammering, då vi ansåg detta var den enklaste kommunikationslösningen. Att använda socketprogrammering är mycket likt tillvägagångssättet som används när man skriver till filer.

5 Implementation

I detta kapitel kommer vi att beskriva agentens implementation. Det kommer även att belysa de val vi har gjort och motivering till dessa val. Vi kommer också att beskriva vår lösning för de olika modulerna. För mer detaljerad information om agentens konstruktion se bilaga A och B.

5.1 Matrix-agenten

Följande avsnitt kommer att behandla de val och lösningar vi har gjort som berör hela vårt program.

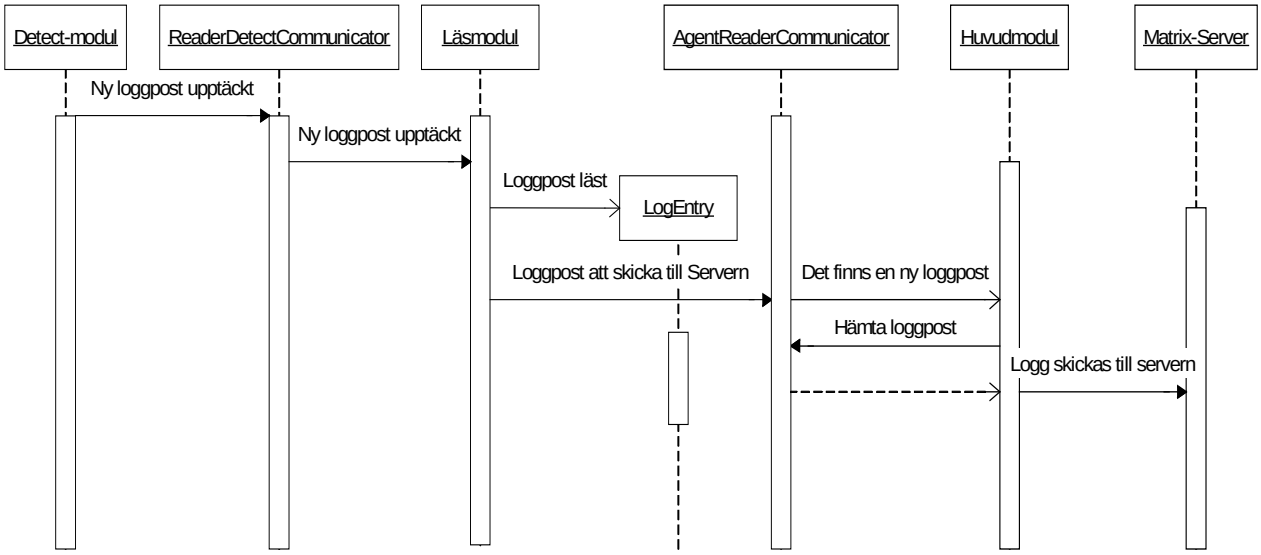
Ett av de viktigaste kraven på vår agent var att den skulle vara plattformsoberoende. Detta innebär bland annat att programmet ska kunna flyttas mellan olika operativsystem och datorarkitekturer utan att behöva kompileras om. Till detta passar ett interpreterande språk. Vi ansåg att Java var det självklara valet med sina många fördefinierade klasser (API:er) [1]. De många färdiga komponenterna i Java förenklade arbetet med programmet jämfört med om vi hade valt något annat interpreterande språk som t.ex. LISP eller Prolog. En nackdel med Java jämfört med kompilerande språk är dock att det måste finnas en Java Virtual Machine (JVM) installerad på datorn för att den ska kunna exekvera Javaprogram. En JVM använder i

storleksordningen 6 MB primärminne. Detta betyder att exekvering av ett Javaprogram alltid kräver minst 6 MB primärminne. Denna nackdel övervägs dock av de många fördelarna och Java var därför det självklara valet.

För att få agenten anpassningsbar mot olika system har vi valt att bygga upp vårt program med moduler. Programmet kan då använda den modul som passar det system där programmet exekverar. Modulerna har också definierade gränssnitt vilket gör att nya moduler enkelt kan läggas till i programmet. För att implementera modulerna har vi valt att skapa superklasser för läs- och detect-modulen. Detta gör det möjligt att efter behov bestämma antalet moduler och sedan skapa rätt typ av modul.

Vi har i vårt program valt att representera en loggpost med klassen LogEntry. Eftersom loggposten representeras av en egen klass kan dess implementation enkelt bytas ut. Detta kan man göra om man vill använda en annan datastruktur för att lagra loggpostens data. I klassen använder vi en hash-tabell för att spara loggpostens olika delar. Klassen LogEntry gör det enkelt att hämta ut information från loggposten.

För att flera moduler ska kunna exekveras parallellt exekverar varje modul i vårt program i en egen tråd. För att kunna synkronisera trådarna och kommunicera mellan dem har vi skapat kommunikationsklasser som är åtkomliga från flera trådar. Dessa används för att skicka meddelanden och objekt mellan trådarna. Sekvensdiagrammet i Figur 8 visar hur meddelanden skickas mellan modulerna efter att en ny loggpost har upptäckts av detect-modulen. Att paren av läs- och detect-moduler går i skilda trådar är troligtvis inte nödvändigt. Om det skulle skrivas en ny post under tiden detect-modulen väntar på läsmodulen skulle denna post ändå läsas eftersom läsmodulen läser till slutet på loggfilen.



Figur 8: Sekvensdiagram över meddelandens gång när en förändring i loggfilen har upptäckts.

Kommunikationsklassen mellan huvudmodulen och alla läsmoduler har en länkad lista för att kunna buffra loggposter. Detta används om t.ex. nätverket är överbelastat och det produceras loggposter fortare än vad som kan skickas till Matrix-servern. Vi använder den länkade listan som en kö (FIFO) så att inte gamla loggposter blir kvar i bufferten, som t.ex. skulle kunna ske i en stack.

5.2 Huvudmodulen

Huvudmodulen är den del av programmet där man med hjälp av konfigurationsfilen väljer vilka moduler som ska användas och hur många par av läs- och detect-moduler det ska vara. Det är nödvändigt att kunna läsa från flera loggfiler då många operativsystem har flera olika loggfiler, t.ex. finns det tre olika loggfiler i Windows XP. Förutom loggen från operativsystemet så kan det vara önskvärt att läsa loggfiler från en eller flera applikationer som exekverar på en dator som t.ex. logg från ett antivirusprogram.

5.3 Läsmoduler

Vi har valt att låta användaren, via konfigurationsfilen, ange sökvägarna till de loggfiler som ska läsas. Det finns två anledningar till detta; För det första gör det programmet generellt då det kan användas i många olika operativsystem och till många olika applikationer. För det andra har de flesta program och operativsystem vid installation en fastställd sökväg, som det

dock finns möjligheten att ändra. Hade inte möjligheten funnits att med konfigurationsfilen ange sökväg till loggfilerna så hade möjlighet att ändra sökväg till den plats operativsystem och applikationer skriver sina loggar försvunnit.

Vi har implementerat en läsmodul för loggfiler i ASCII-format. Den består av två klasser, ASCIIReader och Parse. ASCIIReader läser in varje ny loggpost till en sträng och skapar sedan ett objekt av typen Parse. Klassen Parse har endast en metod. Denna metod konverterar en sträng till ett nytt objekt av klassen LogEntry. En sträng innehållande en loggpost kan ha följande utseende: ”021224 12:55:11 Tomten User Tomten logged on”. Om konfigurationsfilen ser ut som i Figur 7 och ovanstående sträng lästes från Linuxloggen kommer det första ordet (021224) att tolkas som fältet ”date”, andra ordet (12:55:11) som ”time” och tredje som ”user”. Resterande ord kommer att tolkas som att det tillhör fältet ”message”. I de loggfiler vi har undersökt har det sist i varje loggpost alltid funnits ett meddelande som bestått av en mening innehållande ett antal ord. Meningen kan innehålla de tecken (mellanslag och tabb) som används för att avgränsa de övriga delarna av loggposten från varandra. Ett objekt av klassen LogEntry för denna loggpost ser ut som i Figur 9. För att vår implementation ska fungera måste alltså felmeddelanden finnas sist i loggposten.

Fält	Data
Date	021224
Time	12:55:11
User	Tomten
Message	User Tomten logged on

Figur 9: Ett exempel på en loggpost inläst i ett LogEntry.

Eftersom vår läsmodul aldrig stänger loggfilen mellan läsningar, läser den alltid sist i loggfilen. Filpekaren befinner sig då alltid sist i loggfilen. Detta kan få konsekvenser om loggfilen har en maximal tillåten storlek. Hur problemet yttrar sig beror på hur nya poster skrivs till loggfilen när loggfilen nått maximal storlek. Man kan t.ex. tänka sig att den nya loggposten skrivs sist i loggfilen och att den första posten tas bort, eller att den nya loggposten skrivs på den första loggpostens plats. Vår läsare klarar inte något av dessa fall utan är konstruerad för loggfiler med obegränsad storlek där nya loggposter skrivs sist.

Från början hade vi tänkt att förutom implementera en läsare för filer i ASCII-format även implementera en läsare för Windows 2000s loggfiler. Detta visade sig vara en tidskrävande uppgift. Vi har utan att lyckas försökt hitta dokumentation om hur formatet på Windows

loggfiler är uppbyggt. Vi har däremot hittat program som kan konvertera Windows loggar till textfiler och vi har även hittat programkod skriven i Visual Basic och C# för att få tillgång till och läsa Windows loggfiler [6]. Efter ett tag konstaterade vi att det var alltför tidskrävande att fortsätta arbetet med läsmodulen för Windows 2000 och lämnar därför denna modul oimplementerad. Vi har dock konstaterat att modulen är möjlig att implementera.

5.4 Detect-moduler

Vi har implementerat en detect-modul som använder sig av fil-pollning för att upptäcka förändringar i loggfilen. Vi var från början oroliga för att denna modul skulle vara resurskrävande och hade därför tänkt implementera en detect-modul som väntar på ett meddelande från operativsystemet. Detta meddelande skulle operativsystemet skicka varje gång loggfilen har uppdaterats. Fördelen med denna lösning är att vår detect-modul inte använder några systemresurser medan den väntar på nya meddelanden från operativsystemet. Vi vet dock inte om det systemanrop vi då skulle använda kräver motsvarande systemresurser som vår polldetector-modul. En nackdel är också att en sådan detect-modul antagligen blir plattformsberoende. Man blir då tvungen att konstruera en sådan detect-modul för varje system. När vi genomförde prestandatester på vår polldetector-modul visade det sig att modulen inte var så resurskrävande som vi fruktat. Detta gjorde att vi inte fann det nödvändigt att utveckla andra typer av detect-moduler. En stor fördel med polldetector-modulen är att den kan användas i både Windows och Linux. Vår polldetect-modul använder sig av den tidsangivelse operativsystemen märker filerna med för att upptäcka förändringar. Tidsangivelsen anger senaste tidpunkt då filen förändrades. Genom att läsa in denna tidsmärkning och sedan jämföra med tidsmärkningen från förra inläsningen kan den upptäcka förändringar i loggfilen.

När vi konstruerade vår detect-modul ställdes vi inför frågan om hur ofta vi skulle kontrollera om loggfilen blivit uppdaterad. Vi ville inte göra kontrollen för ofta för att inte programmet skulle bli för resurskrävande. Samtidigt ville vi att en uppdatering skulle upptäckas så snabbt som möjligt. Vi ansåg att en sekunds intervall utgjorde en god kompromiss. Provkörning av agenten visade att programmet var relativt resurssnålt med detta tidsintervall. Vi skulle kunnat implementera en detect-modul vars tidsintervall regleras i konfigurationsfilen. Vid implementeringen såg vi inget behov i att användare kan ändra tidsintervallet. Vi har dock senare insett behovet av att kunna anpassa tidsintervallet efter

olika situationer. Om t.ex. agenten körs på en maskin som genererar väldigt lite logg och där risken för att logg försvinner är liten, är det onödigt att kontrollera loggfilen varje sekund.

5.5 Kommunikation mellan moduler

För att ge programmet en viss parallellitet har vi valt att låta agentens moduler exekvera i egna trådar. En nackdel med denna konstruktion är att kommunikationen mellan de olika modulerna blir mer komplicerad.

För att undvika att vi förbrukar onödigt mycket processorkraft så stoppar vi exekveringen av tråden där läsmodulen körs när det inte finns några nya loggposter att läsa. För att åstadkomma detta använder vi oss av synkroniserade metoder i kommunikationsklasserna. Synkroniserade metoder är ett inbyggt sätt i Java att synkronisera processer. Läsmodulen anropar metoden isUpdated() i klassen ReaderDetectCommunicator. Detta innebär att läsmodulen kommer att stanna vid anropet wait(), se Figur 10.

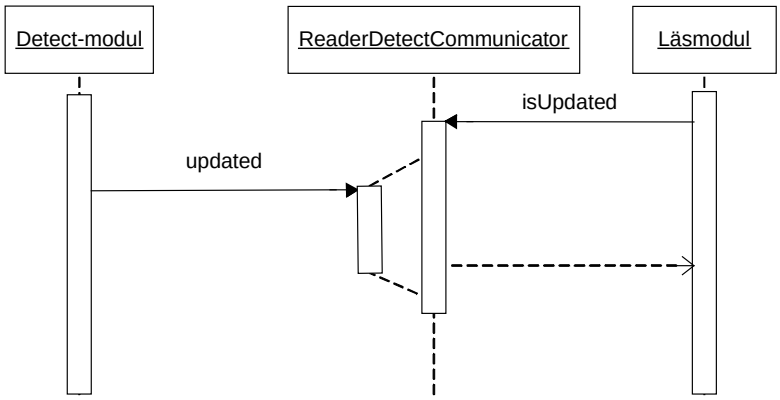
```
public synchronized void isUpdated()
{
        while(!updated)
        {
            try
            {
                wait();
            }catch(InterruptedException ex){}
        }
        updated = false;
 }
```

Figur 10: Kodexempel från klassen ReaderDetectCommunicator.

När detect-modulen upptäcker en förändring i loggfilen anropar den metoden update(), även den i klassen ReaderDetectCommunicator. Denna metod anropar notifyAll(), se Figur 11. Metoden notifyAll() startar alla objekt som har anropat metoden wait() och inte har startats ännu. Figur 12 visar hur modulerna interagerar med varandra.

```
public synchronized void update()
{
      updated = true;
      notifyAll();
 }
```

Figur 11: Kodexempel från klassen ReaderDetectCommunicator.



Figur 12: Sekvensdiagram över processynkronisering av detect-modul och läsmodul.

5.6 Server och kommunikation

Eftersom Matrix-servern och kommunikationen inte ingick i vår huvuduppgift, valde vi det sätt vi ansåg vara enklast när vi implementerade Matrix-servern och kommunikationen mellan agenten och servern. Vårt val för kommunikationen blev därför socketprogrammering. För mera ingående information om servern se bilaga C och D.

6 Erfarenheter och rekommendationer

I följande kapitel beskriver vi de erfarenheter vi har gjort och vad vi har lärt oss under lösandet av uppgiften samt en vägledning till dem som skall vidareutveckla Matrix eller genomföra liknande projekt.

Vi upplevde det svårt att göra en hållbar planering. Svårast var det under uppstarten då vi inte visste dels hur vi skulle strukturera programmet och dels hur mycket arbete de olika delarna skulle kräva. Vi hade heller aldrig använt Java tidigare och visste inte hur snabbt vi skulle kunna lära oss språket samt exakt hur mycket funktioner som fanns färdigt i API:erna. Även att revidera planeringen under arbetets gång var svårt trots att vi då hade viss erfarenhet. Ett annat problem var att vi underskattade tidsåtgången till vissa delar av arbetet. Den största anledningen till våra problem med planeringen var bristande erfarenhet.

Att göra detaljerade planeringar vid uppstarten av ett projekt fungerar mycket bra om man har stor erfarenhet och arbetat med liknande projekt tidigare. I vårt fall hade vi egentligen ingen aning om hur den färdiga koden till programmet skulle se ut. Detta gjorde att den första planeringen vi gjorde inte alls liknar den sista versionen av tidsplaneringen. Om vi skulle genomföra ett liknande projekt igen skulle vår första planering vara mycket översiktlig och

utan detaljer. När vi sedan får större kunskap om vad som krävs för att genomföra projektet skulle vi revidera planeringen och lägga till fler detaljer. De närmaste veckorna skulle vara de som är mest detaljerade och noggrant planerade.

Vi upptäckte också att specifikationer inte alltid är lika tydliga som de laborationsspecifikationer vi har kommit i kontakt med hittills. Vi blev därför tvungna att föra en diskussion med uppdragsgivaren för att säkerställa att deras önskemål uppfylldes. Om man inte har en noggrann uppföljning av specifikationen riskerar man att lägga tyngdpunkten på fel saker i sitt arbete.

Vår första kontakt med Java var enbart positiv. Det är enkelt att lära sig om man som vi har använt C/C++ tidigare. Vi hittade bra handledning till Java på Suns hemsida [2][3][4][5]. Det finns som tidigare nämnts många fördefinierade klasser [1], problemet är att det ibland kan vara svårt att hitta den funktionalitet man söker. Det är också svårt att veta om den funktionalitet man söker finns implementerad. Detta kan leda till att man utför onödigt arbete genom att implementera funktionalitet som redan existerar i API:erna.

På grund av den stora mängd information som genereras är det i ett färdigt Matrix-system nödvändigt att kunna välja vilken typ av information som är intressant. Här ställs man inför ett val, ska urvalet ske i agenten eller i servern? Om det sker i agenten kommer den att kräva mer datorkraft. En fördel är att belastningen på nätverket minskar då ointressant information aldrig skickas till servern. En nackdel är att administratören måste konfigurera vilken information som är intressant i varje enskild agent. Om man däremot väljer att implementera urvalet i servern behöver man endast konfigurera servern för att styra urvalet. Risken med denna lösning är att det kan bli en hög belastning på nätverket och servern. Vi rekommenderar att lägga urvalet i servern för att förenkla konfigureringen. Vid en normal belastning på nätverk och server kommer inte prestandan försämras märkbart. I ett system där loggposterna kommer i skurar borde dock urvalet ske i agenten.

I vår konstruktion av läsmodulen för ASCII-filer ingår klassen Parse (se Figur 5). Denna klass används för att konvertera en loggsträng till ett LogEntry-objekt. Vid konstruktionen av Parse antog vi att den skulle kunna användas till alla typer av filformat. Eftersom vi inte implementerat någon läsare för annat format än ASCII så vet vi inte om Parse fungerar för andra filformat. Om det visar sig att vår Parse enbart fungerar för ASCII-filer är det bättre att infoga dess funktionalitet i klassen ASCIIReader.

7 Slutsatser

Vår primära uppgift var att undersöka om det var möjligt att konstruera en agent som är plattformsoberoende, förhållandevis resurssnål, som exporterar logg snabbt och som klarar ett antal olika loggformat. Vi har kommit fram till att det går att konstruera en agent som är helt plattformsoberoende. Delar av programmet kommer dock att vara anpassade till olika filtyper och filformat. Det går alltså att konstruera en agent som själv anpassar sig till omgivningen. Eftersom agenten ska vara möjlig att exekvera på en mängd olika plattformar passar interpreterande programspråk bäst. Dessa kan aldrig bli lika effektiva som kompilerade språk då de kräver en interpretator. Javas interpretator, JVM, kräver i storleksordningen 6 MB ledigt primärminne för att kunna exekvera. På nyare datorer är denna resursåtgång obetydlig då dessa oftast har minst 128 MB primärminne. JVM:en tar då alltså upp mindre än 5 procent av det totala primärminnet.

Vår sekundära uppgift var att konstruera en prototyp till en agent. Vår prototyp har vi provkört i Windows 2000 och Redhat Linux. Den kan läsa loggfiler i ASCII-format och kan upptäcka förändringar i alla typer av loggfiler i både Windows och Linux. Vår agent klarar dock inte avgöra vilka nya loggposter som har tillkommit i alla typer av loggfiler (t.ex. loggfiler som genereras av Windows). Vår prototyp klarar att upptäcka och exportera en ny loggpost på mindre än en sekund. Det betyder att en eventuell inkräktare som mest har en sekund på sig att ta sig in i systemet och radera loggen för att dölja sitt intrång.

Vår prototyp kan ses som ett bevis på att det är möjligt att konstruera en agent som till stor del uppfyller de ställda kraven (se kap. 3.4). Vår prototyp kan också ses som en stomme för vidareutveckling av Matrix-systemet till en färdig produkt.

Referenser

[1] Sun Microsystems Inc., *All Classes (Java 2 Platform SE v1.4.1),* http://java.sun.com/j2se/1.4.1/docs/api/, 2002-09-02 till 2002-12-17.

[2] Sun Microsystems Inc., *Your First Cup of Java,* http://java.sun.com/docs/books/tutorial/getStarted/cupojava/index.html, 2002-09-02 till 2002-12-17.

[3] Sun Microsystems Inc., *Getting Started,* http://java.sun.com/docs/books/tutorial/getStarted/index.html, 2002-09-02 till 2002-12-17.

[4] Sun Microsystems Inc., *Learning the Java Language,* http://java.sun.com/docs/books/tutorial/java/index.html, 2002-09-02 till 2002-12-17.

[5] Sun Microsystems Inc., *Essential Java Classes,* http://java.sun.com/docs/books/tutorial/essential/index.html, 2002-09-02 till 2002-12-17.

[6] Microsoft Corporation, *EventLog Class,* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemDiagnosticsEventLogClassTopic.asp, 2002-11-12.

A Javadoc för Agenten

Vår Javadoc finns även på nätet: http://hem.bredband.net/larlea/docs/index.html

A.1 Agent

Package

Tree
Deprecated
Index
Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**
SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class Agent

```
java.lang.Object
  |
  +--Agent
```

---

public class **Agent**

extends java.lang.Object

The main class of the program. It creates readers based on the config file and starts them.

Version:

π

Author:

Daniel Törnqvist, Lars-Olof Leander

Constructor Summary

Agent()	

Method Summary

void	**agentDriver**() The method that drives the program.
private java.lang.String[]	**config**(java.lang.String header) Reads every line in a section from the file config.conf and returns it as an array of strings.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Agent

```
public Agent()
```

Method Detail

agentDriver

```
public void agentDriver()
                  throws java.io.IOException,
                         java.lang.InterruptedException
```

The method that drives the program. The method creates communicators and also creates and starts readers depending on the config file. Enters an eternal loop that gets log objects from the AgentReaderCommunicator and sends them to ClientServerCommunication.

Throws:

java.io.IOException

java.lang.InterruptedException

---

config

```
private java.lang.String[] config(java.lang.String header)
```

`throws java.io.IOException`

Reads every line in a section from the file config.conf and returns it as an array of strings.

Precondition: The string Header is a valid header in the configfile. **Postcondition:** An array of strings containing the specified section of the configfile has been returned.

Parameters:

`header` - A string containing the header of the section to be read.

Throws:

`java.io.IOException`

Package

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

A.2 ASCIIReader

Package

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class ASCIIReader

```
java.lang.Object
  |
  +--java.lang.Thread
        |
        +--Reader
              |
              +--ASCIIReader
```

All Implemented Interfaces:

java.lang.Runnable

---

public class **ASCIIReader**

extends Reader

A reader for logfiles in ASCII format.

Version:

π

Author:

Lars-Olof Leander, Daniel Törnqvist

See Also:

Reader

Field Summary

private AgentReaderCommunicator	**arCommunicator**
private ReaderDetectCommunicator	**communicator**
private java.lang.String	**detectorType**
private java.io.BufferedReader	**file**
private java.lang.String	**inputStructure**
private java.lang.String	**logPath**

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

ASCIIReader(AgentReaderCommunicator inComm, java.lang.String detector, java.lang.String path, java.lang.String input) Initializes some of the arguments for the class.

Method Summary

private void	**reader**() Creates and starts detectors.
private void	**readFile**() Reads lines from the log file and returns it as a LogEntry.
Void	**run**() The method called when the thread is started.

Methods inherited from class java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority,

getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

communicator

```
private ReaderDetectCommunicator communicator
```

---

arCommunicator

```
private AgentReaderCommunicator arCommunicator
```

---

detectorType

```
private java.lang.String detectorType
```

---

logPath

```
private java.lang.String logPath
```

---

inputStructure

```
private java.lang.String inputStructure
```

---

file

```
private java.io.BufferedReader file
```

Constructor Detail

ASCIIReader

```
public ASCIIReader(AgentReaderCommunicator inComm,
                   java.lang.String detector,
                   java.lang.String path,
                   java.lang.String input)
```

Initializes some of the arguments for the class.

Parameters:

`inComm` - A instance of a AgentReaderCommunicator. Used to send LogEntrys to the Agent class.

`detector` - Specifies what type of detector to be created and used.

`path` - Specifies the path of the logfile to be read.

`input` - Specifies the structure of the logfile.

Method Detail

run

```
public void run()
```

The method called when the thread is started.

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `Reader`

---

reader

```
private void reader()
             throws java.io.IOException,
                    java.io.FileNotFoundException
```

Creates and starts detectors. Also creates a reader for the logfile. Puts the reader at the end of the logfile. Waits for a signal from the detector via the ReaderDetectCommunicator.

Precondition: All attributes are valid, ie the logPath string contains a path to a logfile etc.

Postcondition: Never returns.

Throws:

`java.io.IOException`

`java.io.FileNotFoundException`

---

readFile

```
private void readFile()
               throws java.io.IOException,
                      java.io.FileNotFoundException
```

Creates a parser. Reads lines from the logfile and send them to the parser. Writes the logObject to the AgentReaderCommunicator.

Precondition: Logfile has been updated.

Postcondition: The new log entry has been read and sent to the AgentReaderCommunicator.

Throws:

`java.io.IOException`

`java.io.FileNotFoundException`

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

A.3 PollDetector

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class PollDetector

```
java.lang.Object
  |
  +--java.lang.Thread
        |
        +--Detector
              |
              +--PollDetector
```

All Implemented Interfaces:

java.lang.Runnable

---

public class **PollDetector**

extends Detector

A class that detects changes in a logfile. It does this by checking when the file was last modified and does this once a second.

Version:

π

Author:

Daniel Törnqvist, Lars-Olof Leander

See Also:

Detector

Field Summary

private ReaderDetectCommunicator	**communicator**
private java.io.File	**inFile**
private java.lang.String	**path**

Fields inherited from class java.lang.Thread

MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

PollDetector(ReaderDetectCommunicator inCom, java.lang.String logPath)

Method Summary

void	**run**() Checks for updates in the logfile.

Methods inherited from class java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

inFile

```
private java.io.File inFile
```

---

communicator

```
private ReaderDetectCommunicator communicator
```

---

path

```
private java.lang.String path
```

Constructor Detail

PollDetector

```
public PollDetector(ReaderDetectCommunicator inCom,
                    java.lang.String logPath)
```

Parameters:

`inCom` - An instance of the class ReaderDetectcommunicator. Used to communicate with a Reader.

`logPath` - A string containing the path and name of the logfile.

Precondition: All attributes are valid, ie the path string contains a path to a logfile.

Postcondition: Never returns.

Method Detail

run

```
public void run()
```

Checks for updates in the logfile. Uses the File.lastModified method to detect updates and when updates are detected, a signal is sent to reader via a ReaderDetectCommunicator.

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `Detector`

See Also:

`Thread.start(), Thread.stop(), Thread.Thread(java.lang.ThreadGroup, java.lang.Runnable, java.lang.String), Runnable.run()`

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

A.4 AgentReaderCommunicator

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class AgentReaderCommunicator

```
java.lang.Object
  |
```

+--**AgentReaderCommunicator**

---

public class **AgentReaderCommunicator**

extends java.lang.Object

A communicator between instances of Agent and instances of subclasses to Reader.

Version:

π

Author:

Daniel Törnqvist, Lars-Olof Leander

Field Summary

private java.util.LinkedList	**loggBuffer**

Constructor Summary

AgentReaderCommunicator()	

Method Summary

void	**add**(LogEntry log) Adds a LogEntry to the linkedlist loggBuffer.
LogEntry	**get**() Waits until there's at least one LogEntry in loggBuffer.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

loggBuffer

```
private java.util.LinkedList loggBuffer
```

Constructor Detail

AgentReaderCommunicator

```
public AgentReaderCommunicator()
```

Method Detail

add

```
public void add(LogEntry log)
```

Adds a LogEntry to the linkedlist loggBuffer.

Precondition: True

Postcondition: A LogEntry added to the buffer.

---

get

```
public LogEntry get()
```

Waits until there's at least one LogEntry in loggBuffer. Then removes the first post in the list and returns it.

Precondition: True

Postcondition: A LogEntry object has been returned.

A.5 ClientServerCommunicator

Package

Tree

Deprecated

Index

Help

Class ClientServerCommunication

```
java.lang.Object
  |
  +--ClientServerCommunication
```

public class **ClientServerCommunication**

extends java.lang.Object

Creates a connection to a server and sends LogEntrys to the server.

Invariant: The output format is defined.

Version:

π

Author:

Daniel Törnqvist, Lars-Olof Leander

Field Summary

private java.lang.String	**ipAddress**
private java.io.PrintWriter	**out**
private java.lang.String[]	**outputStructure**
private java.lang.Integer	**portNumber**
private java.net.Socket	**toMatrixServer**

Constructor Summary

ClientServerCommunication() Reads the output section of the configfile.	

Method Summary

void	**sendToServer**(LogEntry log) Converts the LogEntry object to a string and sends it to the server.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

outputStructure

private java.lang.String[] **outputStructure**

---

ipAddress

```
private java.lang.String ipAddress
```

---

portNumber

```
private java.lang.Integer portNumber
```

---

toMatrixServer

```
private java.net.Socket toMatrixServer
```

---

out

```
private java.io.PrintWriter out
```

Constructor Detail

ClientServerCommunication

```
public ClientServerCommunication()
                            throws java.io.IOException
```

Reads the output section of the configfile. Creates a connection to a server, with the data from the configfile.

Throws:

`java.io.IOException`

Method Detail

sendToServer

```
public void sendToServer(LogEntry log)
                  throws java.io.IOException
```

Converts the LogEntry object to a string and sends it to the server.

Precondition: A connection to a server must be established.

Postcondition: The LogEntry has been sent to the server.

Parameters:

`log` - A LogEntry object that is to be sent to the server.

Throws:

`java.io.IOException`

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

A.6 LogEntry

Package

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class LogEntry

```
java.lang.Object
  |
  +--LogEntry
```

---

public class **LogEntry**

extends java.lang.Object

A instance of this class contains a log entry. It uses a hashtable to store the data with a key defined in the configfile.

Version:

π

Author:

Lars-Olof Leander, Daniel Törnqvist

Field Summary

private java.util.Hashtable	**log**

Constructor Summary

LogEntry()	

Method Summary

java.lang.String	**getField**(java.lang.String key) Returns the data in the field with the specified key.
void	**setField**(java.lang.String key, java.lang.String value) Inserts a string into the hashtable using the specified key.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

log

```
private java.util.Hashtable log
```

Constructor Detail

LogEntry

```
public LogEntry()
```

Method Detail

setField

```
public void setField(java.lang.String key,
                     java.lang.String value)
```

Inserts a string into the hashtable using the specified key.

Precondition: key != null && value != null.

Postcondition: The specified value has been saved with the specified key.

Parameters:

`key` - The name of the datafield.

`value` - The data that is to be inserted into the hashtable.

---

getField

```
public java.lang.String getField(java.lang.String key)
```

Returns the data in the field with the specified key.

Precondition: A field with the specified key exists in the object.

Postcondition: The data in the field with the specified key has been returned.

Parameters:

`key` - The name of the datafield to be returned.

A.7 Matrix

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Matrix

```
java.lang.Object
  |
  +--Matrix
```

public class **Matrix**

extends java.lang.Object

The main class of the program.

Version:

π

Author:

Daniel Törnqvist, Lars-Olof Leander

Constructor Summary	
Matrix()	

Method Summary

static void	**main**(java.lang.String[] args) Creates an instance of the class Agent.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Matrix

```
public Matrix()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Creates an instance of the class Agent. Calls Agent's method agentDriver.

Parameters:

`args` - Not used.

Throws:

`java.io.IOException`

`java.lang.InterruptedException`

Package

Class

Tree

Deprecated

Index

Help

A.8 Parse

Package

Class
Tree
Deprecated
Index
Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Parse

```
java.lang.Object
  |
  +--Parse
```

public class **Parse**

extends java.lang.Object

Class for converting a string into a LogEntry object.

Version:

π

Author:

Lars-Olof Leander, Daniel Törnqvist

Field Summary

private	**inputOrder**

java.lang.String[]	
private int	**numberOfColumns**

Constructor Summary

Parse(java.lang.String input) Divides the String containing the inputstructure into an array of strings and saves it as an attribute.

Method Summary

LogEntry	**logParser**(java.lang.String log) Takes a string and converts it into a LogEntry object.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

inputOrder

```
private java.lang.String[] inputOrder
```

---

numberOfColumns

```
private int numberOfColumns
```

Constructor Detail

Parse

```
public Parse(java.lang.String input)
```

Divides the String containing the inputstructure into an array of strings and saves it as an attribute.

Parameters:

`input` - String containing the strucure of the expected input.

Method Detail

logParser

```
public LogEntry logParser(java.lang.String log)
```

Takes a string and converts it into a LogEntry object.

Precondition: The input structure is correctly specified and numberOfColumns is correct.

Postcondition: The string has been converted to an instance of LogEntry and the LogEntry object has been returned.

Parameters:

`log` - A string containing the line of log.

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

A.9 SectionReader

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS NEXT CLASS | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class SectionReader

```
java.lang.Object
  |
  +-java.io.Reader
        |
        +-java.io.BufferedReader
              |
              +-SectionReader
```

---

class **SectionReader**

extends java.io.BufferedReader

The filter reads a line oriented reader and filters out all lines which start with a ; (comment lines). The reader starts after a section header line [] and returns eof at the end of the file or at the start of a new section. It also returns eof if the section header is not found. Restriction: Since it is line oriented it is defined as a subclass of BufferedReader. The filtering only works in line oriented reading (using readLine()) Character reading reads the physical underlying input stream directly

Author:

Eivind J. Nordby

Field Summary	
(package private) boolean	**debug**
(package	**eof**

private) boolean	
(package private) boolean	**headerLocated**
(package private) java.lang.String	**sectionHeader**

Fields inherited from class java.io.BufferedReader

Fields inherited from class java.io.Reader
lock

Constructor Summary
SectionReader(java.io.Reader in, java.lang.String sectionHeader)
SectionReader(java.io.Reader in, java.lang.String sectionHeader, boolean debug)

Method Summary	
private java.lang.String	**doReadLine**()
private boolean	**isSectionHeader**(java.lang.String line) This is a comment for JavaDoc
private void	**locateHeader**()
boolean	**markSupported**()
java.lang.String	**readLine**()

Methods inherited from class java.io.BufferedReader
close, mark, read, read, ready, reset, skip

Methods inherited from class java.io.Reader
read

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

sectionHeader

```
java.lang.String sectionHeader
```

---

headerLocated

```
boolean headerLocated
```

---

eof

```
boolean eof
```

---

debug

```
boolean debug
```

Constructor Detail

SectionReader

```
public SectionReader(java.io.Reader in,
                     java.lang.String sectionHeader)
```

SectionReader

```
public SectionReader(java.io.Reader in,
                     java.lang.String sectionHeader,
                     boolean debug)
```

Method Detail

markSupported

```
public boolean markSupported()
```

Overrides:

`markSupported` in class `java.io.BufferedReader`

---

readLine

```
public java.lang.String readLine()
                        throws java.io.IOException
```

Overrides:

`readLine` in class `java.io.BufferedReader`

`java.io.IOException`

---

isSectionHeader

```
private boolean isSectionHeader(java.lang.String line)
```

This is a comment for JavaDoc

---

locateHeader

```
private void locateHeader()
                  throws java.io.IOException
```

`java.io.IOException`

---

doReadLine

```
private java.lang.String doReadLine()
                           throws java.io.IOException
```

`java.io.IOException`

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

A.10 ReaderDetectCommunicator

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ReaderDetectCommunicator

```
java.lang.Object
  |
  +--ReaderDetectCommunicator
```

public class **ReaderDetectCommunicator**

extends java.lang.Object

A communicator between instances of subclasses to Reader and instances of subclasses to Detector.

Version:

π

Author:

Lars-Olof Leander, Daniel Törnqvist

Field Summary	
private	**updated**

boolean	

Constructor Summary

ReaderDetectCommunicator()	

Method Summary

void	**isUpdated**() Objects running this method is set to wait until the updated field is set to true.
void	**update**() Sets the updated field to true and wakes objects waiting all reader objects waiting on this object.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

updated

private boolean **updated**

Constructor Detail

ReaderDetectCommunicator

public **ReaderDetectCommunicator**()

Method Detail

update

public void **update**()

Sets the updated field to true and wakes objects waiting all reader objects waiting on this object.

Precondition: The logfile has been updated. **Postcondition:** All waiting threads has been notified.

---

isUpdated

`public void isUpdated()`

Objects running this method is set to wait until the updated field is set to true.

Precondition: True. **Postcondition:** Updated set to false.

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

B Agentens källkod

B.1 Matrix.java

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

/**
 * The main class of the program.
 *
 * @author Daniel Törnqvist
 * @author Lars-Olof Leander
 * @version &#960;
 */
public class Matrix
```

```
{

    /**
     * Creates an instance of the class Agent.

     * Calls Agent’s method agentDriver.
     *
     * @param args   Not used.
     * @exception IOException
     * @exception InterruptedException
     */
    public static void main(String[] args)

    {
        try
     {
         Agent myAgent = new Agent();
         myAgent.agentDriver();
     } catch(Exception ex){
       System.out.println("Error");
       System.exit(0);
     }
    }
}
```

B.2 Agent.java

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

/**
 * The main class of the program.
 * It creates readers based on the config file and starts them.
 *
 * @author Daniel Törnqvist
 * @author Lars-Olof Leander
 * @version &#960;
  */
public class Agent
{
     public Agent()
     {
     }

     /**
      * The method that drives the program.
      *
      * The method creates communicators and also
      * creates and starts readers depending on the config file.
      * Enters a eternal loop that gets log objects from the
      * AgentReaderCommunicator and sends them to
ClientServerCommunication.
      *
         *
      * @exception IOException
      * @exception InterruptedException
      */
     public void agentDriver() throws IOException, InterruptedException
     {
```

```
          AgentReaderCommunicator arCommunicator;
          arCommunicator = new AgentReaderCommunicator();

                         ClientServerCommunication serverClientCom = new
                         ClientServerCommunication();

          Reader[] reader = new Reader[10];                    //superklass

          int i = 0;

          String[] system = new String[10];
          String[] configuration = new String[10];

          system = config("system");

          while ( system[i] != null )
          {
               configuration = config(system[i]);

               if ( configuration[0].equals("ascii") )
                    reader[i] = new ASCIIReader(arCommunicator,
configuration[1], configuration[2], configuration[3]);

               else if ( configuration[0].equals("binary") )
                    reader[i] = new BinaryReader();

               else
               {
                    System.out.println("System exiting");
                    reader[i] = new Reader();      //ska vara en default
reader
                    System.exit(0);
               }

               i++;

          }


          System.out.println("Startar reader");
          int j = 0;

          while ( j < i )
          {
               reader[j].start();
               j++;
          }


          LogEntry lastLog = new LogEntry();


          while ( true )
          {

               lastLog = arCommunicator.get();
               serverClientCom.sendToServer(lastLog);
          }
     }

     /**
      * Reads every line in a section from the file
```

```
     * config.conf and returns it as an array of strings.
     * <br> <br>
     * <b>Precondition:</b>  The string Header is a valid header in the
configfile.<br>
     * <b>Postcondition:</b> An array of strings containing the specified
section of the configfile is returned.
     * @param header A string containing the header of the section to be
read.
     * @return An array of strings where each string represents a line in
the section.
     * @exception IOException
     */
    private String[] config(String header) throws IOException
    {
         String[] configStrings = new String[10];

         try
         {
              SectionReader configFile = new SectionReader(new
FileReader("config.conf"), header);

              int i = 0;

              String temp = new String();

              do
              {
                   temp = configFile.readLine();
                   configStrings[i] = temp;
                   i++;

              }while ( temp != null );

              System.out.println(configStrings[0]);

         }catch ( FileNotFoundException ex )
         {
              System.out.println("File error!!\nFile config.conf not found.
Program terminating");
              System.exit(0);
         }

         return configStrings;
    }


}
```

B.3 Reader.java

```
import java.lang.*;

/**
 * A virtual superclass for readers.
 *
 * @author Daniel Törnqvist
 * @author Lars-Olof Leander
 * @version 0
 * @see ASCIIReader
```

```
 * @see BinaryReader
 */
public class Reader extends Thread
{
    public Reader()
    {
    }

    public void run()
    {
    }
}
```

B.4 ASCIIReader.java

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

/**
 * A reader for logfiles in ASCII format.
 *
 * @author Lars-Olof Leander
 * @author Daniel Törnqvist
 * @version &#960;
 * @see Reader
 */
public class ASCIIReader extends Reader
{
    /**
     * Initializes some of the arguments for the class.
     *
     * @param inComm   A instance of a AgentReaderCommunicator.
     *                 Used to send LogEntrys to the Agent class.
     * @param detector Specifies what type of detector to be created and
used.
     * @param path     Specifies the path of the logfile to be read.
     * @param input    Specifies the structure of the logfile.
     */
    public ASCIIReader(AgentReaderCommunicator inComm, String detector,
String path, String input)

    {
        communicator = new ReaderDetectCommunicator();

        if(detector != null && path != null && input != null)
        {
            arCommunicator = inComm;
            logPath = path;
            detectorType = detector;
            inputStructure = input;
        }
        else
        {
            System.out.println("Error in configfile, statement missing");
            System.exit(0);
        }
    }

    private ReaderDetectCommunicator communicator;
```

```
    private AgentReaderCommunicator  arCommunicator;
    private String detectorType, logPath, inputStructure;
    private BufferedReader file;


    /**
     * The method called when the thread is started.
     */
    public void run()
    {
        try
        {
            this.reader();
        }
        catch(Exception ex)
        {
            System.out.println("Reader error! Does the specified file
exist?");
            System.exit(0);
        }

    }

    /**
     * Creates and starts detectors. Also creates a reader for the logfile.
     * Puts the reader at the end of the logfile. Waits for a signal from
the detector
     * via the ReaderDetectCommunicator.
     *
     * <br>       <br>
     * <b>Precondition:</b>  All attributes must be valid, ie the logPath
string must contain a path to a logfile etc.   <br>
     * <b>Postcondition:</b> Exception has been caught.
     * @exception IOException
     * @exception FileNotFoundException
     */
    private void reader() throws IOException, FileNotFoundException
    {
        Detector detector;

        if(detectorType.equals("poll"))
        {
            detector = new PollDetector(communicator, logPath);
           // System.out.println("polldetect");  //debug
        }
        else if(detectorType.equals("event"))
            detector = new EventDetector();
        else
        {
            detector = new Detector();
            System.out.println("Default detector"); //debug
            System.exit(0);
        }

     file = new BufferedReader(new FileReader(logPath));

        while(file.readLine() != null);

        System.out.println("Startar detector"); //debug
        detector.start();

        while(true)
```

```
        {
            communicator.isUpdated();

            System.out.println("andrad"); //debug
            try
            {
                this.readFile();   //las in uppdaterad log
            }catch(Exception ex){}
         }


    }


    /**
     * Creates a parser. Reads lines from the logfile and send them to the
parser.
     * Writes the logObject to the AgentReaderCommunicator.
     *
     * <br><br>
     * <b>Precondition:</b>  Logfile has been updated. <br>
     * <b>Postcondition:</b> The new log entry has been read and sent to
the AgentReaderCommunicator.
     * @exception IOException
     * @exception FileNotFoundException
     */
    private void readFile() throws IOException, FileNotFoundException
    {
       Parse parser = new Parse(inputStructure);

       String loggLine = new String();
       loggLine = file.readLine();
       LogEntry logObject = new LogEntry();

       //System.out.println(loggLine.length());     //debug

       while(loggLine != null)
       {
            if(loggLine.length() > 0)
            {
                //System.out.println(loggLine);        //debug
                logObject = parser.logParser(loggLine);
                arCommunicator.add(logObject);
            }

            loggLine = file.readLine();
            //System.out.println(loggLine.length());    //debug


       }

    }

}
```

B.5 BinaryReader.java

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

```
/**
 * A reader for binary logfiles. Not implemented yet.
 * @version 0
 *
 * @see Reader
 * @see ASCIIReader
 */
public class BinaryReader extends Reader
{
    public BinaryReader()
    {
    }

    public void run()
    {
        System.out.println("BinaryReader startar");
    }
}
```

B.6 Detector.java

```
import java.lang.*;

/**
 * A virtual superclass for detectors.
 */
public class Detector extends Thread
{
    public Detector()
    {
    }

    /**
     * If this thread was constructed using a separate
     * <code>Runnable</code> run object, then that
     * <code>Runnable</code> object's <code>run</code> method is called;
     * otherwise, this method does nothing and returns.
     * <p>
     * Subclasses of <code>Thread</code> should override this method.
     *
     * @see     java.lang.Thread#start()
     * @see     java.lang.Thread#stop()
     * @see     java.lang.Thread#Thread(java.lang.ThreadGroup,
     *          java.lang.Runnable, java.lang.String)
     * @see     java.lang.Runnable#run()
     */
    public void run()
    {
    }
}
```

B.7 PollDetector.java

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

/**
 * A class that detects changes in a logfile.
 * It does this by checking when the file was last modified
```

```
 * and does this once a second.
 *
 * @author Daniel Törnqvist
 * @author Lars-Olof Leander
 * @version &#960;
 * @see Detector
 */
public class PollDetector extends Detector
{
    /**
     *
     * @param inCom   An instance of the class ReaderDetectcommunicator.
     *                Used to communicate with a Reader.
     * @param logPath A string containing the path and name of the logfile.
     */
    public PollDetector(ReaderDetectCommunicator inCom, String logPath)
    {
        communicator = inCom;
        path = logPath;
    }

    private File inFile;
    private ReaderDetectCommunicator communicator;
    private String path;

    /**
     * Checks for updates in the logfile.
     * Uses the File.lastModified method to detect updates
     * and when updates are detected, a signal is sent
     * to reader via a ReaderDetectCommunicator.
     * <br><br>
     * <b>Precondition:</b>  All attributes must be valid, ie the path
string must contain a path to a logfile.<br>
     * <b>Postcondition:</b> File checked for updates. If updates were
found, a message was sent to a ReaderDetetectorCommunicator.
     */
    public void run()
    {
        inFile = new File(path);
        long mod = 0;
        long modified = inFile.lastModified();

        System.out.println("Startar pollning");
        while (true)
        {

            try
            {
                Thread.sleep(1000);

            }catch(Exception ex){}

            mod = inFile.lastModified();
            System.out.println("Pollar filen");

            if (mod != modified)
            {

                modified = mod;
                communicator.update();

            }
```

```
        }
    }
}
```

B.8 AgentReaderCommunicator.java

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


/**
 * A communicator between instances of Agent and
 * instances of subclasses to Reader.
 *
 * @author Daniel Törnqvist
 * @author Lars-Olof Leander
 * @version &#960;
 */
public class AgentReaderCommunicator
{
    private LinkedList loggBuffer;

    public AgentReaderCommunicator()
    {
        loggBuffer = new LinkedList();
    }


    /**
     * Adds a LogEntry to the linkedlist loggBuffer.
     * <br>       <br>
     * <b>Precondition:</b>  True <br>
     * <b>Postcondition:</b> log added to the buffer.
     * @param logg   The LogEntry that's to be added to the linkedlist
loggBuffer.
     */
    public synchronized void add(LogEntry log)
    {
        loggBuffer.addLast(log);
        notifyAll();
    }

    /**
     * Waits until there's at least one LogEntry in loggBuffer.
     * Then removes the first post in the list and returns it.
     * <br>   <br>
     * <b>Precondition:</b>  True <br>
     * <b>Postcondition:</b> a log removed from the buffer and returned.
     * @return The first LogEntry in the linkedlist loggBuffer
     */
    public synchronized LogEntry get()
    {
        while (loggBuffer.size() == 0)
        {
            try
            {
                wait();
            }catch(Exception ex){}
```

```
        }

        return (LogEntry) loggBuffer.removeFirst();
    }


}
```

B.9 ReaderDetectCommunicator

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

/**
 * A communicator between instances of subclasses to Reader and
 * instances of subclasses to Detector.
 *
 * @author Lars-Olof Leander
 * @author Daniel Törnqvist
 * @version &#960;
 */
public class ReaderDetectCommunicator
{
    private boolean updated;   //behovs kanske inte eftersom bara en trad
vantar...

    public ReaderDetectCommunicator()
    {
        updated = false;
    }

    /**
     * Sets the updated field to true and wakes objects waiting
     * all reader objects waiting on this object.
     *<br><br>
     *<b>Precondition:</b>  The logfile has been updated.
     *<b>Postcondition:</b> All waiting threads has been notified.
     */
    public synchronized void update()
    {
        updated = true;
        //System.out.println("Notify'ar");   //debug
        notifyAll();
    }

    /**
     * Objects running this method is set to wait until
     * the updated field is set to true.
     *<br><br>
     *<b>Precondition:</b>  True.
     *<b>Postcondition:</b> Updated set to false.
     */
    public synchronized void isUpdated()
    {
        while(!updated)
        {
            try
            {
                //System.out.println("Vantar");    //debug
                wait();
```

```
            }catch(InterruptedException ex){}
        }
        updated = false;
    }
}
```

B.10 Parse.java

```
import java.lang.*;
import java.util.*;

/**
 * Class for converting a string into a LogEntry object.
 *
 * @author Lars-Olof Leander
 * @author Daniel Törnqvist
 * @version &#960;
 */
public class Parse
{
    /**
     * Divides the String containing the inputstructure
     * into an array of strings and saves it as an attribute.
     *
     * @param input  String containing the strucure of the expected input.
     */
    public Parse(String input)
    {
        int i = 0;

        StringTokenizer tokenizer = new StringTokenizer(input);

        while(tokenizer.hasMoreTokens())
        {
            inputOrder[i] = tokenizer.nextToken();
            i++;
        }
        numberOfColumns = i;
    }

    private String[] inputOrder = new String[15];
    private int numberOfColumns;

    /**
     * Takes a string and converts it into a LogEntry object.
     *
     * <br><br>
     * <b>Precondition:</b>  The input structure must be correctly
specified and numberOfColumns must be correct.<br>
     * <b>Postcondition:</b> The string has been converted to a instance of
LogEntry.
     * @param log    A string containing the line of log.
     * @return A LogEntry object containing the line of log.
     */
    public LogEntry logParser(String log)
    {
        int i = 0;

        LogEntry logObject = new LogEntry();
```

```
        StringTokenizer logTokenizer = new StringTokenizer(log);

        String temp = new String();

        while(logTokenizer.hasMoreTokens() && i < (numberOfColumns - 1))
        {
            logObject.setField(inputOrder[i],logTokenizer.nextToken());
            i++;
        }

        while(logTokenizer.hasMoreTokens())
        {
            temp = temp.concat(logTokenizer.nextToken() + " ");
        }

        logObject.setField(inputOrder[i],temp);

        return logObject;
    }

}
```

B.11 LogEntry

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

/**
 * A instance of this class contains a log entry.
 * It uses a hashtable to store the data with a key defined in the
configfile.
 *
 * @author Lars-Olof Leander
 * @author Daniel Törnqvist
 * @version &#960;
 */
public class LogEntry
{
    public LogEntry()
    {
        log = new Hashtable();
    }

    private Hashtable log;

    /**
     * Inserts a string into the hashtable using the specified key.
     * <br><br>
     * <b>Precondition:</b>  key != null && value != null <br>
     * <b>Postcondition:</b> The specified value has been entered into the
hashtable under the specified key.
     * @param key    The "name" of the datafield.
     * @param value  The data that is to be inserted into the hashtable.
     */
    public void setField(String key, String value)
    {
        log.put(key, value);

    }
```

```
    /**
     * Returns the data in the field with the specified key.
     * <br><br>
     * <b>Precondition:</b>  A field with the specified key must exist in
the hashtable.<br>
     * <b>Postcondition:</b> The data in the field with the specified key
has been returned.
     * @param key    The "name" of the datafield to be returned.
     * @return A string containing the data in the field specified by the
key.
     */
    public String getField(String key)
    {
        return (String) log.get(key);        //Vi vet att vi bara stoppar
in Strings alltså kommer det bara Strings ut.
    }

}
```

B.12 ClientServerCommunication

```
import java.io.*;
import java.lang.*;
import java.util.*;
import java.net.*;


/**
 * Creates a connection to a server and sends
 * LogEntrys to the server.
 *
 * @author Daniel Törnqvist
 * @author Lars-Olof Leander
 * @version &#960;
 */
public class ClientServerCommunication
{
    private String[] outputStructure = new String[15];
    private String ipAddress;
    private Integer portNumber;
    private Socket toMatrixServer = null;
    private PrintWriter out = null;

    /**
     * Reads the output section of the configfile. Creates
     * a connection to a server, with the data from the
     * configfile.
     *
     * @exception IOException
     */
    public ClientServerCommunication() throws IOException
    {
        int i = 0;

        SectionReader configFile = new SectionReader(new
FileReader("config.conf"), "output");

        String output = configFile.readLine();
```

```
        StringTokenizer tokenizer = new StringTokenizer(output);

        while(tokenizer.hasMoreTokens())
        {
            outputStructure[i] = tokenizer.nextToken();
            i++;
        }

        ipAddress = configFile.readLine();
        portNumber = new Integer(configFile.readLine());


        System.out.println(ipAddress);
        try{
            toMatrixServer = new Socket(ipAddress, portNumber.intValue());

            out = new PrintWriter(toMatrixServer.getOutputStream(), true);
        }
        catch (UnknownHostException e) {
            System.err.println("Could not connect to Matrix-server at:" +
ipAddress + ":" +portNumber.intValue());
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to
server.");
            System.exit(1);
        }


    }

    /**
     * Converts the LogEntry object to a string and sends it
     * to the server.
     *
     * <br><br>
     * <b>Precondition:</b>  Must have a connection to a server and a
output structure must be specified. <br>
     * <b>Postcondition:</b> The log entry has been written to the socket.
     * @param log    A LogEntry object that is to be sent to the server.
     * @exception IOException
     */
    public void sendToServer(LogEntry log) throws IOException
    {
        int i = 0;
        String logString = new String();

        while(outputStructure[i] != null)
        {
            logString = logString.concat(log.getField(outputStructure[i]));
            logString = logString.concat(" ");
            i++;
        }

        System.out.println(logString);
        out.println(logString);
    }

}
```

B.13 SectionReader

```
// File SectionReader.java

/** A filtered reader for setup sections, 980828
 */

import java.io.Reader;
import java.io.BufferedReader;
import java.io.IOException;

/**
 * The filter reads a line oriented reader and filters out
 *  all lines which start with a ; (comment lines).
 *  The reader starts after a section header line [<Header>] and returns
 *  eof at the end of the file or at the start of a new section.
 *  It also returns eof if the section header is not found.
 *
 *  Restriction:
 *  Since it is line oriented it is defined as a subclass of
BufferedReader.
 *  The filtering only works in line oriented reading (using readLine())
 *  Character reading reads the physical underlying input stream directly
 *
 * @author Eivind J. Nordby
 */

// public
class SectionReader extends BufferedReader {
    // Apply lazy header detection in order to
    // follow the convention not to throw IOException in the constructor

    String sectionHeader;  // for my section
    boolean headerLocated;  // is the header located?
    boolean eof;   // logical of physical eof encountered
    boolean debug;

    public SectionReader(Reader in, String sectionHeader) {
        this(in, sectionHeader, false);
    }     // SectionReader(Reader, String)

    public SectionReader(Reader in, String sectionHeader, boolean debug) {
        super(in);
        this.sectionHeader = sectionHeader;
        headerLocated = false;  // triggers localisation at the

     // first read operation
        eof = false;   // eof not yet encountered
        this.debug = debug;  // debugging mode
    }     // SectionReader(Reader, String)

    public boolean markSupported() {
        return false;
    }     //  markSupported()

    public String readLine() throws IOException {
        // Post: returns the next line of the section demanded or null
        String result;

        locateHeader();
        result = doReadLine();
```

```
        if (result != null && isSectionHeader(result))
{
            eof = true;   // blocks further line oriented reading
            result = null;
        }    // end of section
        return result;
    }     // String readLine()

    ////////////////////////////////////////
    // Private support functions
    ////////////////////////////////////////

    /** This is a comment for JavaDoc
  *  @param will describe a parameter
  *
  */

    private boolean isSectionHeader(String line) {
        // Pre: line != null
        // Post:Returns true is the line is a section header
        // A line is a header line if it starts with a [ and ends with a ]
        if (debug) {
            if (line.startsWith("[") && line.endsWith("]"))
                System.err.println("\"" + line + "\" is a header line");
            else
                System.err.println("\"" + line + "\" +  is not a header
line");
        }    // debug
        return line.startsWith("[") && line.endsWith("]");
    }     // isSectionHeader

    private void locateHeader() throws IOException {
        // Pre: true
        // Post:the header is located if it is present and
        // locateHeader == true

        if (!headerLocated) {
            // Read up to the section header if any
            String headerLine = "[" + sectionHeader + "]";
            String line = null;
            if (debug) {
                System.err.println("Searching section header" +
sectionHeader);
            }    // debug
            do {
                line = doReadLine();
                if (debug) {
                    System.err.println("search:\t\"" + line + "\"");
                }   // debug
            } while (line != null && !line.equals(headerLine));
            // line == null || line.equals(headerLine)
            headerLocated = true;
        }    // !headerLocated
    }     // locateHeader()

    private String doReadLine() throws IOException {
        // Basic read line operation for filtering out comment lines
        // Post: return the next physical non comment line, null
        String result = new String();
        if (eof)
            result = null;
        if (!eof) {
```

```
            do {
                result = super.readLine();
                if (debug) {
                    System.err.println("line:\t\"" + result + "\"");
                }   // debug
            } while (result != null && result.startsWith(";"));
            // result == null || !result.startsWith(";");
            eof = result == null; // update the eof flag
        }    // !eof
        return result;
    }     // doReadLine()
}     // class SectionReader
```

C Javadoc för Matrix-servern

C.1 Server

Package

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class Server

```
java.lang.Object
  |
  +-Server
```

---

public class **Server**

extends java.lang.Object

The main class in the Server application. Creates a serversocket and starts new ServerThread objects for each connecting client.

Constructor Summary	
Server()	

Method Summary	
static void	**main**(java.lang.String[] args)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Server

```
public Server()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
                throws java.io.IOException
        java.io.IOException
```

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

C.2 ServerThread

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS NEXT CLASS **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ServerThread

```
java.lang.Object
  |
  +-java.lang.Thread
        |
        +-ServerThread
```

All Implemented Interfaces:

java.lang.Runnable

---

public class **ServerThread**

extends java.lang.Thread

A class that handles a connection to a client.

Field Summary

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary	
ServerThread(java.net.Socket socket)	

Method Summary	
void	**run**() Reads from the socket and prints it on the screen.

Methods inherited from class java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

ServerThread

```
public ServerThread(java.net.Socket socket)
```

Parameters:

`socket` - The socket that the class reads from.

Method Detail

run

```
public void run()
```

Reads from the socket and prints it on the screen.

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

run in class java.lang.Thread

Package

Class

Tree

Deprecated

Index

Help

PREV CLASS NEXT CLASS | **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

D Serverns källkod

D.1 Server.java

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

/**
 * The main class in the Server application.
 *
 * Creates a serversocket and starts new ServerThread
 * objects for each connecting client.
 *
 * @author Lars-Olof Leander
 * @author Daniel Törnqvist
 * @version pi/2
 */
public class Server
{
    public Server(){}

    public static void main(String[] args) throws IOException
    {
        ServerSocket serverSocket = null;
        String input = null;
```

```
        System.out.println("Matrix Server ver. pi/2");

        try {
            serverSocket = new ServerSocket(3729);
        } catch (IOException e) {
            System.out.println("Could not listen on port: 3729");
            System.exit(-1);
        }

        while(true)
            new ServerThread(serverSocket.accept()).start();


    }


}
```

D.2 ServerThread.java

```
import java.io.*;
import java.net.*;
import java.lang.*;
/**
 * A class that handles a connection to a client.
 *
 * @author Lars-Olof Leander
 * @author Daniel Törnqvist
 * @version pi/2
 */
public class ServerThread extends Thread {
    /**

     *
     * @param socket The socket that the class reads from.
     */
    public ServerThread(Socket socket)
    {
        this.clientSocket = socket;
    }

    private Socket clientSocket = null;

    /**
     * Reads from the socket and prints it on the screen.
     */
    public void run()
    {
        try{

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            String temp = new String();

            temp = in.readLine();

            while (temp != null)
            {
```

```
                System.out.println(temp);
                temp = in.readLine();
            }


            }catch(SocketException ex){
               System.out.println("SocketException");
               // System.exit(0);
            }catch(IOException ex){
                System.out.println("IOException");
                //System.exit(0);
            }
    }
}
```