Datasynkronisering åt DISA

– Låghastighetsanpassad datasynkronisering av maskinparametrar

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Ingenjörssektionen / Datateknik

Examensarbete:
Joel Hervén
Daniel Bergquist

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds Universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2011

Sammanfattning

DISA, ett ledande företag inom gjuteribranschen som levererar maskiner till flertalet stora företag i världen, var i behov av ett modernare system för övervakning av maskinparametrar ute hos kund. Ett inledningsprojekt startades hösten 2009 med sex involverade studenter där en första version på en sådan lösning togs fram. Med detta som utgångspunkt har lösningen vidareutvecklats med fokus på datalagringen hos kund samt överföringen till en central server i Danmark. Flera prototyper har tagits fram och utvärderats för att hitta den mest robusta och bandbreddssnåla lösningen. Ett huvudkriterium har varit maximal anpassning även till långsamma anslutningar som kan förekomma i länder med dåligt utbyggd bredbandsstruktur. Som avslutning har den första färdiga prototypen testkörts hos en kund i Norge.

Nyckelord: Databas, MySQL, Låghastighetsanslutning, DISA, Synkronisering, Komprimering.

Abstract

DISA, a leader in the foundry industry that supplies equipment to various companies in the world, was in need of a more modern system for monitoring and storage of machine parameters at the customer. An initial project was started in autumn 2009 with six students involved were the first version of such a solution was developed. On that basis, the solution has been further developed in this project with the focus on data storage and data transfer to a central server in Denmark. Several prototypes have been developed and evaluated to find the most robust and bandwidth-efficient solution. A key criterion was the maximum adjustment even for slow connections that may exist in countries with poorly developed broadband infrastructure. As a final, the first finished prototype was tested at a customer in Norway.

Keywords: Database, SQL, Low-speed connection, DISA, Synchronization, Compression.

Förord

Detta arbete avslutar vår utbildning i Datateknik på Campus Helsingborg.

Främst ett tack till vår handledare Nils Assarsson som hela tiden haft en väldigt positiv och uppmuntrande inställning till projektet, samt även det föreliggande samarbetet som skapade förutsättningarna.

Även ett tack till vår examinator Mats Lilja vars undervisning ligger till grund för många kunskaper som gjort resultatet möjligt.

Innehållsförteckning

1 Inledning

1.1 Bakgrund och problem

DISA är världens ledande leverantör av gjuteriutrustning och ytbehandlingssystem för metall. Med huvudkontor i Danmark och kunder i minst tre världsdelar skapar de ett världsomfattande nätverk. För att underhålla och ge service har DISA därför ett antal tekniker som ofta gör besök hos sina kunder för att felsöka och serva maskinerna. I ett första steg att minska behovet av resor och ge effektivare service har DISA från ett tidigare projekt ett nyutvecklat system för lagring av mätvärden för deras gjuterimaskiner. Då detta arbete påbörjas är nämnda system ännu inte i bruk. Informationen kommer dock endast finnas lokalt i respektive maskinhall utan åtkomst utifrån. En effektiv överföringslösning behövs därför för att skapa full tillgång från en central punkt. Samtidigt som lösningen måste vara framtidssäker måste dagens långsamma internetuppkopplingar i vissa länder tas i åtanke. Höga krav kommer därför ställas på minimal dataöverföring och snabb åtkomst.

1.2 Avgränsningar

Vi förutsätter ett lokalt säkert nätverk och kommer inte fokusera något mer omfattande på säkerheten utanför länken kund -> DISA (och vice versa), men kommer givetvis ha den i åtanke även fast ansvaret kommer ligga hos kunden. När det gäller hårdvaran systemet körs på kommer vi i den mån det går att anpassa oss efter vad som beslutats i grundprojektet som detta examensarbete baserats på.

1.3 Metodik

En empirisk studie av lagringsmöjligheter och överföringsmetoder ska ge grunden för designen, av en lösning vi därefter utvecklar för DISA.

1.4 Förväntat resultat

Målet är en effektiv lagringslösning som ska klara av långtidslagring på ett sådant sätt att den kommer att bidra till DISAs verksamhet. Resulterande överföringsmetod ska vara optimerad för låghastighetsanslutningar. Applikationer som i efterhand kommer att utvecklas till kunder och interna servicetekniker ska enkelt kunna kommunicera genom vald lösning och tydlig dokumentation(API) bör därför finnas.

2 Förstudie

Följande förstudie är en utveckling av bakgrunden som främst presenterar det tidigare projekt som ligger till grund för detta examensarbete och valen av de tekniker och metoder som det medfört.

En kurs vid namn Projekt åk 3 inledde samarbetet mellan sex studenter och Nils Assarson, applikationsingenjör på DISA Industries A/S, Herlev. Målet med projektet var, utöver att tillämpa effektiva projektmodeller, att utveckla och förhoppningsvis realisera det system som avläser maskinernas parametrar och lagrar dem för vidare användning. Detta system kommer vidare i rapporten att benämnas som DACS (Disa Acquisition Software).

Slutresultatet blev en hårdvara vid namn DI-718Bx från DataQ som sköter insamlandet av mätvärdena. Kommunikationen med DataQ boxen sker via en tillhörande ActiveX Control, som är en Windows-komponent och deras tidigare motsvarighet till Java. På dessa grunder valdes projektet att utföras i en Microsoft-miljö med språket C# och utvecklingsmiljön Visual Studio.

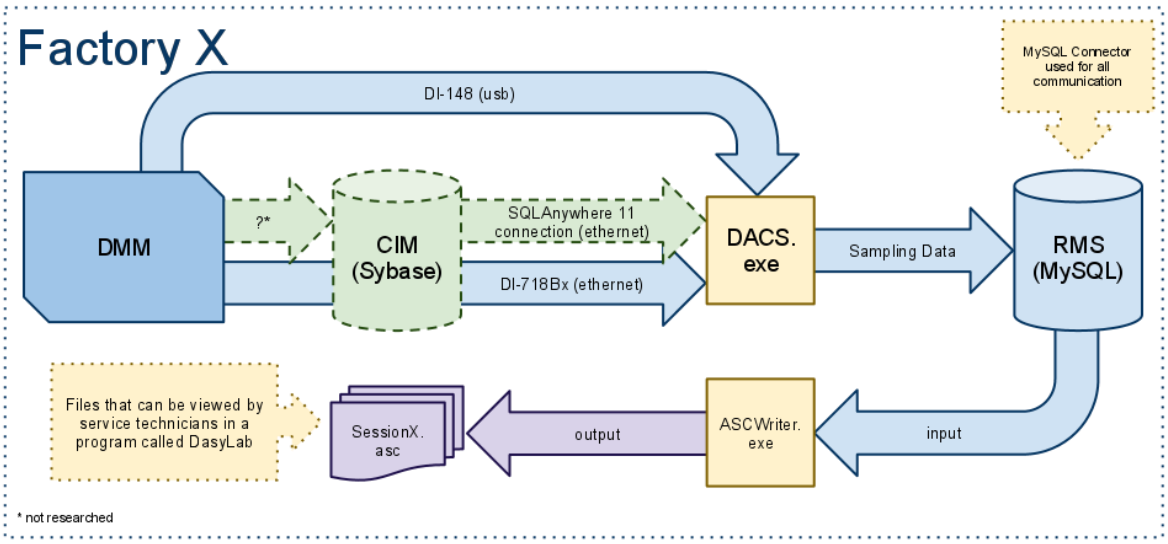
Då C# är ett kraftfullt och mycket produktivt språk som snabbt växer på marknaden blev det ett naturligt val att välja detta som plattform även för den fortsatta utvecklingen.

2.1 Remote Monitoring Service

DACS är en del av det större systemet Remote Monitoring Service (RMS) som är en vidareutveckling av ett koncept kallat Remote Diagnostic Assistance (RDA). RDA gör det möjligt för en servicetekniker att ansluta till den operatörspanel som kontrollerar gjuterimaskinen. De får då bland annat tillgång till vilka inställningar som är aktuella samt vilka felkoder som triggats den närmaste tiden. Detta är i grund och botten ett fjärranslutningsprogram som kommer åt datorn som operatörsprogrammet kör på via ett uppringt modem.

För att få detaljerad information om hur signalerna ser ut som styr maskinen är det nödvändigt att koppla in extern utrustning som kan mäta spänningsnivåer. Detta kopplar sedan serviceteknikern till sin PC där signalerna kan avläsas i speciell programvara. Detta kopplas endast in under felsökning och kan kosta dyrbar tid att få igång (främst att få dit en tekniker), samt bidrar till sladdtrassel. Huvudmålet med RMS är att bygga ett statiskt system av detta, något som alltid är inkopplat. För detta utvecklades DACS.

2.2 Fördjupning DACS



Figur 1 Överblick DACS version 1

DACS i dess befintliga form beskrivs i Figur 1. Kärnan i systemet är processen med samma namn, ”DACS.exe” och dess uppgift är att samla in samplingsdata och spara den i en MySQL-databas vid namn RMS. Hårdvaran som används för att sampla data från en DISA Moulding Machine (DMM) är huvudsakligen DI-718Bx som kommunicerar via Ethernet. En billigare variant som kommunicerar via USB finns också inkopplad. Denna är tänkt att sampla signaler som är ”långsamma”, t.ex. temperaturer.

I Figur 1 kan man se en Sybase-databas vid namn CIM. Detta är en databas som ingår i DISAs befintliga system och är inget vi kan förändra. Databasen är valfri och finns därför inte i alla fabriker. Där den finns kan den bland annat spara modellinställningar samt produktionsstatistik. Fördelen med CIM är att data från deras DMM blir enklare att integrera i egna fabrikslösningar. När CIM finns, eller närmare CIM3Data-tabellen finns för en DMM som DACS övervakar finns möjligheten att starta sampling då modellskifte sker. För att kommunicera med CIM använder DACS programvaran SQLAnywhere 11. SQLAnywhere tillhandahålls av Sybase och innehåller en rad kraftfulla funktioner när det gäller datahantering, synkronisering samt datautbyte. Tyvärr saknas synkroniseringsmöjligheter med just MySQL.

När signalerna från en DMM ska analyseras används ett program kallat DASYLab. Detta har använts under en längre tid av DISAs tekniker. Programmet läser in en fil av ASCII-format, en kommaseparerad fil med några rader metadata i början av filen. Samplingsdata sparas binärt i RMS-databasen vilket inte är kompatibelt med DASYLab. Programmet ASCWriter har utvecklats för att generera ASCII-filer utav binär samplingsdata.

3 Tillvägagångssätt

Under januari månad 2010 hade vi vårt första möte med Nils där förslag på utformningen av examensarbetet diskuterades. Ett antal idéer kom fram som alla var relevanta i DISAs framtidsvisioner. Vi funderade över förslagen ett par dagar innan en projektbeskrivning skrevs ihop och skickades till Nils. Då vi haft mycket kontakt sedan tidigare kom vi snabbt fram till en utformning som kändes givande för båda parter. I samband med detta var vi även över på fabriken i Herlev, Danmark för att testa det system som utvecklats under hösten och som låg till grund för fortsättningen. Även då det inte direkt ingick i detta examensarbete var det en viktig grund för upplägget och för att nå det önskade slutresultatet.

En skiss över strukturen gjordes för att kunna angripa problemets delar en efter en. Först och främst skulle ett val av databas göras som sedan låg till grund för ett antal möjligheter till synkronisering som utvärderades. Ett eget filformat togs fram då det blev den maximala lösningen för komprimeringen av data. I C# utvecklades programvaran som tog hand som paketeringen samt överföringen till motsvarande sida.

Parallellt med detta gjorde vi ytterligare undersökningar för att ta fram rekommenderade specifikationer gällande hårdvaran, då främst den fysiska lagringen på DISAs huvudserver.

Trots att arbetet huvudsakligen ägde rum på Campus Helsingborg hade vi regelbundna möten med Nils för att pejla av arbetsgång, diskutera eventuella nya idéer och planera inför en installation hos den första fabriken i Norge. Inga större ändringar uppkom under arbetets gång, men lite mindre modifikationer fick ständigt göras för att anpassa strukturen till de ytterligare verktyg som DISA parallellt tog fram genom andra projektgrupper.

3.1 Prototyper

En viktig del i utvecklingen var framtagandet av flera olika prototyper. Utgångsproblemet var i sig ganska rättframt och svårigheten låg i att hitta de tekniker och upplägg som skulle uppfylla de definierade kraven på bästa möjliga vis. Flera möjliga lösningar övervägdes under projektets gång, där vi så långt det var rimligt tillämpade metoden utveckla – testa – utvärdera.

3.2 Sluttest Danmark

I slutet av maj gjordes ett sluttest på DISA i Danmark med alla involverade parter som ett sista förberedande inför besöket i Norge. Alla tester gick igenom som förväntat.

3.3 Installation Norge

Läser man på Jøtuls webbsida[1] kan man läsa att Jøtul är ett av Norges äldsta industriföretag. Företaget är en världsledande norsk producent av kaminer och insatser med dotterbolag i flera länder. Produktionen sker i bland annat Fredrikstad i Norge och när vi i denna rapport nämner Jøtul är det denna plats vi syftar på. I slutet av maj gjordes ett tredagarsbesök hos Jøtul där målet var att installera så mycket som möjligt av den utvecklade hårdvaran. Tillsammans med Jøtuls tekniker diskuterade vi hur hårdvaran skulle anslutas till nätverket med avseende på säkerhet och tillgänglighet. Den utvecklade mjukvaran installerades på en befintlig PC tillhandahållen av Jøtul. Vi stötte dock på problem vid installationen av den komponent vi benämner som CIMsync, dvs. synkroniseringen mellan den redan körande Sybase databasen och vår nydesignade MySQL databas. Efter en mödosam nästan två dagar lång felsökning kunde det konstateras att problemet berodde på en äldre Sybase-databas, i detta fall kördes version 8 medan tidigare tester körts mot version 10 och 11. I valet mellan att skriva om mjukvaran eller anpassa den befintliga lösningen valdes det tillsammans med Nils, att föreslå alla kunder med äldre Sybase versioner en uppgradering för att garantera full kompatibilitet. Detta var inget vi hade möjlighet att lösa på plats vilket blev avslutet för den första prototypinstallationen.

3.4 Arbetsfördelning

Då detta arbete är betygsgivande och har utförts av flera personer kan det vara av vikt att nämna hur arbetet har fördelats. Till en början utfördes allt arbete tillsammans, då vi försökte komma fram till en rimlig lagringsstruktur. Vi märkte efter ett tag att vi hade många olika delar att täcka in och delade därför upp oss för att kunna fokusera på olika områden. Det innebär att undersökningen av replikationsförmågor hos olika databaser, samt programmet DisaTX främst ligger under Joel Hervéns ansvar. Samtidigt som det arbetet pågick utvecklades DMI-formatet av Daniel Bergquist. Efter ordinarie planering har ytterligare arbete utförts av Daniel Bergquist, under anställning av DISA, som har kommit med i denna rapport. Det handlar främst om SQLite som filformat samt undersökningen av vilka sökvägar och mappstrukturer som vanligtvis används i Windows-miljö.

[1] http://www.jotul.se (2010-07-18)

4 Databaser

Det finns en stor mängd databashanterare på marknaden, alla olika populära inom sina olika branscher. Vi hade möjligheten att ganska fritt kunna välja och föreslå passande programvaror. Ett önskemål var dock från Nils och DISAs sida givetvis att undvika dyra licensavgifter som inte var nödvändiga. Följande alternativ har undersökts och utvärderats.

4.1 Varför en databas

En möjlighet vore att bara lagra all data i gigantiska textfiler. En textfil är både gratis (i form av licenskostnader), enkel att flytta samt direkt läslig för det mänskliga ögat. En ny textfil skulle kunna skapas för varje sampling och data skulle kunna lagras kommaseparerad rad för rad. Nackdelarna blir dock snabbt flera och ökar snabbt i problematik när system blir mer komplexa och mer svåröverskådliga. Felaktig data, redundans, flera användare och sökning är de huvudsakliga problem man snabbt stöter på för att uppfylla våra kriterier.

En korrekt uppsatt databas är vad vi kallar logiskt koherent, dvs. den innehåller ingen motsägelsefull information. Givetvis måste databasen få förklarat för sig vilken information som hör ihop och hur den hör ihop. Vi förutsätter här den vanligaste typen av relationsdatabaser där detta är synnerligen enkelt för den som har rätt kunskaper. Att åstadkomma samma funktion i en textfil skulle inneburit betydligt fler rader kod, som framförallt kan innehålla dolda fel om inte ordentliga testfall körs.

Redundans är en synnerligen viktig faktor, då vi dels anpassar lagringen för enorma mängder data, och dels att just komprimeringen är ett av projektets huvudsakliga kriterier. En databas gör det lätt att skapa identifikationsnummer som pekar på en större datamängd som uppträder frekvent. Liknande uppdelning i en textfil gör snabbt att de inte längre är särskilt läsvänliga och det som tidigare var en fördel är inte längre sant.

Man behöver inte ens nämna sökningar i filer, eller ännu värre när flera program vill t.ex. söka samtidigt för att det ganska snabbt ska visa sig att användningen av simpla textfiler inte resulterar i en så enkel lösning som det först kunnat verka. Trots att behovet för de mer avancerade databasfunktionerna inte finns i det här projektet, är svaret alltså ändå givet att en databas är den rätta lösningen.

4.2 MySQL

MySQL är en kostnadsfri databashanterare med öppen källkod för den normale användaren. Det är antagligen den mest spridda databashanteraren och syns allra främst inom webbserver-kategorin. Styrkan hos MySQL[2] sägs vara snabbheten och enkelheten, detta tillsammans med mycket hög stabilitet gör det till en mycket bra databashanterare i många sammanhang. Detta leder givetvis till att en hel del funktioner saknas som återfinns i dyrare databashanterare, i många fall är detta dock inget problem och MySQL har även gjort stora förbättringar de senaste åren för att knappa in på detta.

4.3 Sybase

Sybase[3] är en databashanterare som redan används av DISA och en av de ledande inom just industribranschen. Alla standardfunktioner finns och stödet för replikering är omfattande, även tillsammans med flertalet andra databaser. Sybase kräver licens för att få användas, vilken enligt projektets önskvärda kriterium skulle undvikas i de fall fria alternativ fanns.

4.4 PostgreSQL

PostgreSQL är ytterligare en kraftfull databashanterare baserad på öppen källkod, den är ursprungligen mer känd som en databashanterare något långsammare än MySQL men med fler funktioner. Senare versioner av bådadera databashanterarna har dock drivit skillnaderna närmre varandra och mycket funktionalitet som går att hitta i PostgreSQL finns idag även i MySQL. Replikation finns inte direkt inbyggt i databasen utan görs genom ett separat tilläggsverktyg. Detta gör administreringen lite svårare och sättet det är byggt på gör att hastigheten skalar sämre med flera servrar. Upplägget är i övrigt likt det som syns i MySQL.

4.5 Val av databas

Sybase används redan av DISA för deras CIM-databas, och skulle därför kunnat bli ett självklart förstaval på denna utbyggnad. Fallet blev dock helt annorlunda då vi tillsammans med vår handledare valde att utföra ett, i den mån det gick, isolerat arbete ifrån DISAs utvecklingsavdelning. Syftet med detta var kunna se alla möjligheter istället för att riskera snöas in i gamla rutiner.

Kriterierna i vårt val av databas var egentligen inte många. Målet var ett stabilt och säkert sätt att lagra stora mängder information som skulle distribueras till

[2] http://www.mysql.com/why-mysql/ (2010-08-30)
[3] http://www.sybase.com (2010-08-30)

ganska få användare. Valet faller ganska naturligt på MySQL då den på ett enkelt sätt uppfyller målen för det här projektet.

5 Replikation MySQL

Replikation används i normalfallet antingen som en form av ren backuplösning, eller för att avlasta trycket på masterservern genom att tillhandahålla read-only information på flera slavservrar. Det finns många designer med allt från enkelnivåslösningar till pyramidlösningar för maximal skalbarhet. Grundprincipen fungerar med förutsättningen att en slavdatabas endast kan ha en master, MySQL har alltså i nuvarande skede inte stöd för s.k. multi-source replication.

5.1 Multi-source replication

Multi-source replication innebär alltså det helt motsatta fallet än de definierade grundförutsättningarna i MySQL. Istället för en huvudserver som distribuerar lasten på flera underinstanser, är det flera databaser som vill skapa en gemensam spegling av all information. Lösningen vi var ute efter hade alltså inget rättframt stöd via databashanteraren utan en kringlösning måste användas för att komma runt problemet. MySQL utvecklarna varnar dock användare för att vara mycket försiktiga med den här typen av lösning.

En undersökning av företagets framtida utvecklingsplaner för databasen resulterar i följande information:

MySQL worklog(WL#1697) Status: On-Hold — Priority: Low.[4]

Det är alltså ingen funktion man kan räkna med att se i databashanteraren inom en snar framtid. Det finns tredjepartslösningar att tillgå men då vi inte funnit någon som passar in på licens- och kostnadsmålet som vi vill uppnå, ser vi detta som en sista lösning.

5.1.1 Alternativ lösning

Det finns alternativet att köra en ny lokal MySQL-instans för varje produktionsdatabas, dvs. varje databas ute hos kund har en egen spegeldatabas hos DISA. Nackdelen är klumpig utbyggnad och installation, med lika omständligt underhållsarbete. Rent funktionsmässigt under körning fungerar det dock tillfredställande men ur ett framtidsperspektiv valde vi strikt att undvika ett så pass begränsat alternativ.

[4] http://forge.mysql.com/worklog/task.php?id=1697 (2010-04-18)

5.2 Envägs-replikation

Stödet för envägs-replikation från en masterserver till x antal slavar ges fullt stöd i MySQL 5. Själva synkroniseringen fungerar då asynkront vilket ger fördelen att en ständig förbindelse inte är nödvändig. Det går att begränsa exakt vilka specifika tabeller som ska kopieras eller om det ska gälla hela databasen. Alla frågor som körs på masterservern sparas i en loggfil och utförs regelbundet så fort kontakt med slavservern finns, vilket kan vara ofta eller mycket sällan till den gräns att loggfilen når sin maxstorlek och gamla händelser automatiskt börjar skrivas över. Replikering är en mycket kraftfull lösning vid många prestandaproblem, allt från lastbalansering till åtkomsttider. I det studerade fallet blir dock många av dessa aspekter ointressanta då vi jobbar med ett problem som är omvänt normalfallet. Istället för data som från en punkt ska nå flera är det data som från flera punkter ska samlas hos en.

5.2.1 Prestanda

Innan ett definitivt beslut var möjligt gällande metodens användbarhet var givetvis bandbreddsåtgången faktor nummer ett, som en huvudpunkt i hela arbetet. Vi satte upp en testmiljö med två dedikerade MySQL-servar på separata PC-stationer med Windows som operativsystem. En envägs master-slave replikation sattes upp mellan server 1 och 2, medan simulerad sampling kördes på masterservern och utnyttjad bandbredd uppmättes med Microsoft resursövervakare[5]. Samma test gjordes med och utan MySQL compressed protocol aktiverat vilket är en inställning som kan köras i läget på eller av.

5.2.1.1 MySQL compressed protocol i teorin

MySQL compressed protocol är en funktion som inte marknadsförs som en av databasens främre styrkor. Det är klienten som vid handskakning gör en förfrågan till servern att ett kompressionsprotokoll ska användas. Om stödet finns kommer servern/mastern försöka komprimera alla paket som ska skickas. Blir det komprimerade paketet storleksmässigt mindre än originalpaketet är det de som skickas istället, i annat fall sänds originalet. En overhead på 3 byte läggs till det komprimerade paketet.

5.2.1.2 Resultat

Compressed off – ca 100kb/s
Compressed on – *Inga tillförlitliga värden*

Ingen märkbar skillnad mellan komprimering och icke-komprimering uppmättes vid de simulerade testerna. Den direkta orsaken förblir okänd och

[5] Windows 7 Resursövervakare

tillsammans med flera andra problem ger det i helhet en väldigt ostabil lösning vi valde att inte arbeta vidare på.

6 DISA Machine Info (DMI)

För att lösa det speciella synkroniseringsförhållandet vi var ute efter valde vi att utveckla vårt egna system. Grunden för detta system ligger i vårt egenutvecklade filformat ”DISA Machine Info” vilket vi förkortar DMI. Vid varje synkroniseringstillfälle skapas en ny fil av detta format innehållande nya data från den lokala databasen. Denna fil genereras av programmet DMIWriter och importeras senare i centralservern av programmet DMIImporter.

6.1 Filformatet DMI

DISA Machine Info (DMI), ett filformat med filändelsen ".dmi", framtogs för att användas vid ett mellanled vid databasreplikation. Syftet är ett filformat som kan användas för att lagra databasinnehåll för senare import till en annan databas. Formatet ska vara flexibelt och inte bundet till en specifik databas. Det ska även ta upp mindre utrymme än MySQL's egna binlog-system som är ett format MySQL använder vid replikering.

6.1.1 EXIF

Att designa ett filformat är inte den mest triviala uppgiften. Men då ena projektmedlemmen tidigare arbetat med filformatet EXIF som till stor del liknar den lösning vi ville uppnå hade vi något att utgå ifrån. EXIF är ett format som flitigt används i JPEG-bilder sparade av digitalkameror. EXIF innehåller metadata som t.ex. kameramodell, miniatyrbild och inställningar vid fotograferingen. EXIF är utvecklat av JEITA men underhålls inte på något sätt och även om den inte är en erkänd standard av någon internationell organisation som ISO och IEC är den en standard bland tillverkare. Då JEITA inte har specifikationer publicerade på deras webbsida har information istället använts från den inofficiella sidan "EXIF.org".

EXIF använder sig av taggar tillsammans med metadata för att definiera vilka data det handlar om, inte helt olikt sättet som streckkoder sitter på varor för att definiera vilken vara det är. Ett system med taggar lockar till användning av DMI-formatet då man med några få byte kan beskriva vad efterföljande data egentligen representerar.

EXIF innehåller ett virtuellt filsystem. Image File Directory (IFD) kallas de grupper som innehåller metadata. En mappstruktur uppnås genom att en IFD

med speciella taggar kan länka till underliggande IFDs eller efter sin sista tagg länka vidare till en IFD som ligger på samma nivå.

eeee	*Antal poster*
tttt - ffff - nnnnnnnn – dddddddd	*Post 0*
tttt - ffff - nnnnnnnn – dddddddd	*Post 1*
.....	.....
tttt - ffff - nnnnnnnn – dddddddd	*Post eeee - 1*
oooooooo	*Offset till nästa IFD*

Figur 2 Grundstrukturen för en IFD

'tttt'	*2 bytes är tag-numret som säger vad datan representerar.*
'ffff'	*2 bytes är typen av data, t.ex. int eller short.*
'nnnnnnnn'	*4 bytes är antalet komponenter av föregående typ (vektorstorlek)*
'ddddddd'	*4 bytes innehåller datan eller offset som pekar var datan finns.*

Figur 3 Strukturen för en post inuti en IFD (12 byte stor)

De 2 första byte i en IFD berättar hur många poster den innehåller (se Figur 2). Beskrivelsen av en post går att se i Figur 3. Efter sista posten i en IFD ligger en offset på 4 byte till nästa IFD. Ifall värdet är 0 innebär det att inga efterliggande IFDs finns.

Hur bra uppfyller då IFD strukturen från EXIF våra kriterier på ett bra filformat? Den största delen av data vi ska spara med DMI är binära strömmar. I teorin kan headern på 12 byte för en post beskriva en över 12 dygns lång samplingsström på 250Hz på 16 kanaler. Resterande data för en samplingssession är minimal i jämförelse och formatet uppfyller den låga filstorlek vi är ute efter. Formatet är flexibelt i den mån att man helt enkelt kan definiera en ny tagg så fort man ska kunna hantera ny data, t.ex. ett nytt datafält i en databas.

Slutsatsen är att formatet är väldigt intressant och verktyg ska utvecklas för vidare undersökning.

6.1.2 XML

Extensive Markup Language (XML) är ett format som likt EXIF använder sig av taggar för att definiera den innehållande datan. Den stora skillnaden är att XML är ett textbaserat format och varje tagg består av start och stop.

"It is the age of XML: it is everywhere."[6]

Ovanstående citat väcker självklart intresse för XML och formatet har många fördelar:

- Textbaserat vilket gör det läsbart för en människa och inte bara maskin.
- Detaljerat format vilket gör det enkelt för datorn att upptäcka felaktigheter.
- Format med utbredd användning vilket gör att det finns många verktyg för att hantera XML.

Hur bra uppfyller då XML våra kriterier på ett bra filformat? Formatet är flexibelt då element och attribut kan läggas till utan att störa befintlig funktionalitet. Däremot tar filformatet stor plats då det är textbaserat. Dessutom är största delen data vi behöver spara binär, vilket inte är läsvänligt utan konvertering först. Man kan koda den binära datan med Base64 vilket ger en läsvänlig textsträng som inte innehåller radbrytningar och andra formateringstecken.

DMI är inte tänkt att vara ett redigerbart filformat. Är en fil skapad ska ingen gå in och ändra datan. Den stora fördelen att XML-formatet är läsbart av människa är att det ger en bra och enkel metod för att kontrollera så att viss data stämmer. Däremot behövs egentillverkade verktyg för att gå in och titta på den binära datan.

Slutsatsen blir att XML är ett väldigt intressant filformat, men i detta ändamål bidrar inte styrkorna till det önskade resultat vi är ute efter.

6.1.3 EBML

Extensive Binary Markup Language (EBML) är ett format inspirerat av XML men med den stora skillnaden att det är ett binärt format istället för textformat.

---

[6] http://www.w3.org/standards/xml/ (2010-08-17)

"There is also one disadvantage commonly said about XML : it's very verbose. That's why you should have default/assumed values in you EBML-based format as much as possible. So you just describe what is really necessary."[7]

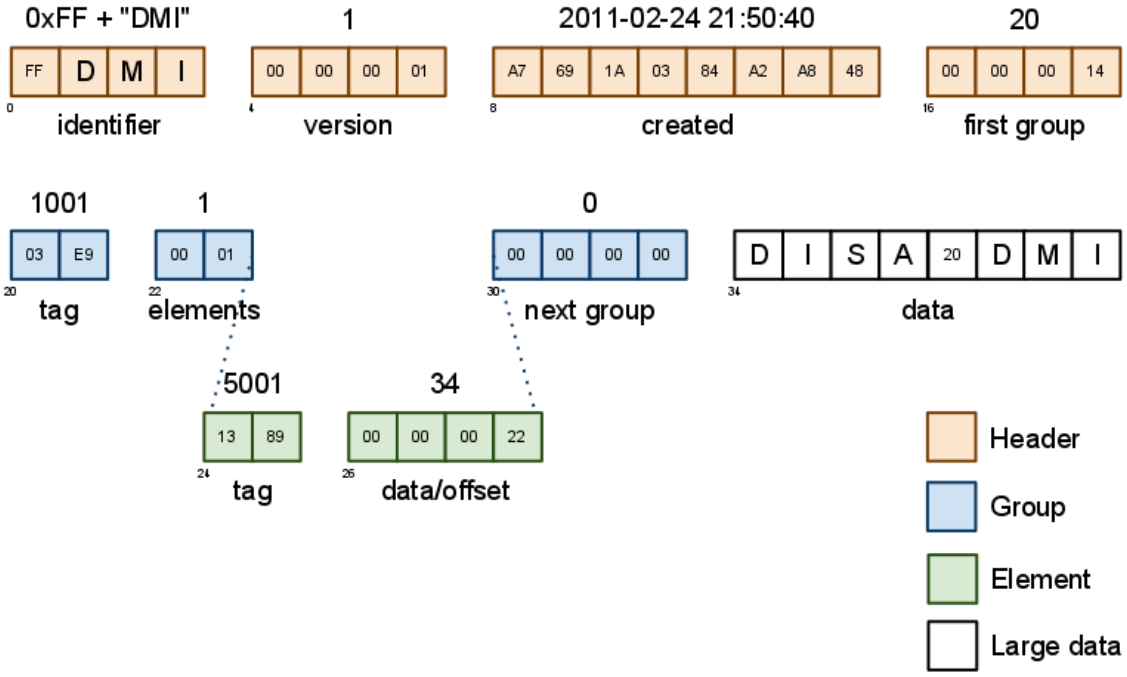
Detta format löser därmed problemen med XML, nämligen stora filer och onödig konvertering av binärdata. EBML skapades för att användas av Matroska, en mediabehållare. Matroska kan samla flertalet audio-, video- och undertextströmmar i en och samma fil.

Nackdelarna med EBML är att det inte finns färdiga bibliotek för .NET som är utvecklingsplattformen vi arbetar med. Vi har heller inte hittat någon ordentlig beskrivning av formatet för att till fullo förstå strukturen.

Slutsatsen blir att EBML är ett väldigt intressant filformat och har fördelar där XML har nackdelar. Tyvärr stämmer även det motsatta då vi haft svårt att hitta verktyg och bibliotek att utveckla med men där XML redan har inbyggt stöd i .NET plattformen.

6.1.4 DMI Revision 1

Av de 3 olika format som jämförts valde vi IFD strukturen från EXIF att bli basen för DMI.



Figur 4 Visuell representation av strukturen för DMI Revision 1

[7] http://ebml.sourceforge.net/ (2010-06-17)

Header:

```
iiiiiiii            4 bytes identifierare
vvvvvvvv            4 bytes versionsnummer
cccccccccccccccc    8 bytes skapelsedatum
oooooooo            4 bytes offset till första session

Group:
gggg                Grupp
eeee                Antal element
tttt - dddddddd     Element 0
tttt - dddddddd     Element 1
....                Element ...
tttt - dddddddd     Element eeee - 1
oooooooo            Offset till nästa Grupp

'gggg'              2 bytes tag för grupp
'eeee'              2 bytes antal element
'tttt'              2 bytes tag för element
'dddddddd'          4 bytes data eller offset till data i
                    ett element
'oooooooo'          4 bytes offset till nästa grupp
```

Figur 5 Textbeskrivning av strukturen för DMI Revision 1

6.1.5 DMI Revision 2

För att förenkla användandet av programkoden för DMI gjordes ett försök att implementera generiska klasser samt använda en klass-hierarki. Fördelen med detta är att programkoden blir enklare att arbeta med då olika typer av databastabeller inte behöver särskilda klasser och funktioner längre. Verktyg utvecklades även för att generera den statiska data taggarna utgör. Med hjälp av de generiska klasserna blev det enklare att använda DMI formatet för att lagra CIM-data. En stor svaghet upptäcktes dock i de genererade filerna innehållande CIM-data. Till skillnad från samplings-datan som till största del utgjordes av långa binära strängar bestod CIM-datan mestadels av enskilda heltal på 4 bytes vardera. Varje fält i en tabell får en header på 6 bytes vilket innebär att metadatan tar större plats än datan den beskriver och en väldig redundans uppstår så fort flera rader sparas i samma fil. Ett annat problem är det faktum att CIM-databasen kan innehålla NULL-värden vilket inte förekommit innan. Detta innebär att ytterligare metadata krävs.

6.1.6 DMI Revision 3

```
Header:
iiiiiiii            4 bytes identifierare
vvvvvvvv            4 bytes versionsnummer
cccccccccccccccc    8 bytes skapelsedatum
gggggggg            4 bytes group offset
tttttttt            4 bytes extended tag offset
```

Figur 6 Textrepresentation av den nya header-strukturen införd i DMI Revision 3

Ett annat problem CIM-databasen bidrog till var det faktum att strukturen av den var väldigt osäker. Olika kolumner upptäcktes ha olika namn i olika implementationer, och vissa implementationer hade egna särskilda kolumner. För att lösa detta problemet används en uppslagsbok i filen där tags speciella för den filen kan definieras.

6.2 Program

6.2.1 DMIWriter

DMIWriter är en "Windows Application" skapad för plattformen ".NET Framework 3.5". Vad detta innebär att programmet har ett grafiskt användargränssnitt och kräver .NET 3.5 för att köras.
Programmets uppgift är att generera .dmi filer utifrån enskilda sessioner eller från angivet intervall. Att generera .dmi filer är tänkt att vara en uppgift som sköts automatiskt och detta program är mest ett hjälpverktyg för att testa funktionalitet samt demonstrera dess syfte.

DMIWriter finns även som ett konsol-program. Detta har begränsad funktionalitet och dess enda uppgift är att generera en .dmi fil innehållande ny data från senaste lyckade körningen.

6.2.2 DMIImporter

DMIImporter är likt DMIWriter en "Windows Application" skapad för ".NET Framework 3.5". Dess uppgift är att ladda .dmi filer för olika tabeller och uppdatera en befintlig databas med filernas innehåll. Programkoden är utdaterad och går därför inte att kompilera.

6.2.3 Bibliotek

DMI.dll
Både DMIWriter och DMIImporter använder sig av biblioteket DMI.dll för att generera och läsa .dmi filer.

Database.dll
Både DMIWriter och DMIImporter använder sig av biblioteket Database.dll för att kommunicera med databasen.

6.3 SQLite som filformat

Den 16 juli 2010 fortsatte RMS projektet i form av en sommaranställning för Daniel Bergquist från detta examensarbete och Marcus Klang som jobbat med analysdelen. Det befintliga DACS program vi hade var varken stabilt eller lättförståeligt. Det största problemet var den invecklade koden som var fylld av tillfälliga fixar vilket gav en oerhört svår kod att jobba med. För att lösa detta bestämde vi oss för att bygga om programmet från grunden och förhoppningsvis få ett mycket bättre resultat denna gång med hjälp av den erfarenhet vi samlat på oss under de två läsperioder vi varit delaktiga i RMS projektet. Eftersom ingen programdel nu var beroende av föregående MySQL-databas kunde vi byta till en databasmotor som eliminerade problemen vi hade med DMI-formatet. Den databasmotorn är SQLite. En minimalistisk databasmotor vars data antingen körs helt i minnet, eller sparas i en enskild fil. Den stora fördelen med detta är att varje samplingssession som körs då kan sparas till sin egna fil. För att sedan synkronisera med centralservern är det bara att söka igenom vilka filer man inte redan har, och metoder för att överföra och synkronisera filer finns det gott om.

Databasstrukturen som användes under utvecklingen av DMI-formatet har ändrats flertalet gånger. Exakta jämförelser av datastorlek kan därför inte göras med den SQLite-databas som används då detta skrevs (2011-01-31). Det vore möjligt att skapa nya taggar och grupper för att anpassa DMI till de nya tabeller och kolumner som används, för att sedan kopiera data från en sampling gjord till SQLite. Men det krävs många ändringar och tillägg i koden för att göra det möjligt. Alltför många för att den spenderade tiden ska vara värd det.

Det går däremot att göra en jämförelse på storleken förr (DMI) och nu (SQLite). Eftersom den största delen av en sampling består av just samplingsvärdena är bytes/sample ett intressant mått. Eftersom DMI enbart är tänkt att användas vid överföringar är storleken på okomprimerade filer inte intressant.

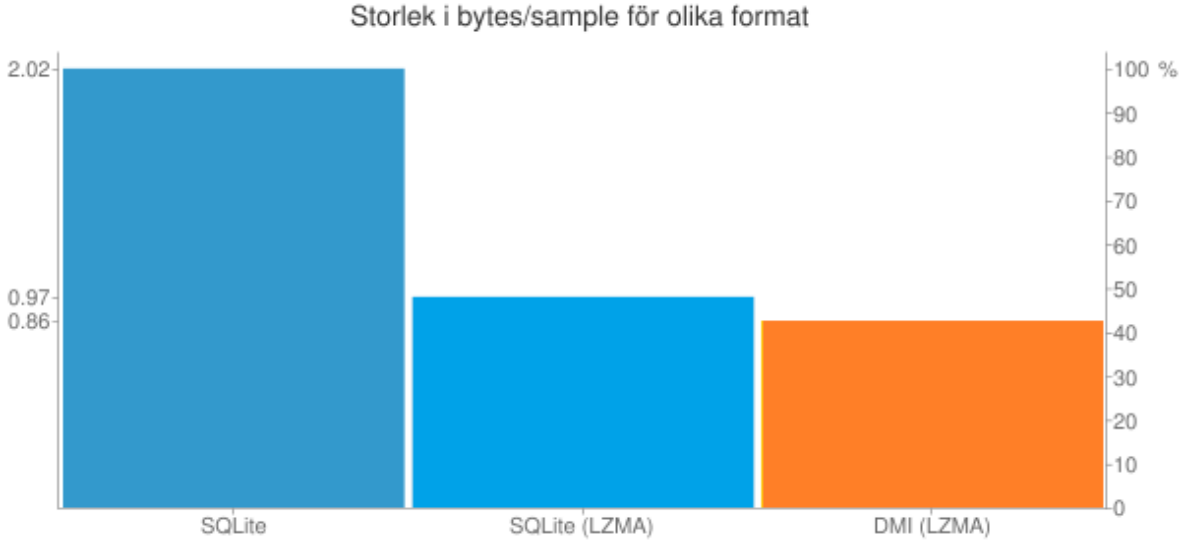
För komprimering av filerna har programmet "7-Zip 9.16" [8] använts, med metoden LZMA på läge Ultra.

Data sparad i SQLite är en sampling från Jøtul med en normalt körande maskin. Data sparad i DMI är en sampling gjord på DISA under testkörning av en maskin. Varje sample är 2 byte stor, 16-bitars heltal. Båda samplingarna har längden 2 minuter och båda samplade 16 kanaler med frekvensen 300Hz som är den maximala för DI-718Bx hårdvaran.

[8] http://www.7-zip.org/ (2010-06-14)

Resulterande värden visar att endast 1 % av SQLite filen är overhead (SQLite-specifik data samt kanalinställningar och andra förhållande för samplingen). De visar även att en komprimerad SQLite-fil tar ca 48 % av originalet (se figur 2). Det format som tar minst plats är komprimerad DMI, vilket tar ca 89 % utrymme jämfört med komprimerad SQLite (se figur 2).

Som tidigare nämnt är detta inga exakta jämförelser. Mätningarna är gjorda på olika maskiner, som kan ge väldigt olika signaler. Vid långa stopptider mellan maskinens operationer får man stillastående signaler som enklare låter sig komprimeras. DMI visar positivt resultat angående filstorlek i jämförelse med SQLite, men det kan alltså mycket väl vara ett vilseledande resultat.



Figur 7 Visuell representation av storleksmetrik för olika format

7 DisaTX

C#.net innehåller färdiga klasser för socketprogrammering via TCP, och det finns även rikligt med guider och exempel att tillgå. Som en första överföringslösning valde vi därför att utveckla ett komplett anpassat program för vårt syfte. Då vi inte jobbat med någon nätverksprogrammering tidigare valde vi att inleda med en mycket enkel prototyp. Denna programdel fick utvecklingsnamnet DisaTX, fortsatt benämnt som DTX. DTX kunde via två simpla GUI överföra korta meddelande mellan en server och en klient.

7.1 Prototyp 2

Efter lovande test utvecklade vi en något mer avancerad prototyp som istället jobbade med textfiler som överfördes från servern till klienten. Även denna prototyp gav förväntat resultat och ett bra underlag för att bedöma om det var en totallösning att satsa på. Två huvudsakliga delar saknades i DTX2 som var nödvändiga för en fullständig lösning. Dels var det segmentering av fil som

inte var implementerad, då programmet skulle vara tvunget att hantera väldigt stora filer vore det inte lämpligt att hantera en så stor buffert, utan istället dela upp filen från början. En buffert i prototypen begränsade alltså maxstorleken på filen. Själva TCP-segmenteringen hanteras av standardklasserna och är allstå inte relaterat till begränsningen. Även en omfattande avbrottshantering med fortsättningsmöjlighet av en fil, i de fall anslutningen brustit behövde implementeras. Vi uppskattade detta till ganska tidkrävande arbete där vi dessutom inte med tillförlitlighet kunde uppskatta prestandan och stabiliteten i programmet.

Slutsatsen blev således att i första hand undersöka färdigutvecklade alternativ att utnyttja, då uppgiften i sig var rätt generell skulle alternativen vara flera. En mindre del av DTX används i slutprodukten för att sända och ta emot enkla kommandon.

8 Background Intelligent Transfer Service (BITS)

8.1 Vad är BITS

Background Intelligent Transfer Service[9] eller förkortat BITS är en Windows-komponent för att asynkront överföra filer via http/https. Det är speciellt anpassat för överföring av stora filer i en miljö där förbindelsen kan komma och gå och arbetet ibland måste utföras över flera dagar med systemomstarter och avbrott. Windows Update är exempel på en tjänst som använder sig av BITS.

8.2 Fördelar med BITS

Som standard körs alla BITS jobb i bakgrunden och använder sig bara av överbliven bandbredd för att ge minimal påverkan på övriga applikationer. Ett BITS jobb kan innebära antingen en nerladdning av en eller flera filer till det egna systemet eller en uppladdning av en fil till en server. BITS utför överföringarna asynkront vilket innebär att programmet som begärde jobbet kan avslutas utan att överföringen påverkas. Så länge en nätverksanslutning uppehålls och användaren som begärde jobbet är inloggad kommer det fortsätta köras.

8.3 Krav för BITS

BITS finns tillgängligt för alla Windows versioner från och med Windows 2000. BITS 1.0 som inkluderades tillsammans med Windows XP tillåter endast nerladdning, medans version 1.5 som lanserades i samband med Windows server 2003 ger stöd för uppladdning. För att utföra en uppladdning krävs det att servern (mottagaren av filen) har Internet Information Server

[9] http://msdn.microsoft.com/sv-SE/library/bb968799(VS.85).aspx (2010-02-11)

(IIS) version 5.0 eller senare installerat, vilket finns att tillgå från och med Windows Server 2003.

9 Fysisk lagring

Då tekniken ständigt går framåt och nya system tas fram, är det inte troligt att maskindata kommer lagras på samma sätt om 50 år även om information fortfarande kommer vara intressant att tillgå. Prioriteringen låg därför på att testa och anpassa systemet för en så standardmässig lösning som möjligt, som lätt kan förändras och bytas ut. Kostnaden är även en faktor som genomgående försökts hållas låg, vilket syns i tidigare resonemang. Inledande bedömning av värdet på datan har gjorts, för att föreslå och ge stöd för lämplig backuplösning och filtrering av data. Det senare är metoder för att rensa ut gammal data samtidigt som vissa milstolpar behålls för att kunna se förändringen hos en maskin över ett längre perspektiv. Då projektet inte kom att innefatta stadiet där DISAs centralserver förbereddes, landade detta på ett diskussionsstadium mellan de iblandade och det mesta har lämnats öppet för framtida utveckling.

10 Filstruktur

Under arbetets gång, både under detta examensarbete och föregående arbete i projektkursen har inga riktlinjer funnits för hur filer ska placeras och vilka kataloger som ska användas. Detta stycke ska utgöra en rekommendation som även förklarar de speciella mappar som Windows XP och Vista använder sig av.

10.1 Speciella kataloger i Windows

10.1.1 Program Files

Både Windows XP och Vista har en särskild katalog kallad "Program Files". Denna katalog kan man komma åt med miljövariabeln %ProgramFiles%. I 64-bit versioner av operativsystemen finns en särskild "Program Files(x86)" för 32-bit program. Sökvägen på en standardinstallation är "C:\Program Files".

Katalogen "Program Files" är till för installationer av program. Lämpligast Installeras programmen i en katalog med namnet av företaget som programmet tillhör. Ett annat alternativ är att direkt lägga programmet i en mapp som innehåller namnet på programmet. I vårt fall använder vi "%ProgramFiles%\Disa\Rms\<Program>" där <Program> motsvarar namnet på ett självstående program, t.ex. DACS eller WebInterface.

10.1.2 Program Data
Både Windows XP och Vista använder denna katalog för att spara programdata som man vill göra tillgänglig för andra program. Det kan handla om en adressbok eller andra typer av "databaser" som flera program ska ha åtkomst till. Mappen är dold som standard eftersom det inte är meningen att användaren ska bläddra igenom mappen och ändra i någon fil.

I Vista kan man komma åt katalogen med miljövariablen %ProgramData% och har sökvägen "C:\ProgramData" i en standardinstallation.

I XP används en lite krångligare sökväg och ingen miljövariabel finns. "C:\Documents and Settings\All Users\Application Data\".

10.1.3 Temp
Temp som är en förkortning för Temporary (tillfällig) innehåller just sådana filer. Alla tillfälliga filer ett program skapar ska hålla sig inom denna mapp. Undantag är filer som symboliserar låsning av andra filer eller backup filer.

10.2 Installationskatalog
Vare sig ett program blir installerat av en installerare, eller om det bara handlar om att packa upp filer till en mapp så kan följande mappstruktur användas.

10.2.1 Config
Katalogen namngiven förkortningen av Configuration (konfiguration) innehåller de konfigurationsfiler programmet använder. Även konfigurationsfiler från plugins får placeras här.

10.2.2 Errors
Denna katalog ska förhoppningsvis hålla sig tom. Till denna skrivs loggar och minnesdumpar vid programkrasher. Programkrasher är särskilt viktiga händelser och det är därför en stor fördel om man enkelt kan hitta data angående dem.

10.2.3 Logs
I denna katalog skrivs diverse loggar ut. Det kan handla om installationsloggar, uppdateringsloggar eller programloggar. Även om en krasch registreras i programloggen eller i någon av de övriga ska den även skriva ut i Errors katalogen.

10.2.4 Plugins
Plugins är ett sätt att utöka ett programs användningsområde eller modifiera befintlig funktionalitet på. För att ett program enkelt ska kunna hitta plugins är det lämpligt att ha en särskild katalog som innehåller dessa. Genomsökningen av plugins bör ske rekursivt, dvs. man kan lägga ett plugin i en underkatalog i plugins. Detta gör det enklare att sortera flera plugins och vid en manuell avinstallation ser man enkelt vilka filer som tillhör ett plugin.

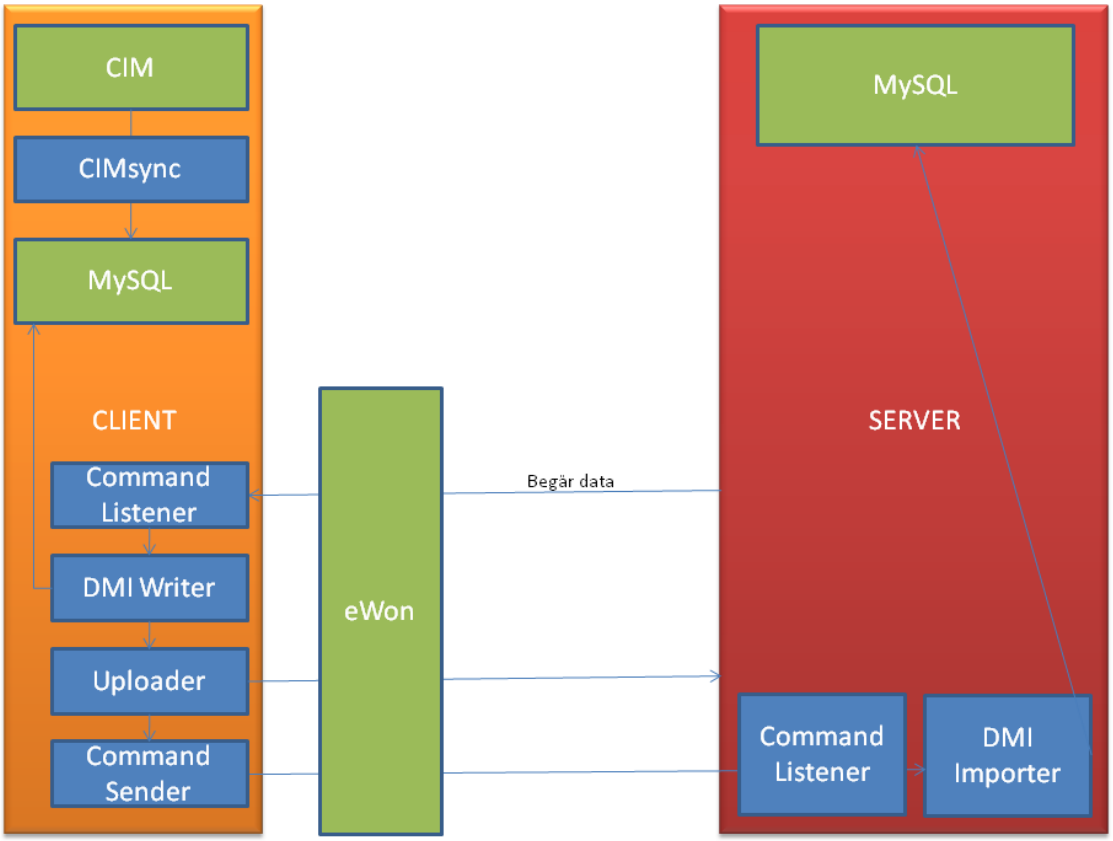
Då plugins innehåller körbara filer som refererar till samma bibliotek som huvudprogrammet uppstår ett problem. För att hitta refererade bibliotek i ett .NET program måste de befinna sig i samma katalog eller i en underkatalog. Detta leder till att dubbla uppsättningar krävs för att behålla Plugin strukturen. En lösning är att installera biblioteken i den global assembly cachen som är en speciell mapp där .NET program letar efter bibliotek. Något mer om detta kommer inte nämnas då vi istället anser att man helt enkelt bör utesluta körbara filer i sina plugins och i de fall de krävs så anses en dubbel uppsättning av biblioteken som den enklaste lösningen.

10.2.5 Resources
Ibland vill man att ens program ska stödja flera språk och då är det mappen Resources som är till för sådana filer. Filer som är nödvändiga för att programmet ska kunna köras ska vara placerade här. Med detta menas inte körbara filer, bibliotek och inställningsfiler.

11 Resultat

Figur 3 motsvarar en översikt över det fullständiga överföringssystemet. För en slutversion av DACS-programmet, se bilaga 1.

Med server avses den lagringsdisk som finns hos DISA i Herlev, medan client syftar till enheten ute hos kund. Gröna block motsvarar en redan befintlig komponent/enhet, dock inte nödvändigtvis i DISAs system utan i den bemärkelsen att den sedan innan är utvecklad av tredje part. Blåa block motsvarar de komponenter som utvecklats specifikt för detta projekt för att sammanknyta det hela.



Figur 3 Grönt motsvarar ”befintliga” komponenter som integrerats med blåa, nyutvecklade komponenter

11.1 Komponenter

11.1.1 CIMsync

Programvara som automatiskt med jämna intervall eller i samband med en client-server synkronisering kopierar alla nya värden från en befintlig Sybase databas till DACS tillhörande MySQL databas. Programmet utför även kontroll på utvalda inställningsrader som i vissa scenarier kan förändras, på grund av att ingen tillhörande produktionsdata har sparats.

11.1.2 eWON

För att upprätta en säker nätverksförbindelse (VPN-tunnel) mellan gjuteri (Jøtul) och DISA används en industrirouter vid namn eWON 4005CD. Denna router är placerad på gjuterisidan. Enheter som är inkopplade på routern behöver ingen särskild VPN-mjukvara eftersom routern redan har inbyggt stöd för detta. Företaget bakom routern erbjuder tjänsten Talk2M, som är ett sätt att upprätta en VPN-tunnel med en eWON. Med hjälp av mjukvara kallad eCatcher upprättar man en VPN-tunnel med en Talk2M-server som i sin tur upprättar en andra VPN-tunnel med en eWON. Därefter agerar Talk2M-servern medlare och en säker nätverksförbindelse är upprättad.

11.1.3 Command Listener/Sender (DTX)
Command Sender skickar via TCP begäran eller bekräftelse på specifika uppgifter. Dessa tas emot av Command Listener som startar önskad process. Båda komponenterna är delar av det tidigare nämna DTX.

11.1.4 DMI Writer/Importer
Skapar en komprimerad fil utifrån ett egenutvecklad format kallat ”DISA Machine Info”. DMI filen komprimeras sedan till det öppna formatet 7z. Filen innehåller endast den data som skapats efter senast registrerade körning. DMI Importer packar upp och importerar maskindata till den hos DISA lokala servern.

11.1.5 Uploader
Skapar ett BITS uppladdningsjobb mot servern innehållande givna data. För att kommunicera med BITS används den fria .NET wrappern[10] SharpBITS.net[11]. När en uppladdning är slutförd anropas Command Sender som meddelar servern att en ny fil finns redo att bearbetas.

12 Diskussion

12.1 Projektmodell

Ett antal tydliga faktorer fanns till grund för val av lämplig projektmodell. DISAs önskemål har sedan ursprungsprojektet varit en prototypbaserad utvecklingsprocess, som stegvis kunde demonstreras hos kund för att få värdefull feedback. Samtidigt fanns inga strikta deadlines för hur och när detta skulle ske utan det fanns stora möjligheter att anpassa efter arbetets framgång.

Med hänsyn till ovanstående valdes ett väldigt agilt arbetssätt. Många strategier i vår utvecklingsprocess går att känna igen ifrån XP metoden, bland annat då vi satsade på att bygga så enkel kod som möjligt, och höll nödvändig dokumentation på en minimal nivå. När en tilltänkt lösning började bli för komplex återvände vi till ritbordet och försökte framställa en enklare lösning.

Då vi varit få personer som jobbat med projektet inkluderat närliggande parter, kändes mer traditionella dokumenttyngda modeller såsom vattenfallsmodellen, mindre givande om ens möjliga att genomföra. Det valda mer iterativa arbetssättet har styrkts genom egen erfarenhet, och vår uppfattning är också att

[10] Kapslar in och tillhandahåller ett bibliotek med ”adapter” metoder
[11] http://sharpbits.codeplex.com/ (2011-01-31)

systemutvecklingsbranschen överlag går längs de mer agila utvecklingsmetoderna.

12.2 Möjlig vidareutveckling

Det finns helt klart stora möjligheter för vidareutveckling. Den största delen är antagligen den övergripande nätverksstrukturen. Hur ska flera anslutna maskiner/fabriker kommunicera på ett fungerande vis? Prioritering vid hög belastning och kösystem är något som inte finns i systemet. Den rent fysiska anslutningen mellan kund och DISA är också ganska begränsat testad/utformad. Underhållet av en enskild ansluten maskin är ganska hanterbar när det gäller installering och inställning av adresser. Men efterhand som fler fabriker ansluts skulle ett mer strukturerat och automatiserat upplägg behöva införas.

Beroende på hur DACS kommer fortsätta utvecklas kan det även öppna upp nya möjligheter i olika val som tidigare inte varit möjliga. Vi anser det ganska sannolikt att sådant kommer ske innan totalprojektet når en komplett status.

12.3 Slutsats

Överlag är vi nöjda med det resultat arbetet utmynnat i, både vad gäller slutprodukten men kanske framförallt med avseende på vad projektet har gett oss rent kunskapsmässigt. Vi ägnade mycket tid åt att testa och undersöka olika typer av lösningar och tekniker. Vi gjorde flera praktiska test vilket resulterat i mycket kod som inte nådde fram till slutprodukten. Detta har givetvis både sina positiva och negativa konsekvenser. På den positiva sidan ser vi inhämtandet av mycket ny kunskap. Vi har också kunnat känna oss nöjda med de val vi i slutändan valde att utveckla mot. Baksidan med det hela är givetvis att mycket tid som kunde gått till att förfina den resulterande mjukvaran istället har lagts på helt andra lösningar som inte kommer få någon praktisk användning. Vi talar här om nätverkskommunikation som slopats på grund av stabilare alternativ, komprimeringsmetoder som helt enkelt inte kunnat konkurrera mot sådant som redan finns. Hursomhelst är vi uppriktigt glada över att vi lagt tiden på att testa allt detta och vilken förbättrad helhetsbild det givit oss. Samtidigt är vi överens om att vi borde satt upp lite hårdare tidsplanering för hur mycket kalendertid som fick gå innan vissa designer definitiv skulle fastställas. Vissa beslut hölls helt enkelt öppna alltför länge för att det skulle vara effektivt.

Det vi främst känner att vi inte hunnit med i detta arbete är en djupare kartläggning av DISAs nätverk, ingen installation har skett på RMS som kan ta emot all data utifrån fabrikerna.

Slutprodukten uppfyller enligt vår bedömning de målsättningar vi från början satte upp. Det saknas visserligen en del finputsning, och förutsatt att DISA väljer att driva huvudprojektet vidare kommer säkerligen mycket vidareutveckling att göras innan det kan användas som en komplett lösning.

13 Terminologi

RMS – *Remote Monitoring Service, det system vårt arbete och DACS är del av.*

DACS – *Disa Acquisition Software, samplar värden från DISAs maskiner.*

DMM – *DISA Moulding Machine, den maskin som skapar sandformar och gjuter godset.*

DataQ – *Tillverkare av samplingshårdvara.*

MySQL – *SQL-baserad databasmotor.*

CIM – *Befintlig databas som innehåller bland annat produktionsstatistik och maskininställningar.*

SQLAnywhere – *En databaslösning från Sybase.*

DASYLab – *Ett program för att presentera samplingskurvor.*

CIMsync – *Replikerar data från CIM till en MySQL-databas.*

DMIWriter – *Genererar tillfälliga filer med segment ur en MySQL-databas.*

DMIImporter – *Importerar genererade filer från DMIWriter till en MySQL-databas.*

DisaTX – *Dataöverföring mellan fabrik och DISA.*

DTX – *Ett lite kortare namn för DisaTX.*

eWON – *Industrirouter med integrerad VPN-lösning.*

VPN-tunnel – *Virtual Private Network, krypterad anslutning över ett osäkert nätverk*

14 Källförteckning

Behrouz, A. (2006). Data Communications And Networking
McGraw-Hill Science/Engineering/Math

Padron-McCarthy & Risch, T. (2005). Databasteknik
Lund:Studentlitteratur AB

Skansholm, J. (2008). Skarp programmering med C#
Lund:Studentlitteratur

MySQL, http://www.mysql.com/ , 2010-08-30

Postgresql, http://www.postgresql.org/ , 2010-08-30

Sybase, http://www.sybase.com/ , 2010-08-30

.NET Dokumentation, http://msdn.microsoft.com , 2010-08-30

15 Bilaga 1: Slutversion DACS