Software Design Document

For a Weather Monitoring System

The project for the course: Software Architecture

15 November 2013

Table of Contents

List of Figures

1.0 Introduction

1.1 Purpose

This Software Design Document (SDD) is the outcome of a project assignment, which supports the learning of the course named Software Architecture (SA). The main purpose of the SDD is to design the major architecture for a fictitious Weather Monitoring System (WMS), to give proper explanation and description of the design, and to rationalise the design by providing synthetic analysis. The WMS is a system that keep tracks of climate data and show the result of its detection to the user of the system. To manipulate the system, an architecture need to be designed depending on the requirements specified by the project description, which was provided by the lecturer of SA. In terms of accomplishing the required project design, the SDD will provide a system overview for the WMS, an analysis for the architectural decision-making and architectural patterns design, and detailed descriptions and design pattern explanations for different components that organise the system.

1.2 Scope

The scope of this SDD is based upon the requirements of the WMS provided by the project description. According to the description, the WMS has the following attributes that need to be implemented:

1. The WMS is a system with mainly four exterior sensors, which are Temperature Sensor (TS), Humidity Sensor (HS), Barometric Pressure Sensor (BPS), and Wind Speed and Direction Sensor (WS), to collect the data for interior system to analyse.

2. There is a User Interface (UI) in the interior system to display the values returned by the four sensors in real time. Either to display the values once a

new reading is made by the sensors or periodically reading the values of sensors is acceptable.

3. The system should be able to provide the last 24 hours' climate data in a line chart when the user requests to see it, which requires database to store the historical data from sensors. As the requested historical data is not too long for storing (just need the database to keep 24 hours' data), the project description, which can be treated as a Software Requirements Specification (SRS), recommended to use Proxy design pattern to implement this mechanism of the system.

4. Due to some unknown factors, the hardware of the WMS is not fixed. The software has to work with a hardware which is initially in a lower cost configuration and will be upgraded to a higher technical specification later. All the drivers and driver Application Programming Interfaces (APIs) will be rewritten for the newly upgraded hardware during the realistic usage of the system. Therefore, the software of the system should have the capability to adapt to the flexibility of the above circumstance, which, in practice, requires the design of the software to cope with different types of hardware independently with regard to architectural perspective.

The elements stated above are all the requirements for the system so far. Some essential parts of the system should also have been included in the SRS was not stated in the project description, including the requirements for the data, the requirements for the UI, the hardware specifications, and the resources required for the software design. The SDD will not discuss the above mentioned elements as a consequence of the absence of them in requirements. This document will focus on the architectural design of the system particularly based on the requirements listed in last paragraphs and lighten other parts that should be appeared in a normal SDD.

Ultimately, the software architecture and components' design should be able to implement the software into the WMS that can read data from exterior sensors, transfer the data from exterior sensors to interior machines, mutually communicate and interact with each other, and display the information on UI when the user requests.

1.3 Overview

Following this introduction, the SDD comprises the following:

- **Systems Overview** gives a broad outline of the architecture;
- **System Architecture** provides a detailed description of the architecture, including its composite components, design rationale, and possible alternatives;
- **Component Design** introduces the detailed design for different components in the architecture, including Class, State and Interaction Diagrams.

As for the data design, UI design, and other essential parts of a typical SDD, this document will just ignore them according to the reasons discussed in the last section.

1.4 Reference Material

[1]Garlan, D. and Shaw, M. (1994) *An Introduction to Software Architecture*. Research Showcase, Carnegie Mellon University. Available from: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1720&context=compsci. [Accessed 14 November 2013].

[2]Shaw, M. and Clements, P. (1997) *A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems*. 1997 IEEE, vol. 0730-3157/97.

[3]Developer Network (2013) *Chapter 3: Architectural Patterns and Styles*. Microsoft. Available from: http://msdn.microsoft.com/en-us/library/ee658117.aspx. [Accessed 14 November 2013].

[4]Gamma, E. *et al.* (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley Longman Publishing Co., Inc.

[5]Qian, K. *et al.* (2010) *Software Architecture and Design Illuminated*. Jones and Bartlett Publishers.

[6]Kanasz, R. (2012) *Design Patterns 2 of 3 - Structural Design Patterns*. Available from: http://www.codeproject.com/Articles/438922/Design-Patterns-2-of-3-Structural-Design-Patterns. [Accessed 16 August 2013].

[7]Booch, G., Martin, R. C., and Newkirk, J. (1998) Chapter 3 of *Object Oriented Analysis and Design with Applications*, 2nd ed. Boston: Addison Wesley Longman, Inc.

1.5 Definitions and Acronyms

API	Application Programming Interface
BPS	Barometric Pressure Sensor
DA	Data Archiving
DC	Data Collection
DD	Data Display
DP	Data Processing
FIFO	First In First Out
HS	Humidity Sensor
SA	Software Architecture
SDD	Software Design Document
SRS	Software Requirements Specification
TS	Temperature Sensor
UI	User Interface
WMS	Weather Monitoring System
WS	Wind Speed and Direction Sensor

2.0 System Overview

The language to develop the system is Java in terms of its popularity and the engineers' familiarity with it.

The whole WMS consists of four parts, which are Data Collection (DC), Data Processing (DP), Data Archiving (DA), and Data Display (DD), based on the working progress and functionality of the WMS. Each part of them can be layered for the data flow through the system.

From the very beginning, the four sensors are comprised by DC package and collect all the data from the environment to get ready for next layer of the system.

DP contains two smaller packages to accept the data from DC and transfer data to DA and DD. One package in DP deals with the real time display data on UI and transfer data directly to DD, whereas the other package manipulate the data structure for the historical track of sensors' data and process the designated data into DA waiting for the calling of DD.

DA package stores the data of the historical track of sensors and integrate all the data into a new data structure that can be read and displayed by DD.

DD package it the top layer of the system to handle the UI and the integration for the data transferred from DP and DA so that it can display the data in an appropriate format in the UI to the user of the system, which is also the principal usage of the WMS.

2.1 System Functioning

The WMS is organised by a four layered system to process the data from the sensors to the UI displaying information to the user. Once the system is open, the launch starts from the root layer, DC. The sensors' drivers will work here to detect the data for the specified sensors. This layer is highly dependent on hardware. As the hardware for the system is flexible, the software of this layer is also subject to change.

The DP layer works independently from DC layer so DP can still work properly when the software changes in DC. DP contains a timer to fetch data from DC with a mechanism that no matter how the software changes in DC it can always fetch data in a simultaneous pace for all the sensors. Behind the timer is the data check mechanism for the data fetched from the timer. It will check all the data come through here and branch the data to different places. The data check also synchronise and integrate data from different sensors and deliver the data to the other two layers in a synthesised way. All the real-time changes will be sent to DD layer and all the data collected following one hour sessions will be treated as historical track data and be sent to DA layer to store.

DA layer contains three components to receive the data, store the data and interpret the data to DD layer. The data store part only store the past 24 hours' data passed from DP layer, because the historical track of UI only needs to display the information for the last 24 hours. Once a new reading is passed from DP, the data store will delete the earliest record and add the new record in. The interpretation part will organise all the data stored in data store into a data structure readable for DD so it can visualise the collected data when DD wants to display the historical track.

DD will deal with the data transfer both from DP and DA. A receiver in DD will receive all the real-time data transferred from DP and integrate the data into a visualisable data structure for the UI to display. The historical track dealer will get the data structure shaped in DA and render it to the paintable components for the UI. The UI part of the DD will just visualise all the data from DP and DA on the display. The visualisation of data from DA will not be implemented unless the user requests to do so.

3.0 System Architecture

3.1 Architectural Design

The pattern used for the whole system is Layered Architecture style, which separate the whole system into four layers as described in System Overview (Figure 1). The Layered Architecture cluster all the packages around the system into four collections based on their functionality. One layer of the system fulfil one particular function in the system and finally organised the whole system to run into the proposed structure. The layers of the system also describe the data flow of the system, from data collection to visualisation display.

In terms of the communication between the layers, this architecture propose pipe-and-filter to deal with the data flow from the sensors to the display. The pipe-and-filter pattern can ensure that the upper layer of the system can get the required data from lower layer by using “filter” inside each layer. The fluent communication between different layers can also be guaranteed by providing pipes across layers to process data. Because of the requirements for the system in dependancy, different layers and components need data transformation to make sure data structures are synchronised in different independent parts. The pipe-and-filter pattern gives the proper control of the data for display, the end purpose of using this system.

Except for the DC layer, the whole system need to give control to the data flow and transform the data into readable information for display. For this purpose, a MVC pattern is designed for the system. The DP layer is the control phase in MVC to process data and make sure the data is provided in a controlled manner. Furthermore, the DA layer and the data structure part in DD act as the model role in MVC to store data and provide an appropriate data format to display. The data integration part and UI in DD perform the display in MVC.

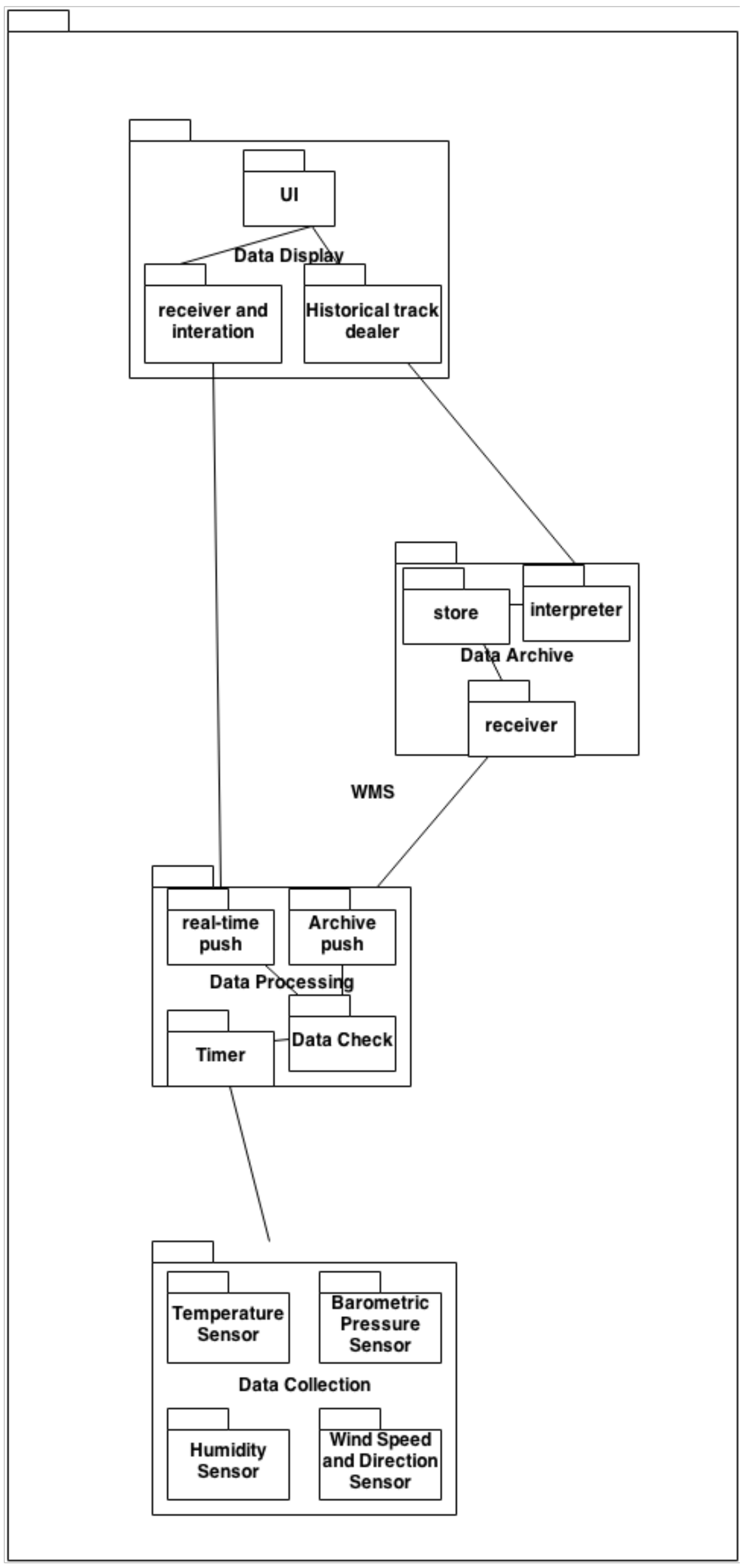


Figure 1. The architecture overview

3.2 Decomposition Description

All the explanations for why the whole system is designed as described above and how the higher level architectural styles are chosen will be discussed in next section, Design Rationale. The analysis for architectural alternatives will also be discussed later. However, this section is going to describe and discuss the decomposition of the system in a comprehensive way. All the rationales for the design of different components will be given inside this section in case of overlapping contents for upcoming sections.

The decomposition description will focus on middle level of the software architecture and the description scope will be inside individual layers of the system. The purpose of this decomposition description is to give detailed structure of the architecture and describe different functionality of different components inside the structure. Some particular parts in the system, such as the communication between DP and DC, the details for DP and data check, and the interaction between DA and DD need to be described in more detailed scale and will not be inclusive in the decomposition section. The the next chapter, Component design, will discuss the design for the above mentioned components in class level and give more detailed explanation for those mechanisms. In this section, decomposition designs are described in three parts, the DP design, the DA design, and the DD design.

1. The data processing design (focus: timer and data check)

The DP layer is mainly comprised by two parts, the timer to get data from DC layer and the data check to push data to DA layer and DD layer. (Figure 2) The DP layer actually plays the "filter" in pipe-and-filter pattern for the data communication through the whole system.

For the sensor reading details in DC layer and how the timer communicate with DC layer, please see the chapter of Component Design for detailed reading.

Here in this section it will only explain how the timer works within DP layer. For the usage of timer, this document chose to periodically read the sensor to get a comprehensive data from all the sensors to the monitor display. This architecture choose a timer to control the sensor reading to organise the flexible hardware and sensors in a comprehensive way so the structure and codes will not change too much if the hardware or the sensors change. The timer has an interface that contains Observer for sensors and AlarmClock to decide the time for reading. The Observer is designed for the timer in case of adding new sensors or the modification of hardware system. By using Observer pattern[1], the timer will not contact with the sensors directly. Once the DC layer changes, the DP layer just need to add a new Observer in timer. The Observer pattern here is to deal with the uncertainty of the hardware system. The AlarmClock control the time when the DC layer read data and transfer to DP layer. Normally, different sensors detect the environment dynamics in different paces. For example, the temperature sensor can detect the environment in every 1 minutes, whereas the barometric pressure sensor will not detect the environment in every 5 minutes because the transformation rate of barometric pressure is lower that temperature. Therefore, the AlarmClock need to choose a proper time (normally this time is the smallest common multiple for all the sensor detecting rate) to read all new data from sensors simultaneously. The AlarmClock has its own algorithm to calculate the time for real-time sensor reading by giving the detection rates of all the sensors as input, so the AlarmClock is also compatible for different hardware systems. In order to read the data for historical track, the AlarmClock also has the other rate to read data specifically for DA layer. Therefore, the AlarmClock has two rates to read data and gives the data two different labels for data check process to identify the difference. One rate is the real-time data reading rate depended on the algorithm's calculation while the other rate is constant for the AlarmClock to control the sensors to read data every one hour.

Later the data check process will identify the destination for the read data by getting the label from AlarmClock. After the determination for the destination, the DP layer will use a method called send to send the data to the proposed destinations. After the DP layer, data has been filtered and put into the pipes to the designated destinations.

2. The Data Archive Design (focus: data structures)

The DA layer will receive the data transferred from DP layer and integrate the data into a comprehensive data structure which contains all kinds of climate data at one time. (Figure 3) The comprehensive data structure will be stored in a queue in the store part of DA with a First-In-First-Out (FIFO) rule. The length of the queue is 24 records to store the last 24 hours' date collected and transferred from the previous layers. Once the number of records exceed the length of 24, the first stored record will be deleted according to the FIFO rule and the requirement that only 24 hours' data is needed for the historical track.

Once the user wants to see the historical track of the last 24 hours' climate information, the DD layer will send a request to DA layer. Later, the interpretation part of DA will start to work. This part of DA will shape a new data structure, which organises all the 24-hour comprehensive records together, for processing to the DD layer to display. The DD layer will not directly use the data structure because the DD layer need to have a control for historical track (the data will not be fetched into DD layer unless request is sent). Here a Proxy pattern[2] is used to implement the organised comprehensive data structure into the DD layer. Detailed diagrams for the Proxy pattern will be showed in the Component Design chapter.

In the DD layer, all the collected and transferred data will be visualised through the UI for the user to see and control. The existence DA layer is for the better control of data to display, which implements the model in MVC pattern.

3. The Data Display Design (focus: real-time data structure and UI)

The DD layer has two ports to receive the data transferred from outside packages and the UI to visualise the data to the user. (Figure 4) As the historical track data has already been arranged into the required data structure for displaying, the historical track part of DD just receive the proxy data structure transferred from DA layer and manipulate the visualisation functionality for the data structure in JPanel (Java graphics component). The purpose of this design is for the convenient implementation in UI. This mechanism can get the visualisation of the historical track ready for UI to invoke and simplify the codes in UI (the painting of the historical track may require long codes to implement).

The DP layer can send the data collected by different sensors together to DD layer without the integration for all the data. To display the sensors' data together in the UI, a data structure to comprehend the real-time data is needed. The receiver of the DD layer will get all the data transferred from DP layer can transform the raw material data into the comprehensive data structure. The data structure will organise all the data for the processing in the display. For instance, the Barometric Pressure itself will have no meanings in display if the trend of it is not calculated for the user to evaluate the weather. Therefore, the data structure will organise the data into the format that will make sense for display.

The UI finally will show the data on screen to the user and perform the end of the data flow. Because the project description has few requirements for the UI, this document will not introduce the design for UI in particular.

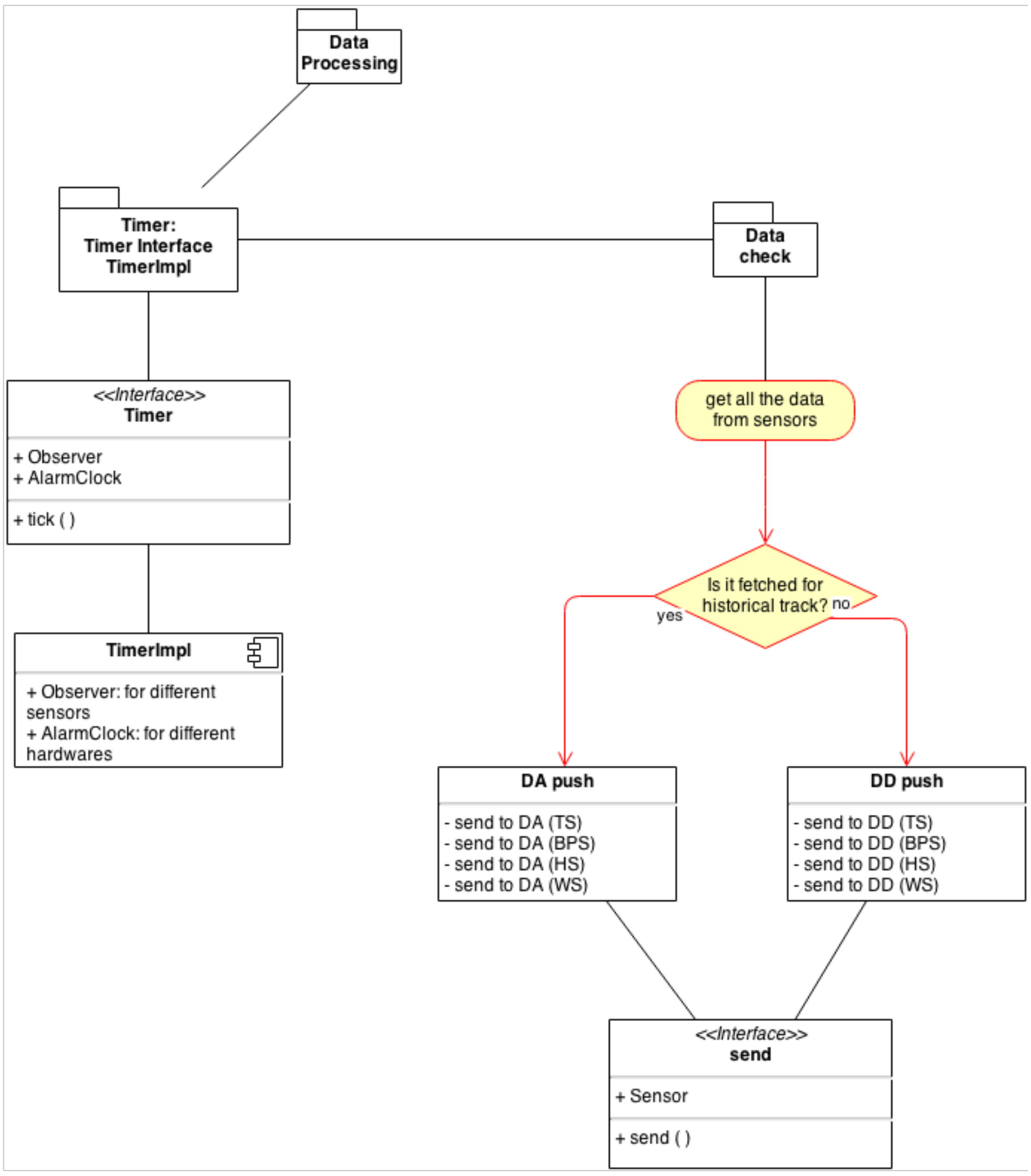


Figure 2. The Data Processing layer

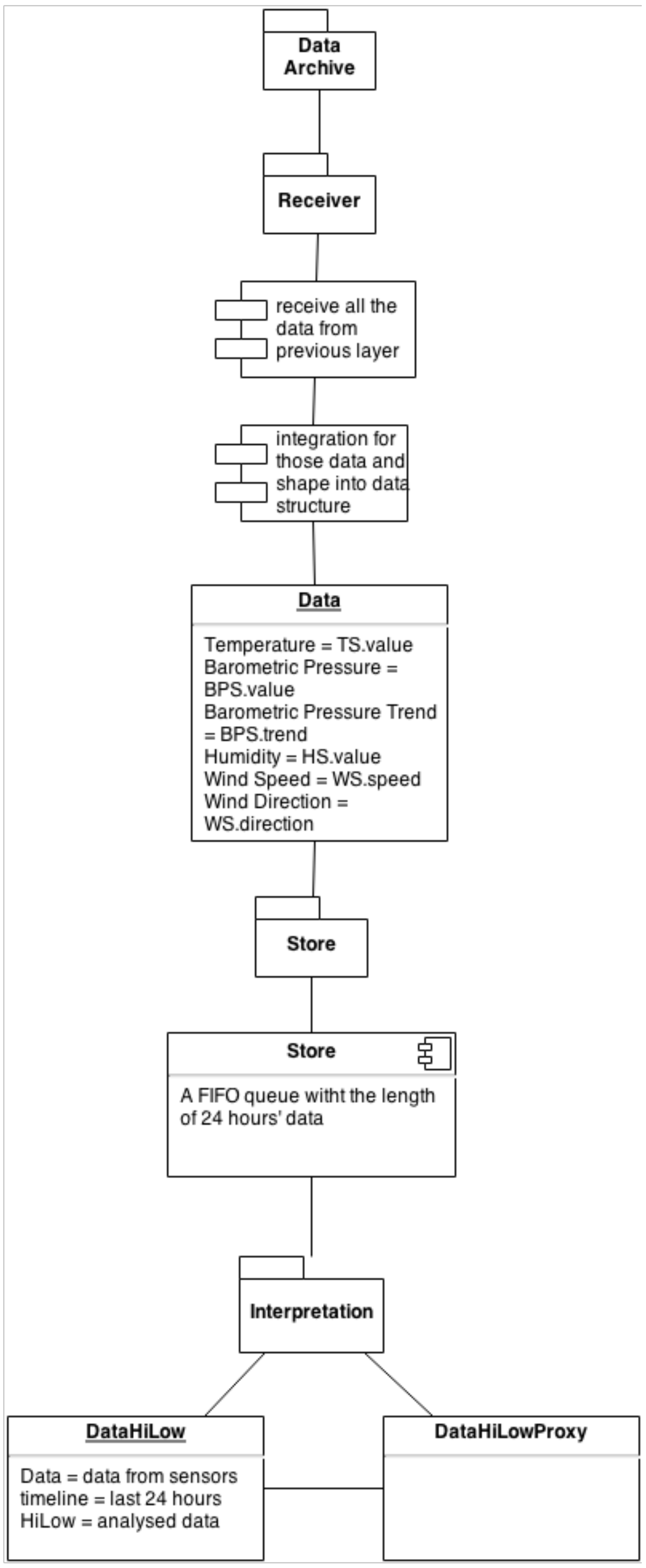

Data Archive
Receiver
receive all the data from previous layer
integration for those data and shape into data structure
Data
Temperature = TS.value
Barometric Pressure = BPS.value
Barometric Pressure Trend = BPS.trend
Humidity = HS.value
Wind Speed = WS.speed
Wind Direction = WS.direction
Store
Store
A FIFO queue witht the length of 24 hours' data
Interpretation
DataHiLow
Data = data from sensors
timeline = last 24 hours
HiLow = analysed data
DataHiLowProxy

Figure 3. The Data Archive layer

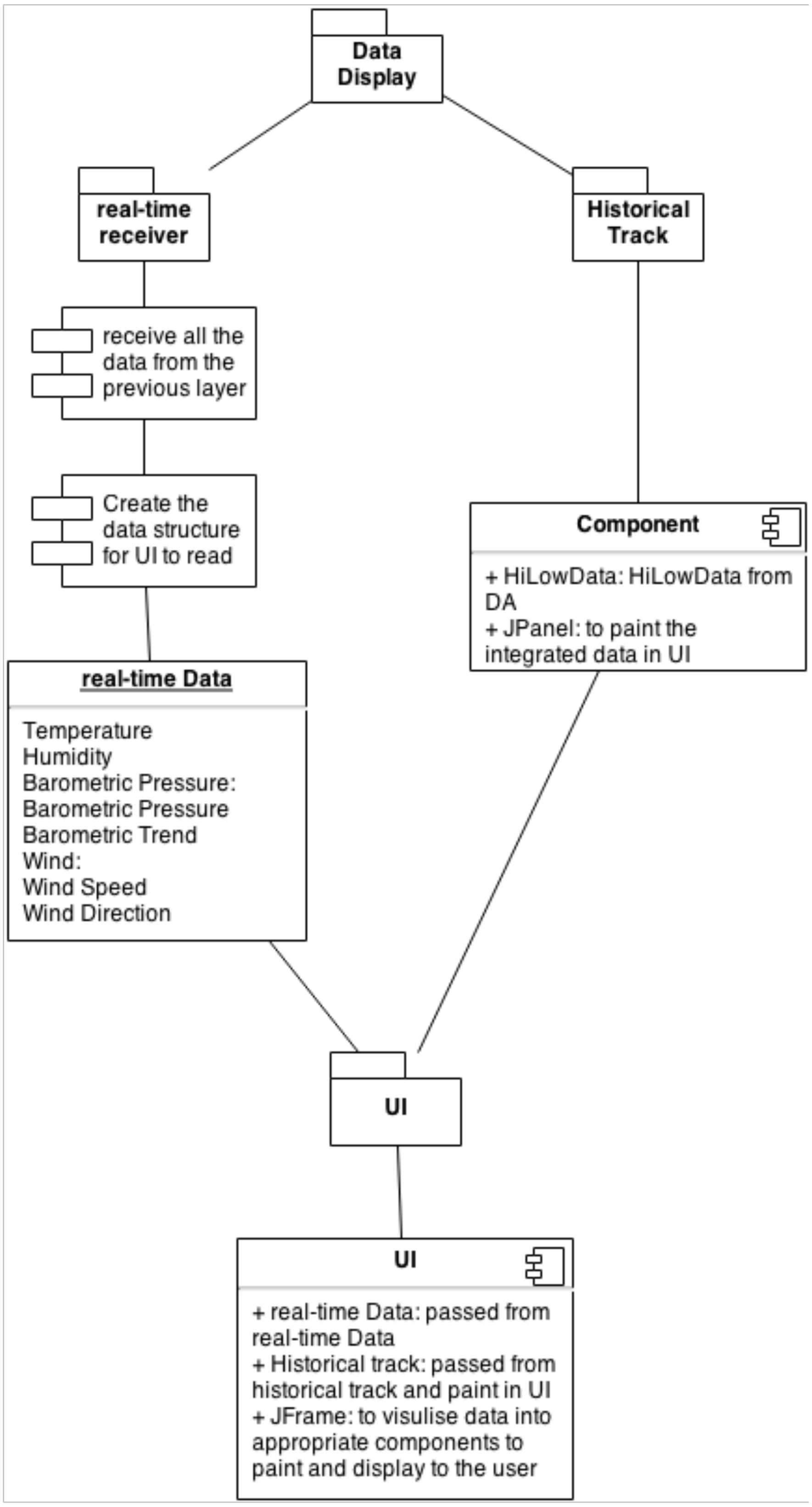

Data Display
real-time receiver
Historical Track
receive all the data from the previous layer
Create the data structure for UI to read
Component
+ HiLowData: HiLowData from DA
+ JPanel: to paint the integrated data in UI
real-time Data
Temperature
Humidity
Barometric Pressure:
Barometric Pressure
Barometric Trend
Wind:
Wind Speed
Wind Direction
UI
UI
+ real-time Data: passed from real-time Data
+ Historical track: passed from historical track and paint in UI
+ JFrame: to visulise data into appropriate components to paint and display to the user

Figure 4. The Data Display layer

3.3 Design Rationale

The system chose Layered architecture style for the whole software design, pipe-and-filter for the data flow between different parts of the system, and MVC for the manipulation of the whole required mechanism. The reasons to choose those patterns are on the subject of the nature of the patterns themselves and the requirements for the system gathered from the assignment description.

Layered style[3] focuses on the grouping of related functionality within an application into distinct layers, which rationalise the decision making of architectural style that functionality of WMS within each layer is related by a common role or responsibility according to the requirements. Based on the rule of layered style that components in layers will not interact with the higher layer, Figure 1 showed a layered structure where the topper layers have lower level and lower layers have higher level in the layered hierarchy. All the lower layers just interact with topper layers to indicate that rule. The WMS structure is also a relaxed layered style because DP layer interacts with two lower levelled layers. The advantages to choose layered style includes Convenience in partial replacement, lower the dependence on components in the system, and better performance on reuse, which cope with the situation that the hardware of the system is subject to change. Layered style also have the drawbacks of low efficiency, which can be illustrated from the frequent communications between different layers.

The pipe-and-filter pattern[4] is used for the data flow inside the system, which means the system has several filters to process and prepare the data for other components and the major data transferring format for the system is stream. The timer and data check in DP layer, the DA layer, and the receiver in DD layer are all filter simulations to provide analysed data for other components to process. In those "filter"s, computation to enrich and add data, removing to refine data, transformation of data, decomposing and merging data are processed to ensure easily maintenance, allowance of specialised analysis such as throughput and deadlock analysis, and separation of tasks within the system. Pipe is also implemented for the system for easy data communicating from the source to sink, and buffer are easily implemented with pipe, especially when the system need to buffer the historical track to put it on display after requests. The pipe-and-filter pattern is the mechanism to enable “real-time” for the system to prepare and communicate data in different components. This pattern also absorb resources as it needs resources to implement the filter and organise the buffer.

MVC pattern[5] is partially used for the system because not all the functionalities of MVC are used in the WMS. MVC supports multiple states in the system and multiple events based on these states to implement complicated interactions between the system. However, the WMS only has one state, “receiving data from sensors”, and two events, “get the real-time data and historical track data to display”, within the system. Based on the requirements of the system, the usage of MVC is not a redundancy here. The system does need “Control” to get proper data for real time and historical track data display and the “Model” to hold those data for the convenient expression by UI. Furthermore, if the system is going to expand the functionally and the complexity, MVC can still use for the new system. Therefore, MVC makes the system extensible, especially when the system is going to be upgraded to a newer high cost hardware depending on the requirements.

3.4 Architectural Alternatives

Normally there are no certain architectural styles and design patterns for one system. There is also no designated fixed architecture for the same requirements of the system. For this WMS, there are simpler structures to implement the same mechanism. For example, the system does not need to provide a comprehensive data structure to store and process the collected data. The sensor can be directly connected to the monitor display to do the ultimate usage of the system. The system just needs to implement some adapter and bridge patterns for the independency and the stability of the different components. As described in the requirements, the system just has four sensors, which means the directly connection architecture will not be too complicated to implement. Even the above proposed structure may consume fewer resources than the design that introduced in this document, it can cause some explicit drawbacks for the system. First of all, without a comprehensive implementation of the data, it will be difficult to check the smoothness of the data flow and the correctness of the processes. Once some bugs occurred, there is not a systematic way to detect it in a looser architecture. Secondly, the organised architecture is more understandable for human so people can easily build the blueprint for the whole system and understand the manual of the system in a shorter time. With clusters within functionality and organised data structures, the designer can build meaningful implementations, add reasonable extensions, and design effective debugging sessions. What's more, as discussed before, the structured and organised architecture can be easily extended within the original framework. No matter the changes to the system are system upgrading, sensor addition or modification, or sophisticate the mechanisms, the implementations will not change the architecture and codes frames too much and flexibly complete the modifications to the system.

There are also other alternatives for the architecture and data flow control. As discussed in last section, MVC is a pattern that more suitable for Event-driven style. However, this document does not design Event-driven style for the system. Event-driven style can get fully use of MVC pattern and help the system to deal with more complicated situations, what's more, the Event-driven style is more extensible for the system to be configured on more complicated hardware. Nevertheless, the system has some implicit reasons to implement Event-driven style. As discussed in previous sections, the system only has one state and two events to implement the data manipulation and the information display. For the basis of the system so far, the Event-driven style will be used redundantly and costly. Even the system add some more complicated modifications, the basic operation that arbitrary data transformation will not be changed for a long time. Arbitrary data transformation does not need too much concern on Event-driven style, so the current designated architectural style for the system is enough even for further updating and modifications.

There are also other patterns for data transfer inside the system such as Batch Sequential pattern. In the perspective of data processing, pipe-and-filter is more suitable for the system than Batch Sequential. The Batch Sequential requires the system to wait for all the results in one process, whereas pipe-and-filter can start to work just after the results produced. For the WMS, all the sensors just start to work once they are powered. After the regular work of the sensors, they will continually produce the collected data from the exterior to interior. Also, the system works from the DC layer to the DD layer arbitrarily. Apparently, pipe-and-filter is more suitable for the WMS to continuously process the data produced by sensors once they start to work.

4.0 Component Design

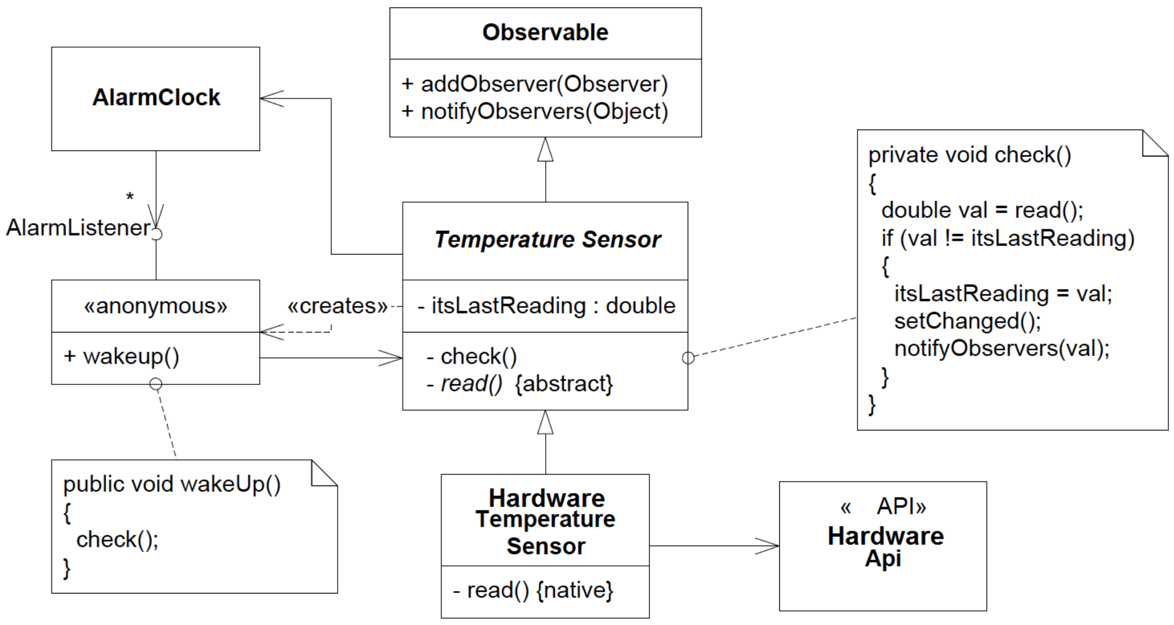
4.1 Interaction Diagrams

In this section, the design for DC layer, the communication mechanism for the DP and DC to adapt to the flexibility of the hardware system, and the communication mechanism for the DA and DD to adapt to the implicit mechanism to fetch historical track data will be introduced and discussed.

The components in DC layer are the sensors' classes based on the driver and APIs for the hardware sensors. Because there is no further information given in the assignment description, we can assume the classes for sensors normally manipulate one function called "read ( )" to wait for the AlarmClock to "wakeup

( )” and “update ( )”. Because the hardware system of the WMS is subject to change, the communication mechanism for AlarmClock and the Sensor classes need to be independently fulfilled by using adapter patterns, bridge patterns, and more importantly the template method patterns.[6] Figure 5 shows the classes and interaction diagram for the circumstance stated above. The Temperature Sensor is treated as an example for the mechanism. Other sensors just work similarly. The DC layer has its own sensor class to implement the functions provided by the hardware APIs. The Observer object and AlarmClock object from DP layer just work independently with DC layer to get the data they needed. (Later they will organised together in DP layer as discussed in Architectural Design section.) Additionally, the AlarmClock implements AlarmListener and anonymous wakeup between the DC layer sensor class to work independently in case of the system changes.

If the DA layer and DD layer want to implement persistent mechanism for the historical track data transfer and the data must be transferred under the requests from DD layer, a proxy pattern is needed for the mechanism.[7] The reasons for using proxy have been discussed in Architectural Design section. In Figure 6, the interaction diagram has shown the detailed proxy pattern in class level. The “Java.io.serializable” helps keep track of the data structure and the proxy object helps the data structure to transfer from DA layer to DD layer. The “StorageKey” can help the system to get correct data from DA. If the DD layer requested multiple times for DA layer, the DD can still identify each requested data by using “StorageKey”.



```
private void check()
{
  double val = read();
  if (val != itsLastReading)
  {
    itsLastReading = val;
    setChanged();
    notifyObservers(val);
  }
}
```

```
public void wakeUp()
{
  check();
}
```

Figure 5. Template method to decouple the timer from sensor

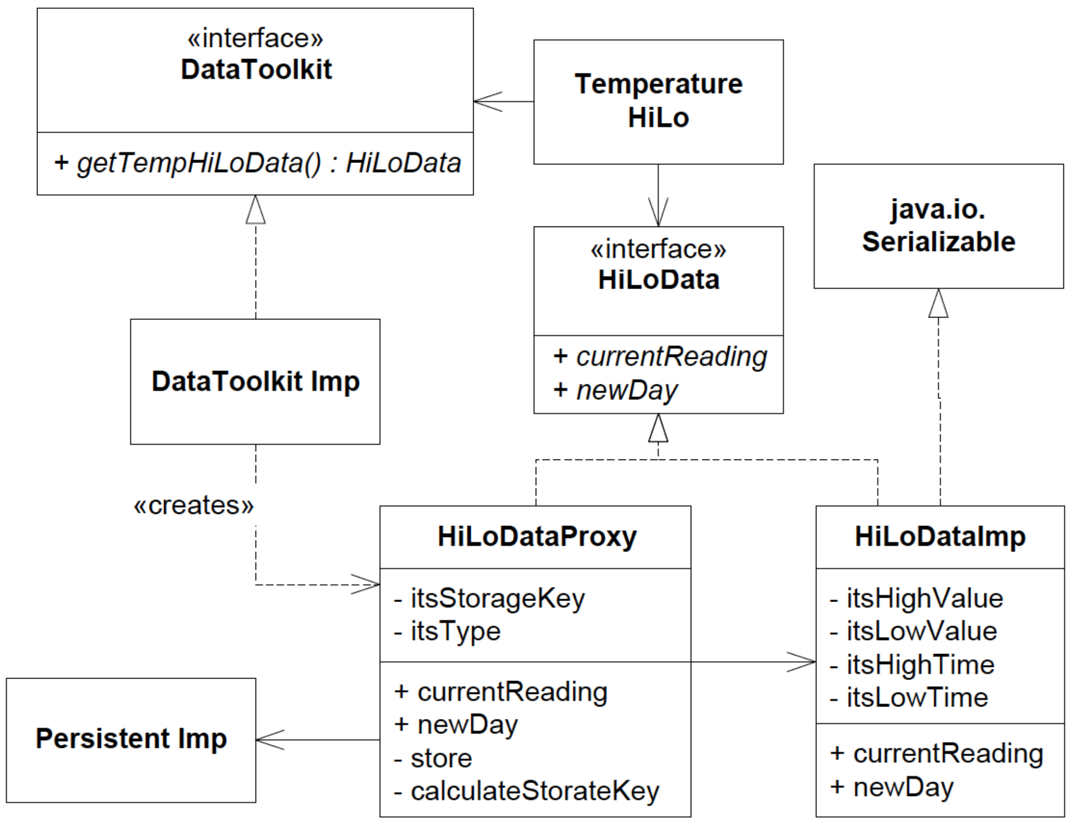


Figure 6. The proxy pattern to deal with the historical track data

5.0 Conclusion

All the above discussions are the architectural design for the WMS so far. All the designs emphasise the rationale around the requirements for the system, which can be simply described as hardware system flexibility and different data structure for display. The architectural design, from high level styles to middle level structure until low level patterns, just followed the requirements and also gives the system extensibility and simplicity for human understanding. Due to the absence of some requirements for the system, the Data Design phase and the User Interface Design phase are not discussed in this document. If time is allowed and more information is given, the SDD can be performed in a more detailed and comprehensive manner.