Fontys ICT

OBSERVER PATTERN

Design patterns

Jan-Niklas Schneider, Georgiana Manolache
9-14-2016

Table of Contents

1 Introduction

The goal of this document is to give an overview over the observer pattern by giving an example implementation which leverages real weather data gathered through the *OpenWeatherMapApi*. Furthermore, reusability, extensibility, and maintainability of this pattern are elaborated. Also, the implementation, its unit test and graphical user interface (GUI) are reviewed.

2 Observer Pattern

The observer pattern is a software design pattern which consists most commonly of two objects, the subject and observer. The pattern describes a relationship of one subject and zero, one, or more observers. The observer can attach or subscribe to the subject and if the subject's data source changes the subject notifies all its observers about the change.

The main benefit the observer pattern provides is a clean separation between business logic and display layer.

3 Implementation

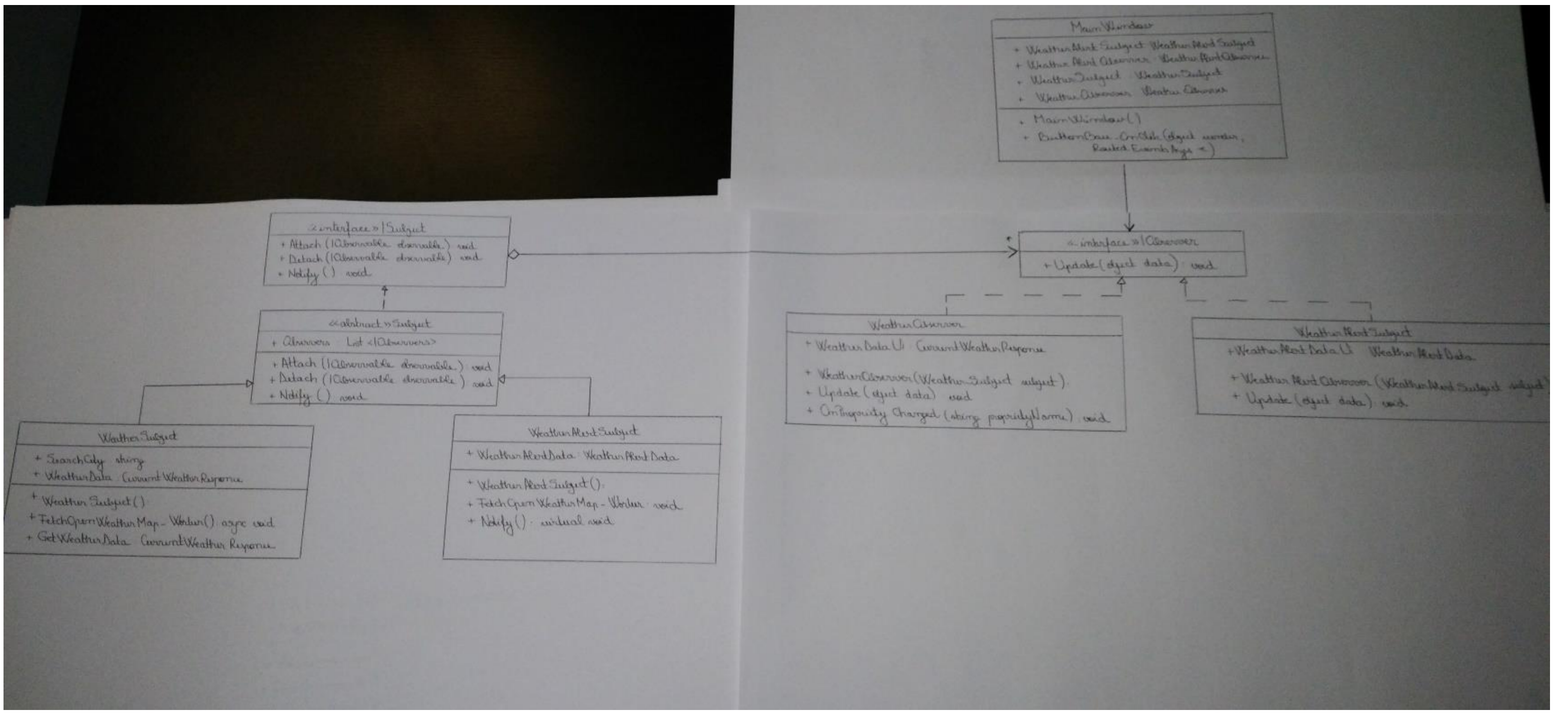

MainWindow
+ WeatherAlertSubject WeatherAlertSubject
+ WeatherAlertObserver WeatherAlertObserver
+ WeatherSubject WeatherSubject
+ WeatherObserver WeatherObserver
+ MainWindow()
+ ButtonBase_OnClick(object sender, RoutedEventArgs e)
«interface» ISubject
+ Attach(IObservable observable) void
+ Detach(IObservable observable) void
+ Notify() void
«abstract» Subject
+ Observers List<IObservers>
+ Attach(IObservable observable) void
+ Detach(IObservable observable) void
+ Notify() void
«interface» IObserver
+ Update(object data) void
WeatherObserver
+ WeatherDataUi CurrentWeatherResponse
+ WeatherObserver(WeatherSubject subject)
+ Update(object data) void
+ OnPropertyChanged(string propertyName) void
WeatherAlertSubject
+ WeatherAlertDataUi WeatherAlertData
+ WeatherAlertObserver(WeatherAlertSubject subject)
+ Update(object data) void
WeatherSubject
+ SearchCity string
+ WeatherData CurrentWeatherResponse
+ WeatherSubject()
+ FetchOpenWeatherMap_Worker() async void
+ GetWeatherData CurrentWeatherResponse
WeatherAlertSubject
+ WeatherAlertData WeatherAlertData
+ WeatherAlertSubject()
+ FetchOpenWeatherMap_Worker void
+ Notify() virtual void

FIGURE 1: CLASS DIAGRAM

The figure above depicts a class diagram of the implementation of a WPF application that showcases weather data in real time.

3.1 Explanation of classes

This subchapter gives a descriptive explanation of the observer patterns implementation, such as methods, properties or fields.

3.1.1 ISubject

ISubject defines the subject interface.

ISubject		
Type	Definition	Explanation
Method	`+ Attach(IObserver observer): void`	Attach an observer to the subject.
Method	`+ Detach(IObserver observer): void`	Detach an observer from the subject.
Method	`+ Notify(): void`	Notify all observers of data source change.

3.1.2 Subject

Subject is an abstract class which implements *ISubject*.

Subject		
Type	Definition	Explanation
Property	`+ Observers : List<IObserver>`	List of attached observers..
Method	`+ Attach(IObserver observer): void`	Attach an observer to the subject.
Method	`+ Detach(IObserver observer): void`	Detach an observer from the subject.
Method	`+ Notify(): void`	Notify all observers of data source change.

3.1.3 WeatherSubject

The *WeatherSubject* implements the abstract class *Subject*.

WeatherSubject		
Type	Definition	Explanation
Field	`- _weatherData : CurrentWeatherResponse`	Holds weather info fetched from OpenWeatherMap API.
Field	`- _weatherMapClient : OpenWeatherMapClient`	Issues Http request to OpenWeatherMap API.
Property	`+ SearchCity : string`	Search term for API.
Property	`+ WeatherData : CurrentWeatherResponse`	Public property of _weaterData.
Method	`- FetchOpenWeatherMap__Worker : void`	Calls the API in an
Method	`+ GetWeatherData() : CurrentWeatherResponse`	Provides callback for observer to retrieve data.

3.1.4 WeatherAlertSubject

The *WeatherAlertSubject* implements all methods form abstract class *Subject*. This is the second subject which is implemented in a pull request method.

WeatherAlertSubject		
Type	Definition	Explanation
Field	`- _weatherAlertData : WeatherAlertData`	Holds weather alert data

Property	`+ WeatherAlertData : WeatherAlertData`	Public property of _weaterAlertData.
Method	`- FetchWeatherAlert__Worker() : void`	Generates weatherAlertData.
Method	`+ Notify() : void`	Notifies subscribeds observers about data change.

3.1.5 IObserver

IObserver defines the observer interface.

IObserver		
Type	Definition	Explanation
Method	`+ Update(object data) : void`	Subject calls update to pull or push data to an observer.

3.1.6 WeatherObserver

Implements the IObserver interface.

WeatherObserver		
Type	Definition	Explanation
Field	`- _weatherDataUi : CurrentWeatherResponse`	Holds observer's data.
Field	`- _weatherSubject : WeatherSubject`	Attached subject.
Property	`+ WeatherDataUi : WeatherAlertData`	Public property of _weatherData
Method	`+ Update(object data) : void`	Subject calls update to pull or push data to an observer.

3.1.7 WeatherAlertObserver

Implements the IObserver interface.

WeatherAlertObserver		
Type	Definition	Explanation
Field	`- _weatherAlertData : WeatherAlertData`	Holds observer's data.
Field	`- _weatherAlertSubject : WeatherAlertSubject`	Attached subject.
Property	`+ WeatherAlertDataUi : WeatherAlertData`	Public property of _weatherAlertData
Method	`+ Update(object data) : void`	Subject calls update to pull or push data to an observer.

3.2 Features

The application uses real-time data collected from a *OpenWeatherMap* API (OpenWeatherMap, Inc, 2016). Furthermore, user can look up weather information from any city, by entering the city name in the top hen side text box.

Due to limited information about with regions weather alert, identifying weather alerts was simulated within the application. This feature is supposed to prove the *push* implementation of the *Observer pattern*. Thus, the application implements both *pull* and *push Observer pattern* requests.

4 Design choices

The implementation of the assignment has been done with regard to reusability, extensibility, and maintainability.

The observer pattern seen in the implementation above exhibits these points in different ways. Reusability can be observed in the class *Subject*. The abstract class derives from the interface *ISubject*.

The pattern is decoupled; hence it is easily maintainable. The pattern allows sending data to other objects effectively without changing the *subject* or *observer* (*WeatherAlertSubject*, *WeatherSubject*, *WeatherAlertObserver*, *WeatherObserver*).

The *WeatherObserver* implements the *pull* request from the *Observer pattern*. *WeatherAlertObserver* class is implemented using *push Observer pattern* request. Firthermore, the data *pulled/pushed* to the observer binds directly to the UI by *INotifyPropertyChanged;* there are no intermediate methods to.

5 Graphical user interface

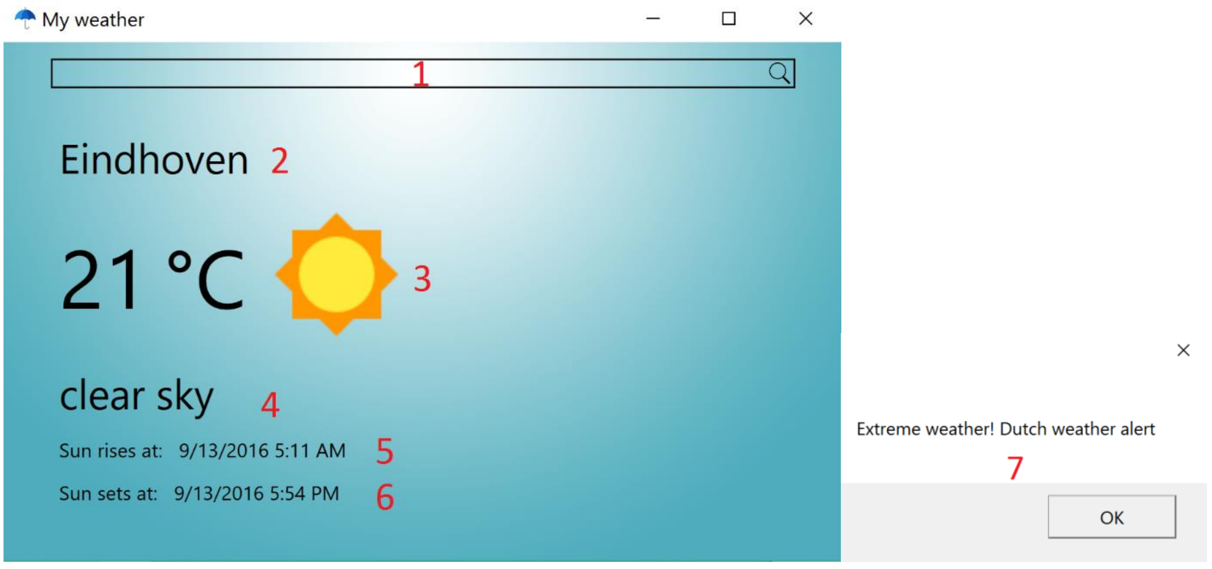

My weather
1
Eindhoven 2
21 °C
3
clear sky 4
Sun rises at: 9/13/2016 5:11 AM 5
Sun sets at: 9/13/2016 5:54 PM 6
Extreme weather! Dutch weather alert
7
OK

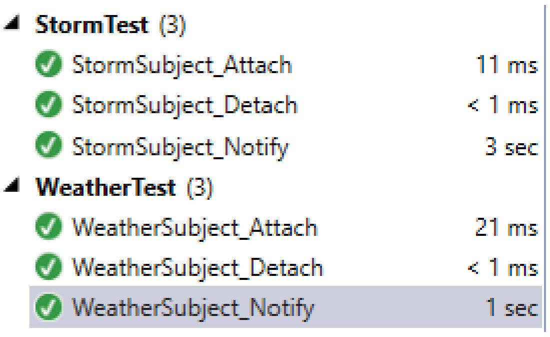
FIGURE 2: USER INTERFACE OF THE APPLICATION

The above figure depicts the user interface where red numbers indicate functionality or controls. More precisely these are:

1. City name can be inserted in the textbox, e.g.: Amsterdam, Eindhoven, London. Once the city name is inserted, the button search is pressed and the UI showcases data from the city.
2. The name of the city.
3. The degrees in Celsius.
4. Short description of the weather status.
5. Extra information
6. Extra information
7. When the push happens, the message box pops up to notify the user of weather alert. This runs every 5 seconds, to just show the *push* request of the *Observer pattern*.

6 Unit tests

For each implemented strategy unit tests have been defined to assert correct behavior. Six tests can be found in *DprObserverPatternTest*, namely Storm tests: *StormSubject_Attach, StormSubject_Detach, StormSubject_Notify* and *Weather tests: WeatherSubject_Attach* , *WeatherSubject_Detach, WeatherSubject_Notify*. Consequently, all test ran successfully.



```
◢ StormTest (3)
  ✓ StormSubject_Attach       11 ms
  ✓ StormSubject_Detach      < 1 ms
  ✓ StormSubject_Notify       3 sec
◢ WeatherTest (3)
  ✓ WeatherSubject_Attach     21 ms
  ✓ WeatherSubject_Detach    < 1 ms
  ✓ WeatherSubject_Notify     1 sec
```

7 References

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns.* O'Reilly Media.

MSDN Microsoft. (2016, September). *Observer Design Pattern*. Retrieved from MSDN Microsoft: https://msdn.microsoft.com/en-us/library/ee850490(v=vs.110).aspx

OODesign. (2016, September). *Observer Pattern*. Retrieved from OODesign: http://www.oodesign.com/observer-pattern.html

OpenWeatherMap, Inc. (2016, September). *Current weather data*. Retrieved from API: http://openweathermap.org/api

SourceMaking. (2016, September). *Observer*. Retrieved from SourceMaking: https://sourcemaking.com/design_patterns/observer