# VÄDERSTATION



## Alvaro Clauré, 2015

# Sammanfattning

Denna rapport beskriver en applikation som simulerar en värdestation som kan erbjuda (***Publishers***) olika vädertjänster som t.ex. att tillhandahålla aktuellt väderdata, väderstatistik, väderprognoser. Dessa tjänster kan prenumereras (***Subscribers)*** av många klienter.

Applikationen bygger på ett känt mönster: ”***The Observer Pattern*[[1]](#footnote-1)”,** där alla som presumerar på tjänster är ***observers*** (prenumeranter)som ständigt ”lyssnar” på något nytt som publicerats (t.ex. väderdata).

* ***Publishers*** publicerar ständigt (data, nyheter, mm)
* ***Subscribers:*** Prenumeranter får automatiskt nyheter, uppdaterad data, mm

# Innehållsförteckning

**Avsnitt Sida**

1. **Inledning 3**
2. **Avhandling 3**
3. **Slutsats 10**
4. **Referenser 10**
5. **Bilagor 11**

# Inledning

Detta projekt är en del av kursen fördjupad programutveckling och följer den utvecklingsmetod som vi har fått tidigare. Applikationen som beskrivs här försöker adressera den problematik som kan uppstå i relationen mellan de som publicerar/producerar (***Publishers)***  information (väderdata, nyheter, mm) och de som vill konsumera/prenumerera (***Subscribers)***  denna information.

Frågan är hur ska prenumeranter ta del av alla nya uppdateringar som inträffar hela tiden? Ska de ständigt fråga efter nya uppdateringar till producenterna? Detta brukas kallas för pull-modell. Alternativet är push-modellen, där producenterna ”pushar” ny information till alla sina konsumenter oavsett om de har begärt det eller inte.

Båda modeller har sina fördelar och nackdelar. Jag valde push-modellen för att det är lättare att hantera. Denna modell bygger på ett känt mönster: ***The Observer Pattern.***

Vidare är av högsta prioritet att kopplingen (mjuktvarumässigt) mellan ***Publishers*** och ***Subscribers*** ska varaså lös som möjligt med syftet att utbygga applikationen som kan tillåta ändringar på båda sidor (*Publishers/ Subscribers* ) utan att påverka varandra nämnvärt. Samtidigt är kravet att all dataöverföring mellan *Publishers och Subscribers* ska ske omedelbart när nya uppdateringar inträffar.

# Avhandling

## 2.1 Vision

Visionen är att skapa en applikation som agerar som en värdestation som erbjuder många olika vädertjänster. Dessa tjänster ska kunna presumeras av andra aktörer (klienter). En central tjänst ska vara att tillhandahålla aktuellt väderdata, dvs. applikationen hämtar data från olika sensorer (temperatur, fuktighet, tryck) och presentera dessa data till alla som presumerar tjänsten.

Viktigt är att klienten inte ska behöva fråga efter väderdata, utan så fort det sker en ändringen i väderstationen (nya mätningar av sensordata) så fortplantas dessa ändringar till klienterna.

Förutom att tillhandahålla aktuellt väderdata ska det finnas mints två andra tjänster till: *tillhandahålla väderstatistik* samt *tillhandahålla väder prognos*. I framtiden ska det vara möjligt att erbjuda flera andra relevanta tjänster.

Kopplingen mellan den som ger tjänster (väderstation) samt den som tar emot tjänster (prenumeranter/klienter) ska vara lös så att eventuella förändringar som görs på båda sidor ska inte påverka varandra.

Klienten ska kunna prenumerera och säga upp befintliga prenumerationer när som helst. Prenumerationen ska vara väldigt enkelt (inga personliga uppgifter, mm) och alla tjänster erbjuds som gratis.

Till att börja med ska applikationen vara lokal, dvs. det ska köras i Windows miljö med grafiskt gränssnitt. I framtiden ska det dock köras över Internet, dvs. det ska vara webbtjänster som anropas via webben av olika klienter (smart phones, suftplatta, datorer, mm). Vidare kommer en del tjänster att kosta pengar och därmed kräva en mer avancerad prenumerations modell

## 2.2 Kravanalys

**Väderstation**

Applikationen agerar som en värdestation som samlar in data från tre olika sensorer som hela tiden mäter temperaturen, fuktighet samt tryck. I början simulerar vi dessa data genom att vi själva mattar in temperatur, tryck och fuktighet vid olika tid punkter.

**Tjänsten: tillhandahålla aktuell värdedata**

Den främsta tjänsten som applikationen erbjuder är: att tillhandahålla aktuell värdedata. Så fort väderstationen registrerar nya mätningar (temperatur, tryck, fuktighet), dvs. dess tillstånd ändras så sker en uppdatering av detta tillstånd och presenteras (meddelas) det direkt på prenumeranters enhet Enheten utgörs av konsolen i början och sedan ersätts med ett grafiskt gränssnitt.

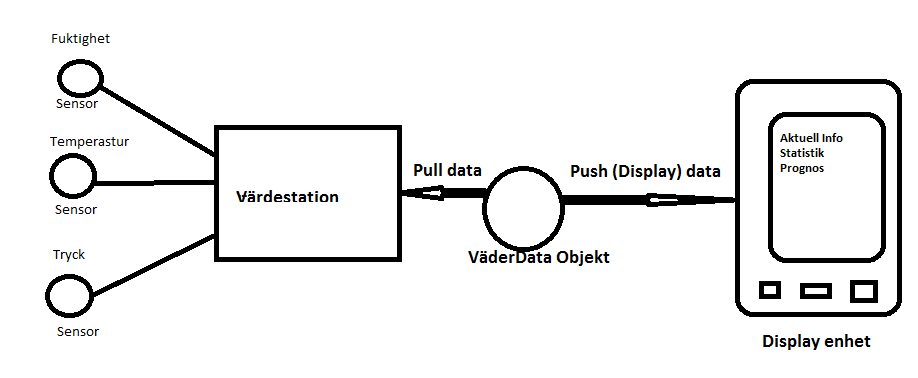
**Två extra tjänster**

Vidare ska två tjänster till förekomma: tillhandahålla väderstatistik samt tillhandahålla väderprognos.

**Prenumeration**

Dessa tjänster liksom den som beskrevs ovan ska kunna prenumereras av andra. Och samtidigt ska man kunna säga upp alla sina prenumerationer.

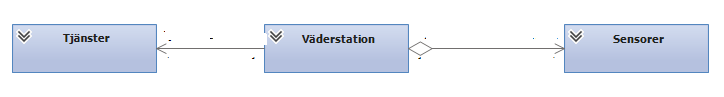
Väderstationens viktigaste uppgift är att samla in sensordata som utgör stationens tillstånd. Tjänsternas viktigaste uppgift är att uppdatera tillståndet (väderdata) från väderstationen och visualisera det på deras enheter

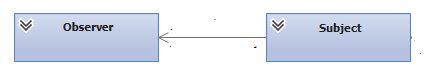


***Fig. 1 En väderstation som mäter sensordata och tillhandahåller olika vädertjänster (Display enhet)***

## 2.3 Domänanalys

Viktiga substantiv som beskriver väl hela verksamheten: S*ensorer* (temperatur, tryck, fuktighet) V*äderstation*, T*jänster* (aktuell väderdata, väderstatistik, väderprognos).





**Fig. 2 Preliminärt domändiagram**

Här väljer jag dock att ändra namn på klasserna värdestation och tjänster. Väderstation bytes till ***Subject*** och tjänsterna till ***Observer***. Varför? Jo. eftersom denna applikation följer ett känt design mönster som heter: ”***The Observer Pattern***”. Och då förekommer dessa begrepp: Subject/Observer. Tanken är att alla som presumerar på tjänster är ***observers*** (prenumeranter)som ständigt ”lyssnar” på något nytt. Eller ett annat sätt att beskriva: det finns ***Publishers*** och ***Subscribers.***

* ***Publishers*** publicerar ständigt (data, nyheter, mm)
* ***Subscribers:*** Prenumeranter får automatiskt nyheter, uppdaterad data, mm

## 2.4 Användningsfall

Viktiga användningsfall som beskriver hur systemet ska fungera.

**Prenumerera tjänst**

* Användaren får en lista på alla befintliga tjänster som alla är gratis
* Användaren väljer en eller flera befintliga tjänster och anmäler sig som prenumerant.
* Användaren behöver ej ange några personuppgifter eftersom prenumerations modellen är väldigt enkelt.
* Systemet ger inte någon bekräftelse pga. prenumerations modellen är väldigt enkelt.

**Säga upp tjänst**

* Användaren väljer en tjänst som prenumererar och säger upp den med omedelbart verkan
* Systemet kollar att det finns tjänsten och om det finns tar bort den

**Uppdatera Aktuell väderdata**

* Användaren får automatiskt uppdaterade värden av temperatur, tryck och fuktighet.
* Användaren får en visualisering av aktuell väderdata
* Systemet ”pushar” ständigt sensordata till samtliga prenumeranter så fort det registrerar nya mätningar.

**Alternativ scenario**:

Användaren kan ej få denna tjänst pga. det finns ingen prenumeration

Ingen uppdatering pga. mätning av sensordata funkar ej

**Uppdatera Väderstatistik**

* Användaren får automatiskt uppdaterade medelvärde, maximumvärde och minimivärde av temperatur och tryck.
* Användaren får en visualisering av aktuell väderstatistik
* Systemet ”pushar” med mellanrum uppdaterade medelvärde, maximumvärde, minimivärde av temperatur och tryck till samtliga prenumeranter.

**Alternativ scenario**:

Användaren kan ej få denna tjänst pga. det finns ingen prenumeration

Ingen uppdatering pga. mätning av sensordata funkar ej

**Uppdatera Väderprognos**

* Användaren får automatiskt en väderprognos.
* Användaren får en visualisering av aktuell väderprognos
* Systemet ”pushar” en väderprognos som gäller högst i tre dagar till samtliga prenumeranter

**Alternativ scenario**:

Användaren kan ej få denna tjänst pga. det finns ingen prenumeration

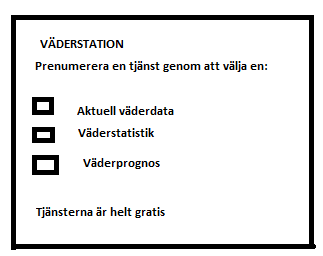
Ingen uppdatering pga. mätning av sensordata funkar ej

# 

**Fig. 3** **Fem användningsfall**

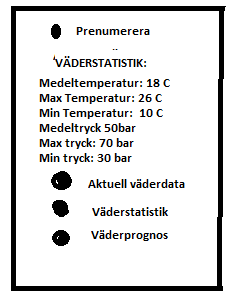
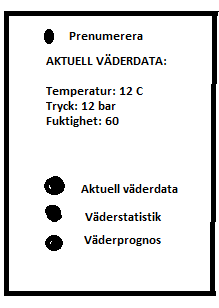
## 2.5 Användargränssnitt

Väderstationen ska visa följande gränssnitt:



**Fig. 4 Grafiskt gränssnitt**

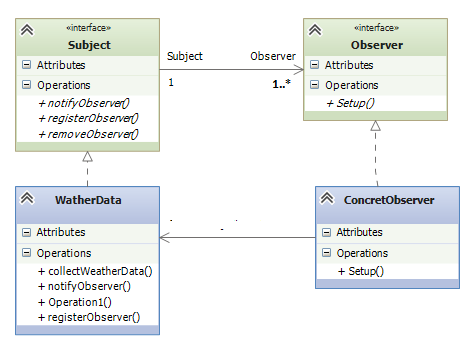
Tjänsterna (i detta fall två av tre) ska visa följande gränssnitt:



***Fig. 5 Tjänst: Aktuell väderdata Tjänst: Väderstatistik***

## 2.6 Klassdiagram

Från domänanalys och de olika avvisningsfallen förfinar vårt klassdiagram enligt följande:



**Fig. 6 Förfina klassdiagram**

Det är en till många relation mellan *Subject* och *Observe*r (Subject känner till många observers). Det är alltså en association (känner till relation. Ej agregering eller komposition) mellan Subject och Observer pga. det ska vara lös koppling mellan dem.

### Subject

Subject är en Interface så att alla som ärver Subject ska implementera:

* + - *registerObserver (Erbjuder prenumeration)*
    - *removeObserver (Ta bort prenumeration)*
    - *notifyObserver (Uppdatera tillstånd)*

Värdestations (Subject) viktigaste uppgifter (oavsett vad det mäter) är alltså att ge möjligheter till att: *prenumerera tjänst, ta bort prenumeration* samt *uppdatera väderdata* (tillstånd). Av denna anledning har jag valt Subject som Interface eftersom det tvingar alla som använder Subject att implementera dessa viktiga egenskaper

Den som implanterar Subject (Concret Subject) har som sin viktigaste uppgift att *samla in* (collect) väderdata

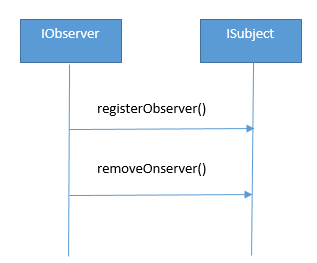
### Observer

Observer är en Interface som tvingar alla som ärver det att implementera Setup() (uppdatera väderdata) eftersom det är grunden för alla tjänster. Observers viktiga uppgift är alltså att ta emot uppdateringar från värdestationen (Subject).

De som ärver Observer kan vara olika tjänster

## 2.7 Sekvensdiagram

Några metoder som behövs för att komplettera klassdiagrammet



**Fig. 7 Sekvensdiagram**

Observer ska kunna registrera sig (prenumerera) hos Subject eller ta bort prenumeration

SensorSata

WeatherData

getTemperature()

getHumidity()

getPressure()

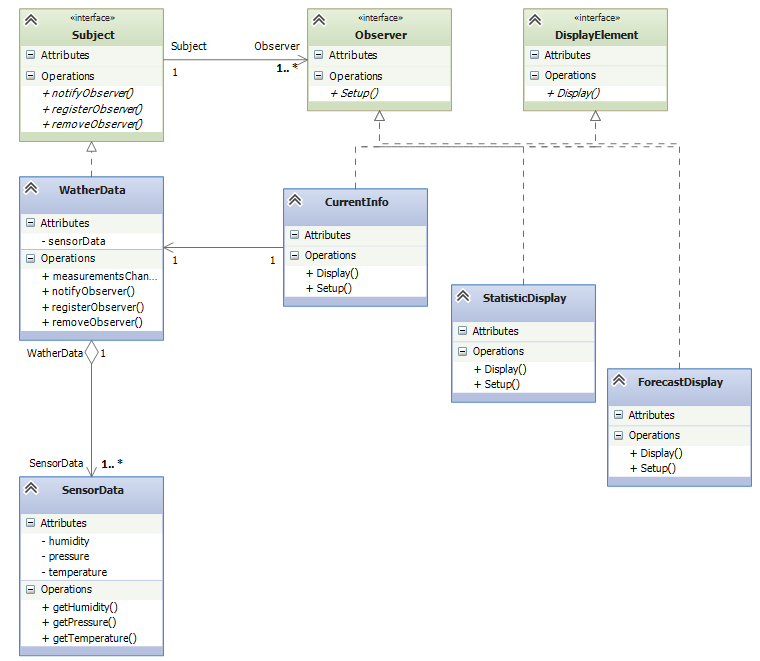
**Fig. 8 Sekvensdiagram**

Väderstationen hämtar data från sensorerna genom att anropa *getTemperature, get Humidity, getPressure.*

## 2.8 Uppdaterat klassdiagram

Klassdiagrammet har utökats med tre tjänster: ***CurentInfo*** (Aktuellt värdedata), ***StatisticDisplay*** (väderstatistik) samt ***ForecastDisplay*** (väderprognos)

Dessa tjänster måste implementera ***Interface Observer*** samt *Interface* ***DisplayElement***eftersom dess viktigaste uppgift är att *uppdatera* och *visualisera* (display) värderdata. Tjänsterrna uppdaterar och visualiserar väderdata på sitt eget sätt.



**Fig. 9 Uppdaterat klassdiagram**

***WeatherData*** är den delen av värderstationen som samlar in väderdata (*temperature, humidity, pressure*) genom att mäta data från sensorerna. Detta görs med hjälp av metoderna: **getTemperature, getHumidity, getPressure.**

WeatherData ärver ***Interface Subject***, vilket innebär att WeatherData även måste implantera: ***registerObserver*** (Prenumerera), ***removeObserver*** (Ta bort Prenumeration) samt ***notifyObserver*** (Meddela prenumeranter)

# Slutsats

Detta projekt har tagit mer tid än vad jag hade räknat med. Jag skulle ha implementerat tjänsten prognos (se klassdiagram) samt utvecklat ett grafiskt gränssnitt, men tiden räcker inte för dessa iterationer. Dock uppfyller detta projekt med råge alla krav för moment 4, tycker jag.

# Referenser

<https://en.wikipedia.org/wiki/Observer_pattern>

# Bilagor

Användarhandledning

1. [↑](#footnote-ref-1)