
Fakulteten för ekonomi, kommunikation och IT
Datavetenskap

William Hemmingsson
Emil Vieweg

# Hantering och presentation av mätdata i en webbapplikation

## Handling and presentation of measurement data in a web application

Examensarbete 15 hp
Dataingenjörsprogrammet

Datum/Termin: 2012-06-07
Handledare: Stefan Alfredsson
Examinator: Donald F. Ross
Ev. löpnummer: C2012:10

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

# Hantering och presentation av mätdata i en webbapplikation

**William Hemmingsson, Emil Vieweg**

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

William Hemmingsson

Emil Vieweg

Godkänd, 2012-06-07

Handledare: Stefan Alfredsson

Examinator: Donald F. Ross

# Sammanfattning

Mätning av exempelvis energiförbrukning och temperatur förekommer i vissa industriella system. Mätning sker med hjälp av en eller flera sensorer. Data samlas in, bearbetas, lagras för att slutligen presenteras. Teknik och konsultföretaget Altran i Karlstad har utvecklat ett sådant system. Mätdata presenteras i en Android-app.

Denna uppsats beskriver ett projekt med syftet att utveckla en plattformsoberoende webbapplikation för presentation av mätdata. Webbapplikationer är inte bundna till ett specifikt operativsystem, utan visas i en webbläsare. En annan viktig del av projektet var dessutom att utveckla ett system som på ett generellt sätt hanterar insamlad mätdata. Två metoder för generering av grafer har jämförts i ett flertal prestandatester. En metod är generering av grafer på servern, som skickas som bild till klienten. Den andra metoden är att skicka mätdata till klienten och där generera grafen.

Projektet har resulterat i en webbapplikation med funktionalitet mycket lik den Android-app som tidigare utvecklats av Altran. Ett komplett system för insamling, hantering och lagring av mätdata har också utvecklats.

Resultaten från prestandatesterna visar att ingen av metoderna är överlägsen i alla situationer. Vilken metod som är optimal beror mycket på prestanda hos klient och server, samt bandbredd.

# Handling and presentation of measurement data in a web application

Measuring of energy consumption and temperature occurs in certain industrial systems. The measuring is performed with one or several sensors. Data is collected, processed and stored and later presented. A technical consulting company, Altran, in Karlstad, have developed such a system. The measurement data is presented in an Android app.

This dissertation describes a project with the purpose of developing a plattform independent web application for presenting measurement data. Web applications are not bound to a specific operating system but are instead shown in a web browser. Another important aspect of the project was to develop a system that in a general way handles collected measurement data. Two methods for graph generation have been compared in a number of performance tests. One method is by generating the graph as an image on the server and then sending the image to the browser. The other way is to send the measurment data to the client and then to generate the graph in the browser.

The project has resulted in a web application with functionality very similar to the Android app that was developed by Altran. A complete system for collecting, handling and storing data has also been developed.

The result from the performance tests shows that none of the methods for generating graphs is best in all situations. Which metod that is best depends on the performance of the server and the client and also on the bandwidth.

Vi vill tacka vår uppdragsgivare Altran och främst Johan Lundin för handledning i utvecklingsarbetet. Vi vill också tacka Stefan Alfredsson för handledning vid uppsatsskrivandet. Även tack till GHI Electronics och Kjell & Company för tillståndet att använda deras bilder i denna uppsats.

# Innehåll

# Figurer

# Kapitel 1

# Inledning

*I detta kapitel beskrivs bakgrunden till projektet, dess syfte och vad som ska utföras. Delar ur kravspecifikationen presenteras. Här finns även en översikt av uppsatsens disposition.*

## 1.1 Syfte och översikt

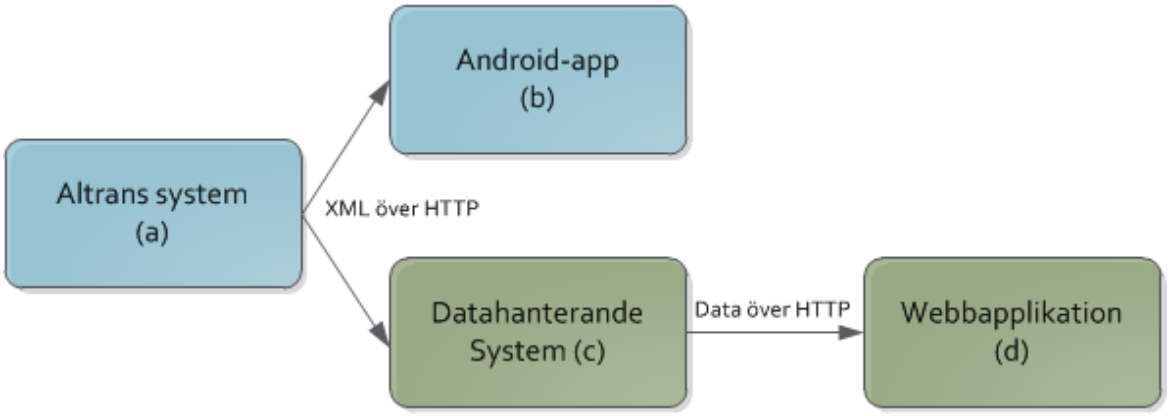
Syftet med projektet är att utveckla en webbapplikation med samma funktionalitet som en existerande Android-applikation som presenterar mätdata i form av grafer och tabeller. Applikationen ska kunna köras på mobila enheter (såsom smartphones och tablets) och gränssnittets design ska anpassa sig efter skärmens storlek. Anledningen till att en webbapplikation ska utvecklas är att den inte är bunden till ett visst operativsystem. Webbapplikationens gränssnitt ska kunna visas i alla moderna webbläsare. Ett underliggande system ska också utvecklas, som på ett generellt sätt samlar in, lagrar och bearbetar mätdata.

Examensarbetet utförs på uppdrag av Altrans karlstadskontor. Altran är ett konsult- och teknikföretag med anställda på många olika platser i världen. En anställd har i sin bostad satt upp flertalet mätsensorer. Dessa mäter temperatur och energiförbrukning. Altran har utvecklat ett system som hämtar mätdata från dessa sensorer och presenterar den i en Android-applikation i form av grafer och tabeller. Figur 1.1.1 illustrerar vad som redan

är implementerat av Altran (a och b) och vad som implementeras i detta projekt (c och d).

Ett av kraven var att implementationen av systemet helt eller delvis skulle bygga på Microsofts teknologier. Dessutom ska webbapplikationens användargränssnitt lätt kunna anpassas för olika företagskunder. Med detta avses till exempel färgschema och logotyp.

Ett annat syfte med projektet var att utvärdera vilken metod för skapandet av grafer som är mest effektiv, med avseende på tid och överförd datamängd. Resultatet från dessa utredningar är till hjälp om applikationens funktionalitet i framtiden ska vidareutvecklas och prestandan optimeras.




Figur 1.1.1: Översiktsbild av projektets delar

## 1.2 Kravspecifikation

Altran skrev en kravspecifikation som tar upp de krav som ställts på implementationen av webbapplikationen och det datainsamlande systemet. I denna sektion listas några av dessa krav, samt krav som tillkommit muntligt under projektets gång. Kraven i denna sektion är ibland skrivna rakt av från kravspecifikationen, ibland omskrivna med egna ord för att bättre passa i sammanhanget.

### 1.2.1 Krav från kravspecifikation

#### Datahanterande system

- Systemet ska begära mätdata från en server, exempelvis över HTTP.
- Systemet ska hantera att mätdata tillhandahålls med XML (se Sektion 2.7)
- Systemet ska hantera att den enhet som tillhandahåller mätdata (server), inte är tillgänglig vid ett tillfälle, det vill säga då enheten inte skickar mätdata eller svarar på begäran.

#### Webbapplikation

- Systemet ska presentera mätdata, grafer eller bilder i en webbapplikation (hemsida)
- Presentationen ska vara anpassningsbar för olika företag (Branding). Detta innebär att presentationen ska innehålla en logotyp, ha visst färgschema och använda vissa typsnitt, som då kan ändras vid installation eller konfiguration av systemet.
- Presentationen ska automatiskt anpassa sig till den enhet som försöker visa presentationen. Detta innebär anpassning för skärmstorlek och förhållande bredd och höjd.

### 1.2.2 Muntliga krav

- Systemet ska vara utvecklat med produkter och tekniker från Microsoft. Detta inkluderar till exempel ASP.NET MVC (se Sektion 3.5.2)
- Webbapplikationens design och funktionalitet ska i stora drag efterlikna den existerande Android-appen. En exakt kopia är inte nödvändig.

## 1.3 Disposition

I kapitel 2, *Det ursprungliga systemet*, beskrivs Altrans ursprungliga system. De olika komponenterna och hur de kommunicerar tas upp.

Alla tekniker, språk och designmönster som är del av implementationen av det nya systemet tas upp i kapitel 3, *Teknikdiskussion*. Alternativ diskuteras också och valen som gjorts motiveras. Denna kunskap är viktig för förståelsen av implementationen av det nya systemet och webbapplikationen.

Beskrivningarna av de utredningar som gjorts görs i kapitel 4, *Utredningar*. Utredningarna syftar till att ge bra riktlinjer för val av grafgenereringsmetod vid en eventuell vidareutveckling av webbapplikationen. Resultaten för utredningarna presenteras här.

I kapitel 5, *Design och Implementation*, beskrivs designen och implementationen av det nya systemet och webbapplikationen ingående.

Resultatet av implementationen presenteras i kapitel 6, *Utvärdering*. Implementationen utvärderas och speciella problem och dess lösningar tas upp.

I kapitel 7, *Projektsammanfattning*, sammanfattas projektet i sin helhet. Förslag på vidareutveckling av systemet presenteras också i detta kapitel.

# Kapitel 2

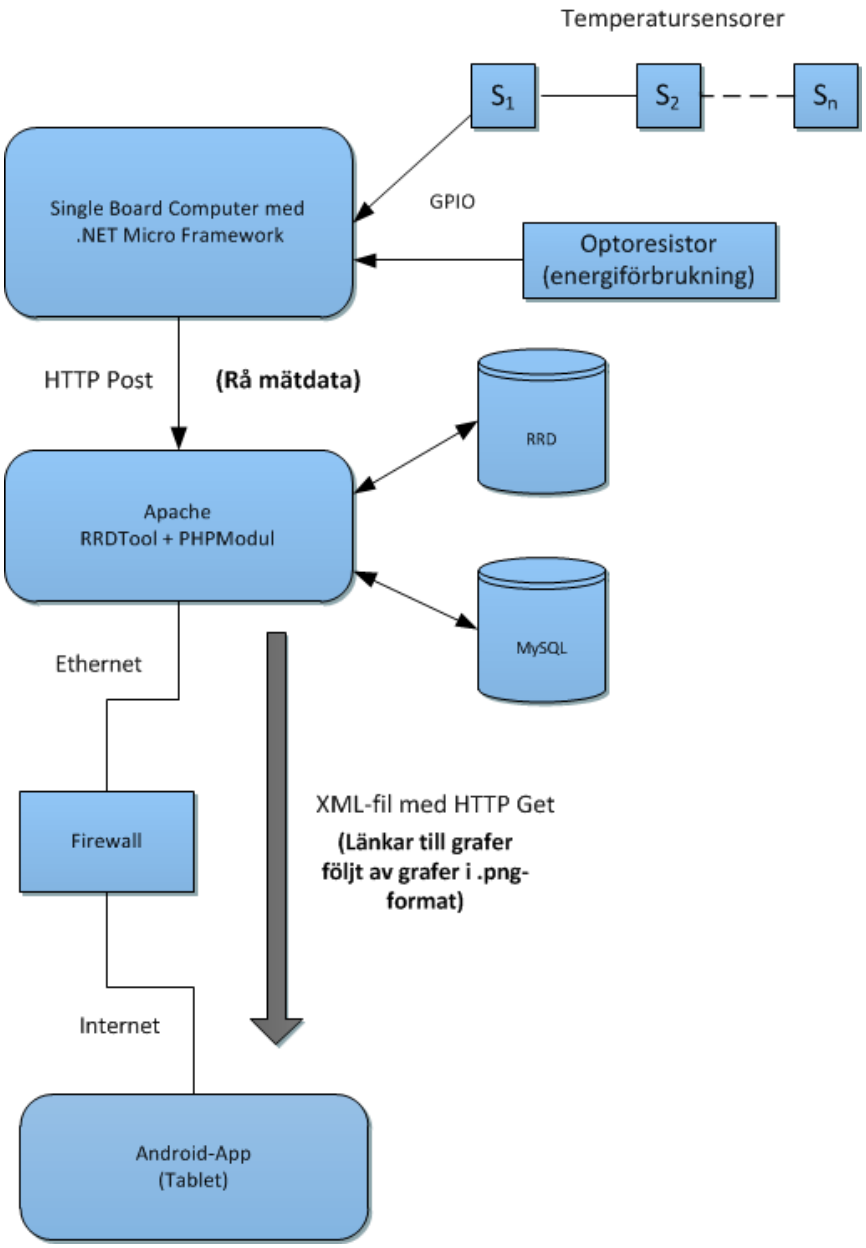
# Det ursprungliga systemet

*I detta kapitel diskuteras det ursprungliga systemet, dess olika komponenter och hur dessa komponenter kommunicerar med varandra. Kapitlet inleds med en övergripande beskrivning. Efter den följer en mer detaljerad beskrivning av varje komponent. Hur dessa komponenter hänger ihop och kommunicerar med varandra beskrivs också. Kapitlet avslutas med en sammanfattning som summerar innehållet.*

## 2.1 Systemöversikt

Figur 2.0.1 visar en övergripande systembild. I en villa finns ett antal temperatursensorer uppsatta ($S_1$ till $S_n$ i figuren). En optoresistor som mäter energiförbrukning finns också. Dessa sensorer är kopplade till en dator som samlar in mätdata. Mätdatan skickas till en webbserver som lagrar den i två olika databaser, RRD och MySQL.

En app utvecklad för operativsystemet Android hämtar en XML-fil från servern med HTTP-Get. När detta sker genererar servern ett antal grafer över temperatur- och energiförbrukningsförändring. I XML-filen finns länkar till dessa grafer samt ögonblicksvärden (senast uppmätta värden från sensorerna). Android-appen hämtar sedan graferna som presenteras för användaren i ett grafiskt gränssnitt.




Figur 2.0.1: Översiktsbild av det ursprungliga systemet

## 2.2 Sensorer och mätdata

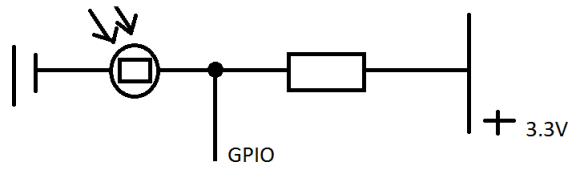
I det ursprungliga systemet finns ett antal temperatursensorer, samt en sensor för avläsning av en elmätare. Temperatursensorerna mäter temperaturen i omgivningen där de är plac-

erade och dessa mätdata avläses av en enkortsdator (Single Board Computer, se Sektion 2.3.1).

Temperatursensorerna är av typ DS18B20 som har en noggrannhet på en halv grad[32]. Senorn använder sig utav 1-Wire-protokollet[31] vilket innebär att temperatursensorn har en 64-bitars identifieringskod som gör det möjligt att koppla flera sensorer till samma GPIO-pinne [2]. Identifieringskoden kan användas för att unikt identifiera en sensor, vilket görs i flera delar av systemet.

Den sensor som mäter elförbrukning kallas optoresistor och sitter placerad över en lysdiod på elmätaren. Sensorn är kopplad som i Figur 2.2.1. Lysdioden blinkar en gång för varje förbrukad wattimme. Enkortsdatorn registrerar varje blinkning. Figur 2.2.2 visar hur en sådan optoresistor kan se ut.

Fortsättningsvis kommer temperaturdata och energiförbrukning att refereras till som mätdata.




Figur 2.2.1: Kopplingschema för optoresistor

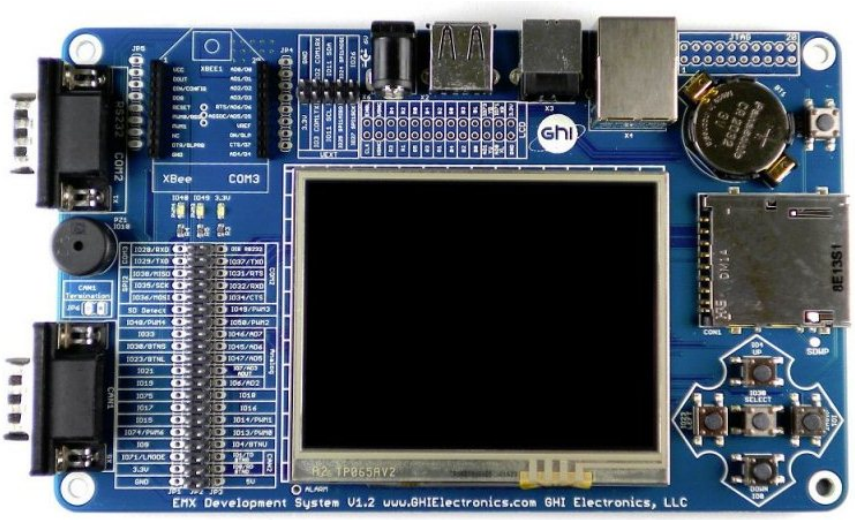
Figur 2.2.2: Optoresistor

## 2.3 Enkortsdator

I det nuvarande systemet används en Single Board Computer, på svenska enkortsdator, för inhämtning av mätdata. En enkortsdator är en dator byggd på ett enskilt kretskort. På detta kretskort inkluderas vanligtvis processor(er), minne samt olika I/O-enheter. Det är möjligt att hämta mätdata till en vanlig persondator, men det finns flera nackdelar med

detta. En persondator är oftast dyrare och är mer energikrävande. Används en enkortsdator kan den nödvändiga mjukvaran redan vara installerad på den, vilket underlättar installation hos användare. En annan skillnad mellan en enkortsdator och en vanlig persondator är att en enkortsdator inte behöver inkludera portar för exempelvis grafikkort och ljudkort.

Figur 2.3.1 visar den enkortsdator som används i det nuvarande systemet. Den är utvecklad av GHI Electronics och kallas EMX Development System [16]. Denna enkortsdator har en LCD-display som visar de senast uppmätta mätvärdena. Den har också nätverksstöd, vilket används för att skicka mätdata till en Apache-server, se Sektion 2.6.1. Dessutom har enkortsdatorn stöd för .NET Micro Framework, som beskrivs i Sektion 2.4.




Figur 2.3.1: Single Board Computer

## 2.4 .NET Micro Framework

.NET Micro Framework (.NET MF), utvecklat av Microsoft, är en .NET-plattform för enheter med begränsade resurser, såsom en enkortsdator [20].

I det ursprungliga systemet har enkortsdatorn som används stöd för .NET MF. Detta gör det enkelt att hantera hämtningen av data. I Figur 2.4.1 är ett exempel på kod som

får en diod att blinka 2,5 gånger per sekund, när man har kortets programmerbara knapp intryckt.

```
//initiera knapp och diod.
 InputPort button = new InputPort(Pins.ONBOARD_SW1, false, ResistorModes.Disabled);
 OutputPort led = new OutputPort(Pins.ONBOARD_LED, false);
 while (true){        //kör hela tiden när enheten är på
     if (!button.Read()){    //knappen returnerar falsk när den trycks in
         led.Write(true);    //tänd diod
         Thread.Sleep(200);    //vänta 200 ms
         led.Write(false);    //släck diod
         Thread.Sleep(200);    //vänta 200 ms
     }else
         led.Write(false);    //släck dioden när knappen inte är intryckt
}
```

Figur 2.4.1: .NET-MF exempel

Programmet, skrivet i C#.NET (se Sektion 3.3.1), som exekverar på systemets enkortsdator har i uppgift att hämta mätdata. Detta görs i en evighetsslinga, med ett intervall på fem sekunder.

För att hämta data från temperatursensorerna används ett 1-Wire-bibliotek [1] [15] till .NET MF från GHI Electronics. Först skapas en instans av OneWire som tar som argument den GPIO-pinne [2] till vilken enheten är kopplad. Efter det körs metoden `reset` som startar sökandet efter enheter på bussen[2]. Hittas någon enhet så returnerar `reset` sant annars falskt. Returnerar den sant så är den kopplad till den första enheten på bussen(identifieringskoden används för att bestämma ordning) och man kan börja sända data direkt mellan enheten och enkortsdatorn. Vill man göra en temperaturmätning på en enhet så skriver man 0x44 till enheten och sen läser man från enheten tills enheten returnerar 0. Efter det måste man köra `reset` igen och be att få läsa temperaturdata igenom att skriva 0xBE och sen läsa från enheten. Hur temperaturdata hämtas visas i Figur 2.4.2. Vill man använda flera enheter på samma buss kan man iterativt söka efter återstående

[1] `http://www.tinyclr.com/support`
[2] `http://en.wikipedia.org/wiki/Bus_(computing)`

```
temperature = ow.ReadByte(); // Läs Least Significant Byte
temperature |= (ushort)(ow.ReadByte() << 8); // Läs Most Significant Byte
Debug.Print("Temperature: " + temperature / 16); //skriv ut temperaturen
```

Figur 2.4.2: Temperaturdata hämtas från DS18B20

enheter med metoden `Search_GetNextDevice`.

Mätdata skickas från enheten till servern med en Query-sträng [45]. En adress som innehåller en Query-sträng har formatet

`http://adress.com/sökväg?fält=värde&fält2=värde.`

Tecknet ? används för att börja Query-strängen och & används för att separera fält-värde par. Vill man skicka en lista som har flera värden kopplade till samma fält så gör man det genom att använda samma fält-namn på olika fält-värdepar.

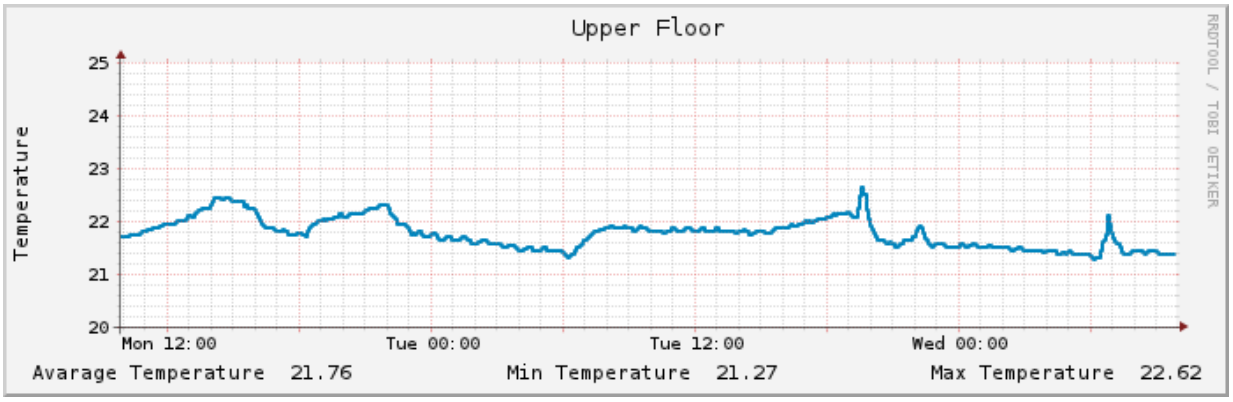
```
http://example.com?Devices=[2844C959030000C9,...]&Values=[20.5625,...]&
Load=3.271&Consumption=4
```

## 2.5 RRDtool och Grafer

All mätdata i det ursprungliga systemet lagras både i en MySQL-databas och i en Round-Robin-Databas (RRD). För att hantera mätdata i RRD samt generera grafer används ett verktyg kallat RRDtool [4]. Datalagring i en sådan databas bygger på Round Robin. Kortfattat bygger denna algoritm på en cirkulär lista med en viss storlek. I en RRD så kan man ha flera RRA (Round Robin Archive). En RRA innehåller en viss mängd mätpunkter med ett viss tidsintervall. Ett exempel är att första RRAn lagrar ett mätvärde var 10:e sekund och har 360 mätvärden vilket blir en timme. Sen väljer man att nästa RRA ska ha en sjättedel av upplösningen av den första RRA och 1440 mätpunkter vilket innebär en mätpunkt per minut i ett dygn. Då är den maximala tiden innan mätdata försvinner ett dygn och en timme och när man skapar databasen har den sin maximala storlek. I en timme så sparas alla mätvärden i första RRA och efter det så sparas ett värde i den

andra RRAn som är medelvärdet av de 6 sista mätningarna i första RRAn. Sen skrivs det äldsta värdet över med det nyaste värdet i första RRAn. Samma sak händer med den andra RRAn när den är full men då försvinner helt enkelt mätvärdena.

RRDtool används för att hantera en RRD, för att lagra mätdata. Det används också i det ursprungliga systemet för att generera grafer över temperaturförändringar och energiförbrukning. Dessa grafer visas sedan upp i en Android-applikation, se Sektion 2.8. Figur 2.5.1 är ett exempel på en sådan graf. Här plottas temperatur (Y-axeln) mot tiden (X-axeln) för övervåningen i hemmet. Bilden visar också medeltemperaturen, den lägsta uppmätta temperaturen samt den högsta uppmätta temperaturen i den aktuella grafen. Dessa värden är beräknade av RRDtool i samband med att grafen genereras.




Figur 2.5.1: Temperaturgraf

## 2.6 Server (LAMP)

LAMP är en förkortning för Linux, Apache, MySQL och PHP. Det är en möjlig konfiguration för en webbserver. Det ursprungliga systemet har en sådan konfiguration på en Linux-dator och i denna sektion diskuteras de olika komponenterna.

### 2.6.1 Apache

Apache är i skrivande stund den mest använda webbservern [37] och används av cirka 2/3 av de en miljon mest trafikerade webbplatserna. Apache är fri mjukvara som utan kostnad går att installera på operativsystemen Linux, Macintosh, Windows med flera. Apache-servern som är installerad på servern är konfigurerad för att kunna köra PHP-skript.

### 2.6.2 PHP

PHP [29] är ett serverskriptspråk som ursprungligen skapades för att göra dynamiska webbsidor . PHP kan integreras i HTML-kod (mer om HTML i Sektion 3.7.1) och exekveras genom en webbserver, som till exempel Apache. Resultatet är en HTML-sida som visas i webbläsaren.

PHP används i det ursprungliga systemet för att sköta kontakten med databaserna där mätdata lagras samt för att generera den XML-fil (se Sektion 2.7) som innehåller mätdata samt länkar till de grafer som RRDtool genererar.

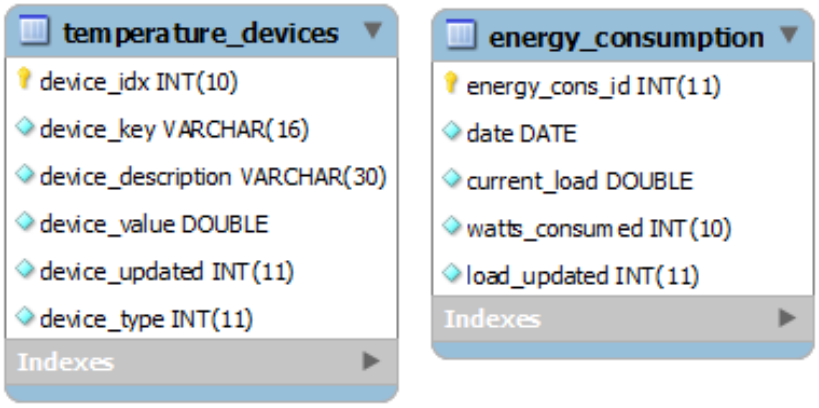
Ett PHP-skript som körs på Apache-servern tar emot data med HTTP-Post[12]. Detta meddelande skickades från enkortsdatorn och innehåller all mätdata. PHP-skriptet extraherar data och itererar över varje unikt id, det vill säga varje sensors unika identifieringskod. MySQL-databasen antingen uppdateras med nya värden eller utökas om nya sensorer tillkommit. Det samma sker för RRD; antingen skapas en ny RRD för den aktuella sensorn, eller så läggs det nya mätvärdet till i den ursprungliga RRDn.

Ett annat PHP-skript körs då Androidappen (se Sektion 2.8) meddelar skriptet med HTTP-Get[12]. Skriptet hämtar data från databaserna, och använder det för att generera XML-dokumentet, som skickas tillbaka till Android-appen. Skriptet genererar också graferna som Android-appen kan hämta med hjälp av HTTP-Get.

### 2.6.3 MySQL

MySQL [10] är en populär databashanterare som finns tillgängligt för flertalet plattformar och används av flera stora företag såsom Wikipedia, Google och Facebook. Den är även öppen källkod och får användas utan licenskostnad.

Det ursprungliga systemet använder sig av en MySQL-databas installerat på servern som sparar viss mätdata. Figur 2.6.1 visar databasens tabeller.




Figur 2.6.1: Databasschema

I tabellen `temperature_devices` som innehåller data om temperatursensorerna som omnämns i Sektion 2.2 finns följande attribut:

#### temperature_devices

- `device_idx` är primärnyckeln
- `device_key` är 64-bitars id från sensorn
- `device_description` är en kort beskrivning vart sensorn sitter till exempel så finns *outside* och *basement1*
- `device_value` är det sista uppdaterade värdet på sensorn vilket är temperatur mätt i grader Celsius

- `device_updated` är när mätvärdena från sensorn senast samlades in och är sparad i POSIX-tid[3]. POSIX-tid är definierad som antal sekunder från första januari 1970 klockan 00:00.
- `device_type` användes aldrig i systemet.

Tabellen `energy_consumption` innehåller data från optoresistorn. En ny rad läggs in i tabellen efter midnatt som innehåller förbrukningsdata för det nya dygnet.

**energy_consumption**

- `enery_cons_id` är en automatisk ökande primärnyckel
- `date` är dagen då data samlades in
- `current_load` är nuvarande förbrukning
- `watts_consumed` sparar hur mycket energi som har förbrukats hittills samma dag och slutar uppdateras nästa dag. Det gör att det är möjligt att få historik över totalt använd mängd energi som används per dag.
- `load_updated` är senaste uppdateringen sparad på samma sätt som i `device_updated` i `temperature_devices`.

## 2.7 XML

Extensible Markup Language (XML) [28] är ett utbyggbart märkspråk som är designat att vara läsbart av både människor och maskiner. Syftet med XML är att kunna utbyta information mellan olika typer av system, som ett gemensamt gränssnitt.

XML används i det ursprungliga systemet i detta syfte. Ett XML-dokument genereras av ett PHP-skript som hämtar data från RRD och MySQL-databasen. Ett utdrag från detta XML-dokument kan ses i Figur 2.7.1.

Detta XML-dokument är centralt inte bara för det ursprungliga systemet, utan även för vidareutvecklingen av systemet. I det ursprungliga systemet hämtas detta XML-dokument

```
<device>
        <device_key>28C0BC5903000040</device_key>
        <device_value>22.6875</device_value>
        <device_description>Upper Floor</device_description>
        <device_updated>1329135870</device_updated>
        <device_img>/rrdtool_img/28C0BC5903000040.png</device_img>
</device>
```

Figur 2.7.1: Utdrag från XML-dokument innehållande temperaturdata

av Android-appen. XML-dokumentet hämtas av klienten genom HTTP-Get. Servern fångar upp denna begäran och exekverar PHP-koden som skapar XML-dokumentet. Servern svarar med att skicka dokumentet.

XML-dokumentet innehåller flera viktiga uppgifter. Det Figur 2.7.1 illustrerar är som nämnt ett utdrag från detta XML-dokument. Taggen `<device>` visar att informationen som följer är relaterat till en enhet, i detta fall en temperatursensor. XML-dokumentet innehåller totalt åtta liknande sektioner, en för varje temperatursensor.

För varje sensor (eller device) ges först och främst sensorns unika värde (`<device_key>`). Varje sensor som produceras får ett unikt värde. Det som är centralt i det ursprungliga systemet är de grafer som genereras på servern. Sökvägen till en sådan graf hittas efter `<device_img>` i XML-dokumentet. Denna sökväg används sedan av klienten för att kunna hitta och ladda ned graferna.

Övrig information om sensorn är dess placering i hemmet samt när det senaste värdet hämtades. Tiden är angiven i POSIX-tid. Slutligen anges också det senast uppmätta värdet, vilket visas i klienten tillsammans med grafen.

Figur 2.7.2 visar ett annat utdrag från samma XML-dokument. Denna del syftar till att visa information relaterad till energiförbrukning. Precis som för temperatursensorerna kan man utläsa en sökväg till en bild, den aktuella energiförbrukningen (angiven i kW) samt när senaste mätningen gjordes.

Utöver ovan angiven information kan man också utläsa den totala mängden förbrukad energi (angivet i Wh) för ett visst datum (i detta fall 2012-02-13) samt medeltemperaturen

```
<electricityconsumption>
        <consumption_graph_12h>/rrdtool_img/load.png</consumption_graph_12h>
        <consumption_now>1.268</consumption_now>
        <consumption_updated>1329137165</consumption_updated>
        <consumption_per_day>
                <consumption_day>
                        <consumption_date>2012-02-13</consumption_date>
                        <consumption_value>37227</consumption_value>
                        <outside_average>-4.0133536769898</outside_average>
                </consumption_day>
```

Figur 2.7.2: Utdrag från XML-dokument innehållande förbrukningsdata

utomhus för det angivna dygnet. I XML-dokumentet följer fler sektioner av taggar för passerade datum, för att ge en historisk bild av energiförbrukningen. Dessa värden visas upp i en tabell i Android-appen (se Sektion 2.8).

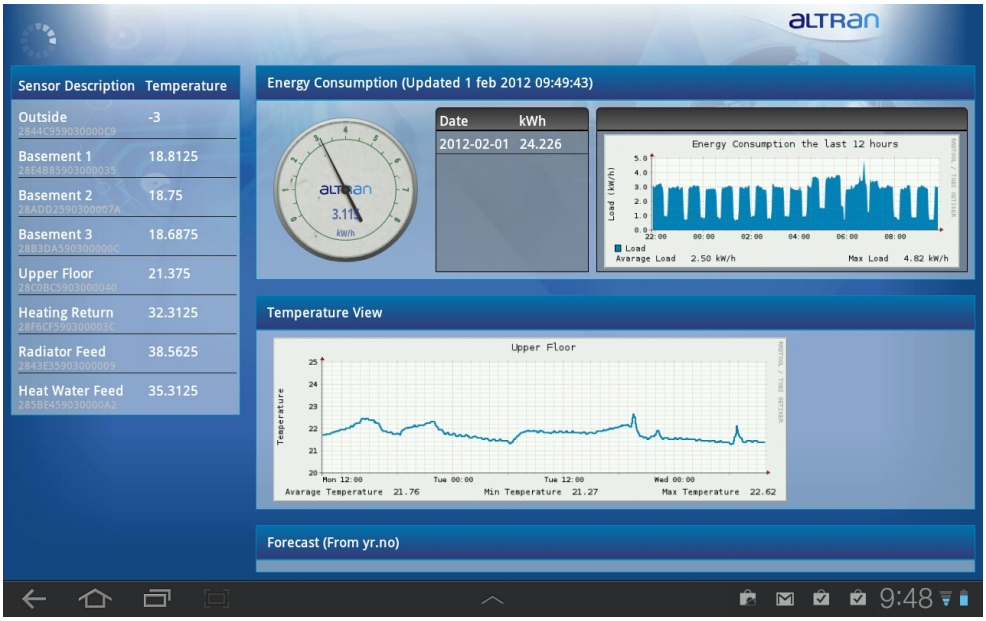
## 2.8 Android-app

Android är ett operativsystem för mobila enheter, såsom tablets och smartphones. Android utvecklades ursprungligen av Android Inc, som köptes upp av Google år 2005 [5]. Android utvecklas av Open Handset Alliance, som leds av Google, och bygger på Linuxkärnan.

För att utveckla program (så kallade appar) till Android kan en variant av Java användas. Detta är fallet i det ursprungliga systemet. Altran har skrivit en app för en tablet (Samsung Galaxy Tab [42]) som presenterar mätdata. Figur 2.8.1 visar appens användargränssnitt.

Appen visar energiförbrukningen (övre graf), nuvarande effekt (mätaren) samt temperaturgraf för övervåningen i hemmet. Till vänster finns en meny med vilken man kan visa de övriga graferna genom att klicka på den motsvarande sensorns beskrivning. Det finns i nuläget åtta grafer som visar temperatur, och en graf som visar energiförbrukningen.

Som nämnts tidigare i detta kapitel genereras graferna på Apache-servern. Dessa genereras när appen begär det, vilket sker kontinuerligt. Appen hämtar det XML-dokument om beskrevs tidigare. Apache-servern genererar då även graferna med hjälp av RRDtool. Ap-




Figur 2.8.1: Grafiskt användargränssnitt för Android-app

pen plockar ut värden och sökvägar till graferna från XML-dokumentet. När sökvägen identifierats kan appen hämta grafen med HTTP-Get, i form av en PNG-bild, laddas ned, sparas i det fysiska minnet och sedan visas.

## 2.9 Sammanfattning

I detta kapitel har det system som utvecklats av Altran beskrivits. De olika komponenter som ingår, både mjukvara samt hårdvara, och de gränssnitt som existerar mellan dem har beskrivits.

Mätdata genereras av ett flertal olika sensorer placerade i en villa. Dessa mätdata skickas till en enkel form av dator, en SBC. På denna dator exekverar C#-kod, i .NET Micro Framework. Programmet hämtar data från sensorerna och skickar dem till en Apache-server. Denna server lagrar mätdata dels i en RRD-databas samt i en MySQL-databas.

En klient, i form av en Android-app, visar upp mätdata i form av grafer. Appen begär ett XML-dokument som genereras på Apache-servern. Servern genererar också grafer av

mätdata med hjälp av RRDtool. När Android-appen mottagit XML-dokumentet hämtar den ut mätvärden samt länkar till graferna som genererats. Därefter laddas bilderna ned och visas i appen.

# Kapitel 3

# Teknikdiskussion

*I detta kapitel diskuteras de teknologier och komponenter som används för implementationen av det nya systemet och webbapplikationen. Alternativ till dessa diskuteras även och de val som gjorts motiveras.*

## 3.1 Datainsamlingsenheter (enkortsdatorer)

Insamling av data görs, precis som i det ursprungliga systemet, av en enkortsdator som är ansluten till systemet på ett mer direkt sätt än den som används i det ursprungliga systemet. Syftet med att ha en ytterligare insamlingsenhet är att låta nya systemet bli mer oberoende av det ursprungliga systemet. Det är också ett en uppvisning av att nya systemet kan hämta data från flera olika källor. I denna sektion diskuteras några olika enkortsdatorer: Netduino, Arduino samt Raspberry Pi. Alla tre varianter kan användas för att samla in data och i denna sektion diskuteras deras för- och nackdelar.

### 3.1.1 Netduino

Netduino[7] är en enkortsdator som har stöd för .NET MF. Det finns flera olika typer av Netduino, varav en är Netduino Plus. För projektet har Altran köpt två stycken av

dessa, i syftet att utöka systemet med fler insamlingsenheter. Det nya systemet tar emot mätvärden inte bara från XML-fil genererad i det urprungliga systemet, utan även från fil genererad på Netduino.

Netduino Plus är den enkortsdator som, utöver den som indirekt används i det urprungliga systemet, hämtar mätdata i form av luftfuktighet och lufttryck från två sensorer. Enkortsdatorn inkluderar tio GPIO-pinnar, två analoga och åtta digitala. Enkortsdatorn har nätverksstöd, vilket är praktiskt om man vill skicka data från den.

Anledningen till att Altran köpte in just denna modell är att den är förhållandevis billig. Den har också, som tidigare nämnts, stöd för .NET MF. De två alternativen till Netduino (Arduino, Sektion 3.1.2 samt Raspberry Pi, Sektion 3.1.3) har inte stöd för .NET MF.

För att testa nätverksstödet och den programmerbara lysdioden implementerades ett enkelt program som konverterar text till Morse-kod. Texten matas in via en Telnet-klient [39] och skickas till enkortsdatorn via TCP. Texten tas emot och varje tecken konverteras till blinkningar.

### 3.1.2 Arduino

Arduino är en serie enkortsdatorer som alla använder en Atmel-processor[33]. En av tillverkarna av Arduino är Smart Projects[1]. Språket som används för att programmera för en Arduino är C++. Utvecklingsmiljön härstammar från Processing [40]. Utveckling av program för Arduino utförs genom att man deklarerar två metoder: `setup()` som körs en gång när kortet startas och `loop()` som körs tills kortet startas om. I Figur 3.1.1 är ett program som får en programmerbar lysdiod på enkortsdatorn att blinka.

### 3.1.3 Raspberry Pi Computer

Figur 3.1.2 visar en enkortsdator, Raspberry Pi, som designades år 2006 av Eben Upton som är anställd på Broadcom som designer av systemchip [14]. Han sade i en intervju: [23]

[1] `http://smartprj.com/catalog/index.php`

> A group of us in Cambridge became concerned that the quality and quantity of applicants to the Computer Science course at the University here was falling year on year, to the point that it was becoming hard to fill the course with good candidates. This in turn has had a knock-on effect on the ability of firms in the Cambridge area to recruit graduates.
>
> There are likely several causes for the decline in applicant numbers, but one factor is the lack of a cheap programmable home computer of the sort that I grew up with in the 1980s. We formed the Raspberry Pi Foundation to develop such a machine.

Vad Upton ville skapa var en enkel, billig dator som skulle starta upp till en Python-tolk (PI står för Python Interpreter). Resultatet blev Raspberry Pi, som på den tiden var en simpel enkortsdator som hade en CPU på 22.1 MHz och 512 KB SRAM [30]

Den kanske största nackdelen med att använda en Raspberry Pi i systemet är att den saknar stöd för .NET MF. Det kommer fortfarande att gå att hämta data, men detta kommer kräva ytterligare utveckling av datainläsning anpassad för just denna plattform. Ett annat problem är att produkten i skrivande stund är svårtillgänglig och bara säljs i

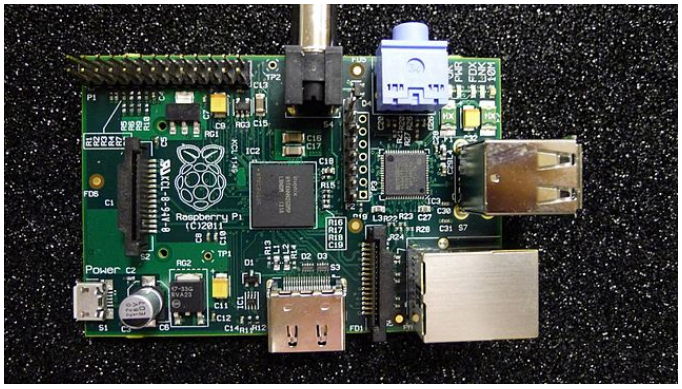
```
#define LED_PIN 13

void setup () {
    pinMode (LED_PIN, OUTPUT);     // Aktivera lysdioden för utsignal.
}

void loop () {
    digitalWrite (LED_PIN, HIGH);  // Sätt på lysdioden
    delay (1000);                  // Vänta en sekund
    digitalWrite (LED_PIN, LOW);   // Stäng av lysdioden
    delay (1000);                  // Vänta en sekund
}
```

Figur 3.1.1: Exempel på Arduino-kod

omgångar.



Figur 3.1.2: Raspberry Pi

## 3.2 Dataformat

Data som skickas till systemet kan struktureras på olika sätt. Struktureringen ger ökad läsbarhet för människor, vilket är bra under utveckling av systemet. I denna sektion diskuteras några olika dataformat som används för dataöverföring. Data skickas antingen till det nya systemet, det vill säga nya mätvärden, eller mellan den datainsamlande modulen och webbapplikation. Alternativet till att använda ett dataöverföringsformat är att låta data skickas som binär data.

### 3.2.1 XML

Mätdata anländer till systemet i form av ett XML-dokument. XML-dokumentets struktur är lik det som används i det ursprungliga systemet, med viss modifikation (se Sektion 5.2.1). XML har valts av tre skäl:

1. Det är det enklaste sättet att bygga vidare på det XML-dokument som redan genereras på servern i det ursprungliga systemet. Detta har inte ställt svåra krav på pro-

duktägaren, som måste tillhandahålla filen med mätdata.

2. Läsbarheten. XML anser vi vara mer läsbart än alternativet JSON och mer strukturerat än rå text.

3. Bakåtkompatibilitet. Äldre program kan fortfarande använda XML-dokumentet även om det utökas med fler mätvärden eller sensorer.

I Figur 3.2.1 följer ett utdrag från ett XML-dokument.

### 3.2.2 JSON

JSON [24], som är en förkortning för JavaScript Object Notation, är likt XML ett språk designat för att att vara läsbart både av datorer och människor. JSON stöds direkt i JavaScript (se Sektion 3.5.5) och lämpar sig då mycket bra för applikationer som använder sig av detta. Det finns två huvudstrukturer i JSON: objekt och listor. Ett objekt är ett eller flera ”attribut-värdepar“. Flera sådana objekt bildar en lista, eller en array. Datatyper för värden som stöds är strängar, heltal och flyttal. Objekt kan också nästlas i andra objekt.

I Figur 3.2.2 följer ett exempel på hur en JSON-fil kan se ut. Innehållet är konvertering av den XML-kod som presenterades i föregående sektion.

Koden illustrerar två olika enheter (Devices) i en array. Varje enhet har tre egenskaper. Syntaxen är mer kompakt än XML. Syntaxen är en delmängd av JavaScript-syntax och skapandet av objekt i JavaScript sker på samma sätt i JSON. Det är enkelt att i JavaScript omvandla en JSON-fil till ett JavaScript-objekt på grund av detta. Detta görs på följande vis:

```
JSON_object = JSON.parse(json_file);
```

Det finns två användningsområden för JSON i det nya systemet. Det första är som alternativ till XML för att skicka data från insamlingsenhet till systemet. Det andra är att skicka data från webbserver till webbläsare. Det andra användningsområdet är bara

```
<device>
      <key>2844C959030000C9</key>
      <value>10.5620</value>
      <updated>1333097452</updated>
</device>
<device>
      <key>28E4B85903000035</key>
      <value>21.5</value>
      <updated>1333097452</updated>
</device>
```

Figur 3.2.1: Lista med enheter i XML-format

```
{
        "devices" :
        [
        {
            "key" : "2844C959030000C9",
            "value": 10.5620,
            "updated" : "1333097452"
        },
        {
            "key" : "28E4B85903000035",
            "value": 21.5,
            "updated" : "1333097452"
        }
        ]
}
```

Figur 3.2.2: Lista med enheter i JSON-format

relevant om grafer ska genereras i webbläsaren, eller om man vill visa större mängder data. Generering av grafer med JavaScript i webbläsaren diskuteras i Sektion 4.2.

### 3.2.3 Rå text

Ett annat alternativ till XML och JSON är att skicka data som rå text. Ett exempel på hur data kan struktureras är att används sig av kommaseparerade värden (CSV, Comma Separated Values). Data formateras som i en tabell, där kolumnnamnen placeras överst i filen. Detta gör det möjligt att identifiera vad värdena representerar. Ett exempel på hur en sådan fil kan ses i Figur 3.2.3. Detta är samma exempel som i avsnittet om JSON och XML, fast i CSV-format.

```
key,value,updated
"2844C959030000C9",10.5620,"1333097452"
"28E4B85903000035",21.5,"1333097452"
```

Figur 3.2.3: Lista med enheter i CVS-format

## 3.3 Serverspråk

Det finns flera möjligheter och alternativ för utveckling av en serverapplikation. Systemet består av två olika serverapplikationer. Den ena är del av webbapplikationen, i form av kod som exekverar på en webbserver. Den andra är den modul som hämtar mätdata och lagrar i databaser. I denna sektion diskuteras främst C#.NET som använts för att utveckla de två olika applikationerna. Även möjliga alternativ tas upp.

### 3.3.1 C#.NET

C# [11], utvecklat av Microsoft, är ett objektorienterat programspråk som används för programmering i .NET-miljön. Syntaxen i C# är väldigt lik Javasyntax.

C#-kod som kompileras blir inte till direkt exekverbar kod, utan ett mellanliggande språk kallat MSIL (MicroSoft Intermediate Language). När programmet sedan ska exekveras, är det MSIL som kompileras till binärkod av den så kallade Just In Time-kompilatorn när behovet finns.

Vi valde att utveckla stora delar av vårt system i C#. Det är ett språk som vi känner oss hemma i och som är både populärt och växande på marknaden [46]. Tack vare .NET-ramverket, kommer språket med många redan färdiga lösningar. I och med ASP.NET (se Sektion 3.5.2), som kan användas i C#, finns bra möjligheter för webbutveckling. Slutligen var det ett krav från Altran att systemet skulle utvecklas för Windows med någon eller några Microsoftprodukter.

### 3.3.2 Alternativ till C#.NET

Det finns flera alternativ till C#. Ett möjligt alternativ är att utveckla den del av systemet som samlar in data och lagrar det i databaserna i C++. För att implementera webbapplikationen skulle till exempel PHP kunna användas. Det finns skäl till att vi inte valde dessa två språk. Det första är att vi tvingas sköta två separata lösningar, en för den datainsam-

lande modulen och en för webbapplikationen. Med C# slipper vi detta - de olika projekten blir delar i samma lösning. Komponenter delas mellan de två projekten, vilket blir svårare att hantera med två skilda språk.

Ett annat alternativ är att utveckla alla komponenter i PHP. Anledningen till att vi inte valde detta är dels att det existerande systemet redan bygger mycket på PHP. Ett krav var även att systemet skulle använda sig av Microsofts produkter. Den andra anledningen är att vi inte har större erfarenhet av PHP, och särskilt inte i de olika ramverk som existerar.

## 3.4 Datalagring

När mätdata tagits emot av systemet behöver den lagras, för att senare kunna presenteras i ett tidsperspektiv. För att lagra data kan relationsdatabaser användas. Data kan också lagras av mer specialiserade datalagringssystem, till exempel om man vill lagra stora mängder data i en längre tidsperiod. I följande sektion diskuteras tre typer av relationsdatabaser: MS SQL Server, MySQL och SQLite. MySQL beskrevs tidigare i Sektion 2.6.3. RRDtool som är en mer specialiserad form av datalagringssystem beskrevs tidigare i Sektion 2.5.

### 3.4.1 MS SQL Server

Microsoft SQL Server[36] är Microsofts databashanterare . Frågespråket som används är en dialekt av SQL som kallas Transact-SQL (T-SQL). Ett krav från Altran var att utökningen av systemet skulle arbeta mot Microsoft SQL Server, för att öppna upp för användandet av andra Microsoft-produkter, såsom C#.NET.

### 3.4.2 MySQL

Det skulle vara fullt möjligt att använda MySQL som databashanterare i systemet. En möjlig lösning är att använda en OBDC-drivrutin [38] för att från C#-kod få åtkomst

till databasen. En anledning till att MySQL inte används är som nämnt kravet från Altran. En annan anledning är att en lösning som involverar MySQL inte blir lika enhetlig, då resterande delar av systemet bygger på Microsofts produkter. Alternativt skulle den existerande MySQL-databasen kunna anropas från systemet, men då skulle syftet med generalisering och lagring av mätdata försvinna.

### 3.4.3 SQLite

SQLite[17] är en databashanterare, licencierad under GLP[13]. Till skillnad från MySQL och MS SQL så körs SQLite inte som en separat process utan körs i samma process där den anropas ifrån. Detta ger SQLite väldigt kort åtkomsttid då funktionsanrop används istället för långsammare inter-processanrop.

Databasen sparas som en fil på hårddisken som är plattformsoberoende och möjligheten finns att spara filen i minne vilket ger väldigt snabb åtkomsttid.

Detta gör SQLite väldigt praktiskt för enheter med begränsad prestanda som mobiler och inbyggda system som vill ha SQL-funktionalitet fast där en SQL server skulle vara för krävande.

Nackdelar med SQLite är dålig prestanda på stora datamängder och att skrivningar till databasen endast kan utföras sekventiellt, dock så kan läsningar utföras parallellt.

## 3.5 Webbapplikation

När data väl lagrats ska den sedan presenteras i någon form för användaren. Det var ett av kraven från Altran att en webbapplikation skulle utvecklas som supplement till den existerande Android-applikationen. Med en webbapplikation binder man inte presentationen av mätdata till en viss enhet eller operativsystem.

I denna sektion diskuteras olika tekniker och språk som är del av implementationen av webbapplikationen. Designmönstret MVC diskuteras, samt Microsofts implementation av

detta med avseende på webbapplikationer. JavaScript och AJAX tas också upp, då det är centralt för dynamisk uppdatering av innehållet i webbsidan.

### 3.5.1 Model-View-Controller (MVC)

MVC är ett designmönster som delar upp en applikation i modell, vy och kontroll.

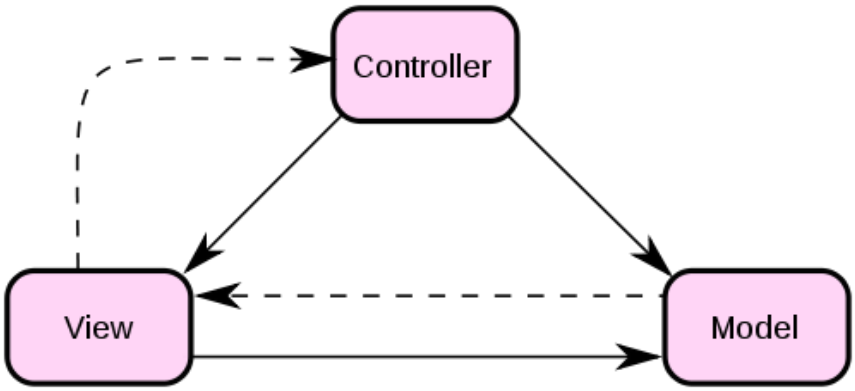
Orsaken till det är att man vill dela upp ett programs gränssnitts-, indata- och affärslogik för att lättare kunna utveckla dessa delar separat.

Flödet i en MVC-applikation brukar se ut så här:

1. Vyn hämtar data från modellen och presenterar datan för användaren.
2. Användaren interagerar på vyn till exempel klickar för att ta bort ett element.
3. Vyn meddelar kontrollen om interaktionen och skickar med relevant data om interaktionen, till exempel vilken element som ska tas bort.
4. Kontrollen tar informationen och omvandlar den så att modellen kan förstå den och tar även hand om validering av data.
5. Kontrollern arbetar mot modellen så att modellens tillstånds ändrats.
6. När modellen ändras så meddelas vyn om detta igenom kontrollen och flödet börjar om på steg 1.

Detta illustreras i Figur 3.5.1. En heldragen linje i figuren betyder direkt association och streckad linje betyder via en observatör.

Man kan även använda en teknik som kallas observation istället för att meddela vyer via kontroller. Då har man en observatör som är kopplad till modellen och sedan registrerar sig vyer på observatören. När modellen ändras så meddelas alla lyssnare om detta via observatören.




Figur 3.5.1: MVC

### 3.5.2 ASP.NET MVC

ASP.NET [43] är ett ramverk för utveckling av webbapplikationer och webbsidor. ASP.NET används i flera miljoner applikationer, och några av dem är Bing, Xbox360 och The British Museum.

Det finns tre olika modeller för att bygga applikationer med ASP.NET. Den som används i implementationen av systemet är ASP.NET MVC. De andra två är Web Forms och Web Pages.

ASP.NET MVC är en implementation av MVC som används för att göra webbapplikationer i .NET. De följer den klassiska MVC-modellen beskriven i Sektion 3.5.1, men med en del skillnader. Vissa av dessa skillnader beror på att det är en webbapplikation vilket ger vissa begränsningar vid jämförelse med en skrivbordsapplikation. Det är till exempel svårare för modellen som kör på servern att meddela vyn, som kör i klientens webbläsare, om ändringar i modellen. Det innebär att felaktig data kan finnas i vyn som inte uppdateras förrän sidan laddas om. En nödvändig del av ASP.NET MVC är routningen som omvandlar webbadressen i ett ankommande HTTP-anrop och kopplar den till ett funktionsanrop som exekveras i en kontroller. Skrivs webbadressen *webbserveradress*/Home/About in så anropas en funktion `About()` i en kontrollerklass `Home`. Funktionen `About()` kan då utföra operationer som till exempel hämta data från en databas och skicka vidare data till

en vy som är kopplad till kontrollen. Vyn innehåller HTML och eventuellt ASP.NET-kod som omvandlar data från kontrollen till ett HTML-dokument som skickas till webbläsaren. Kontrollen kan välja att returnera innehåll direkt från kontrollen utan att passera en vy, till exempel XML-dokument vilket kan användas om man vill använda sig av AJAX (se Sektion 3.5.6). ASP.NET MVC har inbyggt stöd för Query-sträng (se Sektion 2.4) och fält-värde paren kan lätt läsas av inuti kontrollen.

### 3.5.3 IoC

IoC [41] (Inversion of Control) är ett designmönster där en objektkoppling sker först vid körning av programmet. Istället för att ett objekt kopplas till en specifik klass så är det kopplat till ett gränssnitt. Det tillåter mer dynamiska bibliotek där delar av biblioteket kan bytas ut från klientdelen. Det används ofta i samband med test då komplexa system ofta är svåra att testa då de kan kräva mycket konfigurering innan test kan utföras. Med IoC så kan delar av systemet bytas ut till falska versioner vilket underlättar testningen.

### 3.5.4 IIS

IIS [26] (Internet Information Services) är en webbserver utvecklad av Microsoft som är inkluderad i de flesta versioner av Windows. Alla versioner av Windows 7 [35] och Vista [34] har IIS inbyggt förutom Basic och Starter. IIS har stöd för att dela ut statisk innehåll och även köra olika typer av skript på datorn som till exempel ASP och PHP. IIS kan även köra ASP.NET MVC-applikationer och har stöd för debugging av C# och ASP-koden i Visual Studio under körning på webbservern.

Det finns flera alternativ till IIS som webbserver och Apache 2.6.1 är en annan populär webbserver. Då den inte stöder integration av ASP.NET MVC-applikationer är den ej lämplig. Ett alternativ är UltiDev Web Server Pro [47] som har integration med ASP.NET MVC-applikationer men måste installeras separat och dessutom är mindre utvecklad.

### 3.5.5 JavaScript

JavaScript utvecklades av Netscape och Sun Microsystems och första testversionen kom ut 1995. JavaScript är ett skriptspråk det vill säga att JavaScript läses som text och tolkas direkt i webbläsaren. Det medför att man slipper kompilera kod vilket i sin tur innebär att processorarkitekturen som webbläsaren körs på inte har någon betydelse. JavaScript är ett löst typat skriptspråk vilket betyder att man inte deklarerar typ på variabler då alla är av "typ" `var`.

Om man jämför starkt typade språk som tex C++ och C# och svagt typade språk som JavaScript så kräver de olika utvecklingsmetodik. Starkt typade språk är nästan alltid kompilerade språk och förlitar sig på en bra kompilator som hittar många simpla fel när man kompilerar medan JavaScript kräver att du koden körs för att upptäcka fel, såsom syntaxfel.

Javascript läggs oftast tillsammans med HTML-dokumentet och i Figur 3.5.2 följer ett exempel. Större sektioner kod kan skrivas i en JavaScript-fil. Då behövs inte `<script>`-taggen utan kod kan skrivas direkt. Filen kan includeras i html-dokument med `<script type=text/javascriptsrc="external.js"></script>` och används även om man ska inkludera externa JavaScript-bibliotek.

JavaScript används i vår implementation för att dynamiskt uppdatera delar av webbsidan. Detta inkluderar länkar till graferna, som byts när användaren klickar på länkar. Vad som egentligen händer är att webbsidans DOM-träd modifieras, se Sektion 3.7.3 för en beskrivning av DOM.

### 3.5.6 AJAX

AJAX [22] (Asynchronous JavaScript and XML) är en teknik som används för att göra interaktiva webbsidor. Utan AJAX så krävs att all information, såsom data från formulär, skickas till servern. Servern genererar en ny sida som skickas till klienten som blir tvungen

```
<html><head>
<script type="text/javascript">
var i=0;//all kod som inte ligger inne i en funktion
// körs automatiskt när dokumentet laddas.
function displayDate() // funktionsdeklaration
{
//Ett exempel på DOM manipulering
document.getElementById("demo").innerHTML=Date();
}
</script>
</head><body>
<h1>My First Web Page</h1>
<!-- ID används för att kunna identifiera ett element senare -->
<p id="demo">This is a paragraph.</p>
<button type="button" onclick="displayDate()">Display Date</button>
<!-- eventet onclick körs när användare klickar på knappen och då körs
koden inom parantesen. I detta fall kör den metoden displayDate()  -->
</body></html>
```

Figur 3.5.2: Exempel på JavaScript och DOM manipulering

att rita om sidan vilket kräver mycket dataöverföring och kan göra sidan långsam. AJAX består utav XMLHttpRequest[48], DOM (se Sektion 3.7.3), XHTML (se Sektion 3.7.1) och XML (Sektion 2.7). XMLHttpRequest tillåter JavaScript att hämta och skicka data mellan server och klient och kan använda sig av data för att uppdatera delar av sidan. XMLHttpRequest kan utöver XML även hantera JSON och rå text. Data kan skickas både asynkront och synkront men den asynkrona metoden rekommenderas då sidan ”fryser“ under tiden anropet utförs om den synkrona metoden används. Se Figur 3.5.3 för ett exempel som hämtar Mozilla’s hemsida och skriver ut den till webbkonsolen.

XMLHttpRequest kan även användas tillsammans med en Query-sträng (se Sektion 2.4) vilket gör det möjligt att skicka med argument till webbservern.

```
var request = new XMLHttpRequest();
request.open('GET', 'http://www.mozilla.org/', false); // false betyder att
// anropet görs synkront
request.send(null); // väljer man att göra ett asynkront anrop så skickas
//en delegat med istället för null.
console.log(request.responseText);
```

Figur 3.5.3: Hämtning av data med XMLHttpRequest

## 3.6 Grafgenerering

Grafer genereras i systemet, precis som i Altrans ursprungliga system, med hjälp av RRDtool. Det stora alternativet till grafgenerering som sker på servern är grafgenerering som görs på klienten (i webbläsaren). För att åstadkomma detta kan JavaScript användas. För en teknisk jämförelse med avseende på prestanda, se Kapitel 4.

## 3.7 Webbgränssnitt

Webbgränssnittet är den del av webbapplikationen som exponeras för användaren. Gränssnittet är bestående av en webbsida som visas i en webbläsare. I denna sektion beskrivs HTML och XHTML, som används för att strukturera upp innehåll i webbsidan. Begreppen CSS och DOM beskrivs också. Sektionen innehåller även ett avsnitt om webbläsare, vilka varianter och versioner som finns och vilka som webbgränssnittet ska kunna visas på.

### 3.7.1 HTML och XHTML

HTML [44] står för HyperText Markup Language, och är som namnet antyder ett märkspråk. HTML används för att strukturera innehåll på hemsidor, vilket görs i form av taggar. Ett exempel på en tagg är `<h1>` som anger att text som följer taggen är en rubrik, på nivå ett. För att avsluta rubriken används en motsvarade avslutningstagg, i detta fall `</h1>`.

HTML utvecklades ursprungligen av World Wide Web Initiative, på partikelfysiklabo-

ratoriet CERN.

Utöver HTML finns även ett annat snarlikt språk, XHTML. XHTML bygger på XML och är snarlikt HTML i syntax. En stor skillnad mellan de båda är att XHTML, precis som med XML, är utbyggbart. En användare behöver inte endast hålla sig till de fördefinierade taggarna, utan kan utöka med sina egna vid behov. XHTML är dessutom striktare än HTML, vilket till exempel innebär att taggar *måste* stängas. HTML kan tolerera att en tagg inte avslutas. Inbyggt i webbläsare finns hantering av sådana fel, och såvida inte felen är många kommer webbsidan att kunna visas normalt. Om detta fel görs i XHTML, kommer webbsidan inte att visas på ett korrekt sätt. XHTML är också skiftlägeskänsligt, vilket inte är fallet med HTML.

Vi har valt att vår webbapplikations gränssnitt ska vara strikt i sin uppbyggnad, och valet har då fallit på XHTML.

### 3.7.2 CSS

CSS [44] (Cascading Style Sheets) är ett språk konstruerat för att formatera utseendet av HTML-dokument. CSS bildar så kallade formatmallar (eng. *Style Sheets*) som styr hur en HTML-sida visas för användaren. Syftet med stilmallar är att separera struktur från presentation. Tanken är att en given HTML-sida kan visas på ett obegränsat antal sätt, beroende på den stilmall, skriven i CSS, som används. Ett bra exempel på styrkan i CSS kan hittas på webbplatsen CSS Zen Garden[2].

Denna webbsida är uppbyggd av en enda HTML-sida som kan presenteras på en rad olika sätt genom att man byter ut den stilmall som används, i menyn till höger, i den ursprungliga designen.

CSS bygger på att element i HTML-dokument formateras, placeras och ordnas på olika sätt. Det går att formatera befintliga taggar, såsom taggen för en rubrik, <h1>, eller välja att definiera egna taggar. Den färdiga formatmallen kan antingen placeras i samma fil som

[2] `http://www.csszengarden.com/`

HTML-dokumentet eller i en separat fil. Det senare är användbart om samma stilmall används för att formatera flera olika HTML-dokument.

Hanteringen av stilmallar är avgörande för vår implementation. Applikationen som exekverar på webbserven ska kunna ta data från webbläsaren då webbläsaren begär sidan. Data inkluderar information om webbläsare, skärmstorlek och upplösning. Denna information användas sedan för att dynamiskt byta formatmall. Flera olika CSS-mallar specificeras i förväg och byts ut med avseende på ovan nämnda egenskaper.

Det finns, enligt oss, inget bättre alternativ än CSS för formatering av innehållet på en webbsida. I tidigare versioner av HTML vävdes formatering in i HTML-koden i form av attribut för taggarna. Varje tagg formaterades för sig, vilket innebär minskad översikt om webbsidan tenderar till att bli omfattande. Dessutom blir det svårare utföra ändringar av en webbsidas design, särskilt om webbplatsen består av en större mängd HTML-sidor som alla manuellt måste omformateras.

### 3.7.3 DOM

När en webbläsare läser in en sida så tolkas sidan om till en intern representation. DOM [21] (Document Object Model) är ett språk- och plattformsoberoende gränssnitt som ger möjligheten att dynamisk läsa och ändra innehållet, strukturen och stilen på den interna representationen . Detta tillsammans med JavaScript tillåter skapandet av dynamiska webbsidor, där delar av den kan uppdateras utan att hela sidan laddas om. Ett exempel på dynamisk modifiering av DOM med JavaScript visas i Figur 3.5.2.

## 3.8 Sammanfattning

I detta kapitel har de lösningar som är del av systemet samt deras alternativ beskrivits. Data hämtas av systemet i en XML-fil. Anledningen till valet av XML är främst att det redan används i Altrans system. Detta gör det enklare att kravställa XML-filen efter behov,

utan att kräva att en helt annan fil genereras på serversidan. Alternativen är JSON eller en CSV-fil. Båda dessa är helt rimliga alternativ och moduler i systemet kan vid behov bytas ut för att kunna acceptera även dessa typer av dataöverföringsformat.

Den datainsamlingsenhet som används är en Netduino Plus. Valet av Netduino gjordes främst av Altran, då enheten är billig, lättillgänglig och har stöd för .NET MF. Raspberry Pi är billigare, men svår att köpa in i skrivande stund. Arduino har valts bort då de inte har stöd för .NET MF. Enheten i Altrans system används fortfarande för att hämta mätdata i form av temperatur och energiförbrukning.

Serverspråket som används är C#.NET. Val av språk gjordes tidigt i projektet, främst för att det är ett språk vi har erfarenhet av sedan tidigare, men också för att det är språk som efterfrågas på marknaden. Alternativen är PHP eller möjligen en lösning som använder sig av flera olika språk, men med C#.NET och övriga Microsoftprodukter fås en mer enhetlig lösning.

Som databashanterare har Microsoft SQL Server valts. Anledningen är främst att använda ett alternativ till MySQL, som används i Altrans system. Det var också ett krav från Altran att öppna upp för användandet av Microsoftprodukter i det nya systemet. Versionen av Microsoft SQL Server som används är en express-version, och därför gratis att använda. Huvudalternativet till Microsoft SQL Server är SQLite. SQLite används i systemet för utvecklingssyfte, men är inte del av den slutgiltiga lösningen. SQLite kompileras till antingen 32- eller 64-bitars programfiler. Detta innebär en del kompatibilitetsproblem.

Webbapplikationen byggs upp enligt MVC, och använder sig av Microsofts implementation av detta, ASP.NET MVC. MVC är ett designmönster som syftar till att separera presentations- och affärslogik. Denna uppdelning gör det lättare att utveckla systemets olika delar separat. Alternativet till MVC, om ASP.NET fortfarande ska användas, är antingen Web Forms eller Web Pages.

För att dynamiskt ändra på webbsidans innehåll används JavaScript. JavaScript används främst för att trigga servern att generera nya grafer, och att uppdatera HTML-koden

med nya länkar till dessa. Detta kan även lösas med att låta uppdatera hela sidan med ett visst intervall. Ett samlingsnamn för denna teknik och andra kallas AJAX.

Webbgränssnittet är utvecklat i XHTML. Formatering görs av CSS. Det finns inga egentliga alternativ till dessa två om man utvecklar en webbaserad applikation. HTML skulle kunna användas, men vi har inte sett någon fördel med att använda det i stället för XHTML. Kraven för god kodkvalité ökar då XHTML är striktare än HTML, vilket vi anser är positivt.

# Kapitel 4

# Utredningar

*I detta kapitel ges en detaljerad beskrivning av de utredningar som gjorts i projektet. Utredningarna syftar till att göra en jämförelse mellan olika alternativ för grafgenerering och dataöverföring. Kapitlet inleds med en övergripande beskrivning av syftet med utredningarna samt de olika alternativ som finns. Hur testerna har delats upp i olika grupper beskrivs också. Därefter följer en mer ingående beskrivning hur de olika testgrupperna ser ut samt resultat och antaganden. En beskrivning av datakomprimering finns också.*

## 4.1 Översikt

Syftet med utredningarna är att avgöra vilken metod som är mest resurssnål för generering av grafer. Med resurser avses tid samt överförd datamängd. Utredningens resultat avgör vilken metod som i framtiden bör användas vid en eventuell vidareutveckling av systemet, eller om systemet byggs på nytt. Önskemål om ökad användarinteraktivitet (såsom att styra grafers utseende från användargränssnittet) har kommit från Altran. Sådan funktionalitet blir enklare att implementera om grafer skapas på klienten istället för på servern. Detta är således även av detta skäl intressant att göra utredningarna.

För att göra en jämförelse mellan olika former av dataöverföring och grafgenerering har

ett flertal automatiska tester implementerats. Testerna mäter den tid det tar att hämta data och presentera den i en graf på webbsidan. Tiden är en summering av olika operationer, beroende på vilken metodik som används.

Två metoder för grafgenerering har identifierats. Den ena är att använda RRDtool för att generera en graf, vilket är vad som görs i Altrans ursprungliga system och även i det nya systemet. Det andra sättet är att använda något JavaScript-bibliotek. Skillnaden mellan dessa är att RRDtool exekverar på servern medan JavaScript körs på klienten, det vill säga webbläsaren.

Testerna har gjorts på olika sätt beroende vilken metod som använts. För RRDtool mäts den tid det tar för servern att generera grafen, att skicka länk till denna bild till klienten samt tiden det tar att överföra bilden.

För grafgenerering med JavaScript görs tre olika mätningar:

1. Tiden det tar att extrahera data från RRD, skapa en XML eller JSON-sträng samt skicka denna data från server till klient.

2. Tiden det tar för JavaScript att parsa data.

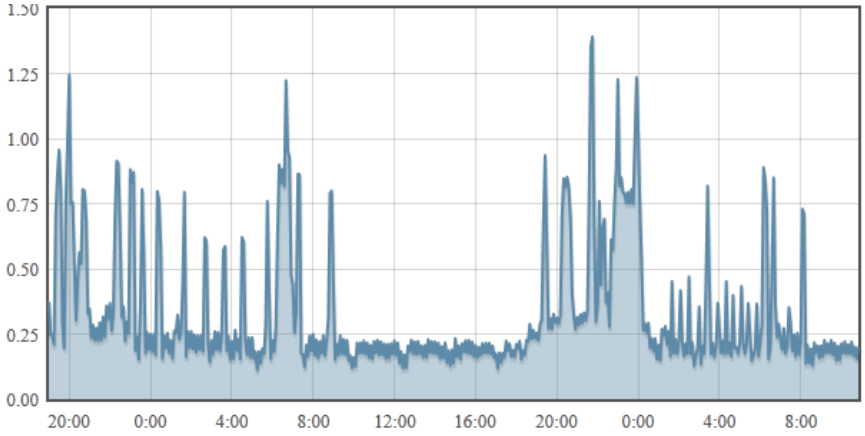
3. Tiden det tar för JavaScript att rendera grafen på skärmen.

Mätning 1 är för operationer som körs på servern medan 2 samt 3 utförs på klienten. Mätning 1 är bestående av tre deltider som är svåra att mäta separat, särskilt från klienten.

Mätningarna har utförts med ökande tidsintervall. Med detta avses det intervall som grafen ska visa. För varje sådant intervall har 100 tester gjorts varefter ett medelvärde har räknats ut. På så sätt fås mera rättvisa värden och avvikelser beroende på störningar i nätverket minimeras.

En ytterligare aspekt som ingår i testerna är huruvida data som skickas via JSON eller XML komprimeras eller inte. I Sektion 4.4 tas detta upp mer i detalj.

Testerna har utförts på en utvecklingsdator och Galaxy Tab mot server. Båda ingår i samma nätverk, och kommunicerar trådlöst över WiFi.




Figur 4.2.1: Graf skapad med jQuery-biblioteket flot

Resultaten för testerna presenteras i Sektion 4.6

## 4.2 Grafgenerering

Grafgenerering med RRDtool görs med hjälp av den wrapper som implementerats. Beskrivning av implementationen av RRDtool-wrappern kan hittas i Sektion 5.5.

Grafgenereringen med JavaScript görs med hjälp av ett jQuery-biblioteket [18] kallat flot, kombinerat med AJAX. flot finns tillgängligt på Google Code [27].

Mätvärden hämtas av ett skript med hjälp av AJAX-metodik. Skriptet transformerar dessa värden till lämpligt format för flot-funktionen som ritar upp grafen. En så enkel formatering som möjligt av grafen har valts. Ett exempel på hur en graf genereras med flot kan ses i Figur 4.2.1.

## 4.3 Dataöverföring

Två alternativ har valts att utredas, JSON och XML, som båda beskrivits i tidigare kapitel. Data som extraheras från RRD packas ihop till ett JSON- eller XML-dokument på servern.

```
[
  [1335863330000, 0.11],
  [1335863340000, 0.29]
]
```

Figur 4.3.1: Struktur hos JSON-dokument som används av flot

Detta görs med en egen implementation som bygger på att data läggs till en sträng som utgör ett giltigt JSON- eller XML-dokument.

JSON-dokumentet har formaterats till den struktur som flot kräver. Denna lösning gör att klienten inte behöver göra ytterligare transformeringar av datamängden innan den skickas till flot. Nackdelen med detta är att dokumentet är svår att tolkas av människor. Strukturen kan ses i Figur 4.3.1.

Strukturen utgörs av en array vars element i sin tur är arrayer med två element, tiden då värdet uppmättes (med Posix-tidsstämpel * 1000) samt själva värdet. Anledningen till att tiden multipliceras med 1000 är att i JavaScript så representeras tid som antalet *millisekunder* sedan 1 januari 1970.

XML-dokumentet en någon annorlunda struktur. Även här multipliceras tiden med 1000. Figur 4.3.2 visar strukturen.

```
<measures>
  <measure>
    <timestamp>1335863330000</timestamp>
    <value>0.11</value>
  </measure>
  <measure>
    <timestamp>1335863340000</timestamp>
    <value>0.29</value>
  </measure>
</measures>
```

Figur 4.3.2: Struktur hos XML-dokument som används av flot

Strukturen har gjorts med avseende på läsbarhet; till skillnad från JSON-dokumentet är XML mer läsbart av människor.

## 4.4 Datakomprimering

Komprimering används för att minimera mängden data som överförs mellan server och klient. För att komprimera innehåll som skickas från en kontroller används ett filter som kopplas till en metod i en kontroller med [compress]. Filtret kollar om requestanropet har header-fält `Accept-Encoding: gzip` satt. Om så är fallet så komprimeras innehållet i kroppen i HTTP-GET i gzip-format[8][9] och headerfältet `Content-Encoding: gzip` läggs till. Senare när innehållet hämtas i JavaScript så dekomprimeras innehållet automatiskt [12].

## 4.5 Testmiljö

Testerna utförs i en webbsida där mätresultaten visas efter körning. Server och klient sitter på samma trådlösa nätverk som har en maxhastighet på 54 Mbit/s. Testerna har utförts på två olika enheter, en tablet och en laptop. Servern och klient-laptopen är av samma modell.

**Tablet**

- Modell: Galaxy Tab 10.1V [42]
- CPU: 1 GHz dual core
- Arbetsminne: 1 GiB
- Operativsystem: Android 3.1
- Webbläsare: Inbyggd Webbläsare

**Laptop**

- Modell: Dell precision m4300 [1]
- CPU: Intel Core 2 Duo T9300 2.5 GHz
- Arbetsminne: 4 GiB
- Operativsystem: Windows 7 Enterprise
- Webbläsare: Firefox 12

## 4.6 Utredningsresultat

I denna sektion presenteras resultaten från utredningarna. Sektionen har delats in in i flera undergrupper som var och en presenterar ett delresultat i form av en eller flera grafer. Slutsatser baserade på resultaten presenteras i Sektion 4.7.

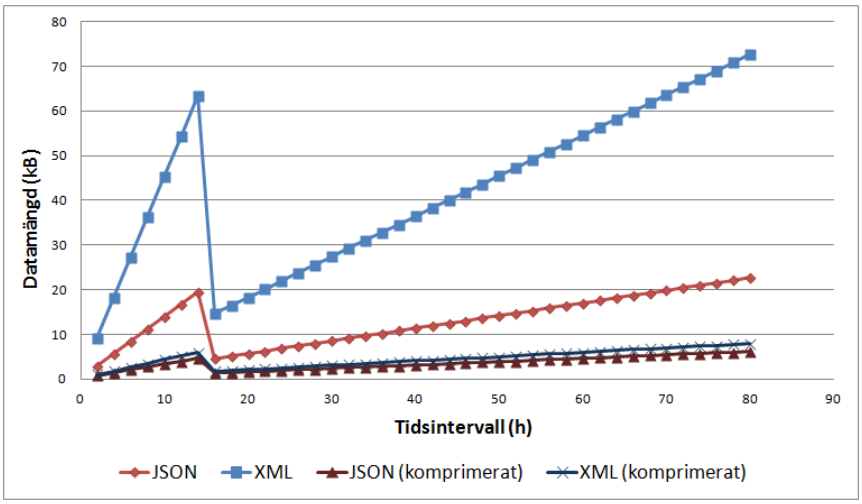
### 4.6.1 JSON eller XML

Nedan presenteras olika grafer som visar resultat från utredning med avseende på dataformat. Det är viktigt att, om grafgenerering ska ske på klienten, att det mest effektiva dataformatet väljs för att minimera tiden det tar att skicka data samt att transformera det till lämpligt format.

Figur 4.6.1 visar storleken för de olika dataformaten då tidsintervallet ökar. Även komprimering ingår.

Figur 4.6.2 visar parsningstider för JSON och XML. Man kan i grafen även se ett tydligt samband mellan antalet mätpunkter och storleken. Den karaktäristiska toppen vid ungefär 15 timmar beror på att systemet vid denna tidpunkt byter RRA, till en med lägre upplösning. Detta resulterar i att antalet mätpunkter minskar, och så även storleken på dokumentet.

Parsning av JSON sker med en inbyggd funktion i JavaScript och parsning av XML




Figur 4.6.1: Jämförelse av datamängd, JSON kontra XML.

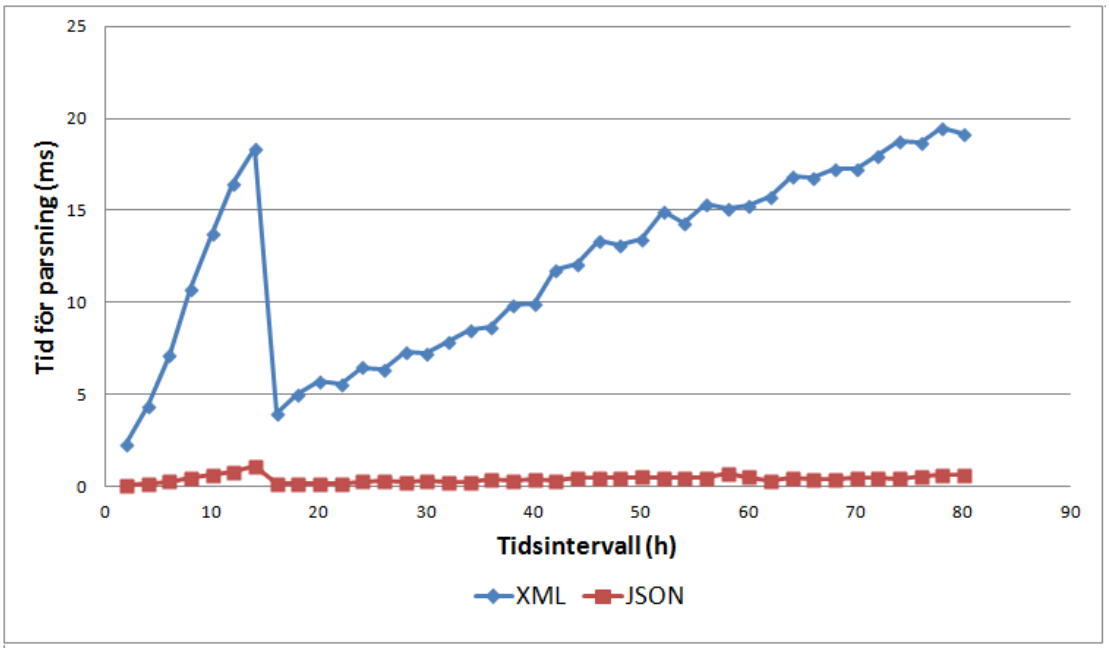
sker med en egenutvecklad funktion. Den stora skillnaden beror på att `JSON.parse()` kör kompilerad kod som är del av webbläsaren, vilket ger mycket bättre prestanda. Alternativet till denna jämförelse är att implementera en JavaScript-parser för JSON för att få en mer korrekt jämförelse. Nackdelen är att man då inte utnyttjar den stora fördelen med den inbyggda parsern.

Figur 4.6.3 visar tiden det tar att skicka data formaterat som XML och JSON. Data är i detta fall komprimerad för att minimera datamängden, vilket åskådliggjordes i Figur 4.6.1.

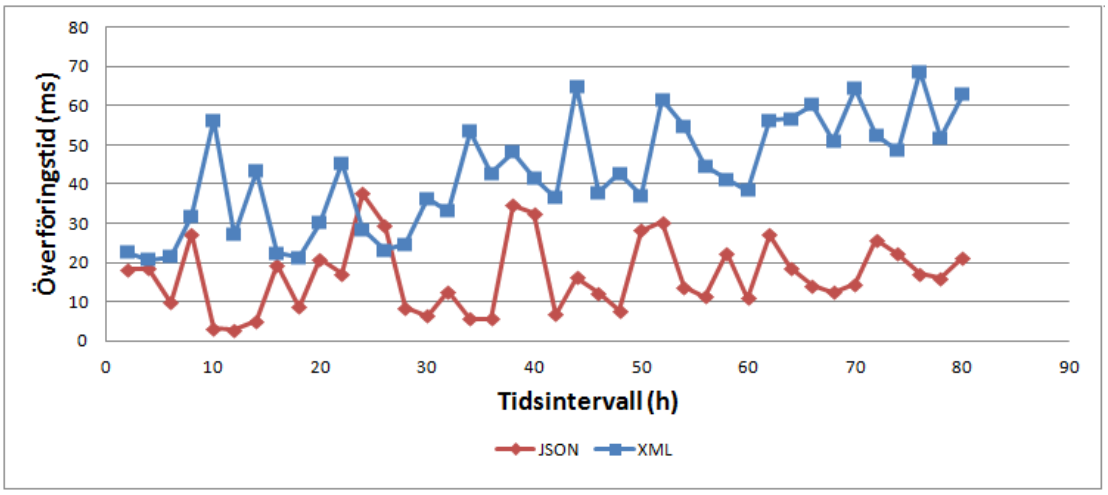
### 4.6.2 Grafgenerering på klient (JavaScript) eller server (RRD-tool)

Baserat på resultat från föregående avsnitt kan vi anta att JSON är det mest resurssnåla sättet att överföra data från server till klient. På grund av att JSON går snabbare att parsa är det ännu ett bra skäl till att använda det.

Figur 4.6.4 visar en jämförelse av total överföringstid. Denna tid är ett medelvärde, för



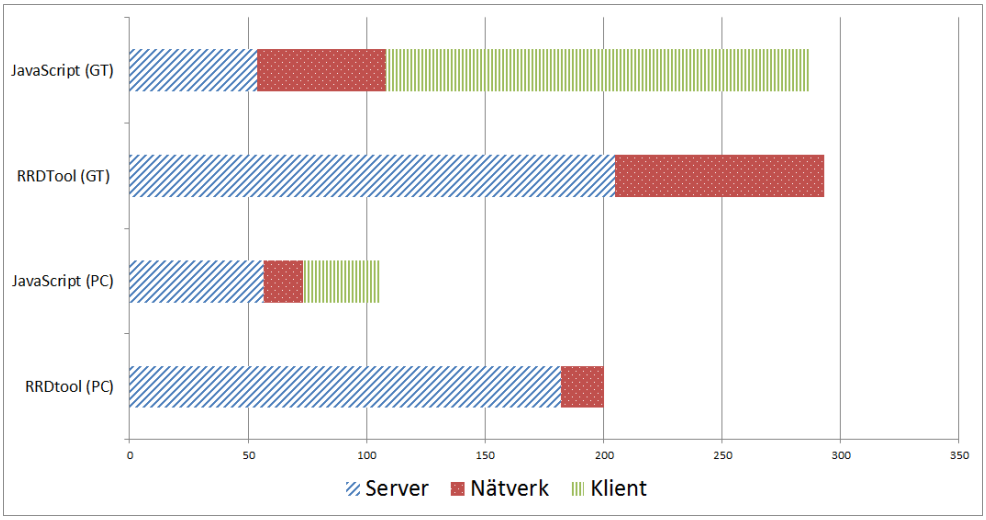

Figur 4.6.2: Jämförelse av parsetider, JSON kontra XML.




Figur 4.6.3: Jämförelse av överföringstider, JSON kontra XML.

olika tidsintervall mellan två och 80 timmar. Grafen visar tid som beror på operationer utförda på server, tid det tar att skicka data över nätverk, samt tid som beror på operationer

som utförs på klienten (webbläsaren). Mätningarna har gjorts för del en PC men också en Samsung Galaxy Tab (GT i grafen). Tid för att parsa JSON på klienten har uteslutits då den är så kort (ungefär 0.3 ms) och ändå inte syns i grafen.




Figur 4.6.4: Jämförelse av total tid

## 4.7 Slutsatser

En första tydlig slutsats som kan göras från Figur 4.6.1 är att ett XML-dokument tar upp mycket mer plats än ett JSON-dokument. Detta beror på att det finns en del överflödig data i ett XML-dokument, i form av beskrivande taggar. Ur figuren kan utläsas att en komprimering löser detta problem mycket bra. Skillnaden mellan komprimerat och okomprimerat XML-dokument är mycket hög. Effekten är inte lika tydlig med JSON, då JSON redan är kompakt i sin uppbyggnad. Skillnaden mellan de komprimerade formaten är inte stor, men komprimerad JSON är något mindre.

I Figur 4.6.2 jämfördes parsningstiden. Skillnaden bestod i att webbläsaren använder kompilerad kod för att parsa JSON, medan JavaScript används för att parsa XML. Slutsatsen är att den inbyggda JSON-parsern är mycket snabbare än den egenutvecklade

XML-parsern och bör användas för att miminera parsningstiden. Det är svårt att avgöra om XML-parsern är optimalt implementerad. Kanske finns möjligheten att göra den mer effektiv, men vi har svårt att se att den skulle kunna bli lika snabb som JSON-parsern.

Ur Figur 4.6.3 kan de totala tiderna för att hämta data från RRD, skapa JSON- eller XML-dokument, skicka detta dokument, extrahera data på klientsidan samt rendera grafen utläsas. Graferna i figuren är inte linjära, men trenden är att parsningstider för XML ökar snabbare än parsningtider för JSON. Dock kan en evigt ökande trend förhindras genom en bra design av RRD. När antalet mätvärden når det maximala antalet för en RRA kommer nästa RRA att väljas. Detta innebär att antalet mätvärden som skickas mellan server och klient effektivt kan reduceras till ett bestämt maxvärde.

Figur 4.6.4 illustrerar relationen mellan grafgenerering med JavaScript och RRDtool, utfört dels på en PC men också på en Galaxy Tab. En första slutsats man kan dra är att tidsskillnaderna inte är enorma. Kortaste totala tiden är ungefär 100 ms, och längsta är ungefär 300, alltså en skillnad på 200 ms. En sådan tidskillnad bör inte påverka användarupplevelsen.

Man ser tydligt att rendering med JavaScript tar många gånger längre tid på Galaxy Tab än på PC. Detta beror antagligen på de begränsade resurser en Galaxy Tab har. Detta resulterar också antagligen i att överföringstiden ökar, trots att båda enheterna befinner sig på samma nätverk.

# Kapitel 5

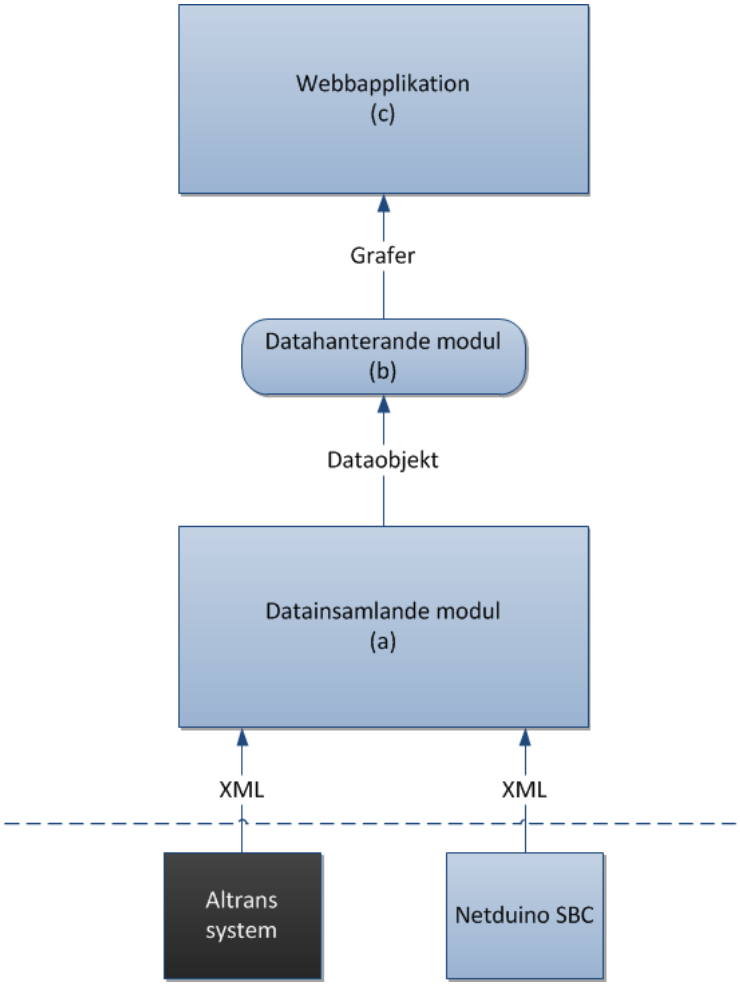
# Design och implementation

*I detta kapitel beskrivs det nya systemets design och implementation. Kapitlet inleds med en översiktlig beskrivning, som tar upp systemets huvudsakliga moduler. Dessa moduler beskrivs sedan mer ingående, på olika detaljnivå.*

## 5.1 Översikt

Systemet kan grovt delas in i tre moduler, som illustreras i Figur 5.1.1. Modul (a) är den del av systemet som hämtar mätdata i form av ett XML-dokument från webbservern i Altrans system. Modulen transformerar mätdata till dataobjekt som skickas till modul (b), den datahanterande modulen. Den datahanterande modulen sköter indirekt kontakten med databaserna i vilka mätdata lagras för att kunna hämtas vid ett senare tillfälle i ett tidsperspektiv. De databaser som används är dels MS SQL och RRD (via RRDtool), men också SQLite för utvecklingssyfte. Denna modul fungerar dels som ett gränssnitt mellan systemet och de databaser som används, men också som ett gränssnitt mellan den datainsamlande modulen och webbapplikationen (c).

I webbapplikationen presenteras mätdata i tabellform och som grafer, som skapas av RRDtool. Det är den datahanterande modulen som arbetar mot RRDtool.




Figur 5.1.1: Systemöversikt

## 5.2 Datainsamlande modul

Syftet med den datainsamlande modulen är att ta emot data i form av ett XML-dokument. Detta XML-dokument är antingen publicerat på webbservern i Altrans system och hämtas med HTTP-Get. XML-dokumentet kan också hämtas från systemets egen enkortsdator, en Netduino Plus som är placerad på Altrans kontor. När XML-filen har hämtats är det den datainsamlande modulens uppgift att transformera strängen med mätdata till C#-dataobjekt. Dessa dataobjekt skickas sedan till den datahanterande modulen.

I denna sektion beskrivs modulens två huvudsakliga komponenter, den insamlande komponenten samt den transformerande komponenten. En beskrivning av XML-dokuments struktur finns också.

### 5.2.1 XML-specifikation

Att bestämma strukturen för det XML-dokument som systemet tar emot är nödvändigt för att systemet ska fungera. Den slutgiltiga versionen av detta XML-dokument bygger på det ursprungliga XML-dokument som används i Altrans system. Det finns vissa skillnader, vilka presenteras nedan.

- Taggarna i den ursprungliga XML-dokumentet, för en enhet (device) var namngivna enligt mönstret `device_[taggnamn]`. Detta är onödigt, då de är subelement till `<device>`. Prefixet `device_` har tagits bort.
- Tillägg av olika taggar har gjorts. De nya är `<unit>` där enheten (som till exempel kW) för värdet anges, samt `<value_type>` som anger vilken av de två typer, som systemet har stöd för, värdet har. Dessa två typer är COUNTER (ökande värde) och GAUGE (varierande värde),
- Ett tillägg av taggen `<interval>` i början av XML-dokumentet som specificerar, i antalet sekunder, hur ofta XML-dokumentet ska hämtas i fortsättningen.

Dessa tre tillägg gör det enklare att generalisera hanteringen av mätdata. Om enheten för värdet som skickas anges, kan denna lagras tillsammans med värdet i databasen. Vilken enheten egentligen är, är helt irrelevant för systemet. Det systemet behöver veta är att det finns en enhet för att sedan kunna presentera den tillsammans med mätvärdena i till exempel en tabell.

Att sätta ett uppdateringsintervall i XML-dokumentet gör det lätt för den som tillhandahåller dokumentet att själv styra hur ofta systemet ska hämta nya värden. I Altrans

system sker hämtning av mätdata var femte sekund. Det är lämpligt att då ange detta intervall i XML-dokument. Anledningen till detta är att tidsintervallet kan variera beroende på vilken data som samlas in.

### 5.2.2 XMLRetriever

XMLRetriever är en klass vars syfte är att hämta ett XML-dokument. Som omnämnts tidigare finns det två sätt att göra detta på. Systemet i sin ursprungliga form kunde endast använda den XML som genereras på det ursprungliga systemets server. Anledningen till att samma XML-dokument användes var för att underlätta utvecklingen. I senare versioner av systemet används det nya XML-dokumentet som beskrevs i Sektion 5.2.1.

För att hämta XML-dokumentet använder sig XMLRetriever av en inbyggd metod i .NET som laddar ned det. Det XMLRetriever behöver för att utföra detta är URL till XML-dokumentet. XMLRetriever kan även verifiera om webbadressen är giltig eller inte, och om så inte är fallet inte hämta något. Systemet fortsätter att exekvera även om XML-dokumentet inte är tillgänglig. Alternativet är att låta systemet upphöra sin exekvering, men det finns flera anledningar till att så inte är fallet. Webbadressen kan vara ”ogiltig“ av flera anledningar. Servern kan ha kraschat, eller så är det stockning i nätverkstrafiken, vilket tolkas som att servern har kraschat. Genom att fånga upp felet, men sedan ignorera det, kan systemet fortsätta att exekvera och behöver således inte startas om efter en eventuell krasch.

Den senaste versionen av systemet kan även hämta ett XML-dokument från en Netduino Plus. Detta görs på samma sätt som hämtningen från Altrans system.

### 5.2.3 XMLParser

Denna komponent är implementerad som en klass vars syfte är att extrahera information från XML-dokument som systemet läst in. Klassen är statisk då den inte har behovet av att lagra data mellan metodanrop.

Klassen har två metoder som båda tar som indata en sträng, själva XML-dokumentet. Den ena metoden extraherar det värde som anger uppdateringsintervallet. Detta intervall används för att programmet ska veta hur ofta XML-dokumentet ska hämtas i fortsättningen. Den andra metoden extraherar all data för varje sensor från XML-dokumentet.

Extraheringen av informationen görs med hjälp av den inbyggda klassen `XmlReader`. En instans av denna klass går igenom hela XML-dokumentet och identifierar de olika taggarna. Efter att ha identifierat en specifik tagg, läses det aktuella värdet för den taggen in. All information som extraheras för varje sensor på detta sätt lagras i ett dataobjekt. XML-dokumentet som hämtas innehåller information om/för mer än en sensor, vilket resulterar i att alla dataobjekt lagras i en lista. Denna lista returneras från metoden som utför dataextraheringen.

## 5.3 Databasdesign

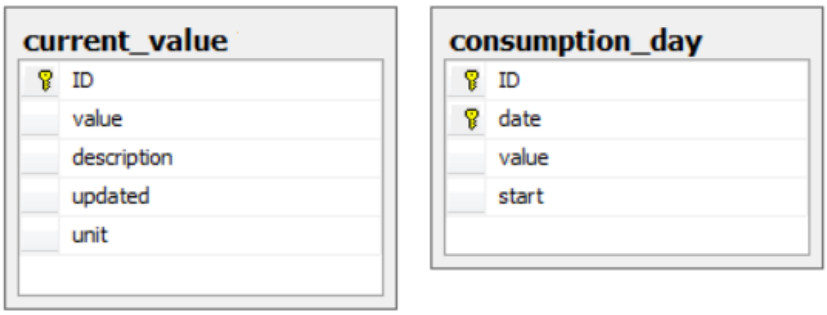
När data har hämtats behöver data lagras. Detta kan göras på tre olika sätt i systemet. Data lagras i en RRD med hjälp av RRDtool, men också i en SQL-databas, antingen MS SQL-server eller SQLite. I SQL-databaserna lagras de senaste värdena samt historik för konsumtionsdata. I RRD lagras värden över tid, vilka används för att generera grafer.

### 5.3.1 MS SQL och SQLite

Databasen har designats att vara så generell som möjligt, men också att vara så simpel som möjligt. Ingen överflödig information lagras.

De båda möjliga databasernas design är likvärdig, och de två tabeller som används kan ses i Figur 5.3.1.

Tabellen kallad `current_value` är en reflektion av XML-dokumentet. Tabellen lagrar de senast uppmätta värdena för att dessa sedan ska kunna visas i en tabell på hemsidan. Tabellen lagrar följande information:




Figur 5.3.1: Databasens två tabeller

- `ID` (sträng) : unikt namn för den sensor som uppmätt värdet.
- `value` (flyttal): det senast uppmätta värdet.
- `description` (sträng): beskrivning av vad som mäts. Som exempel, "Temperatur övervåning".
- `updated` (heltal) : anger tiden, i Posix, för senaste uppdatering.
- `unit` (sträng): den aktuella enheten för mätvärdet.

Den andra tabellen, `consumption_day` lagrar data av typen COUNTER i ett tidsperspektiv. Värdena används för att ge en överblick av förbrukning över tid, i detta fall för varje dag. Tabellen lagrar följande information:

- `ID` (sträng): unikt namn för den sensor som uppmätt värdet.
- `date` (datum): aktuellt datum för insamlad data.
- `value` (flyttal): den totala förbrukningen sedan sensorn anslöts till systemet.
- `start` (flyttal) : den totala förbrukningen vid start för angivet datum.

Anledningen till design av denna tabell beror på att det alltid är den totala förbrukningen som en sensor registrerat som anländer till systemet. Detta är, som nämnts tidigare, ett alltid ökande värde. För en beskrivning av hur data av typen COUNTER bearbetas innan den lagras, se Sektion 5.5.3.

### 5.3.2 RRD

Data lagras i ett tidsperspektiv i en RRD (Round Robin Database). I det nya systemet så har varje sensor tre RRA (Round Robin Archive) med vardera plats för 1000 mätpunkter. Första RRAn har samma upplösning som datakällan vilket, om man har 10 sekunders mätintervall, gör att mätdata lagras i ungefär 2.8 timmar. Nästa RRA har en upplösning på 6 vilket ger ungefär 16.5 timmar och sista med upplösning 30 ger ungefär 83 timmar eller ungefär 3.5 dygn.

Vill man utöka systemet med stöd för lagring över en längre tidsperiod, är det bara att lägga till ytterligare RRAs.

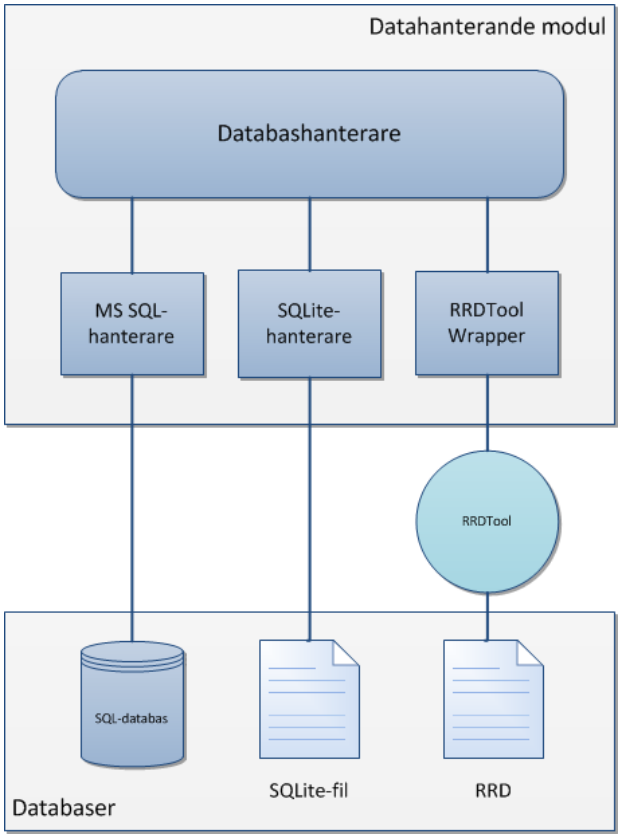
## 5.4 Datahanterande modul

Den datahanterande modulen har som syfte att sköta all hantering av den extraherade mängden mätdata. Hantering innebär kontakt med databaser samt RRDtool. Detta görs inte direkt, då modulen är uppdelad i två lager. Det övre lagret är implementerad som en klass (`DatabaseManager`), och det undre lagret är klasser vars metoder exponeras till det övre lagret. Figur 5.4.1 illustrerar förhållandet mellan de olika komponenterna.

Det två databashanterande komponenterna implementerar båda ett interface, CRUD [19] (Create, Read och Update, interfacet saknar stöd för Delete). Interfacet specificerar de metoder som komponenterna måste implementera för att få önskad funktionalitet. På grund av denna typ av design, är det enkelt att bestämma vilken typ av databas som används. Ska systemet utökas med en ny databas, låter man en ny komponent (klass) implementera CRUD-interfacet och ställer in att denna ska användas istället.

### 5.4.1 MS SQL-hanterare

Denna komponent sköter kommunikationen med systemets SQL-databas. Kommunikationen sker med frågespråket SQL. Som nämnts tidigare implementerar denna komponent ett




Figur 5.4.1: Datahanterande modul

interface med fördefinierade metoder. Dessa metoder implementeras, och anropas sedan från `DatabaseManager`.

Det finns två typer av SQL-frågor som systemet använder sig av. Det ena är en så kallad *Non-Query*. En sådan fråga returnerar inget från databasen utan används för att ändra databasens tillstånd (INSERT). Den andra, kallad *Reader*, bygger på en fråga som hämtar data från databasen (SELECT). Denna uppdelning av SQL-frågor har lett till implementation av två olika metoder som utför dessa frågor. Den ena metoden skriver alltså data till databasen, och den andra läser. Dessa metoder refereras nedan till som huvudmetoder.

Metoderna från CRUD använder sig av huvudmetoderna, vilket gör koden i dessa my-

cket kortfattad, ofta bara en rad. Designen bygger på att göra just CRUD-metoderna så enkla och lättlästa som möjligt. Detta leder till att komplexiteten flyttas till de två huvudmetoderna. Till exempel så specificeras SQL-frågorna i dessa metoder, och väljs ut genom att en parameter till metoden bestämmer vilken typ av fråga som ska exekveras.

Ett alternativ till ovanstående metodik är ADO.NET Entity Framework [25]. Detta är ett ramverk för mappning mellan databaser och dataobjekt. Tekniken har diskuterats under arbetets gång, och om tid funnits skulle den också ersatt ovanstående beskriven metodik. Dock gjordes bedömningen att detta inte var kritiskt för att uppnå projektets mål samt att det skulle vara tidskrävande att lära sig.

### 5.4.2 SQLite-hanterare

Principen för denna komponent är den samma som för MS SQL-hanteraren. Skillnaden är att data inte lagras på en extern server, utan som en fil på samma dator där systemet exekverar. SQLite har använts främst i utvecklings- och testsyfte. Det går enkelt att byta mellan de två typerna av databashanterare, med hjälp av IoC, som diskuteras i Sektion 5.6.2.

Fördelen med att använda SQLite är att systemet inte kräver åtkomst till någon resurs som inte finns på datorn där systemet exekverar. Undantaget är givetvis systemet som tillhandahåller XML-dokumentet.

## 5.5 RRDtool med C#

Systemet använder RRDtool vilket är ett program som används via kommandotolken i Windows för att lagra mätvärden i en RRD på hårddisken. För att lättare använda RRDtool i C# implementerades en wrapper som tar olika argument och omvandlar de till kommandon som körs i kommandoraden. Wrappern är inte beroende av andra delar i systemet och kan användas separat som ett bibliotek, dock krävs att RRDtool är installerat

separat. För tillfället kan systemet endast skapa nya databaser, ej ta bort dem.

Wrappern utgörs av flertalet komponenter vars egenskaper styr hur data lagras och grafer genereras, vilka beskrivs i denna sektion. Ett färdigt alternativ kallat NHawk tas också upp.

### 5.5.1 NHawk

NHawk är en tunn wrapper till RRDtool för att lättare kunna använda RRDtool i en C#-applikation [6]. En wrapper i detta sammanhang omvandlar argument till en kommandorad som exekverar i en kommandotolk. Vi beslutade oss tidigt att inte använda NHawk då NHawk är licensierad under GPL[13]. Om vi vill använda NHawk så måste projektet licensieras under GPL vilket inte är önskvärt av Altran. Utöver detta så saknas viss funktionalitet bland annat möjligheten att extrahera data ur en RRD. Ett annat problem är att buggar kan finnas vilka kan vara svåra, men möjliga, att åtgärda.

### 5.5.2 RRDtool-wrapper för datahantering

På grund av att vi valde bort NHawk utvecklades en RRDtool-wrapper. Detta är en modul bestående av flertalet komponenter. Modulens syfte är att underlätta användningen av RRDtool från systemet. Modulen döljer systemanrop för RRDtool från övriga systemet.

Modulen har implementerats i form av flertalet olika klasser. Den centrala klassen heter `InDatabaseRrd` och omvandlar dataobjekt till en sträng som körs i kommandotolken. Strängen är ett systemanrop för RRDtool.

Modulens övriga komponenter är klasser vars egenskaper motsvarar inställningar vid skapande av RRD, hämtning av data samt grafgenerering. Vid användandet av wrappern är det viktigt att känna till vilka metoder som finns och hur man skapar de objekt som används som parametrar. Parametrarna motsvarar inställningarna.

Alla metoder som wrappern implementerar bestäms i ett interface kallat `IRrd`.

### DS

En DS (DataSource) är en klass som motsvarar en datakälla. Klassen har flertalet egenskaper, där den första är vilken typ av data som datakällan producerar. RRDtool har stöd för typerna COUNTER, GAUGE, DERIVE, ABSOLUTE och COMPUTE men systemet har i nuläget endast stöd för COUNTER och GAUGE. Det går att utöka detta genom att modifiera DS-klassen.

En DS kan också spara information som rör gränser för mätvärden, närmare bestämt min- och maxvärde ett mätvärde kan anta. RRDtool ignorerar mätvärden om de ligger utanför denna gräns. Att sätta dessa gränsvärden förhindrar att orimliga värden lagras i databasen. Ett exempel är att ett fel vid mätningen av utomhustemperatur skulle leda till ett värde på 10.000 vilket uppenbarligen är ogiltigt.

### RRA

RRA (Round Robin Archive) är ett arkiv som innehåller ett antal mätvärden kopplade till en datakälla. Detta har i systemet implementerats i form av en klass. Systemet har stöd för fyra olika sätt att tunna ut gamla mätvärden: AVERAGE, MIN, MAX och LAST. MIN, MAX och LAST tar det minsta, största eller det sista värdet och lagrar detta i en RRA. Systemet har även stöd för att sätta precisionen hos RRAn. Precisionen bestämmer upplösningen relativt det specificerade mätintervallet.

### RRD

En RRD (Round Robin Database) representeras likt en RRA i systemet som en klass. RRDtool har stöd för att koppla flera datakällor (DS) till en RRD-fil. Vi har dock valt att varje DS sparas i sin egen RRD-fil. Detta underlättar borttagandet av DS från systemet då endast filen behöver tas bort.

Ett RRD-objekt lagrar data som är kopplat till hur en RRD ska se ut och bete sig. En av egenskaperna är att mätintervallet, som bestämmer upplösningen för RRDn. Med

upplösning menas avståendet i sekunder mellan mätpunkter. Ett RRD-objekt kan koppla en eller flera RRA till sig, beroende på hur man vill att data ska lagras. Se Sektion 5.3.2 för en beskrivning av inställningarna för datalagringen i systemet.

### 5.5.3 ConsumptionHelper

ConsumptionHelper är en komponent, en del av den databashanterande modulen, som har två uppgifter. Dels ska den uppdatera databasen med data relaterad till förbrukning. Data gällande förbrukningen är av typen `COUNTER`, vilket är ett kontinuerligt växande värde för en förbrukning av något slag.

För att få en översikt av förbrukning, till exempel på dygnsbasis, behöver värden sparas i en databas för att kunna lagras en längre tid. ConsumptionHelper är den komponent som sköter detta, som också räknar ut riktningskoefficenten, som alltså är nuvarande förändring. Detta värde sparas sedan också i databasen.

För att beräkna den totala konsumptionen för givet datum (dag) måste man beräkna:

$v_{day} = v_{now} - v_{start}$

där $v_{day}$ är förbrukningen för en dag,

$v_{now}$ är den senast uppmätta totala förbrukningen och

$v_{start}$ är är den totala förbrukningen uppmätt vid midnatt samma dag.

Ska den aktuella förbrukningen (eller riktningskoefficienten) beräknas görs det på följande sätt:

$$k_{now} = \frac{v_{now} - v_{prev}}{t_{now} - t_{prev}}$$

där $t_{now}$ är aktuell tid,

$t_{prev}$ är tid för förgående uppmätt värde,

$v_{now}$ är den senast uppmätta totala förbrukningen och

$v_{prev}$ är den förgående senast uppmätta totala förbrukningen.

De övriga värdena kan hämtas från databasen. Man bör notera att detta värde är genomsnittsvärdet under tidsperioden.

### 5.5.4 RRDtool-wrapper för grafgenerering

En central del i systemet är den grafgenerering som utförs av RRDtool. För att underlätta användandet av RRDtool, med avseende på grafgenerering, utökades wrappern med en ny metod. Wrappern utökades också med nya komponenter, som styr inställningarna för en graf. Komponenterna är implementerade som klasser och skickas som parametrar till metoden som genererar grafer.

Wrappern har stöd för att visa flera linjer i samma graf, det vill säga att grafen har en eller flera olika datakällor.

#### GraphInfo

Objekt av typen GraphInfo lagrar generell data om en grafs utseende. Grafens fysiska storlek i antalet pixlar är en egenskap. En annan är grafens upplösning i X-led, det vill säga hur stort intervall som ska visas.

Det är också möjligt att lagra information som styr hur linjen ritas. Ett exempel på detta är om linjens ska göras mjukare, eller ritas ut som den egentligen ser ut. Även en beskrivande text kan läggas till, som visas ovanför grafen.

#### Line

Line är en klass vars instanser lagrar information om själva linjens egenskaper. Detta avser dels vilken datakälla som ska användas, samt vilken färg och tjocklek linjen ska ha i grafen.

## 5.6 Stödkomponenter

Systemet har flertalet komponenter som är svårare att dela in i de tre huvudsakliga modulerna. I denna sektion diskuteras några av dem, vars funktion är avgörande för att systemet ska fungera korrekt.

### 5.6.1 Tidskonvertering

En komponent som implementerades för att underlätta utveckling är en konverterare mellan Unixtidsstämplar och den inbyggda klassen `DateTime` i .NET. Mätvärden tidsstämplas i XML-dokumentet enligt UTC med en Unixtidsstämpel. Detta värde är svårläst av människor. Därför konverteras detta värde till det mer läsbara standardformatet DateTime, när värden extraheras från XML-dokumentet.

### 5.6.2 IoC-modul

En IoC-modul, tillhandahållen av Eivind Nordby vid Karlstads universitet, används i systemet vilket ger möjligheten att enkelt byta ut delar av systemet. Modulen används igenom att ett interface skapas som sedan implementeras i olika klasser. Man kopplar i applikationen ett interface till en implementerad klass med metoden
`RegisterType<interface,implementeradklass>()`.
Efter registreringen så kan den statiska metoden
`Resolve<interface>()`
användas vilket returnerar den tidigare kopplade klassen. Detta används i systemet för att lätt kunna byta mellan olika databashanterare. I det nuvarande systemet kan man byta mellan SQLite-hanteraren och MS SQL-hanterare med fyra kodrader och systemet kan lätt anpassas för att använda en extern konfigurationsfil. IoC används även i testsyfte då implementationer finns av databasinterfacen som lagrar data i minnet istället för i en riktig databas.

Implementationen av databasinterfaces har metoder (enligt CRUD) och en statisk lista. När klassen instansieras så verifieras att listan är skapad, och om inte skapas den. Detta gör att man kan skapa många instanser av klassen men listan skapas bara en gång och det är samma lista i alla instanser. Det mest praktiskt vore en klass med en statisk lista och statiska metoder då man då slipper instansiera klassen. Detta är dock inte möjligt om man vill att klassen ska implementera ett interface, då de metoder som implementeras inte får vara statiska.

## 5.7 Webbapplikation

Syftet med webbapplikationen är att presentera mätdata för en användare i en webbläsare. Applikationen är en ASP.NET MVC-applikation som nyttjar AJAX för dynamisk uppdatering av webbsidans olika komponenter.

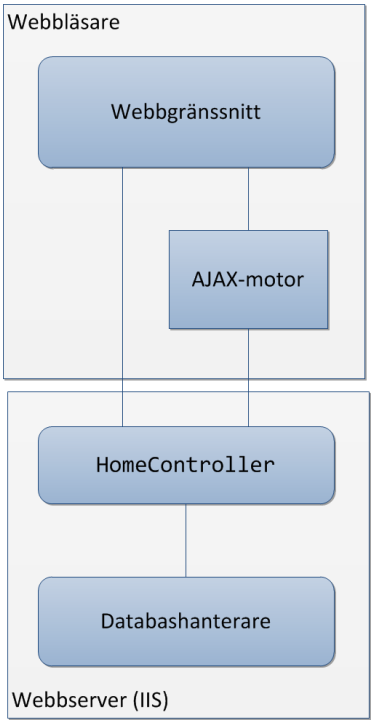
I denna sektion beskrivs webbapplikationens olika versioner, från en simpel sida som visar en graf till den slutgiltiga produkten som är en snarlik replikering av den existerande androidappen, där sidans layout anpassar sig efter skärmens storlek. I denna sektion tas även branding av webbsidan upp.

### 5.7.1 Överblick

Webbapplikationen är byggd med ASP.NET MVC 3. Figur 5.7.1 illustrerar relationen mellan webbapplikationen och det övriga systemet. Arkitekturen är en kombination av MVC och AJAX.

Webbgränssnittet utgörs av en HTML-sida som formateras med CSS. Delar av denna sida genereras på servern med hjälp av C#-kod. Denna kod exekverar då webbsidan för första gången laddas.

Funktionalitet som rör dynamisk uppdatering bygger på AJAX. AJAX-motorn är i realiteten en samling JavaScript-skript som anropas antingen då användaren interagerar




Figur 5.7.1: Webbapplikationens uppbyggnad

med gränssnittet eller efter ett visst tidsintervall för att hålla sidan uppdaterad.

### 5.7.2 Version 1 - Proof of concept

Den första versionen av webbapplikationen byggde endast på MVC. Syftet med denna första version var att visa att underliggande funktionalitet (såsom grafgenerering) fungerar på ett korrekt sätt. Sidans design byggde på den design som automatiskt skapas när ett nytt ASP.NET MVC 3-projekt skapas.

På webbsidan visades till en början en tabell med senaste mätvärden samt en graf, men följdes senare av alla grafer som systemet hade möjlighet att generera, en för varje mätsensor. Majoriteten av logiken var placerad i `HomeController`.

Vid denna version var implementationen av RRDtool-wrapper primitiv. Funktionaliteten för grafgenerering var mycket grundläggande och hade endast de mest nödvändiga funk-

tionerna.

Uppdatering av sidan kunde först bra göras då användaren valde att göra detta. Detta utökades till att, med hjälp av en att inkludera

```
http-equiv='refresh' content='10'
```

i meta-taggen vilket laddar om sidan efter 10 sekunder.

### 5.7.3 Version 2 - Statisk webbsida

Den andra versionen av webbapplikationen var en komplett ombyggnation från tidigare versionen. Designen baserades på Altrans existerande Android-app.

Designen som använts här har följt med i senare versioner, då behovet av en mer omfattande eller komplex design inte funnits. Det var också ett krav från Altran att designen skulle efterlikna Appens design. Designen utgörs av en logotyp som placeras överst. Under logotypen ligger en samling rutor, som presenterar energiförbrukning och temperaturgraf. En väderleksrapport från YR.NO[1] lades även till senare. Till vänster om dessa ligger en meny i form av tabell som visar de senast uppmätta temperaturerna. Figur 5.7.2 visar hur webbgränssnittet såg ut.

För energiförbrukning visas även historisk information i en tabell, med total energiförbrukning dag för dag sedan systemet startades.

Funktionaliteten var begränsad även i denna version. Uppdateringsmetodiken var den samma som den tidigare versionen. Det som saknades var en mer dynamisk uppdatering, likt den i Android-appen. Där uppdateras element i en loop med fem sekunder mellan uppdateringarna.

### 5.7.4 Version 3 - Dynamisk webbsida (AJAX)

Det som saknades för att få en snarlik replika av Android-appen var dynamisk uppdatering, vilket var något som introducerades i den tredje versionen av webbapplikationen. Den

[1] http://www.yr.no


Figur 5.7.2: Webbapplikationens användargränssnitt

dynamiska uppdateringen realiserades med AJAX, som bygger på en samling JavaScript-skript.

Dessa skript anropas dels upprepade gånger med en viss tid emellan, men också då webbsidan laddas om och när användaren trycker på en länk i menyn.

Kontrollen `HomeController` utökades med flertalet metoder som anropas via skripten. Dessa metoder hämtar data från databaserna samt får RRDtool att generera grafer. Datan, som antingen är mätdata, eller länk till bild, skickas tillbaka till webbsidan i form av rå text. Skripten ser till att extrahera informationen som returneras från `HomeController` och uppdaterar webbsidans DOM-träd.

### 5.7.5 Version 4 - Stöd för mobila enheter

Den slutgiltiga versionen inkluderar stöd för flera olika CSS-mallar. Dessa mallar definierar designen för enheter av varierande skärmstorlek. Alla CSS-mallar bygger på den som ursprungligen gjordes för Version 2, med viss modifikation.

Designen för mobila enheter med lägre upplösning utnyttjar bättre skärmens storlek, vilket är den stora skillnaden mellan mallarna. För mindre skärmar finns också en mindre logotyp som inte tar lika mycket plats.

Bytet av CSS-mall och logotyp görs med hjälp av ett JavaScript. Skriptet kan identifiera skärmens upplösning och baserat på fördefinierade gränser göra lämpligt byte.

Alternativet till att byta CSS-fil med JavaScript är att byta hela webbsidan (vyn) på servern. Detta var något som försökte undvikas, på grund av att det innebär ökad komplexitet. Fördelen med detta alternativ är att den anpassade designen byts redan innan sidan laddas. Med den valda lösningen krävs att DOM-trädet byggs innan ändring av mallen görs. Detta resulterar i att den ursprungliga designen hinner visas en kort stund.

### 5.7.6 Branding av webbsida

Ett av kraven från Altran var att webbsidans utseende skulle vara lätt att anpassa för olika kunder. Tre aspekter identifierades:

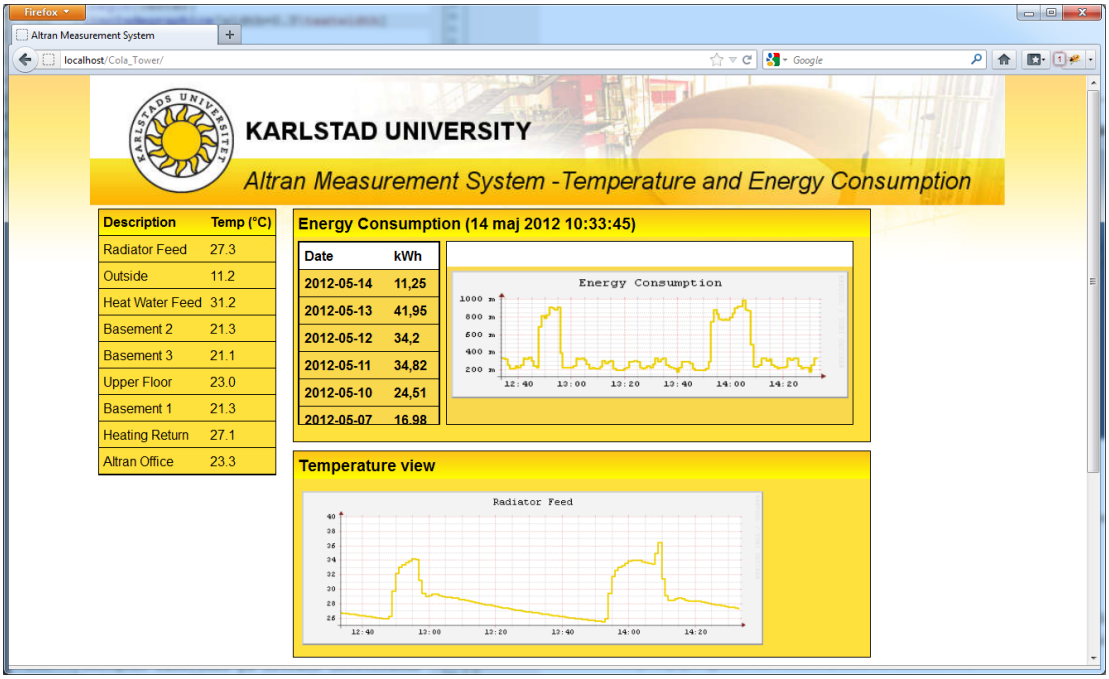
- Färgschema
- Logotyp
- Typsnitt

Det ska vara enkelt att byta dessa tre efter behov, och det löstes med separata CSS-mallar. En CSS-mall inkluderar alla brandingspecifika aspekter. Utöver denna mall finns de mallar som styr övrig design för olika skärmupplösningar. Dessa mallar inkluderar ”branding“-mallen.

Utöver detta så används multipla klasser från CSS-mallarna för att formatera sidans innehåll. Detta är behjälpligt om det finns en ram kring en ruta som har en företagsspecifik färg. Då kan element som kräver denna färg låta använda CSS-klass som specificerar denna ramfärg. Den kan också använda en generell CSS-klass som definierar hur rutan ser ut i övrigt (såsom storlek och positionering) som kan vara baserad på skärmupplösning.

För att byta ut logotyper och grafiska element bestående av bilder räcker det att byta ut dessa bilder mot nya, med samma namn.

Även om den företagsspecifika CSS-filen är kortare än de som bestämmer layout, kan den vara krånglig att modifiera. För att ytterligare underlätta branding, implementerades en simpel Windows-applikation i C#.NET. Applikationen låter användaren välja färger och typsnitt varefter en korrekt CSS-fil automatiskt genereras. Figur 5.7.3 illustrerar en alternativ branding, inspirerad av Kau.se




Figur 5.7.3: Användargränssnitt med alternativ branding

# Kapitel 6

# Utvärdering

*I detta kapitel utvärderas implementationen som beskrivs i föregående kapitel. Även tekniska problem och deras lösningar tas upp.*

## 6.1 Implementation

Vi har lyckats implementera en webbaserad variant av Altrans existerande Android-app. Webbapplikationens grafiska användargränssnitt är en snarlik kopia av Android-appen. Webbapplikationens funktionalitet är mycket likt den i Android-appen. Webbgränssnittets design och layout anpassar sig till webbläsaren i vilken det visas. Detta visade sig vara mycket enklare än vad som antagits i planeringsskedet.

Vi har även lyckats förenkla branding för webbsidan med hjälp av multipla CSS-filer samt en Windows-applikation som automatiskt skapar en CSS-fil med det önskade färgschemat och typsnitt. Denna metodik kan även appliceras för andra projekt.

Utöver detta har vi också implementerat ett system som tar emot mätdata i form av ett XML-dokument. Systemet bearbetar data, som består av mätvärden och metadata om dessa, och lagrar det i två olika databaser. Denna del av implementationen är central för att webbapplikationen ska kunna visa data vid ett senare tillfälle. På grund av dess viktiga

roll, har mycket tid spenderats på denna del. Det är avgörande att systemet inte kraschar, då insamlingen av data skulle upphöra och göra presentationen meningslös.

Systemet kan kantera flera olika källor som tillhandahåller XML-dokument. Det systemet kräver är XML-dokumentets webbadress. Det är också möjligt att XML-dokumenten utökas med fler enheter, vilket automatiskt hanteras av systemet. Om en tillkommen enhet mäter temperatur, kommer den att visas i tabellen i webbapplikationen.

För att systemet ska kunna kommunicera med RRDtool har en så kallad wrapper implementerats. Wrappern utgörs av ett flertal klasser som tillsammans gör det enkelt att specificera hur data ska lagras med RRDtool och hur grafer ska genereras. En stor del av tiden gick till detta moment, då graferna är en viktig del för presentationen av mätdata.

## 6.2 Problem och lösningar

Projektet har kantats av såväl tekniska som icketekniska problem och svårigheter. I denna sektion listas några av de mer intressanta problemen av teknisk natur, samt deras lösning. Dessa kan vara behjälpliga vid en eventuell vidareutveckling av systemet, eller om systemet ska byggas om från grunden.

### 6.2.1 Tidshantering

Det är viktigt, för att grafer ska genereras på ett korrekt sätt, att systemet har en giltig uppfattning om tid. All data tidsstämplas enligt UTC+0, oavsett var datainsamlingen sker. Men även om detta garanterar att data lagras på ett tidsenhetligt sätt, behöver inte presentationen bli korrekt.

Antag att datainsamlingen sker någonstans i Storbritannien. Tidszonen är UTC+0 och data tidsstämplas också i enlighet med detta. Antag sedan att servern som samlar in data är belägen i Sverige, och att datainsamling sker under sommarhalvåret. På grund av tidsskillnaden och sommartid är den lokala tiden UTC+2. Vi kan, för enkelhetens skull,

anta att även webbläsaren som ska presentera data är belägen i Sverige.

Om ett värde mäts klockan 12.00, hur ska det presenteras i en graf? Ska det visas som ett värde insamlat klockan 12.00 eller 14.00? Om vi sedan också antar att webbläsaren där data visas är belägen i Finland, som då har UTC+3 som lokal tid, hur ska värdet presenteras?

Om tidszonen där webbläsaren befinner sig är en annan än den där servern står, måste servern givetvis då, om data ska presenteras i korrekt lokal tid, veta tidskillnaden mellan klientdatorn och UTC+0. Detta löstes med hjälp av JavaScript. Skillnaden i antalet minuter hämtas i JavaScript som sedan skickas tillsammans med övrig data i AJAX-anropen. Skillnaden används senare för att justera mätdata så att tiden visas korrekt för klientens tidszon. Grafgenereringen i RRDtool löstes igenom att sätta en miljövariabel i processklassen som RRDtool körs i.

### 6.2.2 Decimalseparator

Mätvärden som hanteras av systemet måste kunna vara flyttal, om precision är något som anses vara viktigt. Problem uppstod då det var oklarheter gällande vilket tecken som ska användas som decimalseparator. Detta är, om det inte hanteras korrekt, farligt då strängen med mätvärdet ska konverteras till ett flyttal.

Om decimalseparatorn var i form av en punkt och det operativsystem som servern använde var svenskt så kraschade systemet. Operativsystemets språkinställningar avgör vilket tecken som är en giltig decimalseparator, och för svenska system är det komma. Först antogs att decimalseparator skulle vara ett kommatecken, eftersom systemet kraschade när punkt användes. Det fungerade att byta ut punkt mot komma. Men när utveckling skulle fortsätta på en dator med engelska språkinställningar kraschade systemet igen.

När felet identifierats löstes det genom att tvinga systemet att alltid använda punkt som decimalseparator och byta ut komma till punkt vid behov.

### 6.2.3 Bildhantering vid grafgenerering

Det gjordes tidigt ett val att grafer som genereras ska vara personliga för användaren. Med detta menas att grafen och dess namn ska vara unik för varje ansluten användare. Anledningen till detta val var för att kunna utöka systemet med mer användarinteraktivitet. Till exempel var tanken att man via förslagsvis ett webbformulär skulle kunna ändra grafens utseende. På grund av att stöd för detta implementerades, kan man vidareutveckla för att uppnå denna funktionalitet.

Personliga grafer innebar dock ett problem. Varje graf som genereras får ett unikt namn som slumpas fram. Bildfilen sparas på hårddisken. Detta resulterar i att det med tiden skapas ohanterlighet stora mängder bildfiler som aldrig raderas. På lång sikt kan detta få serverns hårddisk att sluta fungera på ett korrekt sätt (ytterligare skrivoperationer blir inte möjliga med mera). Lösningen blev att låta bilder existera en viss tid, varpå de raderas automatiskt. Även om antalet bilder under en kort tid kan öka dramatiskt, kommer de alltid att raderas efter en given tidsperiod.

### 6.2.4 Allmänna säkerhetsrisker

Två allvarligare säkerhetsrisker påträffades och åtgärdades. Den första rörde kopplingen till databasen. På grund av att man kunde med JavaScript och Query-strängar mata in SQL-kod till systemet kunde illvilliga och kunniga användare till exempel radera all data från databasen, eller ta bort databasen helt. Detta kallas SQL-Injektion [49].

För att förhindra attacker mot systemet av detta slag implementerades ett skydd. Skyddet verifierar att SQL-kod inte innehåller tecken såsom enkelcitat på ett ogiltigt ställe. På så vis kan inte kommandon som förstör databasen exekveras.

Det andra säkerhetshålet var mycket allvarligare. På grund av att kommandon till RRDtool körs via kommandotolken kunde även andra systemkommandon köras om ett semikolon matas in följt av det potentiellt illasinnade kommandot. Ett exempel är att låta

kommandotolken gå uppåt i mapphierarkin och där radera mappar såsom Windows eller Program. Säkerhetshålet bestod i att färgkodning för graferna, specificerat med JavaScript, tolkades som strängar rakt av, vilka då kunde innehålla potentiellt skadliga kommandon. Detta löstes genom att färgkoden verifieras, så att den bara kan bestå av bokstäverna a till och med f eller siffror.

## 6.3 Sammanfattning

I detta kapitel har resultat av implementation och utredningar presenterats. Implementationen av systemet och webbapplikationen slutfördes i tid och de krav som ställts har uppfyllts.

# Kapitel 7

# Projektsammanfattning

*I detta kapitel sammanfattas projektet som helhet. Resultaten och slutsatserna summeras. Även förslag på vidareutveckling av implementationen ges.*

## 7.1 Sammanfattning

De krav och mål som sattes upp av Altran i projektets början har uppnåtts om man bortser från tester. Implementationen är slutförd och utredande jämförelser har utförts. Från början antogs att stort fokus skulle legat på implementationen av själva webbapplikationen, vilket inte blev fallet. Fokus låg mer på att utveckla det underliggande system som hanterar mätdata. Mycket tid lades på att utveckla den RRDtool-wrapper som sköter kontakten med RRDtool.

Det var svårt att i ett tidigt stadium göra en korrekt uppskattning av den faktiskta tidsåtgången av de olika momenten. Att lära sig det existerande systemet tog till exempel mycket längre tid än vad som planerats, och författandet av denna uppsats tog längre tid än väntat.

Enligt examensarbetets specifikation och tidsplan skulle en utredning som jämförde XML och JSON göras i ett tidigt skede, för att avgöra vilket format som bäst lämpade sig

för systemet. Detta blev en utredning som hela tiden flyttades framåt i tiden. Anledningen till detta var att behovet för det inte fanns. Systemet tar emot data i form av ett XML-dokument, och ingen anledning till att ändra detta kunde hittas.

Utredningens vinkling ändrades när de väl började utföras. Det blev mer intressant att utreda vilken grafgenereringsmetod som var mest effektiv med avseende på överförd datamängd samt renderingstid i webbläsaren. Bedömning gjordes att JSON var mer lämpat som överföringsformat mellan server och klient. Främst därför att:

1. JSON går att formatera på ett mer kompakt sätt. Datamängden minskar vilket minimerar överföringstiden.

2. JSON är en delmängd av JavaScript och därför mer lämpat då JavaScript ska transformera data till dataobjekt.

3. JSON fungerar bättre med jQuery-biblioteket `flot`.

4. Läsbarhet är av mindre betydelse när data skickas mellan server och webbläsare då data inte är menad att hanteras av människor.

Trots detta gjordes en undersökning som jämförde JSON och XML. Resultatet styrker antagandet att det går snabbare att skicka data med JSON än med XML. Komprimering av XML råder bot på detta, skillnaden mellan de komprimerade formaten är försumbar. Det som kan vara avgörande är att parsning av JSON är snabbare än XML.

En jämförelse av RRDtool och ett jQuery-bibliotek kallat flot gjordes också. Det visade sig att skapandet av grafer med hjälp av RRDtool tar längre tid än att extrahera data och skicka detta till klienten. Dock rör det sig om skillnader i millisekunder, och en användare bör inte uppfatta dessa skillnader.

De jämförande testerna utfördes på en dator och tablet på samma nätverk som servern. Det hade varit bra om testerna gjorts för en enhet ansluten till Internet via till exempel 3G.

Men på grund av svårigheter med att exekvera testerna från en enhet utanför nätverket utfördes inte dessa tester.

Antaganden kan ändå göras. Tid som används för operationer på servern bör vara oförändrade. Tid det tar att skicka data över nätverk borde öka ju sämre anslutningen är. Eftersom operationer för RRDtool huvudsakligen är ”servertid“ borde den totala tiden påverkas mindre än för generering med JavaScript, som är mer beroende av bandbredd och processorkraft. RRDtool kan då antas vara det mest stabila alternativet.

Det finns fler aspekter än prestandan. Funktionalitet och utseende är exempel på två. Det är upp till användaren att avgöra om en graf är estetiskt tilltalande eller inte, men det borde vara enklare att utforma mer användarinteraktiva grafer med JavaScript.

## 7.2 Vidareutveckling

Det finns många intressanta aspekter som inte hanns utreda under projektets gång. Det finns också många olika tillägg till systemet som kan göras i nuläget utan att behöva bygga om systemet från grunden.

Från en användares synpunkt kan webbapplikationen kanske uppfattas som fattig på funktionalitet. Det har i projektets slutskede tillkommit önskemål från Altran. Även tidigare har önskemål framkommit, men de har inte specificerats i kravlistan. Önskemålen rör främst ökad funktionalitet och möjlighet för användaren att själv styra grafernas utseende med avseende på till exempel upplösning och färg. Att kunna ställa in att visa flera linjer i samma graf är också något som diskuterats. Anledningen till att dessa önskemål inte införts i kraven är att de inte är en del av den ursprungliga appens funktionalitet samt på grund av tidsbrist. Det underliggande systemet har stöd för detta, det som saknas är således ett nytt användargränssnitt.

En aspekt som skulle ha undersökts om tiden funnits är säkerhet för överföring av data. I nuläget sker ingen kryptering av data och informationen i XML-dokumentet kan läsas om

nätverkstrafiken övervakas. Data som skickas i nuläget anses inte av Altran vara känslig, och detta är huvudanledningen till att säkerhetsaspekten och kryptering inte undersökts under projektets gång.

Man kan tänka sig att ändra på metodiken kring hur mätdata anländer till systemet. Istället för att låta systemet aktivt hämta mätvärden, kan systemet istället passivt vänta på att det system eller komponent som tillhandahåller mätdata skickar det till systemet. Detta skulle reducera komplexiteten hos systemet, men öka den där mätdata produceras. Den stora fördelen blir dock att insamlingsenheter inte behöver vara publika, det räcker att servern på vilket systemet exekverar är det. Dessutom behöver servern inte känna till enheterna, vilket underlättar om data ska tas emot från flera olika källor.

Det har på flera ställen i arbetets specifikation att systemet ska vara testbart. Test är något som har diskuterats under arbetets gång, men tyvärr tillkom dessa samtal sent i arbetet. Detta gäller även den kravspecifikation som erhölls av Altran. Tester för systemet är något som nästan är nödvändigt för att säkerhetsställa visa av de krav som finns på systemet. Det har under projektets gång producerats en testspecifikation, men inte alla tester har implementerats, mycket på grund av tidsbrist.

## 7.3 Slutord

Projektet har varit lärorikt och involverat flera olika områden, såsom webbutveckling, datakommunikation, systemutveckling och en del kretsteknik. Trots att planeringen inte var exakt, har den i stora drag följts. Det har varit intressant att arbeta mot och hos en produktägare, då man vet att det som producerats kommer användas i framtiden. Det är synd att tiden är begränsad. Om mer tid funnits skulle webbapplikationen kunna utvecklas med mer funktionalitet som produktägaren önskat. Men oavsett om allt som planerats har hunnits med eller inte, blev produktägaren nöjd i slutändan, vilket måste anses vara något väldigt viktigt.

# Referenslista


[1] Dell Precision M4300 Mobile Workstation. http://www.dell.com/us/dfb/p/precision-m4300/pd Besökt 2012-05-23.

[2] GPIO Interfaces. http://www.kernel.org/doc/Documentation/gpio.txt Besökt 2012-05-14.

[3] Posix Time, May 2012. http://en.wikipedia.org/wiki/Unix_time Besökt 2012-05-24.

[4] Alex van den Bogaerdt. RRD Tutorial. http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html Besökt 2012-05-14.

[5] Ben Elgin. Google Buys Android for Its Mobile Arsenal. http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm Besökt 2012-05-14.

[6] Mike Corley. C# Hooks For RRDtool. http://www.codeproject.com/Articles/28763/C-Hooks-For-RRDtool Besökt 2012-05-03.

[7] Cuno Pfister. *Getting Started with the Internet of Things*. O'Reilly, 2011. ISBN: 978-1-4493-9357-1 Tillgänglig online: http://netduino.com/downloads/books/gsiot/Getting_Started_with_the_Internet_of_Things_LED_Controller.pdf.

[8] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informativ), maj 1996.

[9] P. Deutsch. GZIP file format specification version 4.3. RFC 1952 (Informativl), maj 1996.

[10] Paul DuBois. *MySQL (Fourth Edition)* . Addison-Wesley, 2008. ISBN: 0-672-32938-7.

[11] Faraz Rasheed. *C# School*. Synchron Data, 2006.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), juni 1999. Uppdaterad av RFCs 2817, 5785, 6266, 6585.

[13] Free Software Foundation. General Public License. http://www.gnu.org/licenses/gpl.html Besökt 2012-05-22.

[14] Gareth Halfacree. Raspberry Pi interview: Eben Upton reveals all. http://www.linuxuser.co.uk/features/raspberry-pi-interview-eban-upton-reveals-all/ Besökt 2012-05-14.

[15] GHI-electronics. 1-Wire library. http://www.ghielectronics.com/downloads/NETMF/Library%20Documentation/html/b15ea429-dab2-1392-d9b0-b8618f0f24e0.htm Besökt 2012-05-14.

[16] GHI Electronics. EMX Development System. http://www.ghielectronics.com/catalog/product/129 Besökt 2012-05-14.

[17] Grant Allen och Mike Owens. *The Definitive Guide to SQLite*. Apress, 2010. ISBN: 978-1430232254.

[18] J. Garret. *Smashing jQuery*. Wiley, 2011. ISBN: 978-0470977231.

[19] Martin James. *Managing the Data-base Environment*. Prentice Hall, 1983. ISBN: 0-13-550582-8.

[20] Jens Kühner. *Expert .NET Micro Framework*. Apress, 2008. ISBN: 978-1-59059-973-0.

[21] Jeremy Keith. *DOM Scriptning*. Apress, 2005. ISBN: 987-1-59059-533-6.

[22] Jesse James Garrett. Ajax: A New Approach to Web Applications. Februari 2005. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications Besökt 2012-05-24.

[23] Jim Brown. Raspberry Pi Interview With Eben Upton, Mars 2012. http://robots.net/article/3215.html Besökt 2012-05-14.

[24] JSON. JSON. http://www.json.org Besökt 2012-05-14.

[25] Joydip Kanjilal. Entity Framework tutorial, 2008. ISBN: 978-1847195227.

[26] Kenneth Schaefer, Jeff Cochran, Jeff Cochran, Scott Forsyth, Rob Baugh, Mike Everest och Dennis Glendenning. *Professional IIS 7*. Wrox, 2008. ISBN 978-0470097823.

[27] Ole Laursen. flot. http://code.google.com/p/flot/ Besökt 2012-04-20.

[28] Gustaf Liljegren. *XML: begreppen och tekniken*. Studentlitteratur AB, 2003. ISBN: 978-9144065847.

[29] Matt Doyle. Beginning PHP 5.3, 2010. ISBN: 978-0-470-41396-8.

[30] Matthew Humphries. Build the 2006 prototype of Raspberry Pi's $25 PC. http://www.geek.com/articles/chips/build-the-2006-prototype-of-raspberry-pis-25-pc-20111024/ Besökt 2012-05-14.

[31] Maxim. 1-Wire specification. http://www.maxim-ic.com/products/1-wire/flash/overview/index.cfm Besökt 2012-05-14.

[32] Maxim. DS18B20 Programmable Resolution 1 - Wire Digital Thermometer. http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf Besökt 2012-05-14.

[33] Michael McRoberts. *Beginning Arduino*. Apress, 2008. ISBN: 978-1430232407.

[34] Microsoft. Available Web Server (IIS) Role Services in IIS 7.0. http://technet.microsoft.com/en-us/library/cc753473%20%28WS.10%20%29.aspx Besökt 2012-05-09.

[35] Microsoft. Available Web Server (IIS) Role Services in IIS 7.5. http://technet.microsoft.com/en-us/library/cc753473.aspx Besökt 2012-05-09.

[36] Microsoft. Microsoft SQL server. http://www.microsoft.com/sv-se/products/sql-server.aspx Besökt 2012-05-14.

[37] Netcraft. March 2012 Web Server Survey. http://news.netcraft.com/archives/2012/03/05/march-2012-web-server-survey.html Besökt 2012-05-14.

[38] Oracle. MySQL Connector/ODBC. http://dev.mysql.com/doc/refman/5.1/en/connector-odbc.html Besökt 2012-04-11.

[39] J. Postel and J.K. Reynolds. Telnet Protocol Specification. RFC 854 (Standard), maj 1983. Uppdaterad av RFC 5198.

[40] Casey Reas. *Getting Started with Processing*. Prentice Hall, 1983. ISBN: 978-1449379803.

[41] Robert C. Martin och Micah Martin. *Agile Principles, Patterns, and Practices in C#*, chapter 11: The Dependency-Inversion Principle (DIP). Prentice Hall, 2006. ISBN: 978-0131857254.

[42] Samsung. Galaxy Tab. http://www.samsung.com/se/consumer/mobil/mobil/mobilephones/GT-P1000MSANEE Besökt 2012-05-14.

[43] Imar Spaanjaars. *Beginning ASP.NET 4* . Wiley Publishing, 2010. ISBN: 978-0470502211.

[44] Rolf Staflin. *HTML och CSS boken.* Pagina Förlags AB, 2008. ISBN: 978-91-636-0939-82.

[45] T. Berners-Lee, R. Fielding och L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), januari 2005.

[46] Tiobe Software. TIOBE Programming Community Index for April 2012 . `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html` Besökt 2012-04-11.

[47] UltiDev. UltiDev Web Server Pro. `http://ultidev.com/products/UWS-Cassini-Pro/` Besökt 2012-05-09.

[48] Anne van Kesteren. XMLHttpRequest level 2. W3C working draft, W3C, augusti 2009. http://www.w3.org/TR/2009/WD-XMLHttpRequest2-20090820/.

[49] Wikipedia. SQL Injection, 2012. `http://en.wikipedia.org/wiki/SQL_injection` Besökt 2012-04-26.