Software design of control system

Lars Rikard Rådstoga | 223786

2022-04-04

# Introduction

The following report contains object-oriented design work for two pieces of software: a level control system and a temperature control system. Analysis work such as use case diagrams and sequence diagrams will be used to create interaction diagrams and class diagrams. Responsibility driven design will be used by following the GRASP principles to determine what objects should contain what responsibilities. Some additional requirements related to the programs are displayed in Figure 1‑1.

Graphical user interface, application

Description automatically generated

Figure 1‑1 Generated details about the assignment.

# Level Control system

The following chapters contain both analysis and design segments of the level control system.

## Analysis

The most important use case of the system, see Figure 2‑1, is the “Control the level” use case as, in theory, this is the only use case serving a purpose on its own. It will, although probably be poorly tuned without the configuration handling. If the relation between the level and the volume of the vessel is non-linear, then a function of this relation should be provided in the vessel’s specification sheet. And with this non-linear relation, the relation between change in level and the output flow will also be non-linear, so this formula would have to be applied

Diagram

Description automatically generated

Figure 2‑1 Given use case diagram for the level control system.

Figure 2‑2 shows a system sequence diagram of the “Control the level” use case. It is assumed that it is configured by the “Handle Configuration” use case which is why this use case does not include it. The level control is very simple, it will merely check if the level is correct and increase/decrease the input capacity depending on whether the level is lower or higher than desired.

Diagram, schematic

Description automatically generated

Figure 2‑2 System sequence diagram of the level control.

## Design

Parts of the use case document, namely the main success scenario and the extensions are handy when creating a sequence diagram as an interaction diagram. These parts are therefore quickly written out in Table 1 and Table 2.

Table 1 Main success scenario

|  |
| --- |
| 1. Start the system with a start button. |
| 2. Get setpoint from configuration. |
| 3. Get loop delay form configuration. |
| 4. Get level from level sensor. |
| 5. Calculate the error from setpoint. |
| 6. Set the input pump capacity according to the error. |
| 7. Wait out the delay. |

Table 2 Extensions

|  |
| --- |
| 2a. Default 70% if no setpoint exist in configuration. |
| 2b. Default 70% if config can’t be read. |
| 2ba. Display config error in UI. |
| 3a. Default 200ms delay if no loop delay exists in configuration. |
| 3b. Default 200ms if config can’t be read. |
| 3ba. Display config error in UI. |
| 4a. if level sensor doesn’t work, stop the pumps. |
| 4aa. Display error in UI. |
| 6a. If the pump doesn’t respond, display error in UI. |

An interaction diagram in form of a sequence diagram is created and can be found in Figure 2‑3. The GRASP principles have been applied to allocate responsibilities. An immediate oversight of this diagram is the lack of object creation and use of the creator pattern. The LevelControl would however be the creator of Configuration, LevelSensor, Timer and Pumps. Other patterns like controller, information expert, and high cohesion are also applied and a list of responsibilities and allocated classes can be found in Table 3. The run method for the level control should be triggered by the user interface, the controller pattern implies that the responsibility to run the control system should not lie in the same class as the user interface and is therefore place inside the LevelControl class. Most of the other responsibilities are of the information expert pattern, which include both knowing and doing. These responsibilities are therefore placed inside classes that intuitively should know about the information that is needed to fulfill them [1].

A picture containing graphical user interface

Description automatically generated

Figure 2‑3 Interaction diagram in form of a sequence diagram

Table 3 A table containing function descriptions, design patterns used to allocate responsibility and names of the allocated classes.

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Description | Design pattern | Class Name |
| 1 | Run the level control | Controller | Level Control |
| 2 | Get setpoint from config | Information expert + high cohesion | Configuration |
| 3 | Get Delay from config | Information expert + high cohesion | Configuration |
| 4 | Display error message |  | User interface |
| 5 | Get measured level | Information expert + high cohesion | LevelSensor |
| 6 | Display error message |  | User interface |
| 7 | Stop pump | Information expert + high cohesion | Pump |
| 8 | Calculate level error | Information expert | Level Control |
| 9 | Increase pump input | Information expert + high cohesion | Pump |
| 10 | Decrease pump input | Information expert + high cohesion | Pump |
| 11 | Wait for the delay | High cohesion | Timer |

GRASP are nine categories of problems and solutions used in object-oriented programming. They are used in Table 2‑1 to determine relations between responsibilities and classes.

Make a design class diagram

Diagram

Description automatically generated

### Difference between SSD and the design SSD

The design SSD describes the main success scenario from the FDUCD which includes classes from other use cases. Actors from earlier should be represented as classes and the chosen design patterns like GRASP should be used to allocate which classes that should take the different responsibility.

### Difference between interaction diagram and class diagram

The interaction diagram explains the flow of the program (order of execution) and responsibility of functionality. The class diagram displays the responsibility of functionality, relation of objects, and data storage (variables).

### How UP can be used to develop this specific software

For this software (the level control system) I will divide the work into phases and iterations. The Inception phase will be deemed completed as the software is clearly defined. We are currently working on the elaboration phase, just now we put emphasis onto analysis and design. Further design will focus on the layered design and user interface. After this I will focus on choosing the development stack for the application, work on implementing the requirements and testing. When the first use case is implemented successfully and meets the requirements, I will continue with the next prioritized use case.

# Temperature system

The following chapters contain both analysis and design segments of the temperature control system.

* Control temperature in a set of buffer tanks
* Using a set of temperature sensors
* one mixer and one heater for each tank
* each tank will have different number of temperature sensors
* Each tank will be different, they should be specified by a configuration class

Graphical user interface, application

Description automatically generated

## Design pattern

The patterns that I could have used to place each function can be found in Table 2. The first design pattern to consider is the creator

Table 4 Table of GRASP patterns possibly used in the “Control the temperature” use case.

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Description | Design pattern | Class Name |
| 1 | Create timer object | Creator | PowerUp |
| 2 | Create database config object | Creator | PowerUp |
| 3 | Create temperature controller object | Creator | PowerUp |
| 4 | Start/Run the temperature controller | Controller | ControlTheTemperature |
| 5 | Get the number of sensors | InformationExpert | ConfigDB |
| 6 | Create sensor objects | Creator | ControlTheTemperature |
| 7 | Create mixer objects | Creator | ControlTheTemperature |
| 8 | Create heater objects | Creator | ControlTheTemperature |
| 9 | Start/Run mixers | Controller\*\* | Mixer |
| 10 | Get control loop delay | InformationExpert | ConfigDB |
| 11 | Get sensor values | InformationExpert | TempSensor |
| 12 | Get temperature setpoints | InformationExpert | TempSensor |
| 13 | Calculate average temp of a tank | InformationExpert | ControlTheTemperature |
| 14 | Increase temperature | Controller | Heater |
| 15 | Decrease temperature | Controller | Heater |
| 16 | Temperature exceeding thresholds | InformationExpert | ControlTheTemperature |
| 17 | Sensor Error | InformationExpert | ControlTheTemperature |
| 18 | Wait delay | Controller | Timer |

A picture containing timeline

Description automatically generated

Figure 3‑1 Given design sequence diagram of a temperature controller.

Table 5

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Description | Design pattern | Class Name |
| 1 | Run (this should probably be “create”) CheckTempAlarms | Creator | CheckTempAlarms |
|  |  | | |
| 2 | Create horn object | Creator | Horn |
| 3 | Get temperature from a tank | Information expert | ControlTheTemperature |
| 4 | Get lower temperature limit | Information expert | ConfigDB |
| 5 | Turn on the alarm if temperature under low limit | Controller | Horn |
| 6 | Turn off the alarm if temperature above low limit | Controller | Horn |
| 7 | Get upper temperature limit | Information expert | ConfigDB |
| 8 | Turn on the alarm if temperature above high limit | Controller | Horn |
| 9 | Turn off the alarm if temperature below high limit  (alarm will be turned off if it was turned on by being below low limit…) | Controller | Horn |
| 10 | Get alarm loop delay | Information expert | ConfigDB |
| 11 | Wait a delay | Controller | Timer |

## Class diagram

## Object diagram

# Conclusion

# References