University of Groningen

Software Patterns
Team 2

Home Energy Monitoring System

Authors
Putra, Guntur
Fakambi, Aurélie
Schaefers, Joris
Menninga, Wouter

Monday 7th December, 2015

Version 0.4

Authors

Name	E-Mail
Putra, Guntur	G.D.Putra@student.rug.nl
Fakambi, Aurélie	A.Fakambi@student.rug.nl
Schaefers, Joris	J.Schaefers@student.rug.nl
Menninga, Wouter	W.G.Menninga@student.rug.nl

Revision History

Version	Author	Date	Description
0.1	Schaefers	15-11-15	Added a stakeholders section
	Schaefers	15-11-15	Added a key drivers section
	Putra	15-11-15	Added Introduction and System context
	Fakambi	15-11-15	Added non-functional req. and risks
	Menninga	15-11-15	Added high-level and functional req.
	Menninga	15-11-15	Improvements to risks
0.2	Schaefers	20-11-15	Added an initial list of patterns with description
	Schaefers	22-11-15	Improved the analysis chapter, partially restructured and expanded explanations
	Menninga	21-11-15	Improvements to High-level requirements
	Menninga	21-11-15	Improvements to Functional requirements
	Menninga	21-11-15	Added system context / elaborated model
	Putra	22-11-15	Added hardware architecture
	Putra	22-11-15	Edited introduction
	Fakambi	22-11-15	Use-cases
	Menninga	22-11-15	Expanded system context / elaborated model
	Menninga	22-11-15	Improvements to use-cases
0.3	Menninga	28-11-15	Improved Unit of Work and Service Layer in ch. 3
	Menninga	29-11-15	Changed key drivers
	Menninga	29-11-15	Added/modified Layers pattern ch. 3
	Menninga	29-11-15	Added Unit of Work/Service Layer/Layers to ch.6
	Putra	29-11-15	Edited introduction and hardware architecture
	Putra	29-11-15	Added MVC and Template View pattern in chapter analysis
	Putra	29-11-15	Added elaborated architecture of MVC and Template View pattern
	Fakambi	29-11-15	Improvements to the use cases
	Fakambi	29-11-15	Added the shared repository pattern
	Menninga	29-11-15	Improvements to Function req. and use cases
	Menninga	29-11-15	Improved the Elaborated Model
	Schaefers	29-11-15	Updated the analysis chapter and added the front page pattern. Added the front page structure image to the software chapter
0.4	Schaefers	6-12-2015	Restructured the software chapter
	Schaefers	6-12-2015	Enhanced the software chapter. Added description and images for layer function interaction and data flow using pipes
	Putra	5-12-15	Updated introduction chapter
	Putra	5-12-15	Updated MVC diagram in software architecture chapter
	Putra	6-12-15	Improved Non-functional requirements
	Putra	6-12-15	Added system evolution
	Putra	6-12-15	Added Functional requirement evaluation
	Putra	6-12-15	Added Non-functional requirement evaluation

Version	Author	Date	Description
	Putra	6-12-15	Added illustration for MVC and Template View pattern
	Menninga	3-12-15	Improved stakeholder/keydrivers and added stakeholder concern matrix
	Menninga	3-12-15	Improved Service Layer pattern implications
	Menninga	5-12-15	Completed pattern fields in ch. 3
	Menninga	5-12-15	Added interoperability NFRs
	Fakambi	6-12-15	Improvements use-cases
	Fakambi	6-12-15	Shared Repository Pattern and work on evolution
	Menninga	6-12-15	Improved arguments/alternatives Broker, Data Mapper, Domain Model ch. 3
	Menninga	6-12-15	Added Use Case for creating alert
	Menninga	6-12-15	Added evaluation for Unit of Work, Broker, Layers and Service Layer

Contents

List of Figures

List of Tables

1 Introduction

This document describes the software architecture of the Home Energy Monitoring System (HEMS) as the first assignment of Software Pattern course. We proclaim ourself a software architect team working on the Home Energy Monitoring System.

Energy is one of the main concerns in a household. The problem is how to monitor energy usage of a particular house in order to prevent inefficient consumption of energy and even to figure out how to save energy, resulting in lower household expenses for energy. An example of energy consumption for a house is shown in Figure 1.1, which indicates that electricity is one of the main energy consumption of a house. Thus, in the initial product this system focuses only on the electricity usage.

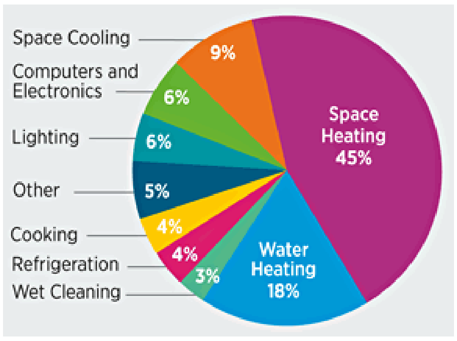

Figure 1.1: An example of home energy consumption [8].

This system will allow the user to collect and store their electricity consumption. This system will also have a web dashboard that shows the graph of electricity consumption, which the user can adjust the time range of the graph to see different data, and other useful data such as which device uses the electricity the most, etc. The data, which will be stored in our database, will be used to do analysis that will help user to predict the upcoming electricity bill in the next month and other required analysis. This system also provides a fault-tolerant computing platform to compute these analyses.

The data collection is not our part, we leave this portion of the system to the third party developers. Basically, we provide a online software service of cloud computing environment, which can be categorized as SaaS[1], to deliver energy management system. SaaS describes when users rent or borrow online software instead of actually purchasing and installing it on their own computers [7].

Figure 1.2 depicts the architectural vision of the HEMS.

[1]Software as a service

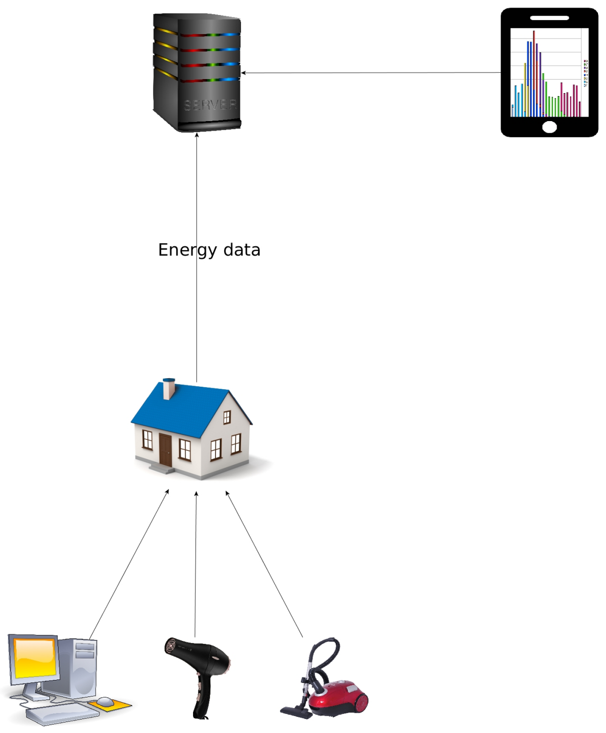

Figure 1.2: Architectural vision

This system provides monitoring for more than one house. Many houses can connect to the system through Internet. Sensors will be located at every device that users are willing to monitor. Real time measurement of electricity consumption will be sent to the HEMS server via Internet protocol by the sensor. Users may see the energy consumption through the web dashboard via computers, tablets, smartphones, or any devices that are able to connect to the Internet.

The rest of the document is structured as follows. The requirements, use cases, stake holders, and key drivers are explained in chapter 2. Chapter 3 describes the analysis and lists the possible pattens to be implemented in this system. Chapter 4 outlines the general system architecture of the system. Hardware selection and architecture are described in chapter 5, while chapter 6 elaborates on software architecture. The evaluation and evolution of this architecture is drawn in chapter 7 and 8.

2 Requirements

This chapter discusses the stakeholder of the system. This information helps us to be able to properly write use cases. These will be used to extract the functional requirements. Afterwards, a risk assessment will take place, to mitigate the risks.

2.1 Stakeholders and their concerns

This section defines all stakeholders of our system and describes the concerns of the stakeholders. A stakeholder is a person, group of persons, or organization that are involved in our system. There are ten stakeholders, ranged from first parties to third parties stakeholders. Several quality standards from the "Software Requirements" book by Microsoft [9] are used. Those quality standards are described in Table 2.1.

Quality Attributes	**Brief description**
Interoperability	How easily the system can interconnect and exchange data with other systems or components
Reliability	How reliable the results of the system are (accuracy). This includes availability, which is the extent to which the system's services are available when and where they are needed
Security	How well the system protects against unauthorized access to the application and its data
Usability	How easy it is for people to learn, remember, and use the system
Maintainability	How efficient and effective the system can be maintained/modified by the maintainers

Table 2.1: Quality attributes of Software Architecture from "Software Requirements" Book [9].

There are five quality attributes, as can be seen in Table 2.1, for measuring stakeholders' concern regarding our system. The stakeholders are listed below and are then explained in more detail below.

- Developers
- Maintainers
- Government
- Home owners

Developers have to make sure the system provides the functionality that the users expect the system to have. They want the external components (sensors) to work with the system. This means that their main concern is interoperability, because users will want to connect any energy consuming device to this site in order to monitor the consumption.

Maintainers are mainly concerned that the website is up and running at all times. Meaning their concern is the reliability of the system. Since they are responsible for the maintenance to the system, they also care about its maintainability.

The government wants to lower the energy consumption of the citizens. It wants to comply with the aims of the EU to reduce the energy consumption by 20% by 2020. Their main concern is that the end users will be able to use the system, so it will be widely adopted. This means they care about the usability and reliability of the system, because the system is only useful to the government if it is used by allot of people and the statistics of the system are reliable.

Home owners want to get an insight in their energy usage. They want to know how to effectively reduce their energy consumption. In order to do this, they might want to receive alerts about sudden increases of energy consumption in certain devices. These alerts have to be accurate and reliable. The main concern of the home owners, thus, is the usability of the system.

Table 2.2 illustrates the stakeholder concern matrix. In our approach every stakeholder is equally important.

		Concerns			
Stakeholder	Weight	Usability	Reliability	Interoperability	Maintainability
Developers	1			100	
Maintainers	1		50		50
Government	1	60	40		
Home owners	1	70	30		
Total		130	120	100	50

Table 2.2: Matrix of stakeholders concern.

2.2 Key-drivers

The stakeholder analysis of the previous section leads to the following key drivers:

- KD- 1 – Usability
- KD- 2 – Reliability
- KD- 3 – Interoperability

Usability has to be the main focus of the system. Users will want to stick to using our system if the usability is better then similar systems of the competitors. Even if competitors have a better availability and or even reliability, there is a good chance that customers will stick to this system if the usability is better.

Reliability is very important, because for this kind of system the trust everyone has is crucial. Making the stakeholders trust the system is the most important aspect of this system. If the system at some points does not provide reliable data, without specifically informing about it, then the all the information from the system will be useless. This not necessarily mean that the system has to be very precise in calculating the statistics. It just has to be very clear about how inaccurate the data is.

Interoperability is important because in order for the system to be useful to the user, it has to collect and process the energy consumption of devices. All the different homes have different devices who each have a different set of relevant statistics to be calculated. The sensor data will be monitored with a energy monitoring plug. However, to be able to receive valuable statistics from this data, the system needs to be able to cope with different types of devices.

2.3 High-level requirements

The high-level requirements describe the high-level functionality of the system. The high-level requirements are used to derive the functional requirements in Section 2.5. The high-level requirements are also used to classify the severity of the risks in Section 2.7.

This table uses different priorities according to RFC-2119[2]. First there is the ‘must’ (‘required’) priority, this requirement is an absolute must. The system could not function without the ‘must’ requirement. Besides ‘must’ there is the ‘should’ (‘recommended’) priority. This priority is highly desirable for the system, but the system could function without this requirement. The last priority is ‘may’ (‘optional’), these requirements are nice to have but not really necessary for the functionality of the system. These different priorities are used through the rest of the document.

Nr.	**Prio**	**Description**
HL- 1	**Must**	**Collecting electricity usage data** The system must collect electricity usage data, which has to be stored so it can be used to compute statistics and detect changes in energy usage.
HL- 2	**Must**	**Computing usage statistics** The system must be capable of computing statistics about the energy usage, like total usage per month, but also periods of peak energy usage. It has to be able to compute such statistics not only per house, but also for individual devices.
HL- 3	**Must**	**Displaying statistics** Data which has been gathered and statistics that have been computed must be displayed to the user in an understandable way. One of the main goals of the system is to make users of the system aware about their energy usage. The data and statistics should be displayed in a way that is intuitive and easy to understand for average consumers.
HL- 4	**Must**	**Configuring the system** The users of the system must be able to configure the system, i.e. register their house/add new devices/subscribe to alerts etc., using the web interface.
HL- 5	**Must**	**User-friendly interface** The web interface of the system should be user-friendly and intuitive, such that an average consumer is able to use it.

Table 2.3: High Level Requirements

2.4 Stories and use-cases

In this section the architectural significant use-cases will be presented.

The figure below represents the Use-case diagram of HEMS.

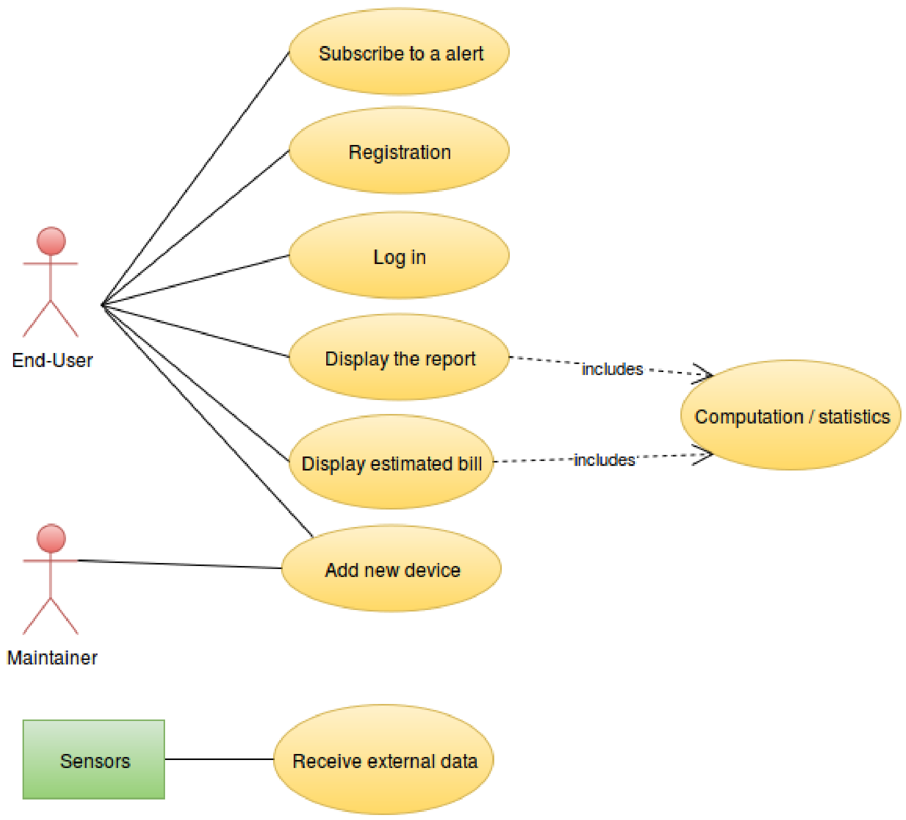

Figure 2.1: Use case diagram

UC- 1	**The End user registration**
Goal	The user has to register on the website to create his account.
Primary actor	End-User
Precondition	The website is implemented to add an user to the database
Main success scenario	1. The user access the HEMS URL 2. The website display a registration form requests the full name, username, email address and address of the user. 3. The system checks if the user isn't already in the database. 4. If not the user is added in the database. 5. The user receives a confirmation link on his email address and clicks on it. 6. The user account is activated
Postcondition	The account is created.
Extensions	4a. The user is already in the database. 1. The user gets to see a message telling him the user is already registered.
Related requirements	FR-9

Table 2.5: UC-1: The End user registration

UC- 2	**The End user login**
Goal	The user wants to be logged into the system.
Primary actor	End-User
Precondition	The End-User has an account.
Main success scenario	1. The user access the HEMS URL and clicks on the "Log in" button. 2. The user enters his username and password. 3. The system checks the validity of the username and passwords.
Postcondition	The user his logged into the system and accesses his interface.
Extensions	3a. The username/password are incorrect. An error message is prompt. Go to step 2.
Related requirements	FR-9

Table 2.7: UC-2: The End user login

UC- 3	**Receiving, collecting and store external energy usage data**
Goal	The sytem needs to receive energy data and store it.
Primary actor	The energy usage sensors
Precondition	The energy data is available
Main success scenario	1. The sensor sends energy usage information to the REST API with a security token attached. 2. The REST API confirms the attached security token. 3. The system stores the energy usage data.
Postcondition	The data is stored in the database.
Extensions	1a. The token is unvalid. Log and ignore the request
Related requirements	FR-1, FR-2, FR-14

Table 2.9: UC-3: Receiving external energy usage data

UC- 4	**Display the estimated bill**
Goal	The End-User wants to see the estimation of his next bill
Primary actor	End-User
Precondition	The user is registered in the system.
Main success scenario	1. The user gets access to his interface. (cf. UC "User log in ") 2. In the menu he clicks on "Bill". 3. The system computes the expected bill (cf.UC "Computation/statistics") 4. The system shows the expected bill to the end user
Postcondition	The estimated bill is displayed
Extensions	cf. extenstions of UC "Computation/statistics". An error message is displayed.
Related requirements	FR-6, FR-8, FR-13

Table 2.11: UC-4: Display the estimated bill

UC- 5	**Display analysis, daily/weekly/monthly report**
Goal	The End-User wants to display several kind of analysis about his energy consumption
Primary actor	End-User
Precondition	The user is registered in the system.
Main success scenario	1. The user gets access to his interface. (cf. UC "User log in ") 2. In the menu he clicks on " Analysis- Chart " 3. The user selects the type of statistic 4. The user select the period for which to compute the statistics 5. The system computes/make the analysis (cf. UC "Computation/Statistics"). 6. The computed result is displayed to the user.
Postcondition	Charts with different item are displayed.
Extensions	The statistics aren't computed : cf. extenstions of UC "Computation/statistics". An error message is displayed.
Related requirements	FR-8, FR-5, FR-4, FR-7

Table 2.13: UC-5: Display analysis, daily/weekly/monthly report

UC- 6	**Configuration : adding new devices**
Goal	The user must be able to configure the system with his home devices' characteristics.
Primary actor	The user
Precondition	The user is registered in the database of the system.
Main success scenario	1. The user get access to his interface. (cf. UC "User log in") 2. The user clicks on the "Settings" button. 3. The user gets access to the Settings section with all the internal sections and chooses "Adding new devices". 4. The user enters the new device' characteristics and submits.
Postcondition	The new device and its related information are added in the database.
Extensions	The device isn't added into the database. An error message is displayed "The device couldn't be added". Go to step 2.
Related requirements	FR-10, FR-11, FR-12

Table 2.15: UC-6: Configuration- adding new devices

UC- 7	**Computation/Statistics**
Goal	The system has to compute several kind of statistics.
Primary actor	System
Precondition	The database of the system has the necessary data.
Main success scenario	1. The statistics part is invoked from the web interface (from UC-4 or UC-5) 2. The relevant data is fetched from the database 3. If the statistic has not been computed before and is not present in the database 4. The statistic is computed on the compute cluster 5. The result is stored in the database and returned
Postcondition	The statistics are computed and the results available/displayed.
Extensions	3a. The statistic has been computed before and is present in the database 3b. The statistic from the database is returned. 4a. The system can't compute the statistics. 4b. The error is propagated to the system in order to display it in a prompt message.
Related requirements	FR-3, FR-4, FR-5, FR-6, FR-7, FR-8, FR-16, FR-19, FR-15

Table 2.17: UC-7: Computation-statistics

UC- 8	**Subscribe to an alert**
Goal	Users want to receive alert about sudden energy increases or when they are using more energy than in previous months/weeks
Primary actor	User
Precondition	The user is registered and logged in
Main success scenario	1. The user navigates to the alerts page in the web interface and clicks on "Create new Alert". 2. The user enters the criteria for when he wants to receive an alert. 3. The user clicks the submit button. 4. The system stores the new alert in the database.
Postcondition	An alert has been stored and will be checked regularly to alert the user when the criteria is met.
Extensions	
Related requirements	FR-8, FR-17, FR-18

Table 2.19: UC-8: Subscribe to an alert

2.5 Functional requirements

This section lists the functional requirements of the system.

Nr.	Prio	Description
FR-1	**Must**	The system must be able to receive electricity usage data from devices.
FR-2	**Must**	The system must be able to store electricity usage data.
FR-3	**Must**	The system must be able to retrieve previously stored electricity usage data.
FR-4	**Must**	The system must be able to compute the total electricity consumption for a given time period for a particular house.
FR-5	**Must**	The system must be able to compute the electricity consumption per device for a given time period.
FR-6	**Must**	The system must be able to compute an estimated electricity bill for the current month based on the electricity consumption to that point.
FR-7	**Must**	A user of the system must be able to select which statistics to compute in a web interface.
FR-8	**Must**	The system must be able to show the computed statistics in a web interface.
FR-9	**Must**	The system must allow users to register an account on the web interface.
FR-10	**Must**	The system must require users to be logged in, before the user can view electricity usage information about his/her house.
FR-11	**Must**	A user of the system must be able to register a new house using the web interface.
FR-12	**Must**	A user of the system must be able to register a new device for his house using the web interface.
FR-13	**Must**	A user of the system must be able to configure the price of a kWH in the web interface.
FR-14	**Must**	The system must be able to send feedback to registered devices about the current electricity usage.
FR-15	**Must**	The system must be able to take the inaccuracy of the sensors into account when computing the statistics.
FR-16	**Must**	The system must be able to display the history of electricity usage.
FR-17	Should	Users of the system should be able to subscribe to alerts in the web interface, alerting them about sudden energy increases or when they are using more energy than in previous months/weeks.
FR-18	Should	The system should send alerts to users by mail when the user is subscribed for this alert and the condition for the alert is met.
FR-19	Should	The system should be able to show the computed statistics in a graph.

Table 2.20: Functional Requirements

2.6 Technical non-functional requirements

In this section, the technical non-functional aspects that are important to the system are described.

2.6.1 Usability

The sytem will be hopefully used by a lot of people who don't necessarily have knowledge in the technology field so the Usability is an important requirement so the end-user can acces all the information he needs.

US- 1	An application is available for tablets and phone with those OS Windows Phone, Android, and iPhone.
US- 2	The end-user needs maximum ten minutes to get a basic understanding of system features through the UI.
US- 3	Every feature/major option of the system can be accessed through the home page (Single Page Application).

2.6.2 Reliability

RE- 1	A margin of error of 5% in the energy measurements is tolerated.
RE- 2	The system should be available 99.9%. of the time which means down for 44 minutes within 6 months. $AV = \frac{\text{MTTF}}{\text{MTTF+MTTR}} = \frac{\text{6 months}}{\text{6 months +44 minutes}} = \frac{\text{4380 hours}}{\text{4380+12 hours}} = 99.9\%$
RE- 3	The system should be down for maximum ten minutes when the user installs a new release (version).

2.6.3 Security

Ensuring the security for the system is a major issue so all the data needed for its good functioning remain protected and consistent.

SEC- 1	Each user is identified and has to log in in order to access his "Home Energy Monitor" account.
SEC- 2	The data stored in the database is encrypted.
SEC- 3	The connection from/to the system is encrypted using secure connection.

2.6.4 Interoperability

INT- 1	The web interface of the system is available and functioning for 95% of the browser market share.

INT- 2	The system exposes a REST interface that allows different electricity usage sensors to submit the electricity usage data.

2.6.5 Scalability

The system should make the increase in workload, resources and users easy to handle that's why the scalability its part of non functional requirements of the system.

SCA- 1	The system should still perform as efficiently as it is suppose to be when the number of users doubled.
SCA- 2	The system should still perform as efficiently as it is suppose to be when the number of devices doubled.

2.7 Risk assessment

The system is confronted by several risks which are determined and mitigated in this section. Taking those risks into account allows to avoid them or at least reduce their impact. The risk management involves the identification of the risks, their probability and potential impact or consequences.

The tables below explain the meaning of the definition for probability and consequence.

Probability	**Likelihood of occurrence**
High	0.65 - 1.00
Medium	0.35 - 0.65
Low	0.00 - 0.35

Figure 2.2: The different probabilities used to classify risks

Severity	**Explanation**
Severe	A risk that can lead to loss of live or casualties.
Significant	A risk that can lead to damages, can delay the project more than 3 months or causes one of the high-level requirements not to be fulfilled.
Moderate	A risk that can lead to one of the high-level requirements not to be fulfilled to an acceptable level.
Minor	A risk that can lead to one of the high-level requirements not being fully fulfilled, but still fulfilled in an acceptable level.

Figure 2.3: The different severities used to classify risks

2.7.1 Technical

T-RISK1	**The statistics provided by the data processing framework are wrong**
Probability of occurrence	Low
Consequences	Moderate. If the statistics computed by the system are not accurate, this will lead to loss of thrust of the end user in the system.
Prevention	Make sure the algorithms used to compute the statistics are correct.
Reaction	Correct the algorithm, if the change is significant also inform end users about the error.

Table 2.26: T-RISK1 – The statistics provided by the data processing framework are wrong

T-RISK2	**The data center storing the energy measurements becomes unavailable**
Probability of occurrence	Low
Consequences	Significant
Prevention	Store the data in a redundant way, preferably in multiple data centers, so that one data center going offline does not lead to downtime of the system.
Reaction	If the data storage does become unavailable, new incoming data should be cached so it is not lost and users should be informed in the web interface that viewing the statistics is temporarily unavailable.

Table 2.27: T-RISK2 – The data center storing the energy measurements becomes unavailable

T-RISK3	**Somebody gains unauthorized access to someone else's data**
Probability of occurrence	Low
Consequences	Significant. Data about electricity usage can be used, e.g. to derive when people are home. Unauthorized data access will lead to a loss of thrust in the system by consumers.
Prevention	Make sure access to the data requires authentication using a password at all time. Enforce users to use a strong password.
Reaction	Make sure the unauthorized access is removed. Inform end users about which data was accessed by the unauthorized party.

Table 2.28: T-RISK3 – Somebody gains unauthorized access to someone else's data

2.7.2 Business

B-RISK1	**Wrong estimation of the budget**
Probability of occurrence	Medium
Consequences	Significant. The final product does not have the features expected.
Prevention	The team needs an accountant or at least someone taking care of the follow-up of the money. Make sure there are regular evaluations to keep track of the money flow.
Reaction	Remove some requirements or features of the product, or change the hardware components used.

Table 2.29: B-RISK1 – Wrong estimation of the budget

B-RISK2	**Wrong estimation of the budget: the money invested in the fabrication and achievement of the product/system is not covered by the sales (shortfall/deficit)**
Probability of occurrence	Medium
Consequences	Moderate. Stopping the sale
Prevention	The team needs an accountant or at least someone taking care of the follow-up of the money.
Reaction	Adding more features to the product in order to make it more competitive in the market.

Table 2.30: B-RISK2 – Wrong estimation of the budget: the money invested in the fabrication and achievement of the product/system is not covered by the sales (shortfall/deficit)

2.7.3 Schedule

S-RISK1	**The project is not finished at the deadline**
Probability of occurrence	Low
Consequences	Significant. Pressure for all the team members, loss of credibility regarding the customers, selling a product with less features than expected.
Prevention	SRA , Schedule Risk analysis : Estimation of the duration of the project by its manager (with the use of probability and statistics) . Meeting for the team members every week to keep track of the timing and take decisions according to the deadline.
Reaction	Postpone the deadline or remove some features when the deadline can't be postponed.

Table 2.31: S-RISK1 – The project is not finished at the deadline

3 Analysis

This chapter describes the architectural decisions related to the patterns applied to the system. In order to document our decisons we followed the publication: *Using Patterns to Capture Architectural Decisions.*

3.1 Layers

Source
Pattern-oriented Software Architecture - Volume 1, P.31 [3]

Issue
The system consists of high-level components (e.g. user interface), which are dependent on lower level components (domain logic, database). Decoupling these components is important.

Assumptions/Constraints

- It is assumed that the system will have higher-level components that depend on lower-level components.

Positions

1. Relaxed Layered System
2. Layers

Decision
The system will use the Layers pattern to divide the application in multiple layers. The top layer will implement the Presentation logic. Below that will be the Service Layer pattern, the third layer will implement the domain logic (domain model) and the bottom layer will be responsible for the data storage.

Argument

1. The relaxed layered system is a variant of the Layers pattern which allows layers to use services of any layer below it, instead of only the next layer. This increases the flexibility and performance of the system. However, this pattern has a large negative impact on the maintainability.
2. Using the layers pattern increases maintainability by decoupling components of different levels of abstraction.

Implications
Using layers will have a positive impact on the maintainability and re-usability of the system. An increased maintainability will help prevent bugs, which on the longer term helps to increase the reliability of the system.

The performance of the system will have some negative impact, since the request have to pass all the layers, even if they only need logic/data in the bottom layer.

Related requirements/decisions
Service Layer, Broker

3.2 Service Layer

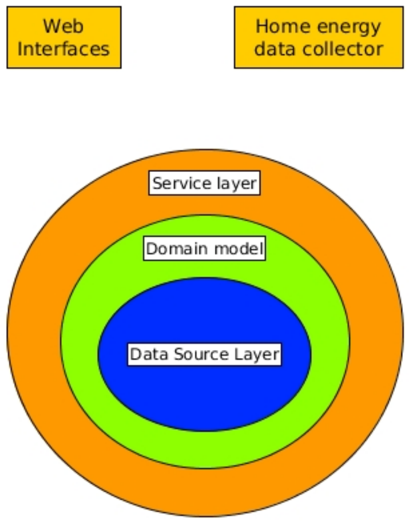

Figure 3.1: Service Layer

Source

Patterns of Enterprise Application Architecture, P.133 [5]

Issue

The system will have different interfaces with different kinds of clients. For example, commands to compute statistics can come from the user interface, but also from the alerting module. These different interfaces have common interactions with the system to invoke the business logic.

Assumptions/Constraints

- The application's business logic and/or data are accessed using several interfaces.

Positions

1. Domain Model
2. Service Layer

Decision

The system will use the Service Layer pattern to define the boundary of the application using a layer of services.

The service layer exposes a set of services to be used by clients and for each service, there is a certain script that will be executed when the service is called. The service layer will be used with the "operation script" variation. This means that the Service Layer consists of thick classes that contain logic.

Argument

1. Using just the Domain Model pattern alone does not make a distinction between application logic and domain logic.
2. The service layer does allow making a distinction between the application- and domain logic, which

Implications

There is a boundary between the application- and domain logic.

As a consequence of using the Service Layer pattern, the functionalities that the system offers have to be decomposed into services.
There will be a service for collecting the electricity usage data and computing the statistics. There will also be a service for configuring the system (adding new devices, creating alerts etc.).

Related requirements/decisions
Layers, INT-1, INT-2, FR-7, FR-1

3.3 Front page

Source

- Pattern-oriented Software Architecture - Volume 4 [4] P.339
- Patterns of Enterprise Application Architecture[5] P.344

Issue
All the different kinds of HTTPS requests that can be made to the system have similar initial handling. If every request has a dedicated handler, the code for handling these similar functions would be duplicated. This decreases the reliability of the system and might have negative affects on the availability as well.

Assumptions/Constraints

- The users use a web browser to view a graphical interface
- There is a central system handling all the HTTPS requests

Positions

1. Page controller
2. Front controller

Decision
The front page provides more functionality to this system. The front page resides in the service layer. Here it will create a pipe of filters and functions using an Intercepting Filter ([6]). This will allow the pipe to be created using a decorator pattern ([6]) The front page will be configured to handle the logging, authentication and initial security of the request. Finally the front page dispatches the requests to the domain using the command pattern ([6]).
The page controller is more intuitive. It defines a controller for each page request. This makes it very easy to track how a page is set up.

Argument

1. By using the front controller, a pipeline can be created that handles the similar activities in handling requests, having all the similar request functionality in one central place.
2. It simplifies configuring the system.
3. Front page reduces concurrency issues, because a new command object is created for each request. Reducing thread-safety concerns. The model, however, can have shared objects that do require thread safety management.

Implications
Having a controller for each request like the page controller has is a clear way of structuring a web server. It increases the understanding of each single request separately, however it decreases the overall understanding because each request could be handled completely different.

Related requirements/decisions
KD-2, KD-3, FR-1

3.4 Domain model

Source
Patterns of Enterprise Application Architecture, P.116 [5]
Pattern-oriented Software Architecture - Volume 4, P.182 [4]

Issue
The domain logic consists of complex functions for serving web request and analyzing data. The func-

tionality of the system must be modifiable and must reduce the amount of duplicate code to a minimum in order to prevent inconsistencies.

Assumptions/Constraints

- The presentation layer uses a front controller pattern

Positions

1. Transaction Script
2. Table Module
3. Domain model

Decision

The domain model pattern will be used.

Argument

1. The complex logic in the domain layer is not well suited for the Transaction Script pattern. There would be a lot of duplicated code.
2. Table Module does fully allow the use of the power of objects. It does not support instance-to-instance relationships. Domain Model is a better choice for handling complex logic.
3. The domain logic is complex and so it requires the use of the domain model pattern. This means that the domain is Object Oriented, with every class representing one specific, individual, meaningful part. The domain model is the most advanced domain pattern, minimizing code duplication and increasing the flexibility of the system. The domain model also integrates well with the front page controller in the first layer.

Implications

The domain logic will be organized in objects that incorporate both behavior and data. This will increase maintainability.

Related requirements/decisions

FR-4, FR-5, FR-6, FR-15, FR-17, KD-3

3.5 Unit of work

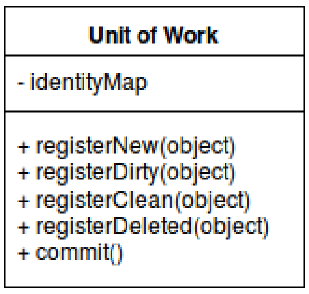

Figure 3.2: The Unit of Work class

Source

Patterns of Enterprise Application Architecture, P.184 [6]
Pattern-oriented Software Architecture - Volume 4, P.541 [4]

Issue

The system has several object stored in a database which can be edited and created. For example, new sensor data of devices becomes available and changes to the configuration (changing device names etc.) can be made. Updating database records on each change leads to a lot of database calls, which is bad for performance.

Assumptions/Constraints

- The domain layer uses the domain model pattern.
- Requests to store and receive data are being made continuously while the server runs.

Positions

1. Active Record
2. Unit of Work with Caller Registration
3. Unit of Work with Object Registration

Decision

Unit of Work pattern will be used to keep track of changes to objects and to coordinate writing these changes to the database in one database call. The variant used is the Object Registration variant, which removes the burden of having to register the object explicitly with the Unit of Work. Registering the object self could introduce bugs.

Argument

1. With Active Record, every change to an in-memory object leads to one or multiple database calls. This reduces the performance significantly, especially when using the domain model pattern.
2. The Unit of Work instead keeps track of these changes to allow writing these changes to the database in a single call.
   When using Unit of Work with Caller Registration, the creator of the object should register it with the Unit of Work explicitly, or else the Unit of Work will not keep track of its changes. This offers the flexibility to have changes to in-memory objects that are never written to database.
3. Using Unit of Work with Object Registration means that the creator of the object does not need to explicitly register it with the Unit of Work. Instead, the created object is implicitly registered (e.g. as part of the logic in the constructor).

Implications

Using Unit of Work will reduce the load on the database (the number of database calls). It does however introduce a delay before the change in the in-memory object is present in the representation in the database.
Callers editing the objects should call the `commit` method on the object after making changes to them, so that these changes are saved to the database.

Related requirements/decisions

KD-1, FR-2, FR-3, RE-2

3.6 Broker

Source

Pattern-oriented Software Architecture - Volume 4, P.237 [4]

Issue

The system uses several servers to compute the statistics. This introduces a lot of challenges, like communication to these servers and dividing the work over these servers. The application code should not have to address these challenges.

Assumptions/Constraints

It is assumed that the workload for computing statistics can be divided over multiple nodes, and that multiple nodes are available.

Positions

1. Broker
2. Publisher-Subscriber
3. Messaging

Decision

The broker pattern will be used. The broker pattern is part of the domain layer.

Argument

- Using a broker creates a layer of abstraction for the specific layer using it. This increases encapsulation, location independence and scale ability. With the Broker pattern, many clients can make remote method invocations on methods hosted by the server. It is best suited for systems that want to hide the presence of the network.
- Publish-Subscriber allows exchanging events in a one-to-many fashion. It does, however, not guarantee that an event will be processed.
- With messaging there is also no guarantee that a message will be consumed.

Implications

- The communication between the system and the servers in the compute cluster are handled by the broker.
- Adding another layer of abstraction which must be used can reduce the performance of the system
- If the broker fails, the sourced that the broker provided can't be accessed any more.

Related requirements/decisions

FR-4, FR-6, FR-6, SEC-2, RE-2, Layers

3.7 Data Mapper

Source

Pattern-oriented Software Architecture - Volume 4, P.540 [4]
Patterns of Enterprise Application Architecture, P.165 [6]

Issue

For the data source layer, there should be a way to store the data where the domain layer is shielded from the details of how the data is stored.

Assumptions/Constraints

- The domain layer uses the domain model pattern

Positions

1. Data mapper
2. Table data gateway
3. Row data gateway
4. Active record

Decision

The data mapper pattern will be used. The data mapper pattern creates the most abstraction between the domain and data source layer.

Argument

1. The data mapper pattern best suits the domain model pattern used in the domain layer. This pattern is the most advanced pattern, and provides the best functionality and abstraction.
2.
3. The logic that will operate on the data will generate statistics of the data, which can become quite complex and will be stored in a different table than the device objects. This is why the Table- and Row Data gateway patterns are not choosen.
4. With the active record pattern, the database access/communication, data and logic of that data is all stored in the same class. This means there is a tight coupling between the data and the domain

logic. Since the logic that will be executed on the sensor could be quite complex, this pattern is not preferred.

Implications

The data mapper is used in the architecture. This means that the in-memory objects in the domain layer need not be aware of the presence of the database. The data mapper objects will implement the details of storing the object in the database.

A gateway of any kind leads to performance overhead, because for each call coming from the domain model pattern, a database call is made.

Related requirements/decisions

Shared Repository, Unit of Work, FR-2, FR-3

3.8 Shared Repository

Source

Patterns of Enterprise Application Architecture, P.322 [6]
Architectural Pattern Revisited - A Pattern Language, P.13 [1]

Name

Shared Repository (also called Repository)

Issue

The external data must be stored, updated, accessed by multiple clients and servers simultaneously and analyzed.

Assumptions/constraints

- It is assumed that the shared repository handles the authentication to it.

Position

In the *Shared Repository pattern* one component of the system is used as a central data store, accessed by all other independent components."
When using the *Active Repository Pattern* the clients are notified when there is a change is the repository which is not a feature of the system.
BlackBoard Pattern " is used in an immature domain in which no deterministic approach to a solution is known or feasible " which is not the case in the HEMS.

Decision

The system uses the Shared Repository Pattern in order to centralize the access to the data.

Implications

After the integration of the Shared Repository Pattern the system has one central repository (the database) and all clients can access it by using a loggin.
This pattern ensures Reusability, Changeability, Maintainability and Integrability (cf. Chapter 7 Evaluation).

Related decisions/requirements

Data Mapper, Unit of Work, FR-1, FR-2, FR-3, SEC-2

The compoments in the figure can either be the users or the computation cluster to perform the statistics

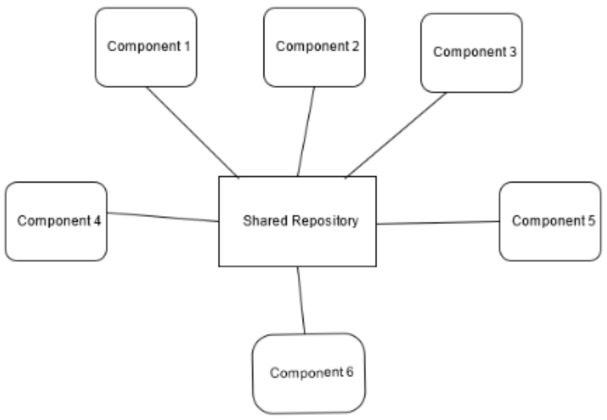

Figure 3.3: Shared Repository

3.9 Model-View-Controller

Model-View-Controller (MVC) started as a framework developed by Trygve Reenskaug for the Smalltalk platform in the late 1970s [5]. Since then it has played an influential role in most UI frameworks and in the thinking about UI design.

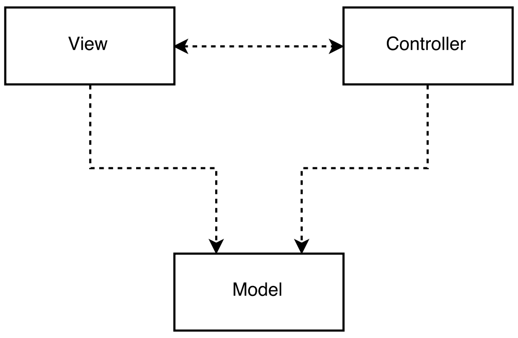

Figure 3.4: Model-View-Controller pattern illustration [5]

Source

Pattern-oriented Software Architecture - Volume 1 P.125 [3]
Patterns of Enterprise Application Architecture, P.330 [5]

Issue

The system must handle the request from user by dynamic web interface. In order to increase re-usability, the views may be decoupled from the logic, resulting in separated logic (controller and model) and view.

Assumptions/Constraints

- It is assumed that the client uses web browser to connect to the system.
- It is assumed that there are multiple views in the system.

Positions

1. Presentation-Abstraction-Control
2. Layers-Model-View-Controller

Decision

The system will use the Model-View-Controller pattern to decouple the view and the logic behind it. This pattern resides in the service layers of this system.

Argument

1. The Presentation-Abstraction-Control (PAC) pattern decompose the system into a tree-like hierarchy of agents, which each agent is made of its own presentation (UI), abstraction, and control. This type of pattern is not well suited to the HEMS since the HEMS only needs decoupled views with the same control and abstraction.
2. The Model-View-Controller (MVC) pattern only decouple the views, models, and controller, which indicates that different views may use the same controller and/or model. This pattern is well-suited with the HEMS and is good for re-usability, which we think has less complexity.

Implications
By implementing the MVC pattern, several views are created to support multiple pages. Each view may use the same controller that provides required process. This will also allows building a filter chain, handling authentication, logging etc. Thus, this will drive better reusability and have less complexity. Furthermore, it is also easier to modify the code since the view and the logic are separated.

Related requirements/decisions
FR-3, FR-7, FR-10, FR-18, FR-19

3.10 Template view

Template view pattern renders information into HTML by embedding markers in an HTML page [5].

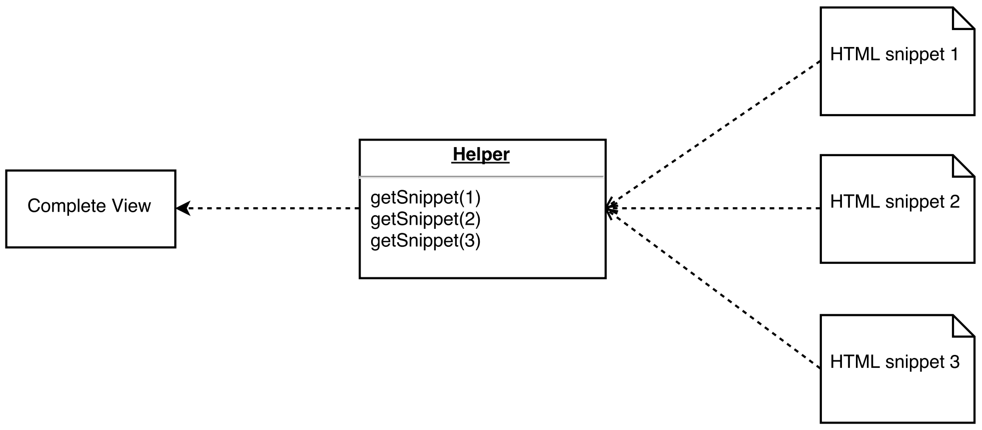

Figure 3.5: Template view pattern illustration [5]

Source
Patterns of Enterprise Application Architecture, P.350 [5]

Issue
In the view component of MVC, the system must handle dynamic web interface in a Single Page Application (SPA) flavored application. The same view (HTML code) will be displayed several time in different page, which will be possibly causing code smells.

Assumptions/Constraints

- This pattern is only applied on web application.
- The system is developed using framework that supports this pattern.

Positions

1. Transform view
2. Two step view
3. Template view

Decision
The system will utilize template view in order to increase reusability and to prevent code smells in the

project. In this way, several HTML code snippets that corresponds to same element of a page will be reused by embedding markers. Thus, this will lead to a better modifiability as well.

Argument

1. Transform view processes domain data element by element and transforms it into HTML. In the system, such task is handled by the models. Thus, if this pattern is used, there will be duplication of work.
2. Two step view turns domain data into HTML in two steps: first by forming some kind of logical page, then rendering the logical page into HTML. These operations are considered as inefficient, as the forming logical page and rendering HTML can be joined. There is also overlapping between models, controllers, and views in this pattern.
3. Template view renders information into HTML by embedding markers in an HTML page, which solves the issues and prevents code smells.

Implications

Several HTML code snippet will be reusable, which is good for reusability and simplicity. Furthermore, code smells can be prevented using this template view.

Related requirements/decisions

FR-3, FR-7, FR-10, FR-18, FR-19

4 System architecture

This chapter outlines the general system architecture of the system. The first section shows the system with its inputs and outputs to and from external factors.

4.1 System context

The system context diagram in Figure 4.1 gives an overview of the entities that interact with the system.

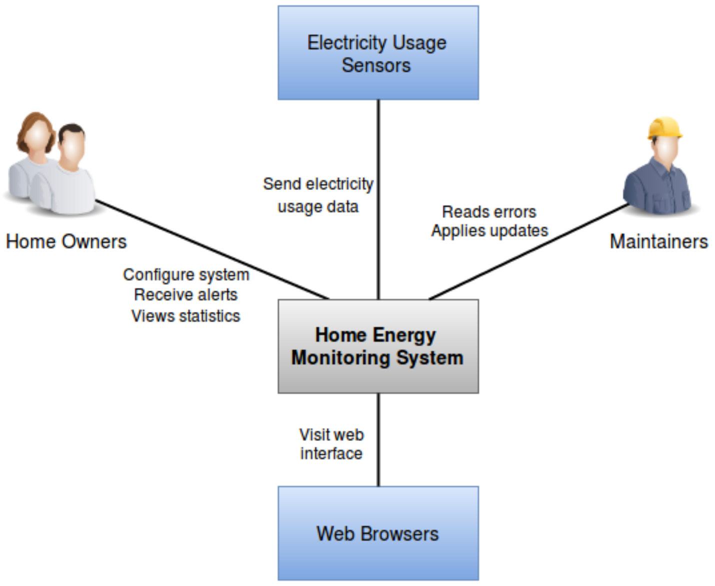

Figure 4.1: System context diagram

4.1.1 Users and roles

Home Owners

The home owners are the main users of the system. They want to know about the electricity usage in their house.

They interact with the system by viewing statistics, configuring the system and they receive alerts if they configured the web interface to send these.

Maintainers

The maintainers of the system will apply updates to the system and read errors that might have occurred in order to solve these.

4.1.2 External systems

Electricity Usage Sensors

The electricity usage sensors are the sensors that measure the electricity usage of the devices. These sensors work by measuring the electricity passing through a power outlet. This means that they in fact measure the electricity usage of a power outlet and not that of a device. This is relevant if multiple devices are connected to the same power outlet .

Web Browsers

The web interface of the system, where the system can be configured and statistics can be viewed, is presented as a web page. Users will use a range of different web browsers (on a range of different devices) to visit this web interface. It is important that the web interface works equally well in all these different web browser.

4.2 Elaborated Model

In Figure 4.2 the elaborated system model is shown. This figure gives a high-level overview of the software and hardware of the system. In this figure, the arrows represent the data flow.

Each house sends collected electricity usage data to the systems ‘Data receival API’, which stores the collected data in a storage cluster. An end-user of the system can visit the web interface using a web browser on any device. Using this web interface, the user can generate statistics data, which is done by a separate software component of the system.

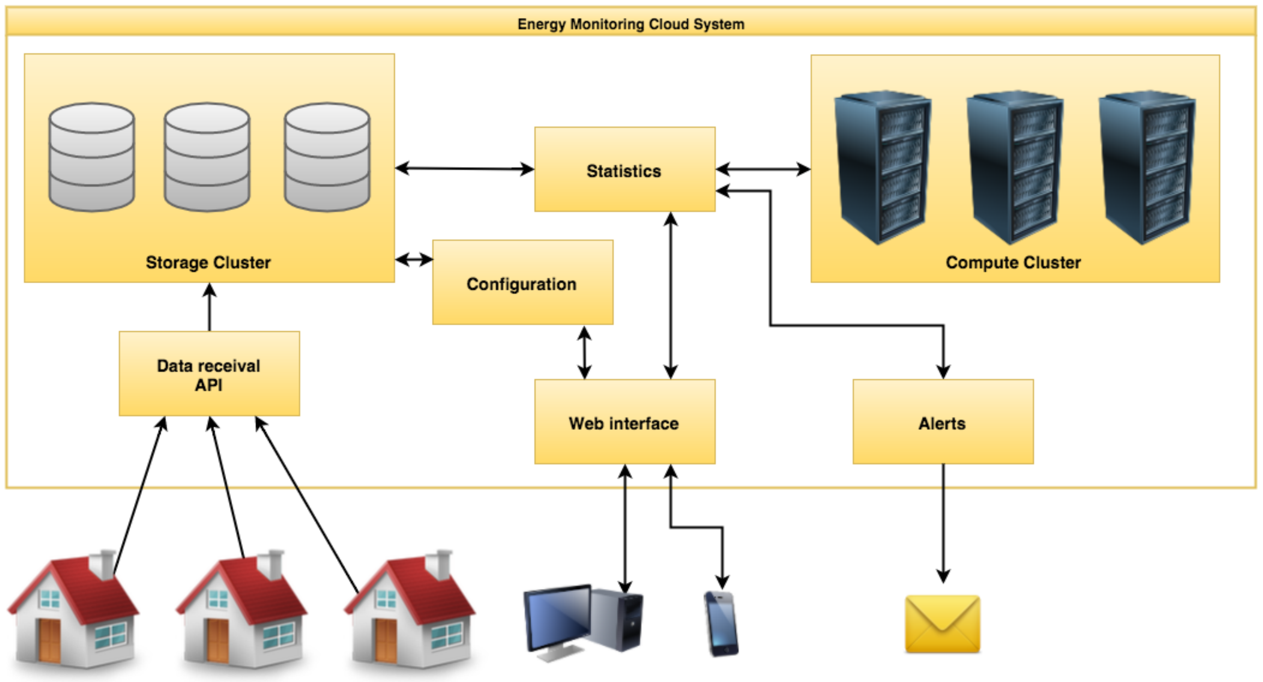

Figure 4.2: Elaborated model

Data receival API
: The ‘Data receival API’ component exposes a REST interface, which is used by the homes to send electricity usage data to the system. The API is responsible for storing the data in the storage cluster.

Storage cluster
: The storage cluster is a set of servers which will store all the data in a replicated way.

Web interface
: Users of the system (home owners) visit the web interface (presented as a website) through all kinds of devices. The web interface allows users to compute and view statistics from the collected usage data. The web interface also allows users to register new devices with the system.

Statistics
: The statistics part of the system is responsible for computing statistics from the collected usage data. It will receive commands to do so through the web interface.

Compute cluster
: The compute cluster is a cluster of servers, which are used to compute statistics from the raw collected electricity usage data.

Configuration
: The configuration part offers the functionality needed to edit the configuration of the system (add/change devices etc.) to the web interface.

Alerts
The Alerts components checks computes statistics (as configured by the user) regularly, to see if a certain criteria is met. If this is the case, the user will receive an alert by email.

5 Hardware Architecture

This section describes the hardware architecture of Home Energy Management System (HEMS). The description will be more high-level along with explanations about the hardware platform and the application interfaces between each components of the system. The rest of this chapter is organized as follows; First section, section 5.1, presents an overview of the hardware implemented in this system depicted in big schema. Decisions made in this system are detailed in section 5.2 with tables. Lastly, the hardware is described in section 5.3.

5.1 Hardware Overview

As mentioned in previous chapters, HEMS focuses on providing services to monitor electricity usages based on installed sensors on each customer's house. Thus, this system works on the cloud part, which is providing data storage, monitoring, and analysis, both in terms of application (software or service) and hardware. This system deals with no electricity collecting devices. Therefore, the electricity collection part is delegated to third party developers.

According to chapter system architecture, the main part of the HEMS hardware consists of storage cluster and compute cluster. The storage cluster is responsible to handle incoming data from sensors through the exposed data acquisition API. This cluster is capable to store real-time data. The compute cluster mainly deals the data presentation of the stored usage data. Furthermore, compute cluster is also needed to perform analysis based on stored data.

The detailed hardware selection to perform and build this system is explained in the following section.

5.2 Hardware Design Decisions

This section defines decisions made regarding the hardware selection. Tables will be used to make our justification in regard to hardware selection more clear.

Name	**Compute cluster selection**
Decision	**HW- 1**
Status	**Approved**
Problem/Issue	HEMS needs a reliable computers to do the analytical processing.
Decision	HEMS will use clustered Dell PowerEdge R530 to act as the main analytic cluster and to provide API to the actors.
Alternatives	*HP ProLiant DL360 Gen9 Base* This server rack has 16GB of memory and 2.4GHz of processor speed. As other server computer, this machine utilizes Intel Xeon E5 2600v3. This server is suitable for high dense computing, however the price is not so suitable for this kind of specification. It does not have LCD screen that will help technician to look the current status of the server. *Lenovo System x3550 M4 7914* This server rack has only 8GB of memory. However, the processor is a bit faster, it runs on 2.6GHz. As other server computer, this machine also utilizes Intel XEON E5-2600. The price is a little bit lower than the others but the memory limitation makes it not so valuable. It has LCD screen that will help technician to look the current status of the server. *Dell PowerEdge R530* This 2U server rack has 16GB of memory and 2.4GHz of processor speed. This machine utilizes Intel Xeon E5-2620V3 with 15MB of cache. This server is suitable for high dense computing. It has LCD screen that will help technician to look the current status of the server.
Arguments	

	Reliability	Performance	Interoperability	Security	Scalability	Cost	**Score**
Weights 3	2	3	2	2	2	2	
Dell PowerEdge R530 5	3	5	4	4	4	5	70
Lenovo System x3550 M4 7914 4	3	4	4	3	4	4	60
HP ProLiant DL360 Gen9 Base 5	3	5	4	3	4	3	64

Table 5.1: Decision – Analytic cluster selection

Name	**Storage cluster selection**
Decision	**HW- 2**
Status	**Approved**
Problem/Issue	The system needs reliable computers to store the data.
Decision	HEMS will utilize Synology RackStation RS814RP to store the data.
Alternatives	*Synology RackStation RS814RP* This storage machine has the fastest connection among the others. This machine will run at SATA with 6 Gbps connection. *70BJ NAS-server* This machine form factor is 1U which is suitable for saving space. However, the connection speed is limited to 3 Gbps. *Thecus N8810U-G NAS-server* This machine also runs in 3Gbps connection. However, the form factor is 2U which makes this machine takes more space in the rack.
Arguments	

	Reliability	Performance	Interoperability	Security	Scalability	Cost	**Score**
Weights 3	2	3	2	2	2	2	
Synology RackStation RS814RP 5	2	4	4	4	4	4	63
70BJ NAS-server 4	2	3	4	4	4	3	55
Thecus N8810U-G NAS-server 4	2	3	4	4	4	3	55

Table 5.2: Decision – Choice of storage machine

5.3 Hardware Description

This section gives an outline of the hardware implemented in this system. This section also elaborates on hardware decisions.

5.3.1 Storage cluster

HEMS will use clusters of computer to manage the database system and to store our data. The cluster enables the database to replicate data. This means no backup is needed, the data is always available in multiple servers. Furthermore, user account information is hashed using bcrypt after being salted with 128 randomly generated characters to increase security. The cluster will also be accessible by the main analytics part as the data will come and go through the main analytics part. The logical schematic of the storage cluster is depicted in Figure 5.1

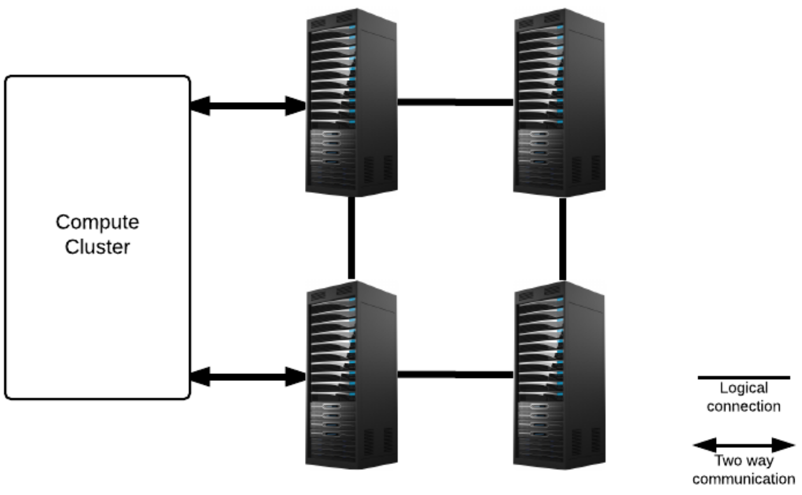

Figure 5.1: Logical schematic of storage cluster of HEMS

As can be seen in Figure 5.1, HEMS will use four database racks to have redundancy in the system. The server is connected as a ring, which is the common way to setup database server. By using this form of architecture, HEMS will be more reliable and fault tolerant. There will also be two physical connection to the main analytic cluster to make this system more fault tolerant in terms of connection. HEMS database cluster will use the same server, Dell PowerEdge R530, for controlling the SATA storage machine.

Related patterns

This storage cluster implements broker, shared repository, and unit of work patterns. Broker pattern is implemented in a mechanism that any other component of this system can connect to the cluster and see this as a single entity, although actually the cluster consists of more than one entity. If this storage cluster is seen as a single entity, then this is also an implementation of shared repository pattern, where a client or other instance connects to the storage cluster and proceeds with corresponding operations. Unit of Work pattern will be used to keep track of changes to objects and to coordinate writing these changes to the database in one database call.

5.3.2 Compute cluster

Compute cluster will be the main brain of HEMS. The data presentation will be handled by this system. The analysis process also runs on top of this hardware. To increase availability and reliability, HEMS will have six server racks to do the processing as depicted in Figure 5.2.

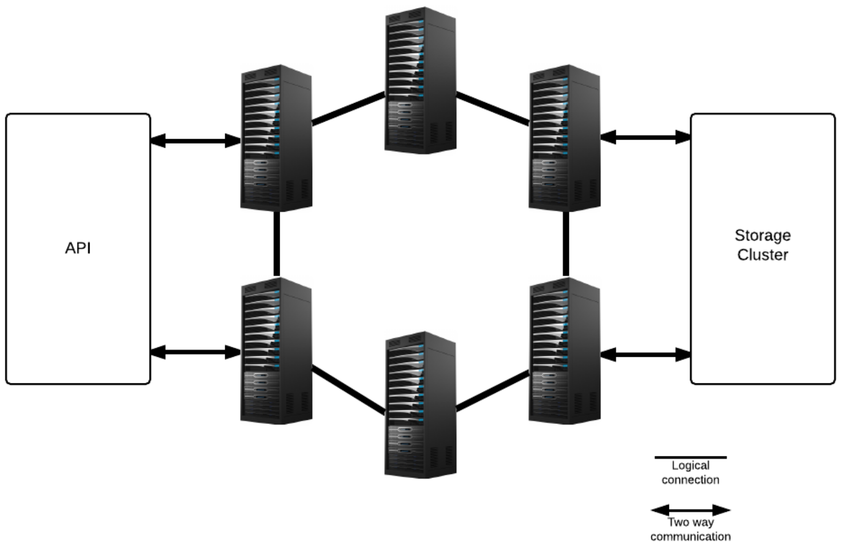

Figure 5.2: Logical schematic of analytic cluster of HEMS

Related patterns

Front page controller pattern is implemented in the compute cluster, which includes decorator and command pattern, because the compute cluster is also responsible for handling incoming connection through API.

6 Software Architecture

The HEMS system is a big and complex system. A developer can not, at any point in time, know the implementation of the entire system at once. "No one's skull is really big enough to contain a modern computer program" (Dijkstra, 1972).
At any level of abstraction, the system needs to be divided into components that have the same concern. This way a developer can work on a single and comprehensible part of the system at a time. It reduces errors in the system and makes the system manageable.
Each of the sections in this chapter explains a single "view" of the system, meaning it describes the system from a related set of concerns.

6.1 Layer decomposition

The layers pattern is used to create a separation of groups of classes that are reusable and share the same concerns.

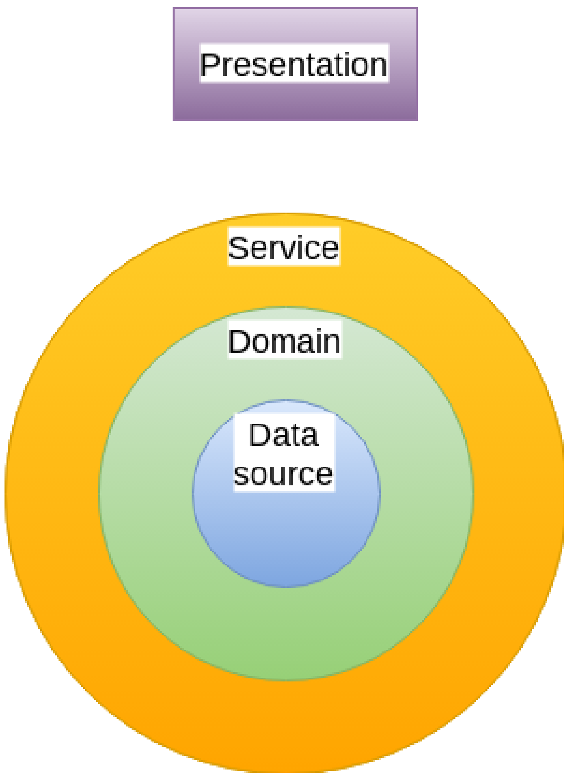

Figure 6.1: Layer decomposition

The system is decomposed into the following layers:

- Presentation layer
- Service layer
- Domain layer
- Data source layer

Presentation layer The presentation layer is concerned with presenting a graphical interface to the user (GUI). A user can use web browser and visit the HEMSwebsite. The browser will then show the energy statistics of that user on that browser.

Service layer The service layer is concerned with providing services to the clients of the system. One of these clients is the web browser of the user, as described in the presentation layer. Figure 6.1 is drawn as a set of nested circles in order to show how the layers communicate. The domain and data source layer can not be directly contacted. In order to receive data and to get statistics, a service in the service layer has to be used.

Domain layer The domain layer's concern is to calculate and provide energy statistics. All the logic for generating the statistics and data that the user wants is generated in this layer

Data source layer The data source layer is concerned with storing and receiving data. The domain layer will use this layer for managing the data it generates. The data source layer has connections with databases and computing nodes. Computing nodes are nodes that provide data by doing computations on other data (i.e. Spark).

The figure below shows these layers, including their responsibilities and concerns. The communication between the boxes is how the concerns are connected to each other. It is not how the actual flow of communication is handled, the communication flow will be discussed in detail later.

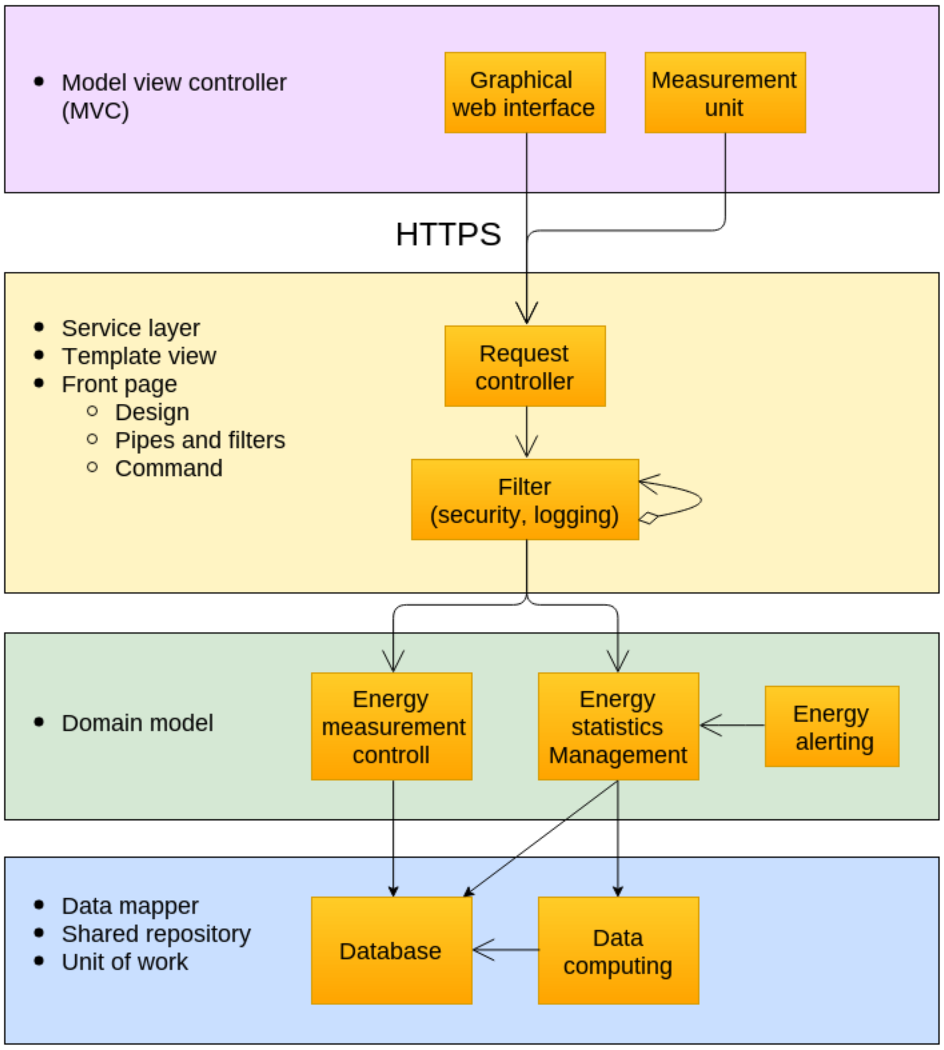

Figure 6.2: Functionality of the system in each layer

Notice that the broker isn't mentioned. This is because the broker pattern is a pattern that handles the communication between the layers, which will be discussed in a later section.

6.2 Data flow and transformation

This section describes how the data is handled and in the system. Requests are made to the service layer. The front page controller pattern is used to create a central set of operations to be executed for each request. These operations include logging and authenticating the request. The pipes and filters pattern to create a pipe that will execute these operations. This is shown below in figure 6.3

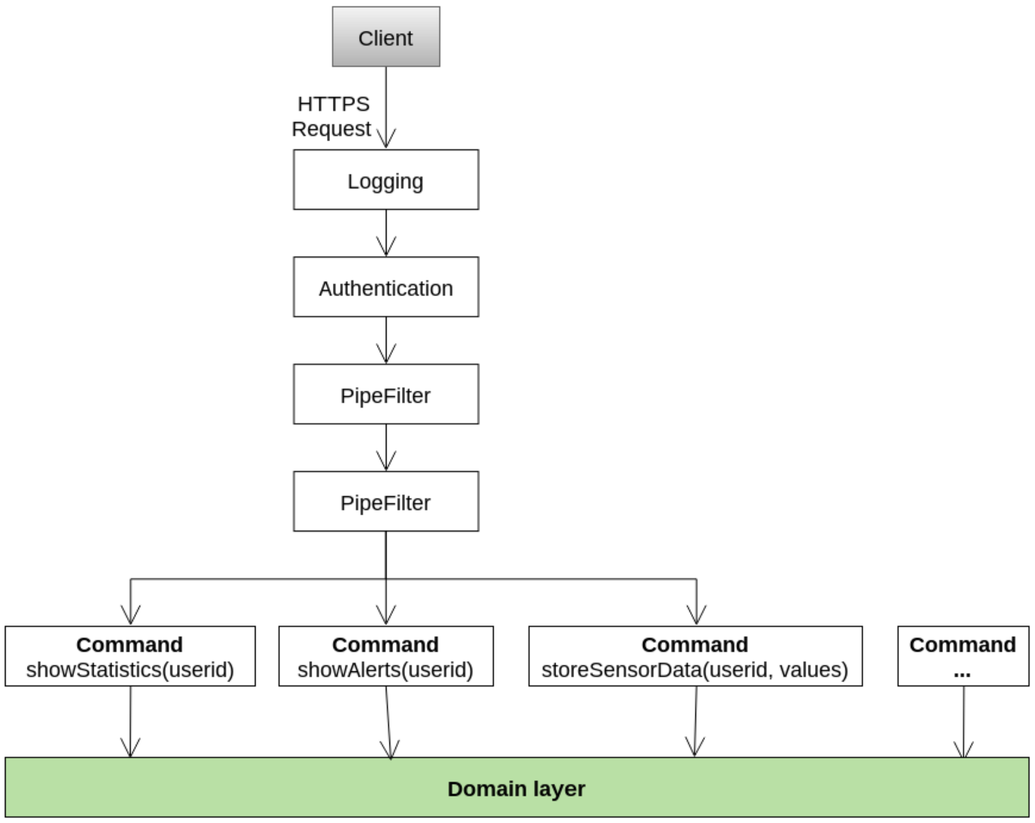
Client
HTTPS Request
Logging
Authentication
PipeFilter
PipeFilter
Command showStatistics(userid)
Command showAlerts(userid)
Command storeSensorData(userid, values)
Command ...
Domain layer

Figure 6.3: Figure showing requests are piped and dispatched

The construction of the pipes will be done using the decorator pattern and the command pattern. The decorator pattern allows the system to add and remove operations to the pipe in a very flexible way. For example like $newLoggingFilter(newAuthenticationFilter(newStandardPipe()))$. It separates the operations, letting them be completely independent of each other.
After the request was handled by the pipe, it is dispatched to the domain layer, by using the command pattern. The command pattern is where the service layer and domain layer come together. The service layer decides what command it should execute, but the implementation of that command resides in the domain layer. The command defines an abstract "Command" class that defines a single function which is used by the service layer. The domain layer then executes this command, which will do all the operations the command was created for. The image below shows a class diagram of how this information flow.

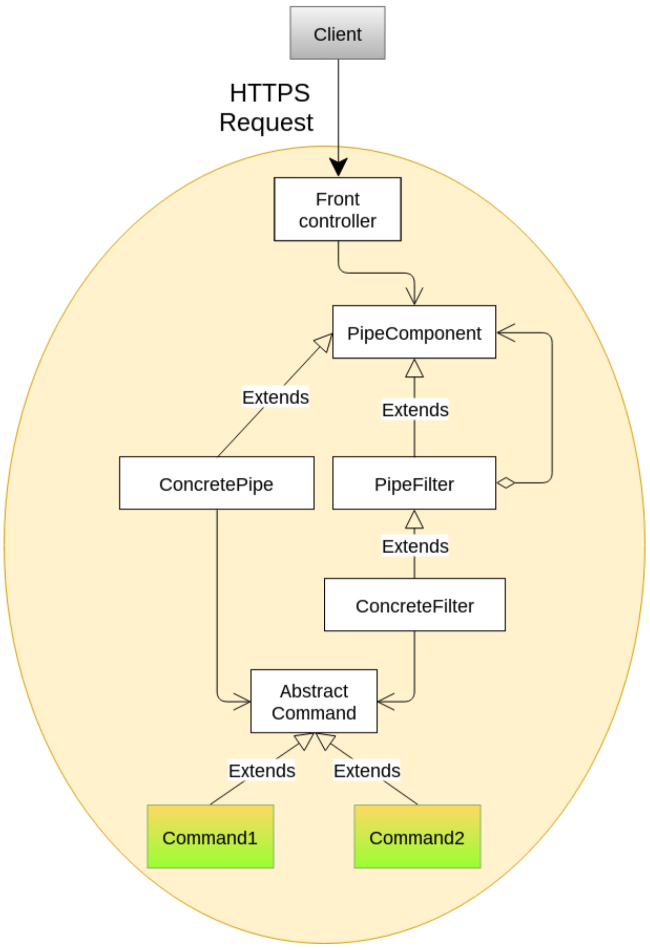

Figure 6.4: Class diagram of the font page controller creating the pipes and filters

6.3 Data repository

There are multiple components accessing a central repository of data. They do this by creating a query object and passing this to the repository. This way, the database is completely abstracted from the domain layer. The database could be replaced by a text file, without the domain layer having to change. The figure below shows an illustration of how the domain layer and the data source layer are connected.

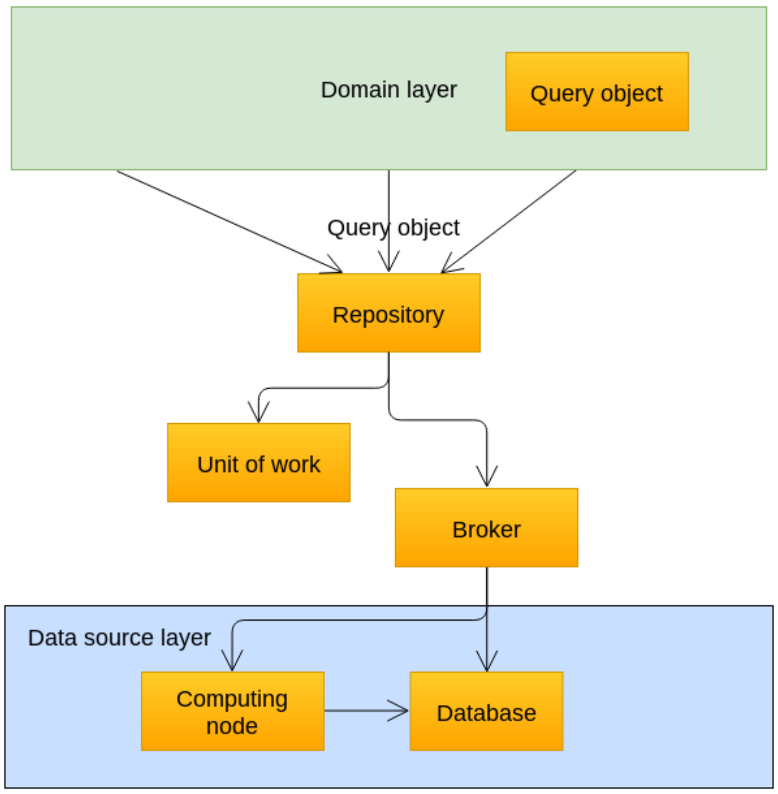

Figure 6.5: Connection between domain layer and data source layer

The Unit of Work pattern is used to keep track of the changes made to objects and newly created objects. Whenever an object is created, changed or deleted, the Unit of Work is told about this. Whenever the object can be saved to the database, the `commit()` method of the Unit of Work is called, which translates the stored changes into database transactions.

A sequence diagram showing an example of this can be seen in Figure 6.6. This is an example of the user configuring the system. Here, the StatisticsController constructs a new Device object, which is fetched from the database and then registers itself with the Unit of Work. When the StatisticsController changes the name of this device, the device object registers itself as dirty with the Unit of Work. When the device object is saved, it calls `commit()` on the Unit of Work, which leads to the device updating the appropriate fields in the database.

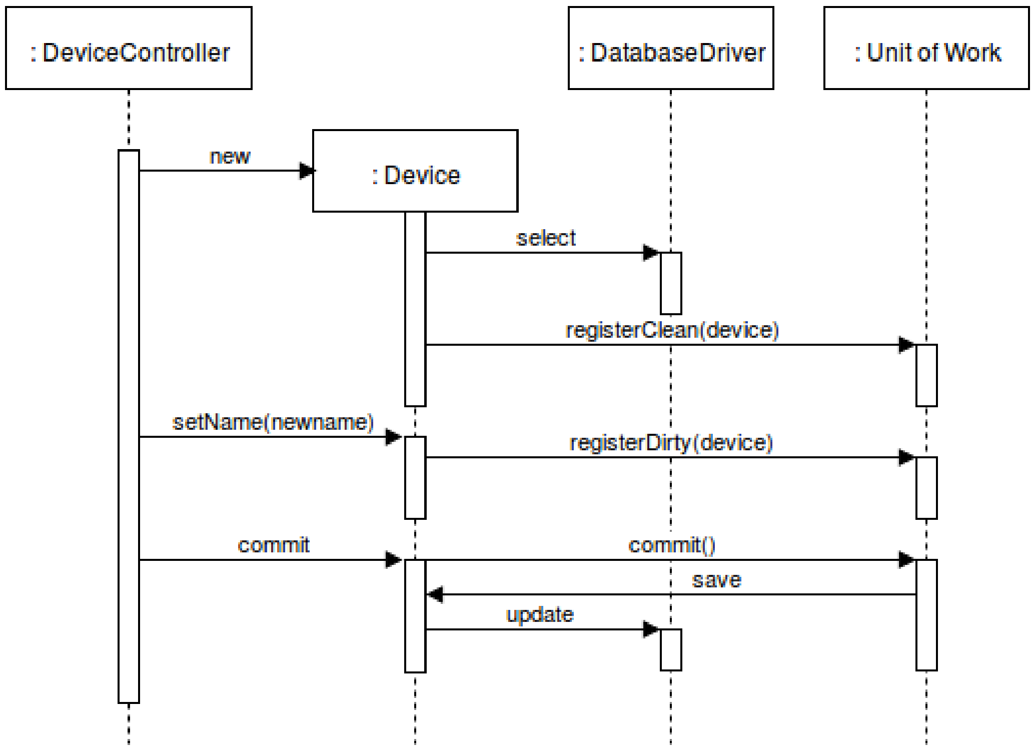

Figure 6.6: Sequence diagram showing an update to a Device-object using Unit of Work

The repository pattern mediates between the domain and data layer. The repository pattern is used by the broker pattern to get the sources the domain layer requests. The repository clients create a criteria object, specifying what kind of data is wanted. For example $criteria.equals(Device.NAME, Computer)$. Then the clients use this criteria by invoking repository.matching(criteria) to receive the data from the repository. The client just asks the data, it has no further knowledge about any interaction with any data source/data base.

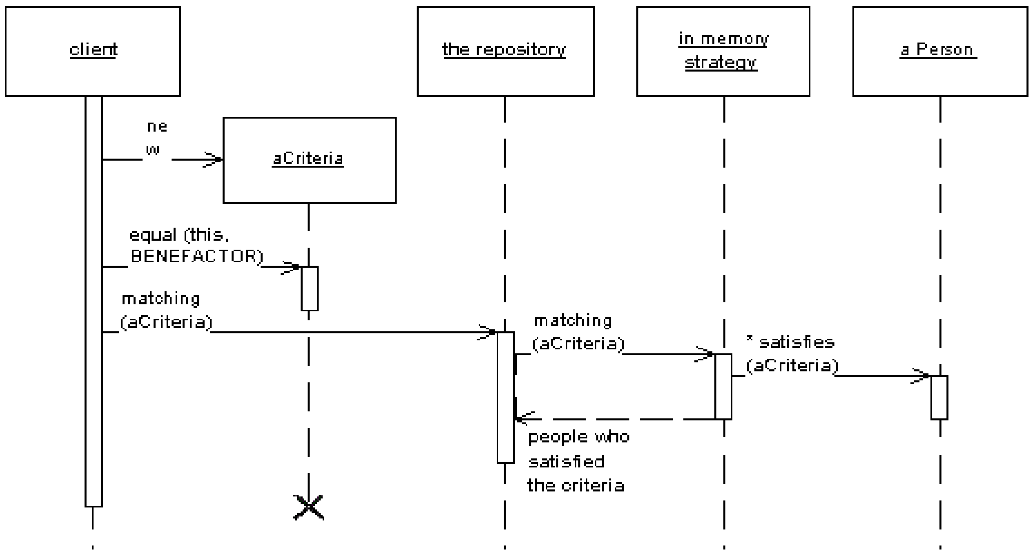

Figure 6.7: Repository View

The broker pattern is used to hide how the system interacts with the data source. Allowing the system to for example: use RPI, use a network or multiple networks, use a VPN. All without the rest of the system having to know about it.

6.4 Interaction decoupling

As mentioned in chapter 3, MVC pattern is applied to decouple user-interface and the logic behind it. In this way, reusability is increased because the same models or controllers can be coupled with the same view. Modifiability is also increased because it becomes easier to modify a particular user interface or data model without interfering the logic, and vice-versa. Figure 6.8 depicts an example of MVC implementation in the HEMS.

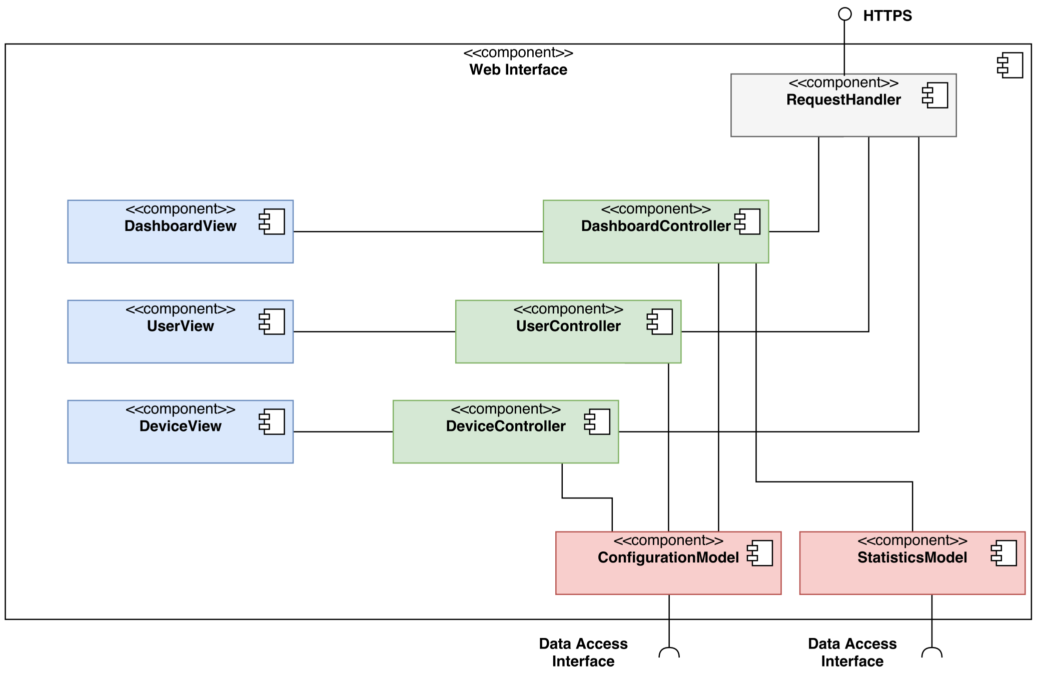

Figure 6.8: Model-view-controller pattern implementation

Template view is implemented in this system to make the HTML code reusable in different pages. This will also make the view structure more simple. Code duplication can be prevented because instead of duplicating the code, the HTML will use a certain template.

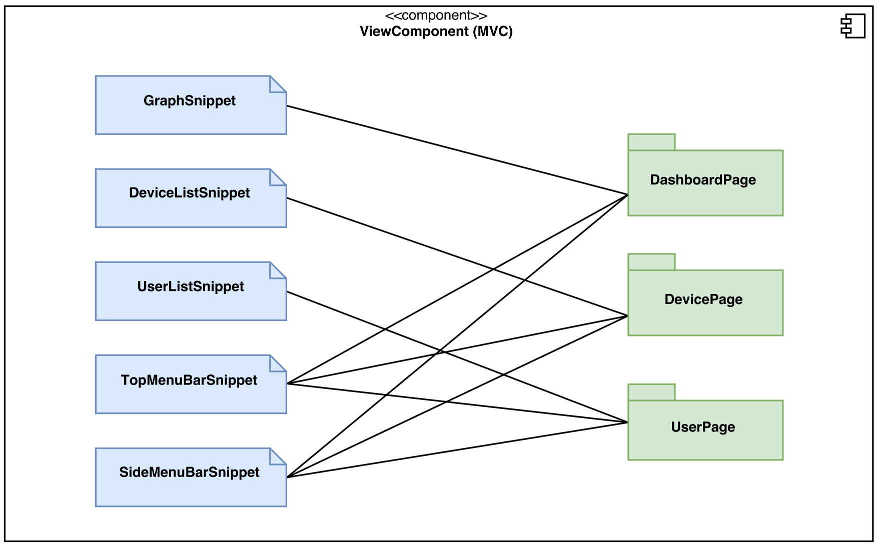

Figure 6.9: Template view pattern implementation

Figure 6.9 shows an example of implementation of the template view pattern. Each page of the system (presented in green color) will combine several HTML code snippets (presented in blue color) together. `TopMenuBarSnippet` and `SideMenuBarSnippet` are used several times, as each page contains top menu bar and side menu bar. This is also good for expandability because the new page may just combine existing page template to create new web page.

6.5 Component interaction

This section will describe the elaborated model on the basis of the patterns used in the architecture. For each patterns, this section will describe how it is implemented and how it affects the quality attributes of the system.

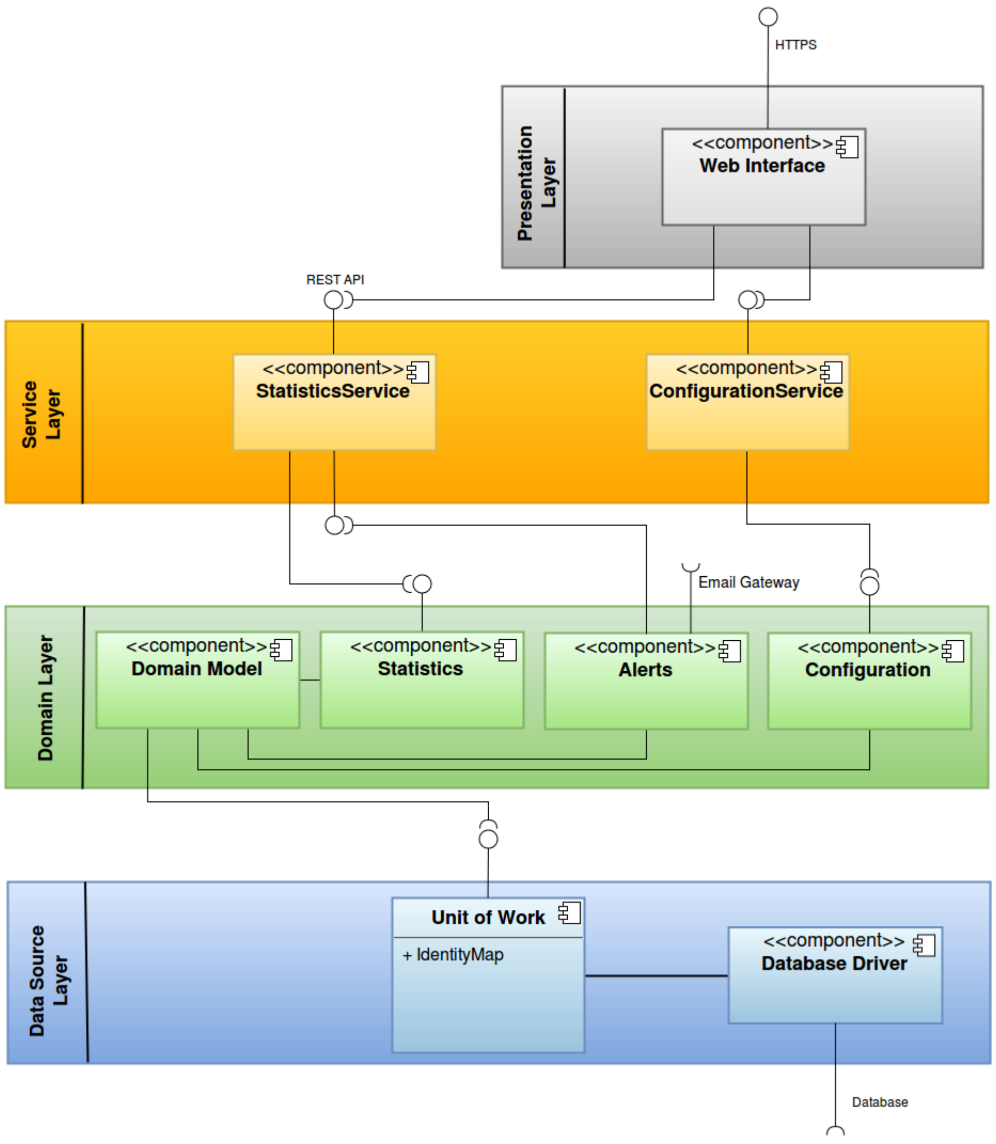
HTTPS
Presentation Layer
<<component>> Web Interface
REST API
Service Layer
<<component>> StatisticsService
<<component>> ConfigurationService
Email Gateway
Domain Layer
<<component>> Domain Model
<<component>> Statistics
<<component>> Alerts
<<component>> Configuration
Data Source Layer
Unit of Work
+ IdentityMap
<<component>> Database Driver
Database

Figure 6.10: The Layers

The system is divided into four layers. The first layer is the presentation layer. This layer is responsible for handling the interaction with the end user. It contains the Web Interface, which is accessible to the user over HTTPS.

The second layer is the Service Layer. This layer offers services, which can be used by other (external) components.

The third layer is the Domain Layer and is responsible for the domain logic. The Domain Model contains all the classes, has an in-memory representation of the data and contains the logic which is inherent to the objects. It uses the Unit of Work pattern to keep track of the changes to objects, so not every change will lead to a new database call.
The components in the Domain Layer are connected to the Domain Model, so they have access to the classes in there.

The Alerting component is responsible for sending alerts by email to the end user, for which it depends on an external Email Gateway. It also uses the StatisticsService in order to compute statistics, which it needs to decide if the user should be alerted by email.
The Configuration and Statistics components expose their functionality to the Service Layer.

The Data Source layer contains the Unit of Work, which keeps track of the changes to the objects and translates those changes to database transactions when the object is committed. The layer also contains a Database Driver, which handles the communication with the database.

7 Architecture evaluation

In this chapter the architecture of the system will be evaluated. After taking architectural decisions and building the system the architects have to make sure that the patterns applied suit the architecture, especially the quality attributes and key drivers.

7.1 Key-Driver Validation

Layers

The layers improve the maintainability of the system, they support reusability. The performance is reduced, since the propagation of request/calls through the layers can be inefficient.
Benefits & Liabilities

+ Maintainability
- Performance

Service Layer

The Service Layer pattern defines the boundary of the application using a layer of services to be used by clients. This increases the interoperability, since the set of operation offered by the service layer are available to many kinds of clients.
Benefits & Liabilities

+ Interoperability

Front-Controller

By having a central point that handles all requests, all the requests get prepossessed for the domain layer in the same way. All the common functions like security and logging are handled by the front controller for each request.
It increases usability by for example not showing the users with different authentication results for accessing a similar page. The web page results are more consistent.
It increases security because the security the request security is handled before any other operation in the system. And the security is handled the same for every request.
Reliability increases, because all the the common functions are centrally handled and so there are no duplicate implementations leading to unexpected results.
The front page increases maintainability because the way each request is received and initially handled can be modified in a central place in a very flexible way. It allows for easy modifications of the piping functions by using the decorator pattern.
A downside of the front page controller is that it is a single point of failure. If the front page controller is faulty or doesn't work, then the entire system fails. And if a function in the front page takes a long time to execute, then the performance of the entire system decreases. However, the front page controller makes testing the system easier since the functions are handled in one place.

In conclusion, the front page has the following benefits and liabilities:
Benefits & Liabilities

+ Usability
+ Reliability
+ Modifiability
+ Security

- − Performance

Domain Model

The domain model fits well with the front page controller and the domain mapper. It aims to reduce all the code duplication. This increases the maintainability.
It increases the reliability, because the adding failure recovery and handling exceptions can be handled in a more flexible way.
Benefits & Liabilities

- + Reliability
- + Maintainability

Broker

The reliability of the system increases by the use of the Broker pattern. The pattern guarantees the delivery of messages to the destination. This guarantees that statistics are computed by a server in the compute cluster. The modifiability of the system increases, since the brokers separate the messaging logic from the domain logic. This means that changes to the messaging logic do not affect the rest of the system.

The interoperability of the system increases as well, since the broker allow the system to interoperate with the servers in the compute cluster.

The broker pattern does however introduce a slight overhead, which reduces the performance.
Benefits & Liabilities

- + Reliability
- + Modifiability
- + Interoperability
- − Performance

Model View Controller

Model View Controller decouples the view, the logic, and the data modeling of the interface. Thus it is easier to modify the view without interfering the logic and data model. This results in good modifiability and better update to the usability since updating the view is relatively easy to increase the usability.

Benefits & Liabilities

- + Usability

Template view

Template view separates each components of a web page to individual snippets. The snippets are combined together to form a web page using a HTML helper. Since the elements are separated, code reusability is increased. Updates are also easy to be implemented, since updating one snippet may results in updated elements in several pages at once. Thus, template view increases the usability of the system because of its ease of web page update.

Benefits & Liabilities

- + Usability

Unit of Work

The Unit of Work will significantly reduce the load on the database. The increased performance has a direct impact on the usability of the system. Faster processing will make the web interface of the system more responsive to the actions of the user, which increases the usability.

While the unit of work does introduce a delay before the change in the in-memory object is present in the representation in the database, this delay is very short and does not affect the functional correctness of the system or the reliability of its results.
Benefits & Liabilities

+ Performance
+ Usability

Shared Repository

The repository gives a lot more control over how the data is handled. The benefits are:

- Reduces code (and code complexity)
- Increases performance
- Separated domain and data layers, increasing flexibility and changeability

Performing analysis on the data also consists of executing complex queries on the data source. The database that executes these queries, however, might change. The system could also decide to use multiple databases and data sources. Using the Repository pattern, processing these changes is simplified and can be made in a flexible way that does not effect the rest of the system. The repository also allows for multiple configurations to exist. So an extra repository could be created for testing purposes, only using an in-memory database to increase the test execution speed.
Benefits & Liabilities

+ Reusability : because the data store is central
+ Integrability : because the data store is central
+ Maintainability : thanks to the Separation of concerns
+ Modifiability : thanks to the Separation of concerns

7.2 Requirements validation

7.2.1 Functional requirements

Functional requirements evaluation is depicted in Table 7.1 below.

Nr.	Priority	Fulfilled	Pattern(s)	Remarks
FR-1	Must	Yes	Layers	Receving data is handled by the layers pattern. The process starts with reception of sensor data from service layer, then the data is forwarded to domain layer to perform several failure checks. Lastly, the data is stored through data-source layer.
FR-2	Must	Yes	Unit of Work, Data Mapper, Shared Repository	Storing the data is mainly handled in data-source layer. This layer receives the data from the domain-layer. Unit of Work keeps track of changes to objects and coordinates storing the data to the database in one database call. The data is stored in single repository, so to say.

Nr.	Priority	Fulfilled	Pattern(s)	Remarks
FR-3	Must	Yes	Unit of Work, Data Mapper, Shared Repository	Previously stored data is available in the repository, which belongs to data-source layer. Unit of Work also keeps track of changes to objects.
FR-4	Must	Yes	Broker, Layers, Unit of Work	Computation process involves multiple pattern: domain and data-source layer. This process is also a high read-write process, thus unit of work pattern is useful for tracking the changes of objects and coordinates read/write process to the database. Broker makes sure that the parallel computation is taken care properly.
FR-5	Must	Yes	Broker, Layers, Unit of Work	This process is also similar with previous process (FR-5), but this is more time-range restricted.
FR-6	Must	Yes	Broker, Layers, Unit of Work	Computing bill process involves multiple pattern: domain and data-source layer. This process is also a high read-write process, thus unit of work pattern is useful for tracking the changes of objects and coordinates read/write process to the database. Broker makes sure that the parallel computation is taken care properly.
FR-7	Must	Yes	Model-View-Controller, Template view	All operations done in web interface are taken care by the template view and MVC pattern.
FR-8	Must	Yes	Model-View-Controller, Template view	All operations done in web interface are taken care by the template view and MVC pattern.
FR-9	Must	Yes	Front page, Model-View-Controller, Template view	Registration, which is done through web interface, is taken care by the template view and MVC pattern as well. The front page will handle the logging, authentication and initial security of the request.
FR-10	Must	Yes	Front page, Model-View-Controller, Template view	The front page will handle the logging, authentication and initial security of the request. User interface is provided through MVC and template view pattern.
FR-11	Must	Yes	Front page, Model-View-Controller, Template view	In order to add a house, a user must be logged in. The front page will handle the logging, authentication and initial security of the request. User interface is provided through MVC and template view pattern.
FR-12	Must	Yes	Front page, Model-View-Controller, Template view	In order to add a device, a user must be logged in. The front page will handle the logging, authentication and initial security of the request. User interface is provided through MVC and template view pattern.
FR-13	Must	Yes	Front page, Model-View-Controller, Template view	In order to configure the price of the electricity, a user must be logged in. The front page will handle the logging, authentication and initial security of the request. User interface is provided through MVC and template view pattern.

Nr.	Priority	Fulfilled	Pattern(s)	Remarks
FR-14	Must	Yes	Layers, Service Layer	This is multi-tier operation, which involves all layers. The feedback is sent through the service layer.
FR-15	Must	Yes	Layers, Domain layer, Unit of Work	The accuracy is provided by the sensor itself. The computation process is taken care in the domain layer.
FR-16	Must	Yes	Model-View-Controller, Template view	The history, computed in FR-3, is shown to the user through web interface, which involves MVC and template view pattern.
FR-17	Should	Yes	Model-View-Controller, Template view	The alert is shown in the web interface, which employs MVC and template view pattern.
FR-18	Should	Yes	Layers, Domain layer	This is a multi-tier operation, which includes domain and service layer. The email is sent through service layer.
FR-19	Should	Yes	Model-View-Controller, Template view	Graph is shown in the web interface, which employs MVC and template view pattern.

Table 7.1: Evaluation of functional-requirements

7.2.2 Non-Functional requirement

This subsection presents non-functional requirements evaluation in tables.

7.2.2.1 Usability

Nr.	Fulfilled	Pattern(s)	Remarks
US-1	Yes	Model-View-Controller, Template view	Fulfilled with web application, which employs MVC and template view pattern.
US-2	Yes	Model-View-Controller, Template view	Fulfilled by using MVC and template view pattern.
US-3	Yes	Model-View-Controller, Template view	Fulfilled by using MVC and template view pattern.

Table 7.2: Evaluation of non-functional requirements: usability

7.2.2.2 Reliability

Nr.	Fulfilled	Pattern(s)	Remarks
RE-1	No	-	Sensor accuracy is out of scope of this project as this project does not deal with sensor hardware.
RE-2	Yes	Layers, Service Layer, Unit of Work, Broker	High availability is achieved by separating the service in layers and parallelizing the processes.

Nr.	Fulfilled	Pattern(s)	Remarks
RE-3	Yes	Layers, Service Layer, Unit of Work, Broker	Separated layers tame care smooth new version deployment in the HEMS.

Table 7.3: Evaluation of non-functional requirements: reliability

7.2.2.3 Security

Nr.	Fulfilled	Pattern(s)	Remarks
SEC-1	Yes	Front page, Model-View-Controller, Template view	Login page is handled by MVC and template view pattern. Front page pattern also helps to serve the login request.
SEC-2	Yes	Shared Repository	Repository pattern is used to encrypt the data in the database.
SEC-3	Yes	Layers, Service Layer	Layers and service layer pattern make sure the connections are secured using HTTPS channel.

Table 7.4: Evaluation of non-functional requirements: security

7.2.2.4 Interoperability

Nr.	Fulfilled	Pattern(s)	Remarks
INT-1	Yes	Model-View-Controller, Template view	The web interface are mainly handled by MVC and template view pattern.
INT-2	Yes	Layers, Service Layer	REST interface are exposed in service layer.

Table 7.5: Evaluation of non-functional requirements: interoperability

7.2.2.5 Scalability

Nr.	Fulfilled	Pattern(s)	Remarks
SCA-1	Yes	Layers, Broker, Unit of Work, Data Mapper	The system uses layers which are possible to be replicated.
SCA-2	Yes	Layers, Broker, Unit of Work, Data Mapper	The system uses layers which are possible to be replicated.

Table 7.6: Evaluation of non-functional requirements: scalability

8 System evolution

Energy Monitoring is a field which always grows because of the new environmental issues, such as global warming. This adds opportunities for the HEMS to grow by adding new features. “Various studies have shown a reduction in home energy use of 4-15% through use of home energy display” [10]. This percentage show the efficiency and the impact on the everyday life of such systems and that more and more users will use them. This chapter describes the system evolution that is targeted to be achieved within three to five years.

8.1 Monitoring other energy consumption

As can be seen on Figure 1.1, heating is one other major energy consumption of a particular house. In the first version of the HEMS, only electricity consumption is measured. In the next release, the HEMS will also support gas consumption, because it is main energy source for space and water heating. The same features will be applied in this measurement, such as predicting upcoming gas bill and analytics regarding the usage.

The system will allow the user to compute and display to the user his greenhouse gas emission for example.

8.2 Home automation functionalities

Knowing how the user consumes is good but controlling it is even better. One of the goal of the HEMS is to make the user more energy efficient. The website/application will allow the user to control devices remotely (turning them on/off or changing light luminosity or heating intensity for example) in the next release. The system will be able to advice the user about his energy consumption (ex: you should turn off this light because the room is empty, etc).

8.3 Native mobile application support

Current interactions are only delivered by web interface. Although this interface supports computer (large screen) and mobile device (small screen), native mobile application has richer user interactions. Native mobile application, both in Android and iOS operating system, will be developed in order to expand this system capabilities.

8.4 Integration with payment company

Users are only able to see electricity consumption and predict upcoming electricity bill in this first version of the HEMS. Future version will also be able to pay the electricity bill directly through the system. The HEMS will be expanded to integrate with third party payment company to make this possible.

8.5 Building the HEMS own sensor

Currently, sensors installed in the houses are not the concern of the HEMS, because the first version is aimed to build robust platform to support energy measurement in cloud environment. Although the system also has support to any third party sensors by exposing REST API, specialized sensor build for the HEMS will deliver better support, because latest updates about the system will be firstly compatible and tested with the own sensor of the HEMS. The future release of the HEMS will also include its own sensor.

A Time Tracking

Week 1

Person	Task	Hours
Putra	Meetings, writing introduction and system context.	4
Fakambi	Meeting, First approach on non functional requirements and risks	4
Schaefers	Meetings, Added a stake holders and key drivers description.	5
Menninga	Meeting, high-level and functional requirements and improvements to risks	5

Week 2

Person	Task	Hours
Putra	Meetings, editing introduction and adding hardware architecture.	5
Fakambi	Meetings, work on the use-cases	6
Schaefers	Meetings, researched patterns to use and created initial analysis draft	9
Menninga	Meetings, system architecture, use case improvements	8

Week 3

Person	Task	Hours
Putra	Meetings, processing feedback, working on MVC and template view patterns in analysis and software architecture chapter	12
Fakambi	Meetings, updated use-cases, work on the Shared repository pattern	12
Schaefers	Meetings, researching patterns in the various books, describing front controller and command patterns	14.0
Menninga	Meetings, feedback, working on Layers/Unit of Work/Service Layer patterns (ch. 3 & 6), improved elaborated model	13.0

Week 4

Person	Task	Hours
Putra	Meetings, processing feedback, adding evaluation and evolution, quick review	17
Fakambi	Meetings, Process feedback from the other group and our Todolist, Improvements to use cases and Shared Repository Pattern(analysis ,evaluation), work on Evolution chapter Global review	15
Schaefers	Meetings, researching patterns and ways to define them, restructuring software chapter and reviewing the report of an other group	18
Menninga	Meetings, evaluation, processing feedback (see also revision history)	18

Bibliography

[1] Paris Avgeriou and Uwe Zdun. Architectural patterns revisited – a pattern language. 2005.

[2] S. Bradner. Key words for use in RFCs to indicate requirement levels, 3 1997. RFC 2119.

[3] K Henney F Buschmann and D.C. Schmidt. *Pattern-oriented Software Architecture - Volume 1 - A system of patterns.* 2007.

[4] K Henney F Buschmann and D.C. Schmidt. *Pattern-oriented Software Architecture - Volume 4.* 2007.

[5] Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[6] Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[7] Paul Gil. What is 'saas' (software as a service)? `http://netforbeginners.about.com/od/s/f/what_is_SaaS_software_as_a_service.htm`, 2013. [Online; accessed 5-December-2015].

[8] Rik Langendoen. How to reduce home energy-related costs. `http://greenifynow.com/wp/?p=795`, 2013. [Online; accessed 22-November-2015].

[9] K. Wiegers and J. Beatty. *Software Requirements.* Developer Best Practices. Pearson Education, 2013.

[10] Wikipedia. Home energy monitor. `https://en.wikipedia.org/wiki/Home_energy_monitor`, 2015. [Online; accessed 6-December-2015].