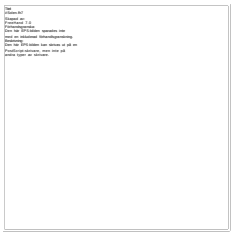


Datavetenskap

Andreas Lövgren

Thomas Persson

Loggvisningssystem

Examensarbete, C-nivå

2004:03

Loggvisningssystem

Andreas Lövgren

Thomas Persson

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

Andreas Lövgren

Thomas Persson

Godkänd, 2004-01-15

Handledare: Robin Staxhammar

Examinator: Stefan Lindskog

Sammanfattning

Denna uppsats tar upp hur utvecklingen gick till för att skapa ett loggvisningssystem, en applikation som kan visa och söka igenom loggfiler som genereras från Uddeholm Technologys datorsystem UTCAS, Uddeholm Technology Converter Automation System. Anledningen till att Uddeholm Technology efterfrågade en sådan applikation var att de på ett enkelt sätt vill kunna upptäcka eventuella fel som uppstått i processer och vid kommunikationen mellan processer. Applikationen skall alltså förenkla felsökning, vilket hittills har utförts genom manuell genomsökning av loggfiler.

Ett grundkrav var att applikationen skulle utvecklas med MS Visual Studio .NET. Detta eftersom Uddeholm Technology inom en snar framtid skall övergå till denna teknik. Man vill också på ett enkelt sätt kunna komma åt loggfiler, både inom samma och mellan olika företag.

Själva arbetet inleddes med att undersöka vilka verktyg som .NET tillhandahöll. För att göra loggvisningssystemet lättillgängligt valde vi att distribuera det på Internet genom att skapa en webbapplikation. Funktionalitet för att snabbt kunna hämta ut önskade delar ur valda loggfiler utvecklades. Ett grafiskt användargränssnitt, där man enkelt kan jämföra olika processers loggfiler, skapades. Projektet genomfördes med framgång och både vi, som utvecklare, och företaget var nöjda med den slutgiltiga produkten.

LogViewer

Abstract

This report describes the development of a log viewing system, an application that can display and search through log files generated from Uddeholm Technology's computer system UTCAS, Uddeholm Technology Converter Automation System. The reason that Uddeholm Technology demanded such an application was that they wanted a simple way to discover possible errors generated from interprocess communication aswell as errors generated within processes. Thus, the application shall facilitate error discovery by making it easier to search through log files, which so far has been done manually.

A requirement for the application was that it should be developed using MS Visual Studio .NET. The reason for this is that the company is planning to use .NET in the future. The application should also make it easier to access log files, both within the same and between different companies.

The project work started by investigating useful tools offered by .NET. The log viewer system was made easily available by distributing it on the Internet through a web application.

Functionality, which allows the user to quickly retrieve desired parts from selected log files, was developed. A graphical user interface, where one easily can compare different process's log files, was created. The project was a success and the product satisfied both the company and us, the application developers.

Innehållsförteckning

Figurförteckning

Tabellförteckning

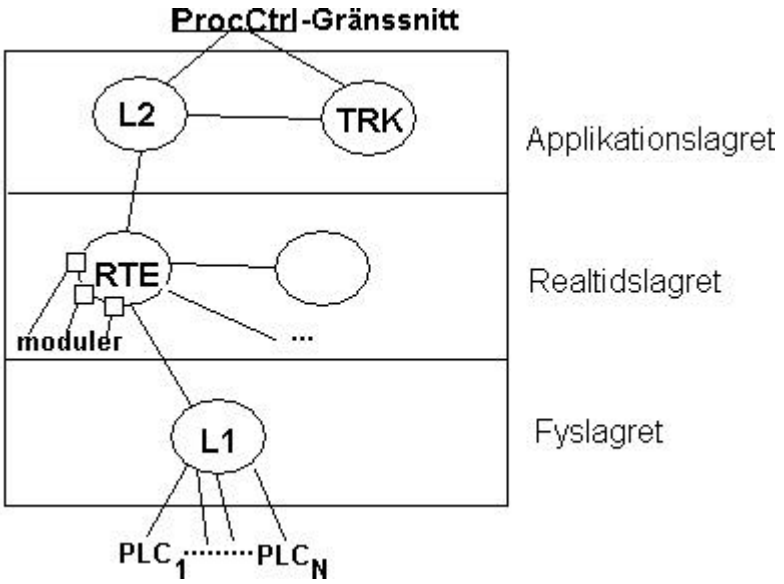
1 Inledning

1.1 Bakgrund

Uddeholm Technology AB är ett dotterbolag till Uddeholm Tooling AB. Företagets verksamhet är indelat i tre olika affärsområden:

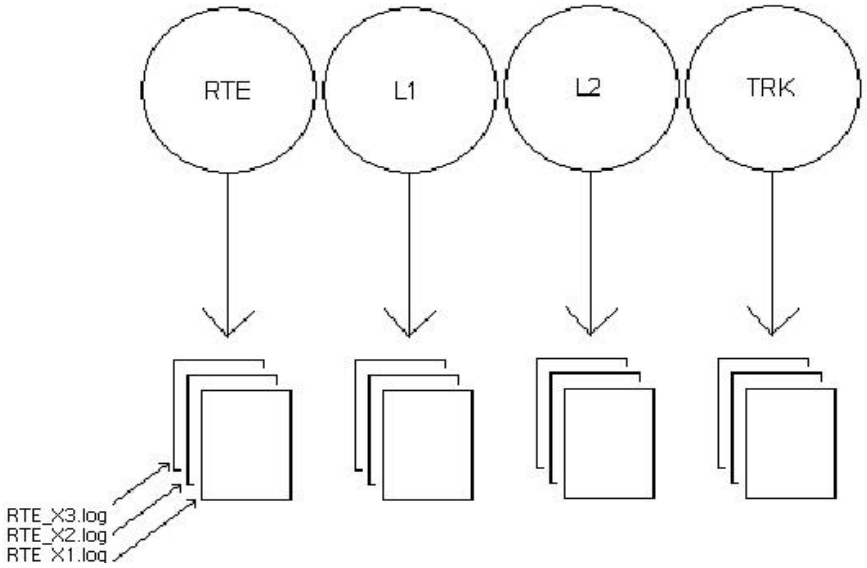
- Tillverkning och försäljning av granulerat stål - ”Granshots”.
- Försäljning av anläggningar för tillverkning av granulerat stål samt konverterprocesser, se bilaga A, och i samband med detta även konsulttjänster.
- Försäljning av UTCAS, ett datorsystem för styrning av stålugnar.

Det utvecklade datorsystemet kan säljas tillsammans med ovannämnda anläggningar men även som fristående produkt att användas vid kundens redan befintliga anläggningar.



Figur 1.1: Lagerstacken i UTCAS

Ovan beskrivs Uddeholm Technologys befintliga system ”UTCAS” (Uddeholm Technology Converter Automation System) med hjälp av en lagerstack. ProcController är gränssnittet mot användaren av systemet. L2, TRK, RTE och L1 Connector är exempel på olika processer i Uddeholms system. PLC används för att styra diverse maskiner i industrin såsom gasreglage och materialtillflöden. L1 Connector ansluter sig till dessa när något specifikt ska utföras.



Figur 1.2: Generering av loggfiler från olika processer

UTCAS omfattar, som nämnts, ett flertal processer. Varje process delas in i ett flertal moduler, där varje modul genererar loggfiler. Dessa loggfiler namnges sedan enligt <processnamn>_<nummer>.log. <nummer> hos en viss loggfil är alltid ett högre än den loggfil som skapades tidigare av samma process. Då en modul vill kommunicera med en annan process använder man sig av en generell meddelandemodul. Meddelande som sänds mellan processerna loggas också i loggfilerna.

Då en loggning av en viss händelse sker så sätts en tidstämpel som ser ut på följande sätt [YYYY-MM-DD HH:MM:SS + nivå av loggning(modul nivå)]. Syftet med loggfilerna är att kunna verifiera funktionaliteten och att kunna spåra felorsak i händelse av fel.

Systemet är utvecklat i MS Visual C++ och avsett att köras på MS Windows 2000 plattform. Anledningen bakom detta val är att det är det mest spridda operativsystemet hos företagets kända och potentiella kunder. Att produkten passar in i kundens övriga IT-struktur och att kunden kan förstå och underhålla systemet, är i många fall en förutsättning för att en affär skall komma till stånd. Företaget är dessutom inte stort nog att kunna tillhandahålla parallella systemlösningar, utvecklade för andra plattformar. MS Visual .NET är Microsofts vidareutveckling av MS Visual Studio 6.0 och antagandet är att den inom några år kommer att vara den plattform som är mest spridd och etablerad hos företagets kunder. Av denna anledning överväger företaget nu en framtida övergång till .NET-teknik. Det gäller att vara tidigt ute och följa med i utvecklingen för att inte systemstrukturen skall bli betraktad som föråldrad. Företaget är dessutom övertygat om att .NET-tekniken kommer tillföra ny och förbättrad funktionalitet, inte minst vad gäller

distribution över nätverk och kommunikation med kringliggande system i den operativa miljön.

1.2 Vad behöver förbättras?

Loggfilerna som de olika delprocesserna genererar kan variera från 0kb till några Gb men ligger vanligtvis mellan 100kb till 200MB, beroende på hur detaljerad loggning man valt. Vid felsökning krävs ofta att loggfiler från olika processer jämförs. Om det exempelvis sker ett fel vid kommunikation mellan processer så undersöks loggfilerna för de inblandade processerna. Detta för att man ska kunna finna vilken process som orsakade felet och varför det skedde. Felsökningen är både tids- och arbetskrävande eftersom den idag utförs manuellt. Man öppnar en loggfil över en viss process och söker upp en händelse/tid, därefter söker man upp samma händelse/tid i loggfilen för den process man vill jämföra med. Dessutom är det vanligt att en aktiv loggning sträcker sig över flera filer, varför man kan bli tvungen att öppna flera olika filer för att finna sökt händelse/tid. Det finns därför önskemål om en produkt där man på ett snabbt och enkelt sätt kan få tag på de delar i en loggfil som är relevanta för en specifik felsökning. Därtill vill man att produkten ska innehålla möjlighet för att jämföra loggfiler från olika processer.

Man vill också på ett enkelt sätt kunna komma åt loggfiler inom och mellan företag. Som det ser ut idag så krävs det att man skickar loggfilerna manuellt, exempelvis via email. Detta kan vara mycket tidskrävande om filerna är stora. Därför önskas det också att man kan komma åt den del av loggfilen som är specifik för felsökningen utan att behöva ladda hem hela filen.

1.3 Mål

Målen är att:

- Skapa en klientserver-applikation som skall sköta kommunikation av loggfils-data mellan klientapplikationen och servern, där loggfilerna kommer att finnas. Gränssnittet, klienten, skall vara en webbapplikation som kan köras i en webbrowser. Vad gäller transport av loggfiler från det verkliga systemet till servern kommer dessa att laddas upp manuellt. Det är alltså inget vår applikation skall tillhandahålla.
- Skapa ett enkelt användargränssnitt för klienten. Här skall tillhandahållas funktionalitet för datum/tid-sökning i loggfiler och visning av sökresultat. Det ska fungera så att användaren anger starttid/datum och sluttid/datum och får sedan ut all

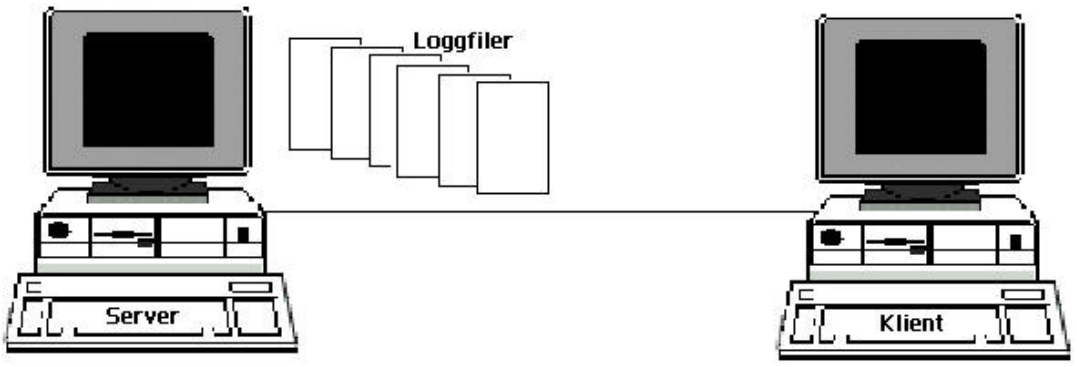
loggdata mellan dessa tider som resultat. Sökning skall kunna utföras på sådant sätt att tider kan jämföras med varandra för att det ska gå att hitta närmast matchande tider vid sökning. Dessutom önskas funktionalitet för samtidig visning av sökresultat från loggfiler genererade av olika processer. Som nämnts tidigare genereras loggfiler från flera processer och det är essentiellt för felsökningen att kunna jämföra flera loggfiler samtidigt. Detta p.g.a. att, som nämnts i 1.2, det kan uppstå fel vid kommunikation mellan processer och man vill då jämföra inblandade processer för att kunna upptäcka var och varför felet skedde.

- Skapa funktionalitet för att kunna hämta och sammanställa sökresultat från flera loggfiler. Det skulle kunna hända att en användare anger start- och sluttid där starttiden befinner sig i en loggfil och sluttiden i en annan. Vi måste då kunna hämta en del av resultatet från en fil och resterande ur nästkommande fil(er).

2 Analys

I detta kapitel beskrivs tidiga idéer och tankar om hur de mål som sattes upp i avsnitt 1.3 skulle tillgodoses.

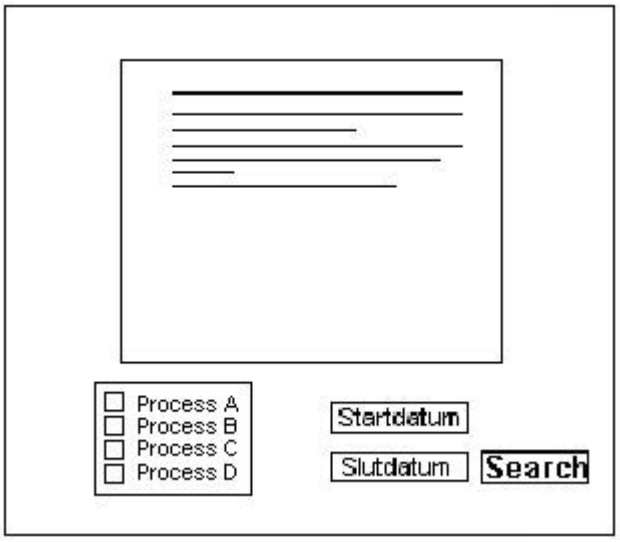
2.1 Klientserver-applikation



Figur 2.1: Kommunikation av loggfiler mellan klientapplikation och server

Tanken var att tillhandahålla ett användargränssnitt som skulle kunna visas i en webbrowser. Loggfilsdatan skulle hämtas från en server så någon form av kommunikation mellan servern och klienten (användargränssnittet) skulle behövas. Exakt på vilket vis detta skulle ske var i detta läge oklart. Vad som dock var klart var att .NET-teknik skulle användas. Efterforskningar hade givit insikt om att .NET´s främsta användningsområde var att på ett enkelt sätt kunna göra applikationer tillgängliga på webben. En översikt av .NET ges i kapitel 3.

2.2 Användargränssnittet utseende

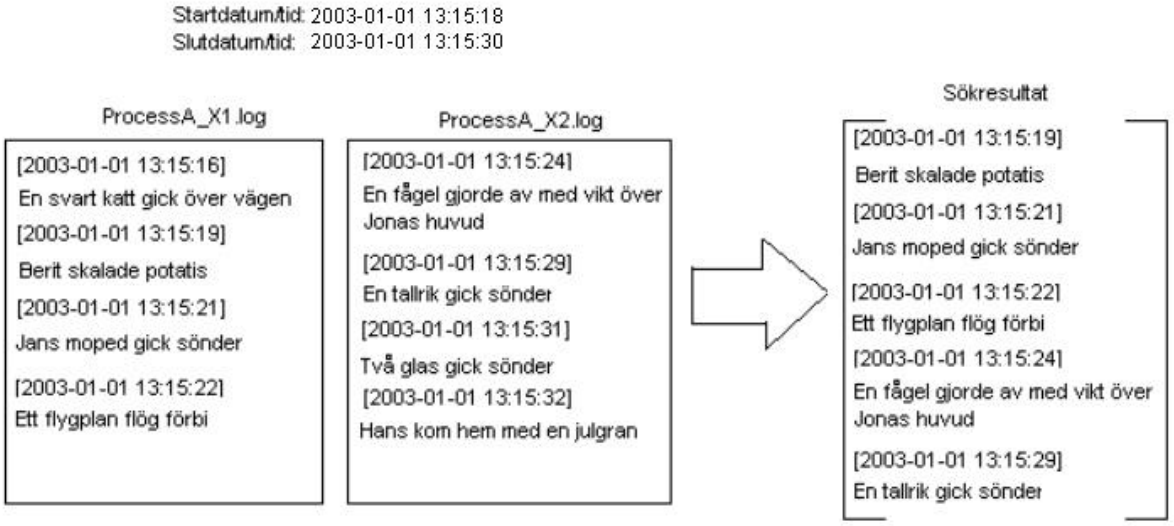


Figur 2.2: En första tänkt bild av användargränssnittet

Det tänkta gränssnittet skulle bestå av två fält för inmatning av start- och slutdatum, en knapp för att utföra sökning, en eller flera resultatareor för visning av sökresultat. En vallista skulle också finnas för att kunna välja vilka processers loggfiler som önskas inkluderas i sökningen, med andra ord de processer som man vill jämföra.

Figur 2.3: Ett bibliotek där mappar med loggfiler kan placeras

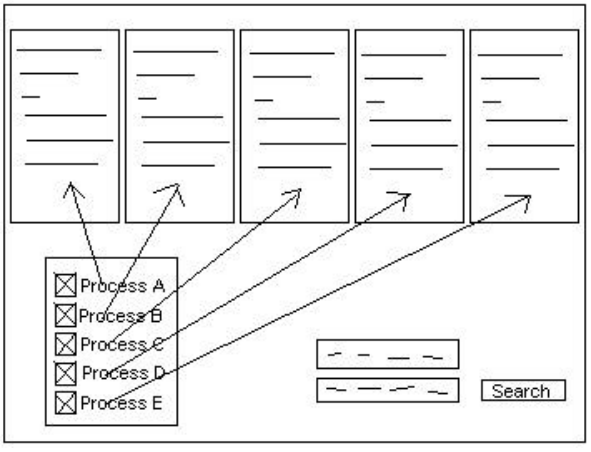
Tanken var då att tillhandahålla ett loggfilsbibliotek, i vilket man skulle kunna lägga in mappar innehållande loggfiler från olika processer. En mapp för varje specifik process. För varje sådan mapp som finns i biblioteket så skulle det bli ytterligare ett val i vallistan.



Figur 2.4: Exempel på en sökning

För de processer som valts vid en sökning skulle det ur loggfilerna från dessa processer hämtas ut sökresultat som ligger mellan de två angivna tidpunkterna. En beskrivning av tankegången till detta förfarande illustreras i *figur 2.4* ovan.

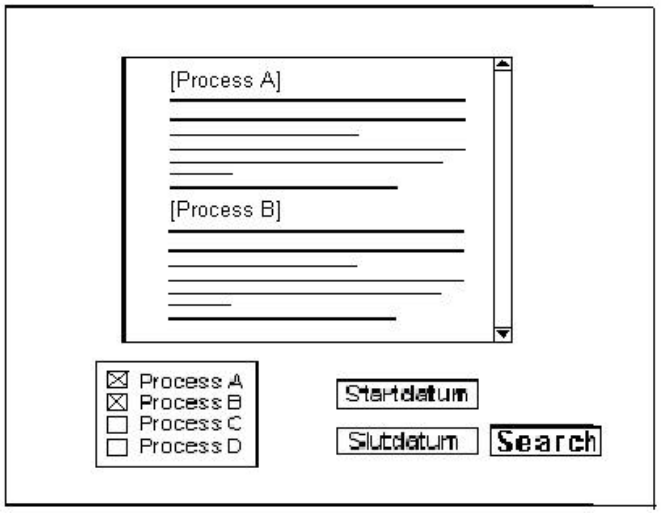
Det diskuterades en hel del kring hur visning av sökresultat skulle ske. En tanke var att tillhandahålla resultatarea för varje vald processtyp. Det gick dock inte att föreställa sig hur dynamiskt uppritande skulle kunna ske på ett snyggt sätt. Om exempelvis fem processer valdes så skulle fem resultatareor behöva ritas upp, såsom illustreras i *figur 2.5*. Detta skulle resultera i att varje area blir ytterst liten och då skulle inte sökresultat kunna visas på ett tillfredsställande sätt.



Figur 2.5: Användargränssnitt med en resultat-area för varje vald process

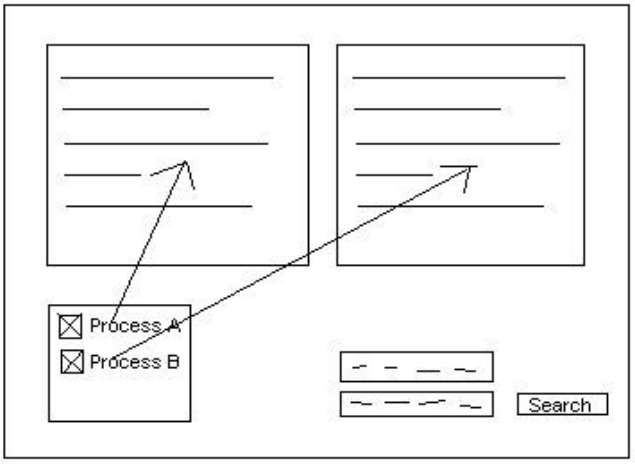
En annan tanke var att använda endast en resultatarea. Hur det skulle kunna se ut illustreras i *figur 2.6*. Fördelen med detta är att man har möjlighet att välja att inkludera ett obegränsat

antal processer. Att visa resultatet på ett sådant sätt skulle dock göra det svårt att jämföra flera processer, eftersom det skulle kräva att man rullar upp och ned i resultatarean.



Figur 2.6: Gränssnittet med endast en resultat area

Den slutgiltiga tanken kring visningen av resultat blev istället att tillhandahålla två separata resultatareor för att kunna jämföra loggfiler från två olika processer samtidigt.

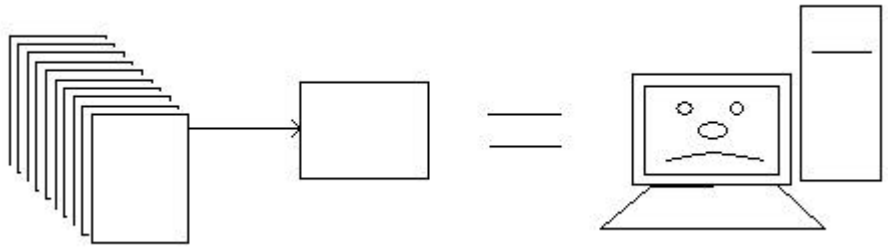


Figur 2.7: Gränssnitt med två resultat-areor

Resultatet från den ena processens loggfiler skulle då placeras i ena resultatarean och resultatet från den andra processen i andra resultatarean. Dessa resultatareor borde lämpligen placeras bredvid varandra för att det skulle vara enkelt att jämföra resultatet. Hur det skulle kunna se ut visas i figur 2.7 ovan. Anledning till valet av just två resultatareor och inte fler var att då antalet areor översteg två så blev det ytters litet utrymme till att visa resultatet på i varje area.

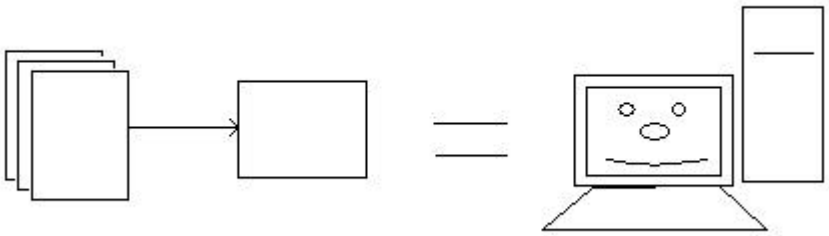
2.3 Hämtning av sökresultat från flera filer

Som nämnts tidigare kan en aktiv loggning sträcka sig över flera filer. Vid en sökning kan man då bli tvungen att samanställa resultat från flera loggfiler, om användaren anger start- och slutsökdatum på så sätt att de befinner sig i olika filer. En idé var att slå samman alla filer från en viss process och lagra allt innehåll i ett temporärt lagringsutrymme för att kunna förenkla sökningen.



Figur 2.8: Sammanslagning av alla loggfiler från en specifik process

Men det är en oacceptabel lösning då detta sätter orimliga krav på datorns primärminne eftersom filerna kan vara mycket stora. Vad som istället kan göras är att på något sätt ta reda på vilka filer som innehåller sökresultat, d.v.s. loggfilsdata mellan angivna tider, och enbart sammanfoga dessa.



Figur 2.9: Sammanslagning av ett utvalt antal loggfiler från en specifik process

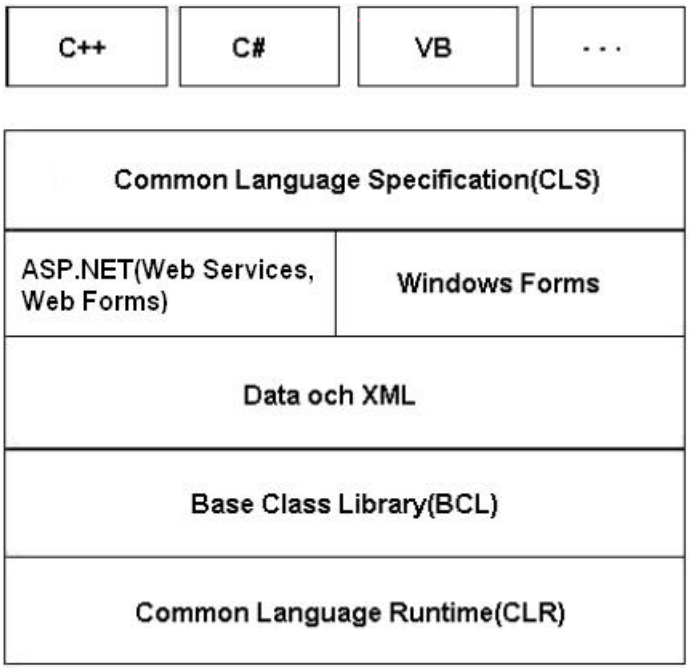
En annan lösning på detta problem skulle varit att använda sig av enbart filhantering för att få ut resultat-strängen. På så vis skulle man helt och hållet ha undvikit att mellanlagra data i primärminnet. Nackdelen med detta är att det kan vara stora filer som skall gås igenom och det skulle bli märkbart mer tidsineffektivt med enbart filhantering jämfört med att hantera datan i primärminnet.

3 .NET

I detta kapitel ges en översiktlig beskrivning av .NET´s arkitektur. De delar som kom att användas vid utvecklandet av vårt system beskrivs mer utförligt och en motivering till varför de valdes ges.

3.1 Arkitekturen

Nedan syns en bild av .NET´s arkitektur [13], med dess olika lager.



Figur 3.1: .NET arkitekturen

- CLR är .NET´s virtuella maskin som har hand om minneshantering, språkintegrering, undantagshantering och dynamisk bindning under programkörning.
- BCL har hand om grundfunktionalitet såsom in- och utmatning.
- Data och XML-lagret tillhandahåller funktionalitet för ADO.NET och XML. ADO.NET (ActiveX Data Objects) är en grupp klasser för databashantering. XML förklaras ingående i 3.2.1.
- Nästkommande lager består av två delar, nämligen ASP.NET och Windows Forms. Lagret innefattar klasser för skapande av grafiska användargränssnitt. ASP.NET

(Application Service Provider) används vid skapande av Internet applikationer (Webbtjänster och Web Forms) och Windows Forms vid skapande av Windowsapplikationer.

- CLS är en samling regler för att se till att program skrivna i olika programspråk kan fungera tillsammans.

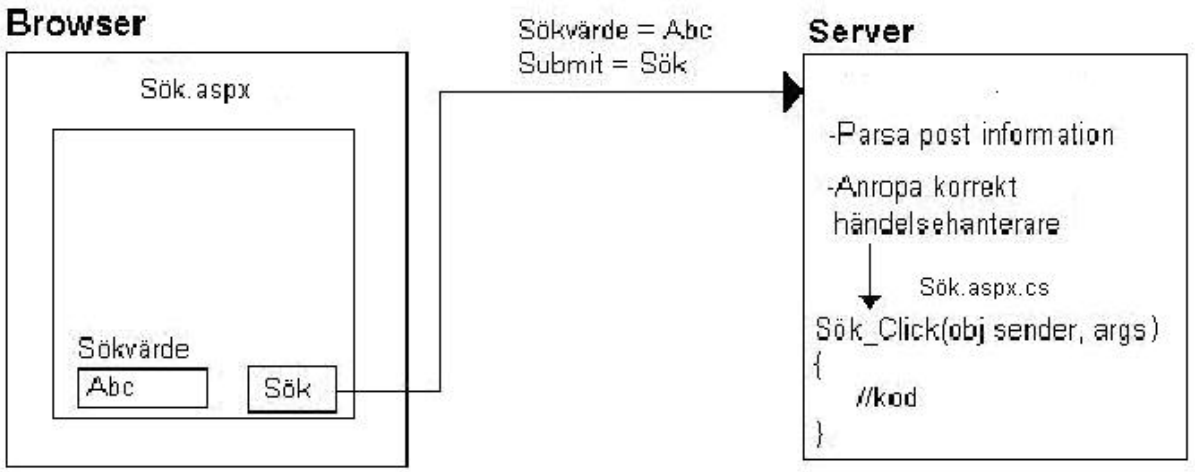
Eftersom vår produkt skulle publiceras på Internet så var ASP.NET intressant för oss.

3.2 ASP.NET Web Application

ASP.NET Web Application är en typ av Internet applikation. Det är möjligt att skapa en sådan applikation i utvecklingsmiljön, MS Visual Studio .NET. Detta görs genom att välja just projektet ASP.NET Web Application. Dock fanns det inte stöd för ASP.NET Web Application projekt i C++, vilket vi hade föredragit, åtminstone inte i de versioner av utvecklingsmiljön som vi kom i kontakt med. Det stöddes däremot i C# och Visual Basic. Vad som är speciellt med just ASP.NET Web Application är att man på liknande sätt som i Visual Basic kan rita upp användargränssnitt. Det finns tillgång till textfält, textareor, knappar, rullningslistor, samt en hel del andra komponenter. På ett enkelt sätt kan också komponenter tilldelas händelsehanterare. En händelsehanterare innehåller de instruktioner som man vill ska ske då en viss händelse skett, såsom att en knapp har tryckts.

Har väl en komponent i gränssnittet ritats ut så är det bara att dubbelklicka på komponenten för att skapa händelsehanteraren. Därefter är det bara att fylla i den kod för den händelse som önskas ske. ASP.NET Web Application består främst av två delar. En del för synliga komponenter (gränssnitt) och en del för logik (händelsehantering). Visual Studio sparar var och en av dessa i en separat fil. De synliga komponenterna skapas i en .aspx-fil och logiken i en separat klassfil; .aspx.cs eller .aspx.vb, beroende på om C# eller Visual Basic används. Denna klassfil kallas code-behind class file.

Det skapade gränssnittet (.aspx-filen) är det som visas för klienten (webbrowsern) då applikationen anropas och på servern är alla händelsehanterare för gränssnittets komponenter belägna. ASP.NET Web Application fungerar på så vis att händelser sker i klienten och tas om hand i servern med hjälp av ASP.NET ramverk.



Figur 3.2: Kommunikation mellan klient och server

För händelser som sker på klienten så krävs det att händelseinformationen fångas på klientsidan och ett händelsemeddelande sänds till servern m.h.a. HTTP. Ramverket måste översätta meddelandet för att få reda på vilken händelse som skett för att kunna anropa korrekt händelsehanterare på serversidan. ASP.NET hanterar alla mekanismer för att fånga, sända och översätta händelsen. När man skapar händelsehanterare så behöver man inte tänka på de avancerade mekanismer som krävs för att sända och ta emot information.

3.3 Webbtjänstteknikens uppbyggnad

En webbtjänst är en serverapplikation som kan anropas via standardwebbprotokoll såsom HTTP och SOAP. Dessa tjänster kan nås över hela Internet, man anropar en metod och får ett resultat i retur. Med webbtjänster går det enkelt att ansluta sig till tjänster över Internet. Webbtjänster är inte tjänster till en slutanvändare utan till ett system, vilket innebär att webbtjänstteknologin är transparent för en slutanvändare. En webbtjänst kan utföra allt ifrån mycket simpla till mycket komplexa uppgifter. Webbtjänster kan skapas i vilket programspråk som helst eftersom de är operativsystems-, plattforms- och programspråksoberoende. En av anledningarna till att alla webbtjänster kan kommunicera med varandra är att de pratar samma språk; **XML** (eXtensible Markup Language). Webbtjänsterna använder XML för att beskriva sitt gränssnitt och för att koda sina meddelanden. En webbtjänst har två viktiga egenskaper; den är självbeskrivande och upptäckbar. Självbeskrivande innebär att tjänsten innehåller en läsbar dokumentation som beskriver tjänsten, detta för att underlätta integrationen av tjänsten för andra. **WSDL**-protokollet (Web Services Description Language) används till detta. Upptäckbar innebär att när en tjänst skapats bör den vara lätt att publicera, detta för att intresserade parter lätt ska

kunna hitta tjänsten. **UDDI** (Universal Description, Discovery and Integration) används till detta. Medelanden är skickade med hjälp av **SOAP** (Simple Object Acess Protocol).
I kommande underkapitel ges mer ingående beskrivningar av de delar som ingår i webbtjänsttekniken.

3.3.1 XML

XML (Extensible Markup Language) [11] är i grunden en uppsättning regler för att, med hjälp av *taggar* (förklaras senare i texten), dela upp ett dokument i delar och kunna identifiera dessa delar. Dessa regler har utarbetats av World Wide Web Consortium (W3C). Huvudidén med XML är att man ska kunna separera innehållet i ett dokument från utseendet. Formatet är speciellt anpassat för att överföras via Internet. XML-dokument är hårdvaru- och plattformsoberoende. Både XML och HTML härstammar från SGML (Standard Generalized Markup Language), vilket medför att de har liknande karakteristik och syntax. SGML specificerar en standard för att beskriva strukturen i ett dokument, en struktur med vilken man sätter upp hierarkiska modeller för dokumenten. SGML har dock två stora nackdelar: dess komplexitet och kostnad. För att komma runt SGMLs krånglighet och dyra verktyg så skapades HTML (HyperText Markup Language). HTML är dock begränsat i sin utformning, med en begränsad uppsättning taggar såsom <BODY>, <P> och <TITLE>. I XML däremot kan man skapa sina egna taggar för att strukturera upp ett dokument.
En rubrik i HTML kan exempelvis skrivas:

```
<H1>Detta är en rubrik</H1>
```

I XML kan man istället skapa en egen tag för att senare bestämma dess funktion:

```
<rubrik>Detta är en rubrik</rubrik>
```

XML är ett språk med vilket man kan beskriva andra märkspråk (Markup Language). Alla XML-dokument är uppbyggda på liknande sätt, med element som innehåller data. De vanligaste elementen består av en starttagg, data i form av vanlig text och en sluttagg. Starttaggen kan även innehålla attribut. XML-dokumenten är i sig rena textfiler, vilket innebär att de kan hanteras på alla typer av plattformar och system. Datat är det egentliga innehållet i dokumentet medan namnet på elementet tillsammans med attributet fungerar som beskrivning på vad datat är. Man brukar kalla det här för *märkt* eller *taggat* data. Vilka märkord (element- och attributnamn) som får förekomma i ett dokument, hur de kan nästlas och vad de får innehålla, definierar man i en s.k. DTD (document type definition).

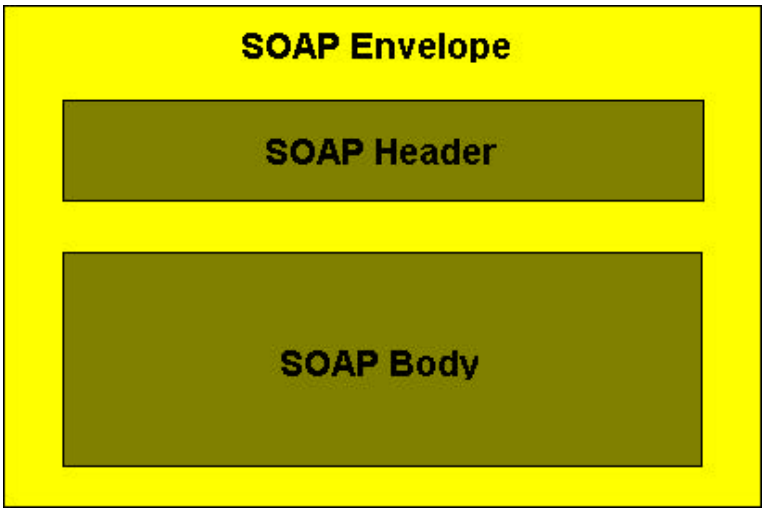
3.3.2 SOAP

SOAP (Simple Object Acess Protocol) [9] ingår i webbtjänster som meddelandehanteringslager. SOAP är en standard för hur data packas ihop till ett strukturerat meddelande som kan förstås av båda avsändare och mottagare, oavsett plattform och programmeringsspråk. SOAP-standarden består av regler för hur meddelandets ”kuvert” skall utformas, samt för hur datavärden av olika typer (strängar, heltal o.s.v.) skall kodas. Innehållet i kuvertet är dock upp till sändaren och mottagaren att komma överens om formatet på. Kuvertet innehåller information om brevet i sig, dess sändare och mottagare. Dessutom innehåller det förstås meddelandet.

Datatyper representeras olika i olika programspråk och två språk kan därför inte alltid utbyta data direkt. Datan måste då först kodas till ett format båda parterna förstår. En del av SOAP-specifikationen (SOAP encoding rules) behandlar därför 'encoding styles'. En 'encoding style' är ett antal regler för hur datatyper skall representeras i ett SOAP-meddelande. I SOAP-encoding-specifikationen finns regler för ett antal datatyper, bland annat heltal, strängar, tid och datum. Man kan dock själv skapa egna 'encoding styles' om de fördefinierade inte räcker till.

SOAP består av fyra delar:

I. **SOAP envelope** används för att förklara vad som finns inne i meddelandet och vem som skall ta hand om det.
II. **SOAP encoding rules** används för att kunna byta ut applikationsdefinierade datatyper och istället representera dessa i XML-kod.
III. **SOAP RPC representation** används för att bestäma hur funktionsanrop och svar anges med XML-kod i ett SOAP-meddelande.
IV. **SOAP binding framework** innehåller regler för hur ett SOAP-meddelande skall transporteras inom eller ovanpå ett annat protokoll.



Figur 3.3: SOAP kuvertet

Själva meddelandet är ett XML-dokument som består av följande:

- **SOAP envelope** är toppelementet i XML-dokumentet och representerar SOAP-meddelandet. Denna är obligatorisk.
- **SOAP header** kan användas för att utöka funktionaliteten i kommunikationen. Detta är en möjlighet och inte ett krav, därför är headern valfri.
- **SOAP body** innehåller den information som skall skickas till mottagaren.

3.3.3 WSDL

WSDL (Web Services Description Language) [12] är en specifikation som definierar hur en webbtjänst beskrivs med hjälp av XML. Ett WSDL-dokument beskriver vilka funktioner en webbtjänst erbjuder, hur den kommunicerar och var den kan nås. WSDL är oberoende av plattform och programspråk, den används främst för att beskriva tjänster. Till varje webbtjänst som publiceras på Internet medföljer ett WSDL-dokument som listar dess tjänster och instruktioner för dess användning. Ett WSDL-dokument fastställer typen på meddelanden som en webbtjänst kan skicka och ta emot, samt specificerar vilken data den anropande applikationen måste tillhandahålla för att webbtjänsten ska kunna genomföra sin uppgift. WSDL använder sig av ett antal element, dessa är:

- **<definitions> –** namnet på WSDL-dokumentet.
- **<types> –** definierar datatyperna som används mellan klienten och servern.
- **<message> –** anger meddelandets format.
- **<portType> –** definierar en mängd abstrakta funktioner. Varje funktion refererar till ett inkommande och ett utgående meddelande.

- **<binding> –** definierar vilket kommunikationsprotokoll som skall användas vid överförandet av ett anrop, alltså vilket protokoll som skall bindas till ett specifikt portType element.
- **<port> –** anger adressen för en binding, alltså vart meddelandet skall skickas.
- **<service> –** Används för att samla ihop ett antal relaterade portar och på så vis definierar en viss webbtjänst fysiska befintlighet.

3.3.4 UDDI

UDDI (Universal Description, Discovery and Integration) [9] lagrar och ger information om tillgängliga webbtjänster. Det tillåter användare att publicera och söka efter webbtjänster på Internet. När en förfrågan sänds från en applikation svarar UDDI-registret med att ge information om upptäckta tjänster, deras status och tillgängligheten av andra tjänster som stämmer överens med förfrågan. Data som samlas i UDDI är indelad i tre kategorier; *Vita sidor* som innehåller allmän information om ett företag, såsom dess namn, adress, telefon o.s.v. *Gula sidor* som innehåller information om verksamheten. *Gröna sidor* innehåller teknisk information om en tjänst. Denna information kan vara programspråket, plattform med mera. Den har också en adress för att kunna anropa en tjänst. Den tekniska arkitekturen för UDDI består av tre delar:

UDDI data model (datastrukturen) anger hur företag och tjänster beskrivs.

UDDI API beskriver hur data publiceras och söks.

UDDI cloud services tillhandahåller information om hur en nod i UDDI registret skall hanteras.

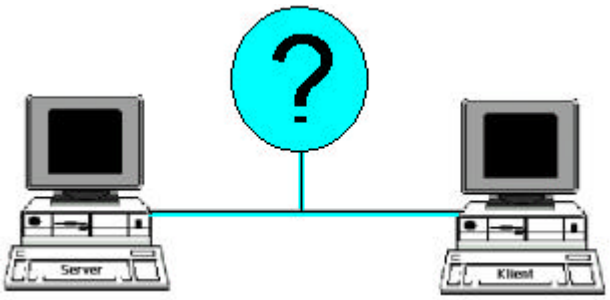
För att kunna kommunicera med UDDI används SOAP-meddelanden.

3.4 Internet Information Service (IIS)

För att göra en webbapplikation tillgänglig på Internet behövs en IIS (Internet Information Service), det är en webbserver för Windows. IIS är gratis men kräver att Windows är installerat på datorn i fråga. För att kunna publicera dokument eller webbapplikationer måste de placeras i en virtuell katalog. När du skapar en Web Application läggs projektet ut automatiskt i en virtuell katalog nämligen(\inetpub\wwwroot\”Projektnamn”).

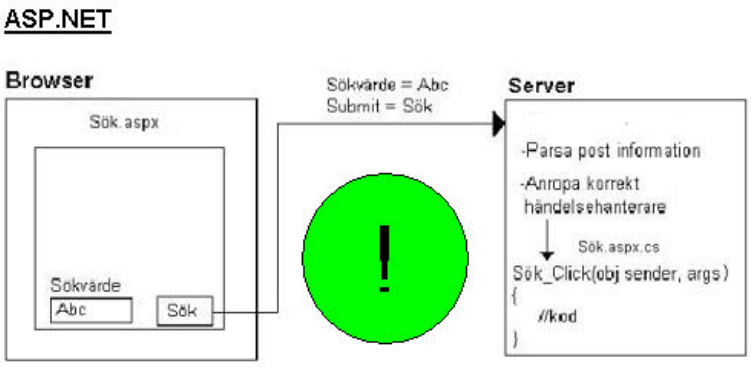
4 Beskrivning av konstruktionslösningen

I kapitel 2 nämndes att någon form av kommunikation mellan klient och server skulle behövas eftersom loggfilsdata skulle överföras. I det läget var det oklart hur kommunikationen skulle ske, men det stod klart att .NET-teknik skulle användas.



Figur 4.1: Den tidiga bilden av klientserver kommunikationen.

I kapitel 3 gjordes en undersökning av vilka tekniker som .NET hade att erbjuda, det som eftersöktes var någon teknik som skulle göra det enkelt för oss att skapa funktionalitet för kommunikation mellan klient och server. Det vi kom fram till var att ASP.NET Web Application, som kan skapas i utvecklingsmiljön MS Visual Studio .NET, var ett intressant verktyg. Som nämnt i kapitel 2 så ger verktyget användaren möjlighet att skapa snygga gränssnitt för visning i webbläsare samt möjlighet att enkelt skapa händelsehanterare för knappar och dylikt i gränssnittet. Kommunikationsmekanismerna som krävs för att överföra data mellan klientdelen (gränssnittet) och serverdelen (händelsehanterarna) sköts automatiskt av ASP.NET´s ramverk.



Figur 4.2: Klientserver kommunikation med ASP.NET-teknik

ASP.NET Web Application tillgodosåg alltså behovet av klientserverkommunikation och gav dessutom möjlighet till att kunna rita upp ett användargränssnitt på ett enkelt sätt. Av dessa

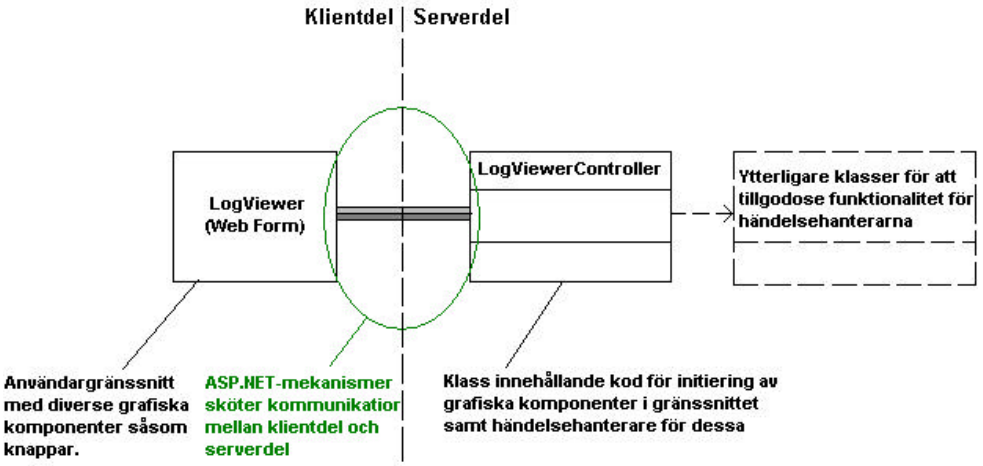
anledningar valde vi att använda oss av detta verktyg. Följande kapitel ger en överblick av systemets design.

4.1 Systemöversikt

LogViewer, en så kallad Web Form, är användargränssnittet som kommer visas i browsern då applikationen anropas.

LogViewerController tillhandahåller händelsehanterare för de grafiska komponenterna som visas i LogViewer såsom knappar och rullninglistor.

Ytterligare klasser, som vi skapat, tillhandahåller funktionalitet som används av händelsehanterarna i LogViewerController vid initiering av gränssnittskomponenter och vid sökning. Dessa innehåller grundläggande funktionalitet såsom filinläsning och strängklippning.



Figur 4.3: Enkel bild över systemstrukturen.

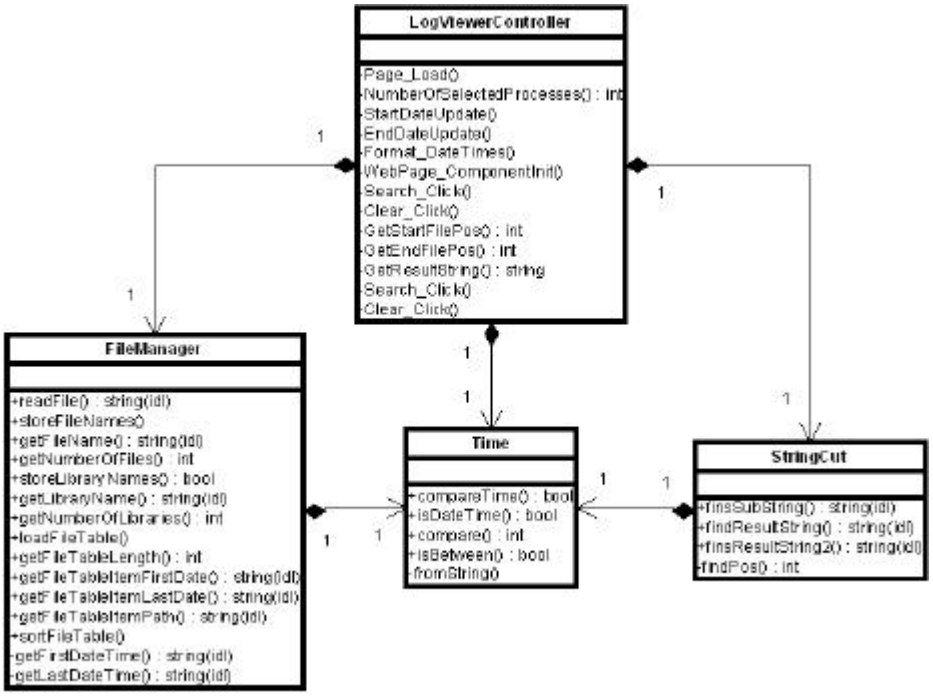
I avsnitt 3.1 nämndes att det ASP.NET Web Application består främst av två delar, en del för grafiska komponenter (gränsnitt) och en del för logiken (händelsehanterare). I systemstrukturen, som visas i figur 4.3, är LogViewer delen med de grafiska komponenterna och LogViewerController är delen med logiken för dessa komponenter. Denna uppdelning är inget som vi har bestämt utan det är så det är upplagt i .NET.

5 Implementation

Utvecklandet av applikationen skedde i MS Visual Studio .NET. Vi använde oss av projektmallen ASP.NET Web Applications. I detta kapitel beskrivs implementationen av systemet.

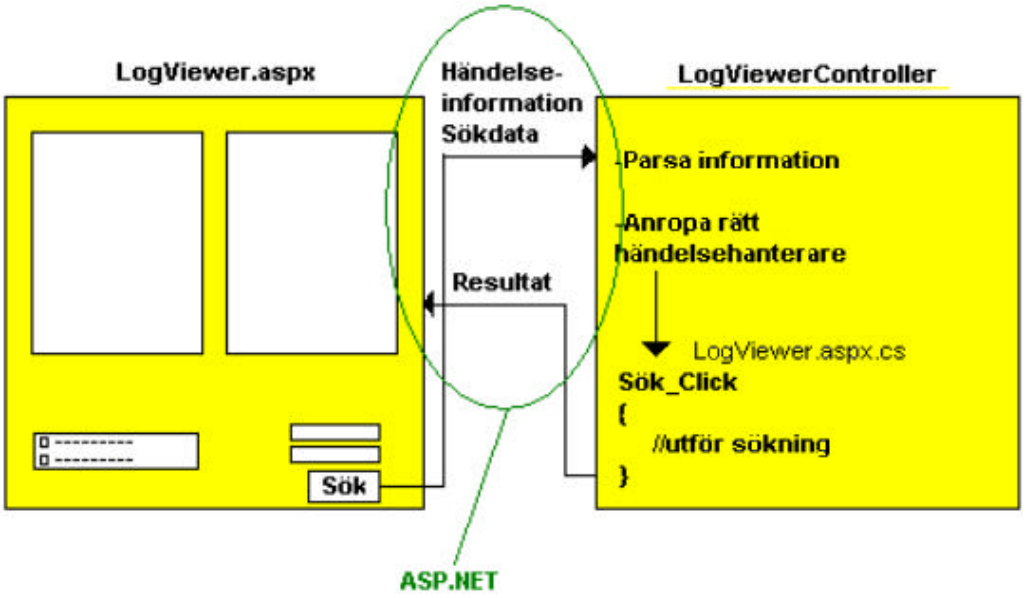
5.1 Klasser som ingår i systemet

LogViewerController, FileManager, Time och StringCut är de klasser som utgör funktionaliteten på serversidan av systemet.



Figur 5.1: Klasser på serversidan

Klientsidan av systemet utgörs av en *gränssnittssida* (LogViewer.aspx), vilken kan visas i en browser, detta är ingen klass utan en fil bestående av bl.a. HTML- och ASP-kod. Klientsidan innehåller alltså inga egenskapade klasser. Då en händelse sker, exempelvis att användaren trycker på en knapp i gränssnittet så sker ett anrop till servern. Informationen parsas och rätt händelsehanterare anropas. Eventuella returvärden sänds tillbaks till klientsidan. Kommunikationen mellan klient och server sköts av ASP .NET.



Figur 5.2: Gränssnittet LogViewer.aspx och LogViewerController

LogViewerController

LogViewerController är huvudkontrollklassen och det är i denna klass alla händelsehanterare är belägna. LogViewerController har en instans av alla andra klasser eftersom händelsehanterarna måste kunna tillgodose all funktionalitet.

FileManager

Filhanteringsklassen har hand om all filhantering då det gäller inläsning av filers innehåll till strängar, namn på bibliotek och namn på filer. I denna klass finns också en *filtabell* som lagrar information om loggfilers startdatum, slutdatum och sökväg. FileManager-klassen använder sig av Time-klassen eftersom filtabellen ska sorteras i datumordning. Detta för att se till att filtabellen refererar till loggfilerna på sådant sätt att föregående post i filtabellen refererar till en fil med lägre datum än nästkommande post. FileManager använder också Time för att bestämma vilka filer som innehåller relevant resultat för en specifik sökning eftersom detta kräver att jämförelse mellan datum utförs. Mer om vektorns syfte och hur filer med relevant resultat bestäms finns i avsnitt 5.2.

Time

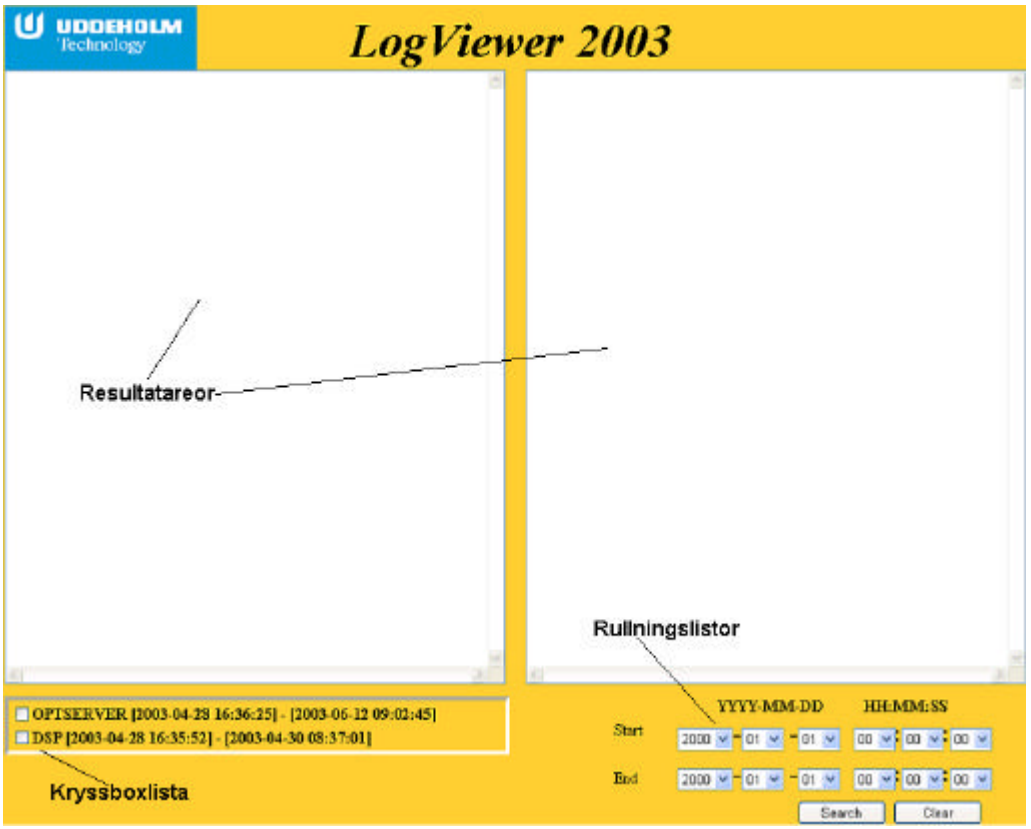
I denna klass finns det möjlighet att använda sig av olika jämförelse-funktioner för tid. Man kan undersöka om ett datum är tidigare eller senare än ett annat datum.

StringCut

Denna klass tillhandahåller funktionalitet för att klippa ut viss data ur en sträng. I denna klass finns en binärsöknings-funktion som snabbt och effektivt kan söka upp och välja ut en delmängd av en sträng, där två datum anges som avgränsare för delmängden.

5.2 Applikationen

I detta avsnitt beskrivs användargränssnittets utseende, hur gränssnittets olika komponenter initieras och hur en sökning fungerar. En verklig bild av användargränssnittet syns i figur 5.3. Detta kan som tidigare nämnts visas i en browser, genom att ange adressen till sidan. Denna adress är densamma som det virtuella bibliotek som det bestämts att applikationen skall köras i (detta görs vid installationen av produkten, se bilaga B). Då ett ASP.NET Web Services-projekt skapas så måste först denna adress anges, exempelvis ”http://localhost/LogViewer”. Localhost innebär ip-adressen på den egna datorn. När applikationen skall anropas kan det se ut på följande vis: http://192.168.0.1/LogViewer. Det första som sker vid anrop av sidan är initiering av grafiska komponenter. De komponenter som ingår i gränssnittet är följande. En knapp (Button) för att utföra sökning, 12 rullningslistor (DropDownList) för inmatning av datum och tid, en kryssboxlista (CheckBoxList) för att välja de processer som önskas att ha med i sökningen samt två textareor (TextArea) där resultatet från sökningen placeras. Knappen och textareorna kräver ingen initiering, detta krävs däremot för kryssboxlistan och rullningslistorna.



Figur 5.3: Det slutgiltiga gränssnittets utseende

5.2.1 Initiering av kryssboxlistan

Som nämndes i kapitel 2 så planerade vi att använda oss av ett *loggfilsbibliotek* där loggfiler från olika processer kan placeras, i sina respektive bibliotek (processbibliotek). För att kunna initiera kryssboxlistan, som ska innehålla val för de processer som kan inkluderas i sökningen, så krävs kännedom om sökvägen till loggfilsbiblioteket samt vilka underbibliotek (processbibliotek) som finns i loggfilsbiblioteket.

Figur 5.4: Kryssboxlistan med val för processer som kan inkluderas i sökning

Som nämns i avsnitt 5.4 så kan denna sökväg ställas in manuellt av serverns administratör. Med hjälp av denna sökväg så avgörs vilka processbibliotek som existerar i loggfilsbiblioteket, detta sker med hjälp av *storeLibraryNames(filsökväg)* i klassen FileManager.

Funktionen tar emot sökvägen till loggfilsbiblioteket som enda parameter. storeLibraryNames() bestämmer vilka processbibliotek som finns i loggfilsbiblioteket samt

sparar namnen på dessa bibliotek. Biblioteksnamnen hämtas sedan med hjälp av *getLibraryName()* i klassen FileManager. Nedanstående pseudokod lägger till alla funna bibliotek i kryssboxlistan tillsammans med första och sista datumet bland alla filer i respektive bibliotek. Datumen används för att användaren av applikationen ska kunna veta vilka tider som är rimliga att söka på i respektive process' loggfiler.

while(fler bibliotek finns)

{

hämta biblioteksnamn

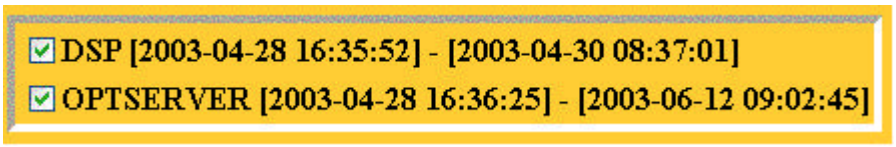
bestäm första och sista datumet bland alla filer i aktuellt bibliotek

lägg till biblioteksnamnet tillsammans med första och sista datumet i kryssboxlistan

välj nästa bibliotek

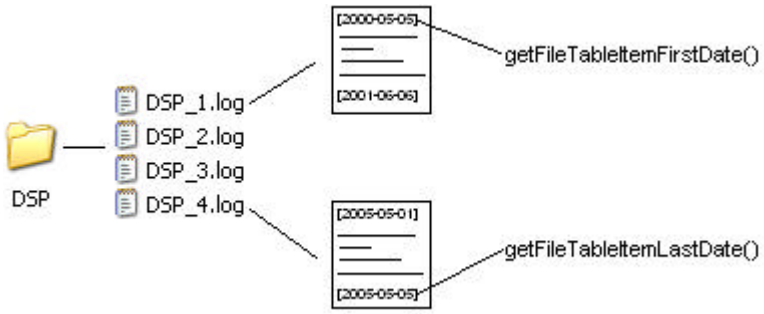
}

I figur 5.5 syns resultatet som kommer visas i kryssboxlistan.



Figur 5.5: Process tillsammans med första och sista tid i processens alla loggfiler

För att få ut första datumet i första filen så används *getFileTableItemFirstDate()* och För att få ut sista datumet i sista filen så används *getFileTableItemLastDate()*, båda i klassen FileManager.



Figur 5.6: Hämtning av första datumet i första filen och sista i sista filen för en process.

När väl alla varv i loopen passerats så har namnen på alla loggfilsbibliotek lagts till i kryssboxlistan och initieringen är klar.

5.2.2 Initiering av rullningslistorna

I rullningslistorna för tid och datum krävs viss initiering. För månads-, tim-, minut- och sekundlistorna är detta mycket enkelt. Månadslistorna initieras med valbara element från 01-

12, timlistorna med 00-23, minut- och sekundlistorna med 00-59. Initieringen av års- och månadslistorna är dock lite mer avancerad. Då det gäller årslistorna så är problemet att det finns inget sista år, så det går inte att tillhandahålla en viss valbar mängd. Av denna anledning finns det, som nämns i 5.4, möjlighet för administratören att själv ställa in det första och det sista året som skall finnas i rullningslistan. Nedanstående pseudokod beskriver förfarandet.

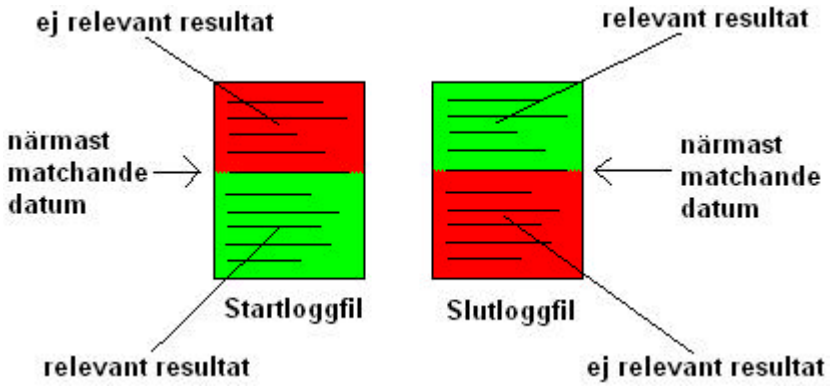
Läs in konfigurations filen
Plocka ut start- och slutår
Lägg till alla år från och med startår till och med slutår i årslistorna.

När detta är utfört så är initieringen av årslistorna klar.
När det gäller dagslistorna så är det lite annorlunda. De måste inte bara anpassa sig beroende på vilken månad som valts utan även vilket år, eftersom februaris dagar blir 29 istället för 28 vid skottår. Därför laddas startsidan om varje gång som en ny månad eller ett nytt år har valts, detta eftersom ny information från servern måste hämtas. Vid en sådan omladdning undersöks årtal och månad varefter antal valbara dagar korrigeras.

5.2.3 En sökning i detalj

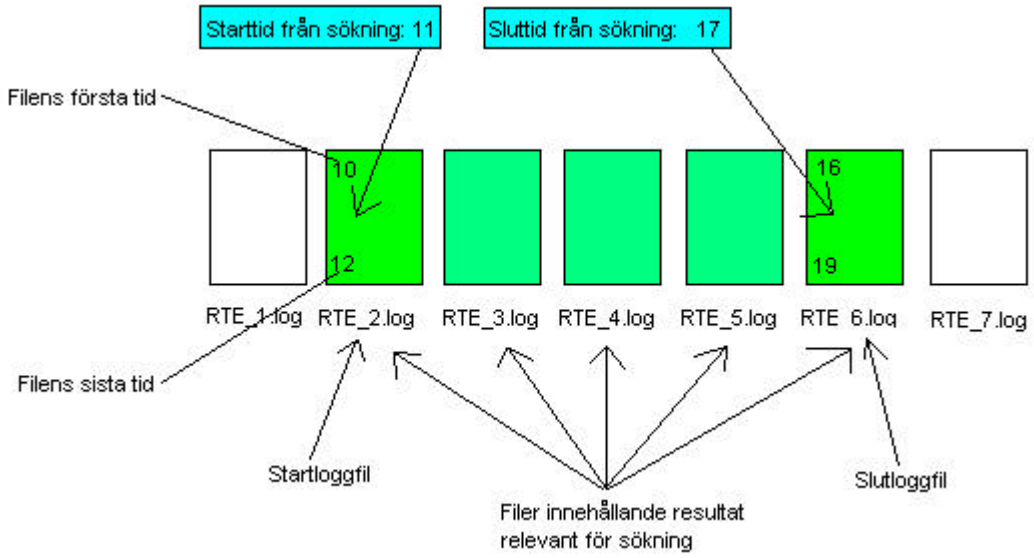
När en sökning utförs så anges de processer som önskas inkluderas i sökningen samt ett startsökdatum och ett slutsökdatum. Dessa datum används för separata sökningar i var och en av de processer som valts. Detta utförs på grund av önskemålet om att jämföra tidpunkter där processer har samarbetat med varandra, vilket då ungefärligen kommer att innebära samma tidpunkter i loggfilerna för dessa processer. Denna jämförelse utförs som nämnts för att undersöka felorsaken i händelse av fel.
Varje loggfil innehåller ett flertal datum. Då en sökning utförs kan startdatumet befinna sig i en loggfil och slutdatumet i en annan som nämnts tidigare. Det gäller då att på ett effektivt sätt ta reda på vilka loggfiler som innehåller data som är relevant för sökningen. För att undvika sökning igenom alla filer, vilket skulle vara otroligt tidskrävande, så undersöks bara första och sista datumet i varje loggfil. På så vis går det snabbt att avgöra vilka loggfiler som är relevanta för en specifik sökning. Om startdatumet från sökningen befinner sig mellan startdatumet och slutdatumet i en viss loggfil, så innebär detta att det här är första loggfilen som innehåller resultat; *startloggfilen*. Om slutdatumet från sökningen befinner sig mellan startdatumet och slutdatumet i en viss loggfil så innebär detta att det här är sista loggfilen som innehåller resultat; *slutloggfilen*. Alla filer mellan dessa, om det finns några, innehåller då

också sökresultat, förutsatt att filerna är ordnade enligt växande datum. Av denna anledning utförs en sortering av filerna. Startloggfilen och slutloggfilen kommer med stor sannolikhet att innehålla delar som inte tillhör sökresultatet, medan alla loggfiler mellan dessa enbart kommer att innehålla sökresultat. Bilden nedan beskriver hur detta skulle kunna se ut.



Figur 5.7: start och slutloggfil bestående av delvis icke-relevant resultat

I exemplet i figur 5.8 nedan så bestäms RTE_2.log som startloggfil och RTE_6.log som slutloggfil. Alla loggfiler mellan dessa kommer då enbart innehålla resultat som är relevant för sökningen medan just start- och slutloggfilen innehåller data som måste sållas bort.

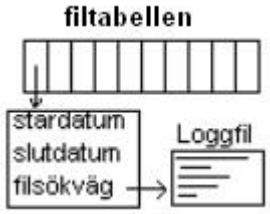


Figur 5.8: Hur det kan gå till att avgöra vilka filer som är relevanta för en viss sökning.

Själva förfarandet att bestämma sökresultat börjar med att bestämma det bibliotek som den aktuella processens (den process av de ikryssade processerna som för tillfället behandlas) loggfiler befinner sig i. För att åstadkomma detta så används *sökvägen till loggfilsbiblioteket* kombinerat med *namnet på vald process i kryssboxlistan*.

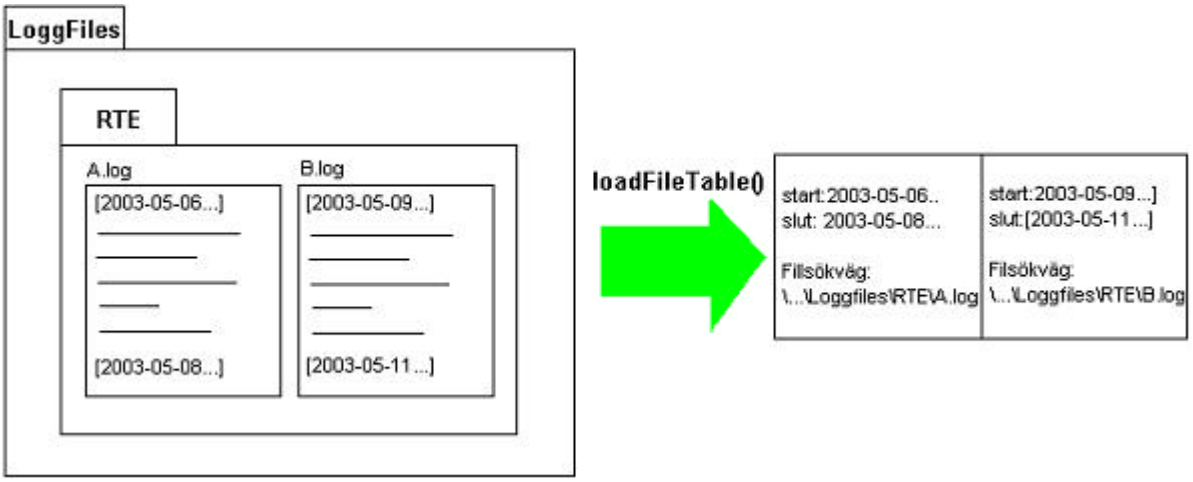
Exempel: processbibliotek = “C:\Program\LogViewer\LogFiles\” + “RTE”

Härnäst laddas filtabellen med information om de loggfiler som finns i aktuellt processbibliotek.



Figur 5.9: En tänkt bild på hur filtabellen ser ut.

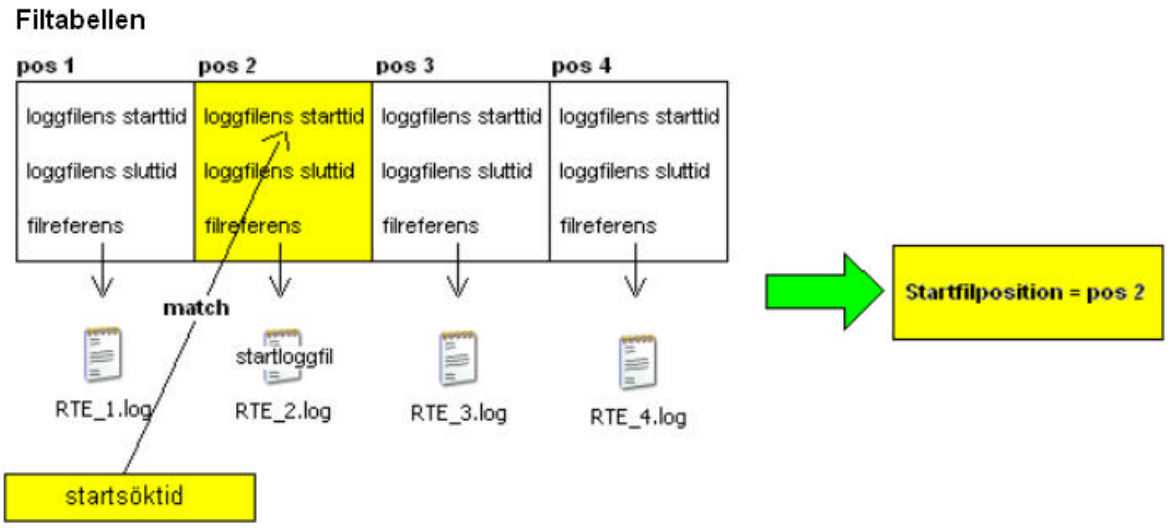
Varje loggfils *första datum*, *sista datum* och *sökväg* lagras på varsin plats i filtabellen. Detta sker genom funktionen *loadFileTable()* i klassen FileManager. Denna funktion tar emot sökvägen till aktuellt loggfilsbibliotek (processbibliotek) som inparameter.



Figur 5.10: Ladda filtabellen

Därefter sker en sortering av filtabellen med hjälp av *sortFileTable()* i klassen FileManager. Informationen i filtabellen används för att bestämma vilka av den aktuella processens loggfiler som är relevanta för sökningen. Vad som måste göras är att bestämma de *två positioner* i filtabellen som avgränsar de filer som är relevanta för sökresultatet från de som inte är det; *startfilspositionen* och *slutfilspositionen*.

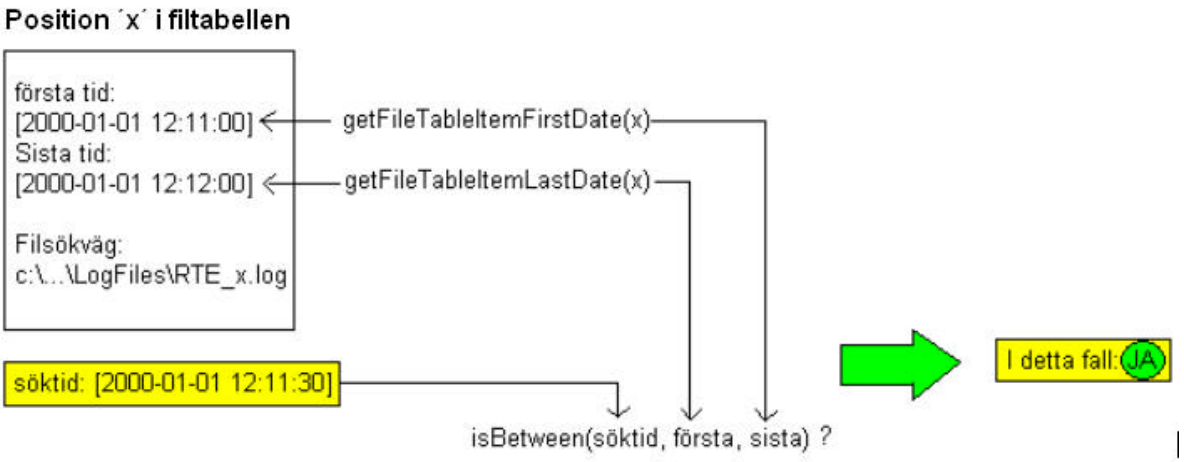
Först bestäms startfilspositionen m.h.a. *startsökdatumet*, därefter bestäms slutfilspositionen m.h.a. *slutsökdatumet*. Om exempelvis startsökdatumet *befinner sig mellan första och sista datumet i en viss fil,* alltså om filen i fråga är startloggfilen, sätts isåfall startfilspositionen till den postition i filtabellen som refererar till just denna fil. Samma förfarande utförs också förstås med slutsökdatumet.



Figur 5.11: Finna startfilspositionen

Undersökningen om ett datum ligger mellan två datum utförs med hjälp av funktionen *isBetween(sökdatum,datum1,datum2)* i klassen DateTime. Denna tar emot 3 datum som parametrar och avgör om första datumparametern ligger mellan de två andra.

Funktionerna *getFileTableItemFirstDate()* och *getFileTableItemLastDate()* i klassen FileManager hämtar ut första respektive sista datumet ur filtabellen på angiven postition.

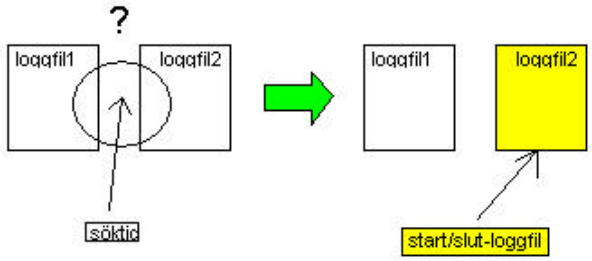


Figur 5.12: isBetween(), getFileTableItemFirstDate() och getFileTableItemLastDate()

Ett problem som kan uppstå här är om start- eller slutsökdatumet som angivits ligger mellan sista datumet i en fil och första datumet i nästa fil, alltså hamnar i den kritiska sektorn som syns på bilden nedan. Loggfilen för detta sökdatum kommer då aldrig att hittas.

Figur 5.13: Den kritiska sektorn

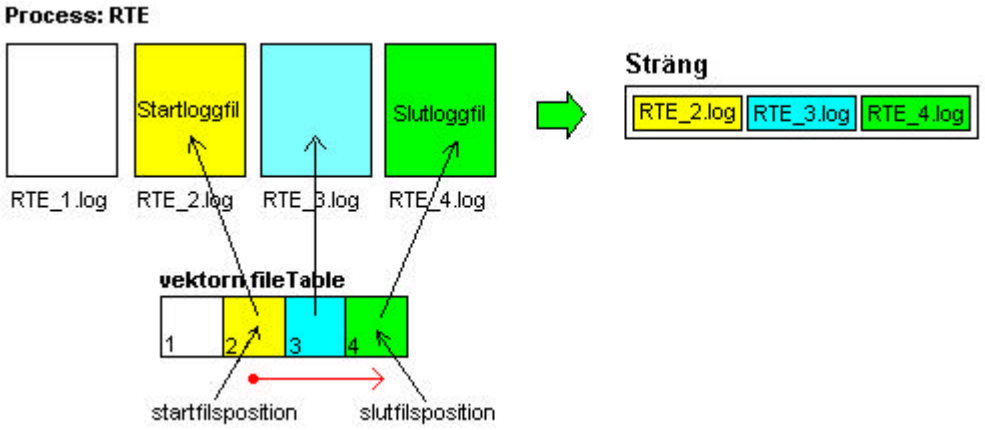
Av denna anledning utförs även en specifik granskning för att täcka av detta fall, då datumet ligger i denna sektor. En undersökning sker alltså för att se om sökdatumet ligger mellan sista datumet i en loggfil och första datumet i nästkommande loggfil. Även denna undersökning utförs m.h.a. isBetween(). Om detta stämmer så bestäms nästkommande loggfil vara start/slut-loggfilen. I fallet med med startloggfilen beror detta på att relevant resultat inte kan uppstå förrän i nästkommande loggfil då söktiden ligger mellan två loggfiler.



Figur 5.14: Vad som sker om start/slut-söktid hamnar mellan två loggfiler.

När det gäller slutloggfilen så tycker man att den borde bestämmas till loggfil1 om tiden hamnar mellan loggfil1 och loggfil2. Det skulle dock kunna inträffa att loggfil2 börjar med loggdata tillhörande ett datum i loggfil1 och om nu slutdatumet skulle sättas till loggfil1 så skulle i detta fall data gå förlorad. Av denna anledning sätts även slutloggfilen till loggfil2.
Efter att relevanta loggfiler bestämts, alltså *startloggfil*, *slutloggfil* och *filerna mellan dessa*, så sparas innehållet i dessa till en *sträng*. Startloggfilen läggs till först och därefter alla filer fram t.o.m. slutloggfilen. Förfarandet beskrivs i nedanstående pseudokod.

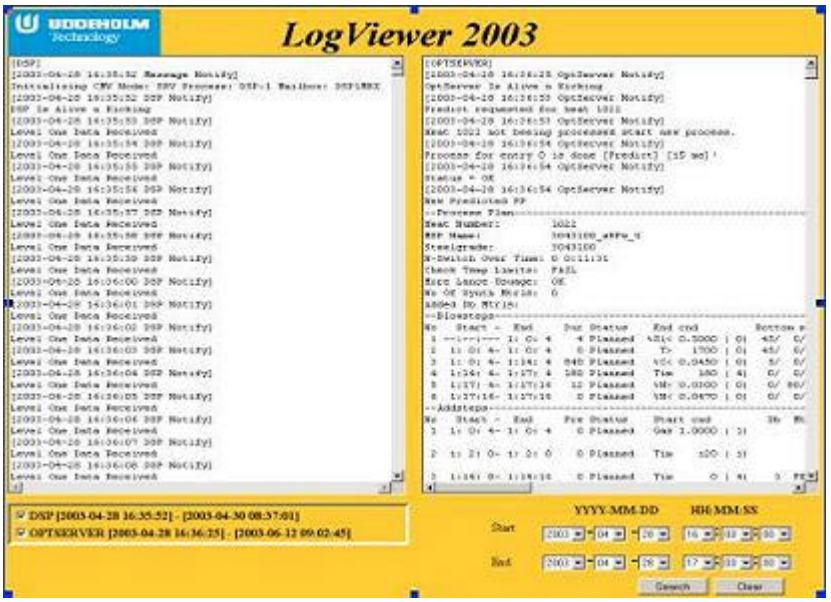
starta på startfilspositionen i filtabellen
while(slutfilspositionen i filtabellen ej nådd)
{
    lägg till den fil som refereras till på aktuell position till resultatsträngen
    stega till nästa position i filtabellen
}



Figur 5.15: Loggfiler med relevant resultat läggs till en sträng.

I figur 5.15 ovan, visas en bild på förfarandet. Loopen arbetar sig sedan igenom alla mellanliggande loggfiler fr.o.m. startfilspositionen (2 i exemplet ovan) fram t.o.m. slutfilspositionen (4 i exemplet ovan) och samlar ihop innehållet i alla filer till en sträng. På detta vis har alla de loggfiler som inte innehåller något sökresultat sållats bort, i detta fall endast RTE_1.log. Ur den sträng som skapats från de relevanta loggfilerna tas sedan exakt sökresultat fram med hjälp av startsökdatum och slutsökdatum. Utförandet sker med hjälp av *findResultString()* i klassen StringCut(). Denna funktion tar emot 3 parametrar, en sträng (datan att söka i) och två datum som avgränsare. Resultatet från funktionen är all loggdata som ligger mellan de två angivna datumen, alltså det slutgiltiga sökresultatet.

Hela detta förfarande utförs separat för var och en av de processer som har valts vid sökningen, så om förfarandet exempelvis gäller processen RTE så genomsöks enbart loggfiler genererade av RTE, loggfilerna från olika processer blandas ej. Till sist så placeras resultat från var och en av de processer som valts i separata resultatareor i gränssnittet. Eftersom bara två resultatareor finns i gränssnittet så kan man bara jämföra två processer samtidigt, dock kan antalet processer att välja mellan vara obegränsat.



Figur 5.16: Resultat av sökning i det slutgiltiga gränssnittet

5.3 Specifika problem

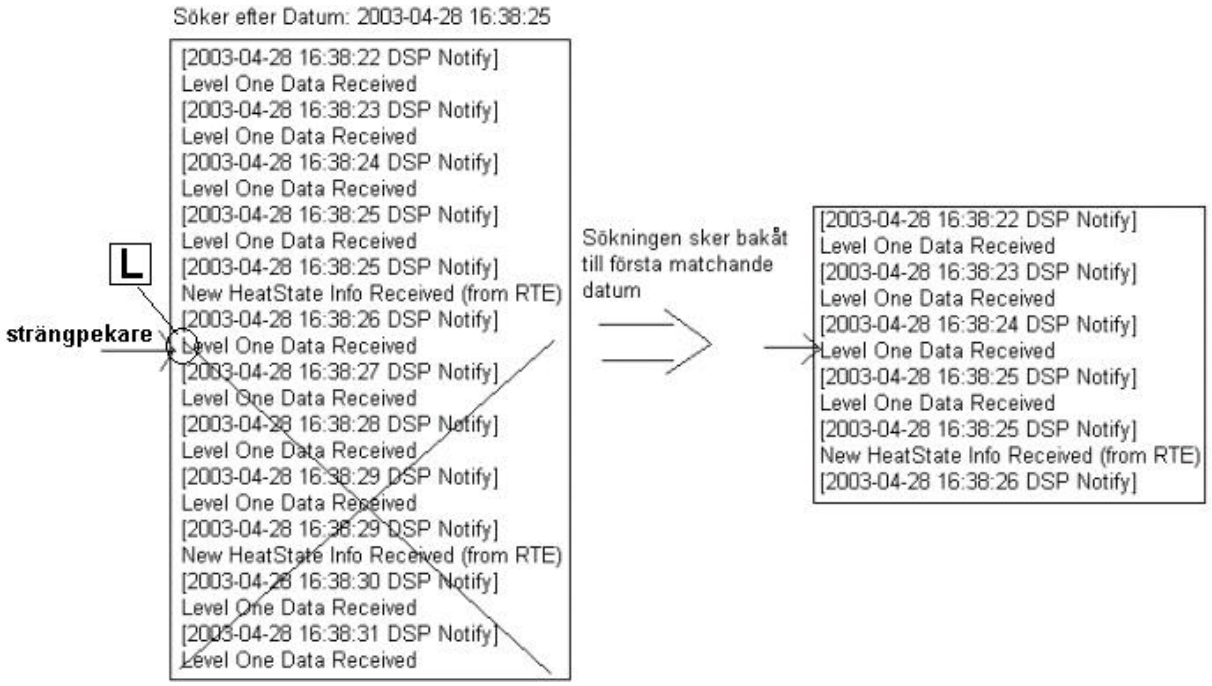
Här får läsaren en inblick i vilka problem som uppstod under implementationsfasen och hur dessa löstes.

5.3.1 Sammanfogning av klasser

De fyra klasser (Serversidan) som har beskrivits i 5.1 är inte skrivna i samma programspråk. LogViewerController-klassen är skriven i C#.NET medan resterande är skrivna i C++.NET. Anledningen till att LogViewerController, som är vårt namn på den klass i ASP.NET Web Application som innehåller händelsehanterare, är skriven i C# är att just projektet ASP.NET Web Application inte fanns tillgängligt i C++. C++ är det språk vi behärskar bäst och därför föredrar att använda. Nackdelen är att två olika språk måste kombineras. Detta löstes genom att använda DLL-filer (Dynamic Link Library) [14]. En DLL-fil är ett filformat i MS Windows för dynamiskt laddbara programbibliotek och klassbibliotek. Genom att i Visual C++.NET skapa ett så kallat ”Class Library”-project och i detta implementera klassen så gick det att skapa en sådan DLL-fil. Denna fil kunde sedan enkelt inkluderas och användas i C#-projektet. Detta förfarande utfördes på FileManager, StringCut och DateTime så att de kunde användas av LogViewer.

5.3.2 Sökningsalgoritm för filsträngen

Den filsträng som hämtas ut från de loggfiler som är aktuella i en viss sökning måste genomsökas då korrekt sökresultat skall hämtas ut. Om sökningen täcker av ett långt tidsintervall kan denna sträng bli mycket stor. Det kan också vara så att strängen innehåller en stor mängd datum, som då alla måste gås igenom vid sökningen eftersom jämförelse krävs för att närmast korrekta datum skall hittas. Dessa faktorer leder till att en sökning kan ta oacceptabelt lång tid om en linjärsökningsalgoritm används. Av nämnda anledningar blev det ett måste att basera sökningen på en betydligt effektivare sökningsalgoritm än linjärsökning, nämligen binärsökning. Sökningar som tog mer än en minut med linjärsökningen kunde utföras på någon sekund med binärsökning. Binärsökningen används för att finna de korrekta positionerna för start respektive slutdatumet. När start och slutpositionerna har hittats i strängen sker en klippning från start t.o.m. slut och detta är resultatsträngen som kommer att visas för användaren av systemet. Det uppstod dock vissa problem i samband med binärsökningen. Algoritmen arbetar på så vis att sökning initieras i mitten av strängen varefter undersökning sker för att bestämma vilken av stränghalvorna som kan tänkas innehålla sökt data. Sökningen sker baklänges i strängen, när strängpekaren sätts att peka på ’L’ exempelvis i bilden nedan så klättrar algoritmen bakåt, tecken för tecken, tills en vänster-hakklammer hittas. När vänster- hakklammern har hittats så indikerar detta på att det kan vara ett datum, då läses 19 tecken in. I bilden nedan blir det datumet 2003-04-28 16:38:26. Om det sökta datumet är mindre än datumet i strängen så måste det korrekta datumet finnas i den övre halvan, om datumet i strängen stämmer överens med det sökta datumet är det funnet annars finns datumet i den nedre datahalvan. I detta exempel finns datumet i den övre halvan eftersom sökdatumet 2003-04-28 16:38:25 är ett tidigare datum än 2003-04-28 16:38:26.



Figur 5.17: En halvering av filsträngen

Den halva som med all säkerhet inte innehåller sökresultat skalas bort och samma förfarande utförs återigen på den kvarvarande datamängden. Detta görs fram tills det att datumet har hittats eller att ingen mer sökdata existerar.

Ett fall kan uppkomma då man inte hittar något datum i strängen. Det är om sista halveringen av strängen har skett på ett sådant sätt att det inte finns något datum i den övre halvan av strängen, som illustreras i figur 5.18 på nästa sida. Då utförs istället en linjärsökning på hela kvarvarande mängden data.

Psedokod för hur förfarandet går till:

Loopa så länge det finns ett datum(om inget datum har hittats byt till linjärsökning för att kolla sista data mängden efter närmast matchande datum)

{

halvera strängen

placera strängpekaren på mitten

sök bakåt i strängen tills ett datum hittas

jämför datumet

om datumet inte matchar kapa bort den halva som datumet inte finns i.

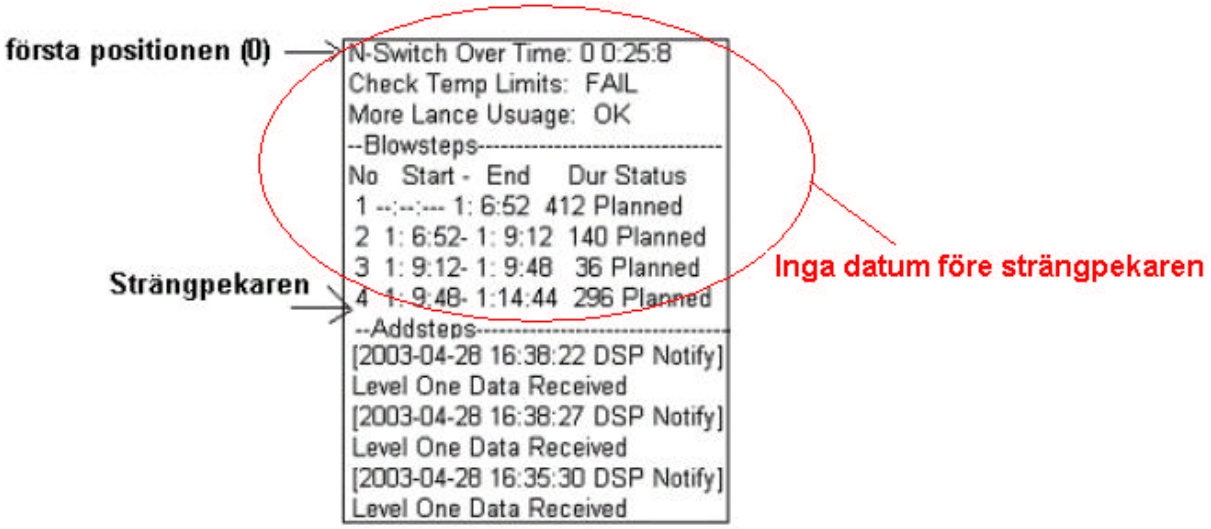
}

Exempel: om man söker efter 2003-04-28 16:38:25 och som synes i figur 35 nedan har strängpekaren placerats så att inget datum finns i den övre halvan alltså kommer algoritmen

att svara att inga mer datum är funna och det sista datumet som undersöktes innan denna halvering skulle då vara den mest korrekta vilket det inte är.

Det närmast matchande datumet i mängden är [2003-04-28 16:38:27].

När ett sådant fall inträffar, alltså inget datum hittas sista gången, så traverseras den sista strängen igenom linjärt för att hitta det närmast korrekt matchande datumet. Detta datum är funnet så fort datumet i strängen blir större än eller lika med det sökta datumet. Linjärsökningen startar från första positionen(0) i strängen och söker sig nedåt linjärt.



Figur 5.18: Specialfall då linjärsökning krävs

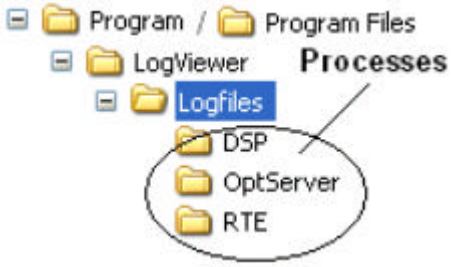
Två ytterligare specialfall:

Fall 1: Om startdatumet som har angivits är tidigare än första datumet i strängen, så sätts datumet automatiskt till det första datumet. Det finns ingen anledning att utföra en binärsökning..

Fall 2: Om slutdatumet är senare än sista datumet i strängen så sätts datumet automatiskt till det sista datumet. Det finns ingen anledning att utföra en binärsökning.

5.4 Serverkonfigurering

Under serverinstallationen (se bilaga B) av produkten skapas ett bibliotek, LogViewer, i programmappen på datorn.



Figur 5.19: Efter installationen

I denna mapp finns en ini-fil där man kan göra vissa inställningar för servern. Dessa inställningar läses sedan in när startsidan laddas upp i browsern. Nedan syns ett exempel på inställningar i ini-filen.

[Sökvägen till processerna]
path**=C:\Program files\LogViewer\Logfiles**;
[Ange start och slutår som ska visas i gränssnittets rullnings lista för år]
startyear**=2000**;
endyear**=2004**;
[Maximalt antal tecken som tillåts att visas på skärmen/resultatarea]
max_length**=1000000**;

Vad som utförs först är att hela LogViewer.ini läses in till en sträng med hjälp av readFile() i klassen FileManager. Sökvägen plockas exempelvis ut från strängen med hjälp av *findSubString()* i klassen StringCut.
findSubString(sträng,”**path=**”,”**;**”)
findSubString() tar emot tre variabler, datan att söka i, startavgränsare och slutavgränsare. Funktionen returnerar den delsträng av sökdatan som ligger mellan avgränsarna. Resultatet blir då i detta fall ”C:\Program files\LogViewer\Logfiles\”. Därmed har sökvägen till loggfilsbiblioteket hämtats ur ini-filen.

Vad som också är rimligt är att administratören av servern själv kan bestämma vilka år som skall finnas i listan. Av denna anledning finns det även möjlighet att bestämma detta i ini-filen. Därför hämtas ”startyear” och ”endyear” från ini-filen in på samma sätt som sökvägen. Maximalt antal tecken som tillåts att visas på skärmen/resultatarea (max_length) kan också ställas in i ini-filen för att undvika att alldeles för ospecifika sökresultat visas i resultatareorna och på så vis förbrukar en massa minne i onödan. max_length hämtas in på samma sätt som sökvägen, startyear och endyear.

6 Testning

De viktigaste testerna bestod av att söka på specifika sökresultat och jämföra resultatet från sökningen med de förväntade. Resultatet från samtliga tester var positiva.
Vi testade även att mata in felaktiga värden. Resultat visas i Tabell 6.1 nedan.

Tabell 6.1: Resultat från felinmatningstesterna

Inmatning	Resultat
Båda söktiderna är tidigare än första datumet i vald process' loggfiler.	Fungerar. Användaren informeras om att tidsintervallet inte finns med i aktuell process' loggfiler.
Båda söktiderna är senare än sista datumet i vald process' loggfiler.	Fungerar. Användaren informeras om att tidsintervallet inte finns med i aktuell process' loggfiler.
Slutsökdatumet är tidigare än startsökdatumet.	Fungerar. Användaren informeras om att slutsökdatumet inte kan vara tidigare än startsökdatumet.
Mer än två processer är ikryssade.	Fungerar. Användaren informeras om att högst två processer åt gången kan jämföras.

Vi testade hur produkten fungerade då den anropades från den dator som servern kördes på, hur den fungerade då den anropades från andra datorer i nätverket och från datorer i ett externt nätverk. Resultatet kan ses i Tabell 6.2 nedan.

Tabell 6.2: Resultat från befintlighetstesterna

Klienten har accessats från	Resultat
Den egna datorn(localhost)	Fungerar
En datorn från samma nätverk	Fungerar
En dator från ett extern nätverk	Fungerar

Vi testade även vad som sker om det inte finns några processmappar eller filer att söka i, men resultaten från samtliga tester var positiva. Testet med att utelämna processmappar och filer utfördes enbart från den egna datorn, det fanns ingen anledning att prova de andra två fallen

(d.v.s. från annan dator i samma nätverk och externt nätverk). Om det finns filer och processmappar att söka i eller ej beror inte på befintligheten hos den anslutande datorn.
En anledning till att systemet blev så stabilt beror troligen på den kontinuerliga testningen som utfördes under utvecklingen av produkten samt p.g.a. den felsäkra datum/tids – inmatningen i gränssnittet. Vi använde oss dessutom av kontraktsprogrammering vilket gjorde att så länge vi upprätthöll kontraktet så returnerade funktionerna det förväntade resultatet.

7 Resultat och rekommendationer

I detta kapitel mäter vi de mål som vi satte upp i kapitel 1.3. Vi nämner även vilka begränsningar som finns i applikationen och hur de skulle kunna förbättras.

7.1 Resultat

I avsnitt 1.3 satte vi de mål som vi ville hinna uppnå under projekttiden. Det första målet var att skapa en klientserver-applikation för att kunna överföra loggfilsdata mellan klientapplikationen och servern. Våra föreställningar var att detta skulle bli både svårt och tidskrävande att utveckla. Färdiga verktyg i Visual Studio.NET gjorde dock att klientserver-utvecklingen blev mycket enklare än vi hade trott, enda problemet var egentligen att lära sig hur verktygen fungerade.

Andra målet var att utveckla ett användargränssnitt för klienten. I detta gränssnitt skulle det finnas möjlighet för datum/tid-sökning. Funktionalitet för att jämföra tider med varandra skulle utvecklas för att kunna finna närmast matchande tider. Därtill skulle det finnas möjlighet att jämföra sökresultat från flera processer i gränssnittet.

Utvecklandet av gränssnittet trodde vi inte skulle bli något problem. Däremot trodde vi att det skulle bli svårt att utveckla funktionalitet för att jämföra tider. Om vi inte skulle lyckas med detta så skulle matchning av datum istället få ske exakt. Nackdelen med detta är att om de tider som anges vid sökning inte skulle finnas i loggfilerna så skulle det inte gå att få ut något resultat. Visningen av sökresultat hade vi tänkt göra så enkel såsom att placera sökresultat från alla processer i en och samma resultatarea. Gränssnittet blev bättre än vi förväntat oss. Vi utvecklade ett gränssnitt och implementerade funktionalitet för jämförelse mellan tider. Visning av resultat sker på sådant sätt att resultat från varje vald process visas i separata resultatareor.

Tredje målet var att skapa funktionalitet för att hämta och sammanslå sökresultat från flera loggfiler. Som nämnts tidigare så kan det hända att en användare anger start- och slutdatum på så vis att startdatumet matchas mot ett datum i en loggfil och slutdatumet mot ett datum i en annan. Vi var vid projektets inledning osäkra på om vi skulle lyckas tillgodose sådan funktionalitet. Det blev däremot inga större problem, vi lyckades utveckla funktionaliteten till den grad som ställts upp i målet.

Vi kan slutligen säga att vi är mycket nöjda med resultatet av vårt arbete. Vi lyckades uppnå varje enskilt mål som ställdes upp i projektets inledningsfas.

7.2 Begränsningar och möjlighet till vidareutveckling

I detta avsnitt tar vi upp begräsningar i systemet och möjlighet till förbättring.

7.2.1 Begränsningar i systemet

1. Ur säkerhetssynpunkt är programmet bristfälligt. Vem som helst kan få access till servern (ip-adressen är det enda som behövs) och undersöka vad loggfilerna innehåller för data. Man kan tycka att systemet skulle vara oanvändbart när det har en sådan säkerhetsbrist, men så är inte fallet. Systemet kommer att användas inom ett intranet, där det redan finns ett säkerhetssystem installerat.
2. En annan begränsning är att filer från olika processer inte kan blandas hur som helst med varandra utan måste vara strikt separerade i olika mappar. Detta är förstås bra med tanke på strukturen men problemet är att någon måste skapa mapparna manuellt.
3. Då man jämför loggfilerna i de båda fönstren finns det inget enkelt sätt att snabbt hoppa ner till det motsvarande datumet i det andra fönstret, man måste själv leta upp motsvarande datum i andra fönstret.
4. Om ett orimligt högt tidsintervall väljs så kan datorns fysiska minne ta slut. Detta skulle kunna hända om exempelvis servern eller klienten bara har 500 MB ledigt primärminne och ett tidsintervall har valts så att man slår samman så många filer att strängen överskrider 500 MB.

7.2.2 Möjlighet till vidareutveckling

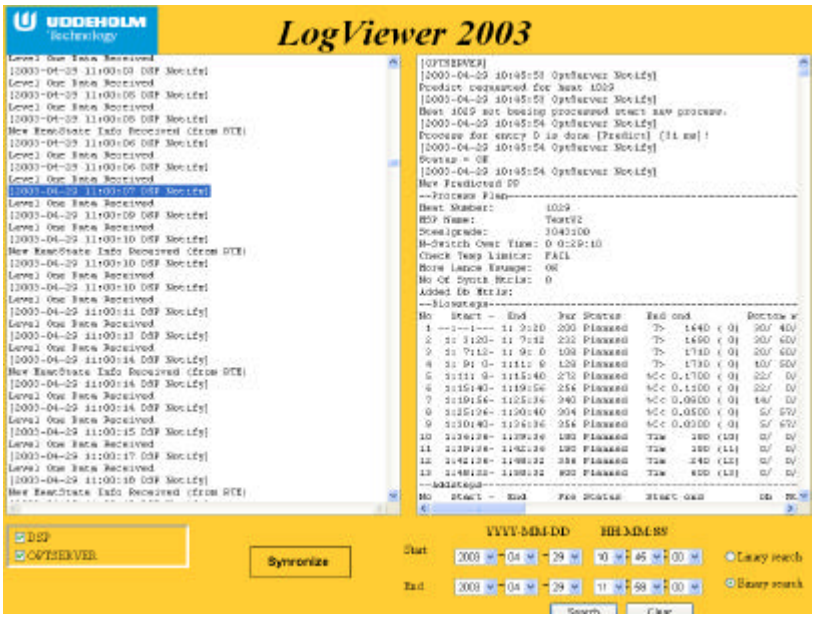
1. Vad det gäller säkerheten skulle man kunna lägga in ett inloggningssystem för att förhindra att obehöriga användare får access till systemet. Detta skulle vara en bra utveckling om man vill använda produkten på Internet.
2. Man skulle kunna koda om programmet så att loggfiler från olika processer kan blandas hur som helst med varandra och på så vis slipper vara strikt separerade. Men detta skulle vara en försämring av produkten då man ser till struktur. Vad som istället skulle kunna göras är att skapa ett program som automatiskt skapar mapparna och placerar filerna i dessa.

   Exempel på hur det skulle kunna fungera:

Figur 7.1: Hur en automatisk skapning av mapparna skulle kunna ske.

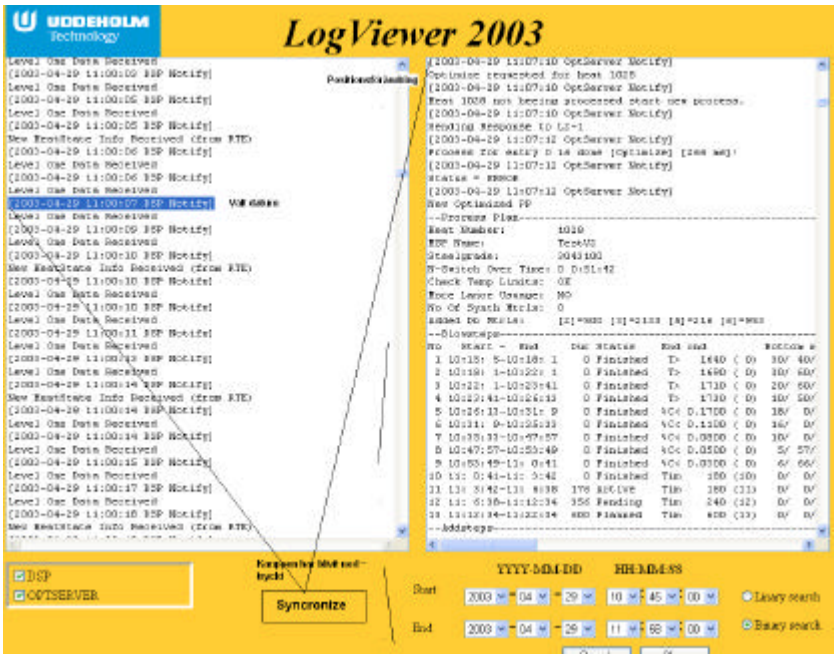
Programmet ovan skulle fungera så att om man trycker på ’Create libraries’ så skapas en mapp för varje process som finns. Loggfilerna flyttas sedan till sina respektive mappar. Programmet behöver dock inte nödvändigtvis vara fristående, funktionaliteten kan lika väl inkluderas i LogViewer.

3. För att användaren ska kunna ändra positionen i fönstret direkt och på så vis slippa skrolla ner till det datum man vill jämföra med så kan man utveckla positionerings-funktionalitet. Detta ska då gå att läsa in det datum som användaren trycker på i första fönstret. Sedan ska positionen för datumet sökas upp i det andra fönstret som automatiskt ompositioneras i det andra fönstret.



Figur 7.2: En valig sökning i LogViewer

”Syncronize” är den tänkta extra knappen som skulle implementeras i gränssnittet och förses med funktionaliteten. Om man exempelvis som i bilden ovan trycker på [2003-04-29 11:00:07] och sedan trycker på knappen ”Syncronize” så ska det högra fönstret stegas fram till det närmast matchande datumet som i bilden nedan.



Figur 7.3: Reglering av position med ”Syncronize”

Detta är en vidareutveckling som vi gärna skulle ha velat se i det slutliga systemet men tiden blev tyvärr för knapp.

4. Vi har inte kommit på något bra sätt att förhindra att minnet tar slut på klientsidan. Vad gäller servern så kan man se till att installera ordentligt med primärminne för att minska risken för att minnet tar slut.

8 Summering av projektet

Då vi började studera plattformen .NET så trodde vi att det skulle krävas mycket tid för att få lärdom om .NET innan själva arbetet kunde påbörjas. Vi trodde även att det skulle bli svårt att finna information om .NET då det var relativt nytt. Att skapa någon slags klientserver-kommunikation var en del av vår uppgift och det ansåg vi inte vara trivialt, särskilt inte när det skulle utvecklas med verktyg vi aldrig varit i kontakt med.

Med anledning av denna föreställning hade vi små förhoppningar att lyckas hinna med att utveckla själva produkten så mycket som vi skulle önska. Men överraskande nog så blev det en enkel sak att sätta sig in i .NET., det var inte heller svårt att få tag på information. Att skriva en klientserver-applikation, som hade varit det arbete som vi trodde skulle vara svårast och ta längst tid, var i själva verket relativt enkelt.

All den funktionalitet som krävs för kommunikation mellan datorer autogenererades av färdiga verktyg. Vi behövde inte alls koncentrera oss på den delen. Arbetet bestod främst av att hitta verktygen och förstå hur de fungerade. Resterande delen av uppgiften blev däremot mycket svår, d.v.s. sökning i loggfilerna och behandling av resultat.

Det krävdes kontinuerlig analys av krav från företaget samt kontinuerlig lösning av avancerade problem. Vi var också tvungna att sätta oss in i delar av företagets befintliga system, vilket krävde en hel del tid. Det som var till stor hjälp var att företaget var ganska litet så det var enkelt att få bra kontakt med företagspersonalen. Dessutom var personalen både hjälpsamma och kompetenta, var det något vi behövde hjälp med så ställde dom upp.

Vi är nöjda med den färdiga produkten, alla mål som ställdes upp i projektets början blev tillgodosedda. Det finns dock viss funktionalitet vi önskar att vi hade haft tid till att vidareutveckla. Vi är även nöjda med att ha fått goda kunskaper i .NET.

Referenser

[1] Julian Templeman och Andy Olsen. *Visual C++ .NET Steg för Steg*. Pagina Förlags AB, 2002.

[2] Robin A. Reynolds-Haertle. *OOP med Microsoft Visual Basic .NET och Microsoft Visual C# .NET Steg för Steg*. Pagina Förlags AB, 2002.

[3] Michael Hyman och Bob Arnson. *Visual C++ .NET I Ett Nötskal*. Pagina Förlags AB, 2002.

[4] Microsoft. *Visual Studio .NET walktroughs*. 2002. Handbok medföljande MS Visual Studio.NET-paketet.

[5] Nätverk & Kommunikation, Nummer 11, 22 augusti 2002.

[6] H.M Deitel och P.J. Deitel. *C++ How to Program*. Prentice Hall, 2nd edition, 1998.

[7] MSDN, **http://msdn.microsoft.com/**, 2003-11-25

[8] Webbtjänster,**http://www.handels.gu.se/epc/archive/00002660/01/Nr9_RHG.pdf**, 2004-01-02

[9] SOAP, **http://www.w3.org/TR/2001/WD-soap12-part2-20011002/**, 2003-12-05

[10] UDDI, **http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf**, 2004-01-02

[11] XML, **http://www.acc.umu.se/~alpha/xml/**, 2003-11-26

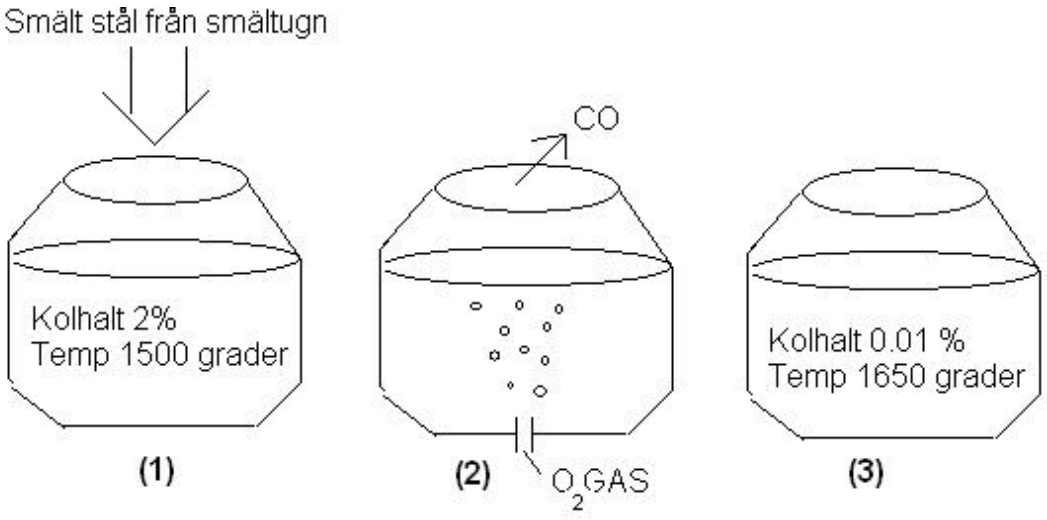
[12] WSDL, **http://www.w3.org/TR/wsdl**, 2004-01-02

[13] .NET, **http://www.netologi.se/default.aspx?Contents=VadArNet_1**, 2003-12-06

[14] DLL, **http://susning.nu/DLL**, 2004-01-19

A Konverterprocess

En process för att minska kolhalten i stål.



Figur A.1: Konverterprocess

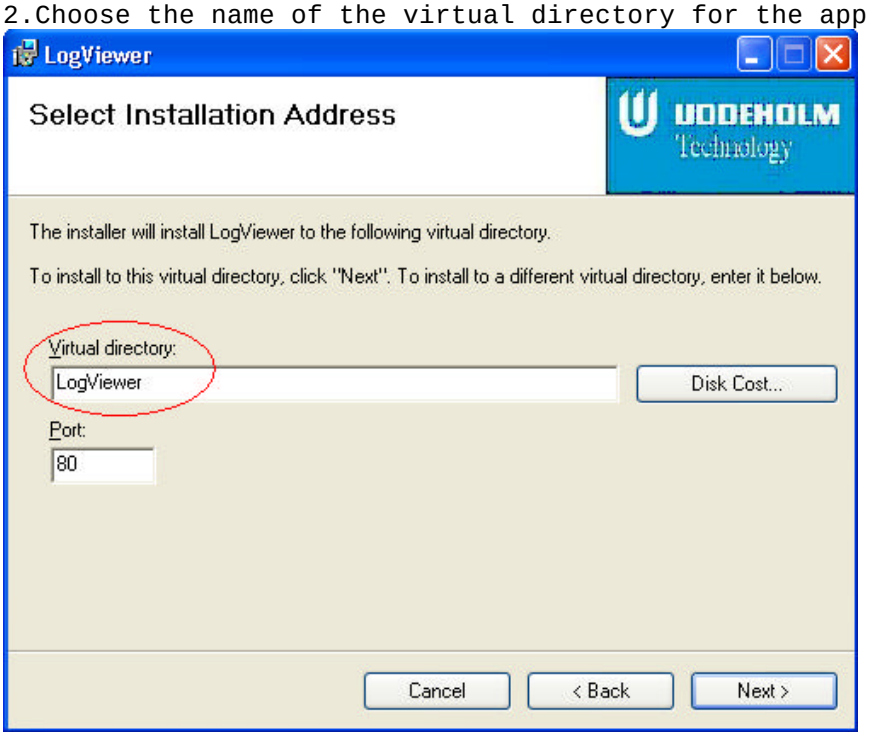
1) Smält stål från smältugnen tillsätts i konvertern.
2) Syre tillsätts och värmen stiger, en exoterm reaktion sker C + O ←→ CO
3) Kolhalten i stålet har minskat.

B How to install LogViewer server

[Installation manual for LogViewer-Server]

1.Run Setup.exe

2.Choose the name of the virtual directory for the application to run on



Figur A.2: Virtuella sökvägen

3.Complete setup wizard

4.If you got an Operating System with english language, the installation directory will be C:\Program Files\LogViewer\.
On the other hand, if you got a swedish Operating System, the directory will be C:\Program\LogViewer\.
In \LogViewer\LogViewer.ini you can choose different settings for your application.
All processes that you are going to check needs a separate library in "Logfiles". Location on the disk for Logfiles= C:\"Programsfolder"\LogViewer \Logfiles\"Process1".
Example:

Figur A.3: Processbibliotek

The logfiles from the processes should be placed in the correct library(the one you been creating).

5.The server setup is finished.

6.Now is it possible to access the homepage at **http://(IP-address)/(**Virtual Directory)
Virtual Directory: The name you chose for your Virtual directory.
IP-address: Address to the machine where the server runs.

Good luck!

[LogViewer –Server requirements]

- IIS Internet Information Service 5.0(server add on for frontpage2000)
- Windows operating system (Windows XP/Windows2000)
- Security settings for the \programfolder\ should be set to read/write.

C Klasser och metoder

C.1 C++ Klassen Time

C.1.1 Time.h

```
/*Defines*/
#define YEAR_INTERVALL 0,4
#define MOUNTH_INTERVALL 5,2
#define DAY_INTERVALL 8,2
#define HOUR_INTERVALL 11,2
#define MINUTE_INTERVALL 14,2
#define SECOND_INTERVALL 17,2

namespace DateTimeDll
{
     public __gc class Time
     {
     public:
          Time();
          //Funktionen kollar om 'searchTime'>='fileTime'
          //Pre:  'fileTime'  &&  'searchTime'  ska  båda  vara  strängar  formaterade
          //enligt: yyyy-mm-dd hh:mm:ss
          //Post:  OM  'searchTime'>=  'fileTime'  så  har  true  returnerats  ANNARS
          //false
          bool compareTime(String* fileTime,String* searchTime);

          //Kollar om en sträng är ett giltigt datum
          //Pre:true
          //Post: OM strängen består av 19 tecken och har '-' på pos 4 &&'-' på
          //pos 7 && ':' på pos 13 && ':' på pos 16 så har true returnerats ANNARS
          //false
          bool isDateTime(String* dateTime);

          //Denna  funktion  tar  in  två  datumsträngar  gör  om  dessa  till  DateTime
          //objekt och utför en jämförelse på dessa. Om första objektet var större
          //än det andra så har 1 returnerats om de vara lika 0 ANNARS -1.
          //Pre:  'time1'  och  'time2'  ska  båda  vara  strängar  formaterade  enligt:
          //yyyy-mm-dd hh:mm:ss
          //Post: OM 'time1' > 'time2' så har '1' returnerats
```

```
        //OM 'time1' == 'time2' så har '0' returnerats
        //OM 'time1' < 'time2' så har '-1' returnerats
        int compare(String* time1,String* time2);

         //Undersöker om datumsträngen 'time' ligger mellan datumsträngarna
         //'first' och 'last'
         //Pre:'time','first' och 'last' ska alla vara strängar formaterade
         //enligt: yyyy-mm-dd hh:mm:ss
         //Post: OM 'first' <= 'time' <='last' så har true returnerats ANNARS
         //false
         bool isBetween(String* time, String* first, String* last);

    private:
         //Funktionen plockar ut de olika datumdelarna ur strängen och gör om dom
         //till int värden.
         //Pre: Strängens format skall vara: yyyy-mm-dd hh:mm:ss
         //Post: de privata 'int' variablerna year, month, day, hour, min, sec
         //har satts.
         void fromString(String* timeStr);

        int year, month, day, hour, min, sec;
    };
}
#endif
```

C.2 C++ Klassen FileManager

C.2.1 FileManager.h

```
/*Constants*/
const int LINE1=4;
const int LINE2=7;
const int COLON1=13;
const int COLON2=16;
const int DATE_TIME_LENGTH=19;

namespace FileManagerDll
{
     typedef __gc struct FileTable
     {
          String* path;
          String* date;
          String* endDate;
     };

     public __gc class FileManager
     {
     public:
          FileManager();

          //Läser in angiven fil till en sträng
          //Pre: true
          //Post: OM sökvägen existerade så har filens innehåll returnerats som en
          //sträng
          //ANNARS har felmeddelande skrivits till skärmen och tomsträng ("")
          //returnerats.
          String* readFile(String* path);

          //Lagrar namnen på alla de filer som finns i angivet bibliotek 'path'
          //i vektorn filenames.
          //Pre: Angiven sökväg existerar
          //Post: Namnen på alla de filer som finns i angivet bibliotek
          //har lagrats i vektorn filenames.
          void storeFileNames(String* path);
```

```
//Hämtar filnamnet som finns i 'filenames' på angiven position 'pos'
//Pre: 0 <= pos <= getNumberOfFiles() && storeFileNames() har anropats
//tidigare.
//Post: Filnamnet på angiven position har returnerats som en sträng
//innehållande
//enbart små bokstäver.
String* getFileName(int pos);

//Hämtar antalet filnamn som finns lagrade i vektorn 'filenames'
//Pre: storeFileNames() har anropats tidigare
//Post: Antalet filnamn som finns lagrade i vektorn 'filenames' har
//returnerats.
int getNumberOfFiles();

//Lagrar namnen på alla de kataloger som finns i angivet bibliotek 'path'
//i vektorn 'libraries'.
//Pre: true
//Post: Namnen på alla de kataloger som finns i angivet bibliotek
//har lagrats i vektorn 'libraries' och true har returnerats om
//sök vägen till biblioteket inte fanns har false returnerats.
bool storeLibraryNames(String* path);

//Hämtar katalognamnet som finns i 'libraries' på angiven position 'pos'
//Pre:0 <= pos <= getNumberOfLibraries() && storeLibraryNames() har
//anropats tidigare.
//Post: Katalognamnet på angiven position har returnerats som en sträng
//innehållande enbart små bokstäver.
String* getLibraryName(int pos);

//Hämtar antalet katalognamn som finns lagrade i vektorn 'libraries'
//Pre: storeLibraryNames() har anropats tidigare
//Post: Antalet katalognamn som finns lagrade i vektorn 'libraries' har
//returnerats.
int getNumberOfLibraries();

//Här laddas filtabellen 'table', i vilken ett objekt av typen FileTable
//läggs på varje plats, innehållande sökvägen till filen, startdatum och
//slutdatum. Detta utförs för alla de filer som befinner sig i angiven
//katalog.
//Pre: Sökvägen 'path' måste existera och varje fil i denna katalog måste
//innehålla minst ett datum, formaterat enligt "[YYYY-MM-DD HH:MM:SS",
//'[' används för att tala om att det är ett möjligt datum, valfri data
//kan placeras direkt efter datumet utan konflikt.
//Post: vektorn 'table' har laddats med ett FileTable-objekt för varje fil
//i katalogen.
void loadFileTable(String* path);
```

```
    //Returnerar Filtabellens storlek.
    //Pre: loadFileTable har tidigare anropats
    //Post: Filtabellens storlek har returnerats
    int getFileTableLength();

    //Hämtar ut startdatumet på angiven position 'pos' i Filtabellen.
    //Pre: loadFileTable har tidigare anropats
    //Post: startdatumet på angiven position har returnerats
    String* getFileTableItemFirstDate(int pos);

    //Hämtar ut slutdatumet på angiven position 'pos' i Filtabellen.
    //Pre: loadFileTable har tidigare anropats
    //Post: slutdatumet på angiven position har returnerats
    String* getFileTableItemLastDate(int pos);

    //Hämtar ut filsökvägen på angiven position 'pos' i Filtabellen.
    //Pre: loadFileTable har tidigare anropats
    //Post: filsökvägen på angiven position har returnerats
    String* getFileTableItemPath(int pos);

    //Sorterar filtabellens innehåll i datum-ordning, med tidigaste datumet
    //först
    //Pre: loadFileTable måste ha anropats
    //Post: filtabellen har sorterats.
    void sortFileTable();

private:
    //Hämtar första datumet ur angiven fil 'path'.
    //Pre:Minst ett datum finnsi filen, formaterat enligt " [YYYY-MM-DD
    //HH:MM:SS",
    //'[' används för att tala om att det är ett möjligt datum, valfri data
    //kan placeras direkt efter datumet utan konflikt.
    //Post: OM sökvägen existerade så har filens första datum returnerats som
    //en sträng
    //ANNARS har felmeddelande skrivits till skärmen och tomsträng ("")
    //returnerats.
    String* getFirstDateTime(String* path);
```

```
        //Hämtar sista datumet ur angiven fil 'path'.
        //Pre: Minst ett datum finns i filen, formaterat enligt "[YYYY-MM-DD
        //HH:MM:SS",
        //'[' används för att tala om att det är ett möjligt datum, valfri data
        //kan placeras direkt efter datumet utan konflikt.
        //Post: OM sökvägen existerade så har filens sista datum returnerats som
        //en sträng
        //ANNARS har felmeddelande skrivits till skärmen och tomsträng ("")
        //returnerats.
        String* getLastDateTime(String* path);

        String* filenames[];
        String* libraries[];
        FileTable* table __gc[];
    };
}
```

C.3 C++ Klassen StringCut

C.3.1 StringCut.h

```
/*Constants*/
const int NOT_FOUND=-1;
const int DATE_TIME_LENGTH=19;

namespace StringCutDll
{
    public __gc class StringCut
    {
    public:
        StringCut();

        //Klipper ut en delsträng ur 'str' klippningen sker från 'first' tom
        //'last'   Exempel:str="path=C:\;"   first="path="   last=";"   så   blir
        //delsträngen="C:\"
        //Pre:true
        //Post: OM start('first') och sluttecken('last') fanns i 'str' så har dess
        //substräng returnerats ANNARS har "" returnerats
        String* findSubString(String* str, String* first, String* last);
```

```
        //Tar emot en sträng('result') och söker upp första förekomsten av ett
        //datum som är större än eller lika med 'startDateTime' när detta har
        //hittas har allt ovan detta datum klippts bort. Därefter görs samma
        //förfarande med 'endDateTime' i den nu förminskade strängen när
        //'endDateTime' också har hittats sker en klippning
        //från 'startDateTime' till 'endDateTime' och detta är det slutgiltiga
        //resultatet.
        //Pre: 'startDateTime' <= 'endDateTime'
        //Post: OM 'startDateTime' och 'endDateTime' har hittats så har strängen
        //som fanns från 'startDateTime' till 'endDateTime' returnerasts. ANNARS
        //OM 'startDateTime' har hittats men inte 'endDateTime' så har allt från
        //'startDateTime' tom slutet av strängen returnerats. ANNARS OM
        //'startDateTime' inte hittats så har "ERROR!" returnerats.
        String* findResultString(String* result, String* startDateTime, String*
        endDateTime); //Linjärsökning


        //Söker upp start- och slutposition i angiven sträng (result) och plockar
        //ut en delsträng som ligger mellan positionerna. Start- och slutposition
        //söks upp genom en algorithm som fungerar ungefär som en binärsökning.
        //Pre: 'startDateTime' <= 'endDateTime'
        //Post: OM 'startDateTime' och 'endDateTime' har hittats så har strängen
        //som fanns från 'startDateTime' till 'endDateTime' returnerasts. ANNARS
        //OM 'startDateTime' har hittats men inte 'endDateTime' så har allt från
        //'startDateTime' tom slutet av strängen returnerats. ANNARS OM
        //'startDateTime' inte hittas har allt från börja av sträng t.o.m.
        //'endDateTime' returnerats.
        String* findResultString2(String*result, String*startDateTime, String*
        endDateTime, String* lowestTime, String* highestTime); //Binärsökning

    private:
        //Söker upp en position i en sträng för ett datum.(binärsökning kombinerat
        //med linjärsökning)
        //Pre: Datumet måste vara formaterat enligt "[YYYY-MM-DD HH:MM:SS"
        //Post:En position där datumet befinner sig i strängen, har returnerats.
        int findPosition(String* result,String* dateTime);
    };
}
```