ISSN 0280-5316
ISRN LUTFD2/TFRT--5903--SE

Model Based Engineering of a Reverse Osmosis Water Purification Plant

Sofia Mejvik
Håkan Olin

Lund University
Department of Automatic Control
August 2012

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
August 2012

Document Number
ISRN LUTFD2/TFRT--5903--SE

Author(s)
Sofia Mejvik
Håkan Olin

Supervisor
Mattias Wallinius, Adevo Consulting AB, Lund, Sweden
Karl-Erik Årzén, Dept. Of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Model Based Engineering of a Reverse Osmosis Water Purication Plant (Modellbaserad utveckling av en vattenreningsanläggning baserad på omvänd osmos)

Abstract
As engineering systems become more and more advanced, the need for collaboration and communication between dierent system areas increases. Systems Modeling Language (SysML) is a graphical modeling language developed to provide a modeling capability independent of the system area. In this thesis, a
SysML model will be developed for a reverse osmosis water purication plant. The purpose of the model is to document requirements stated by stakeholders and to use these requirements as a basis for the development of a new control system. The control system developed is composed of several state machines, with each state machine controlling its own part of the plant. Also, as the reverse osmosis plant wastes a lot of water, a control strategy is developed in order to feed otherwise wasted water back into the system. The validation of the control strategy is done against a mathematical model of the membrane process, which is also derived. First the control system is simulate using Matlab/Simulink and later implemented in C code on a PLC.

Keywords

Classification system and/ or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-107

Recipient's notes

Security classification

Acknowledgments

This thesis is the result of work performed between February and August 2012 at Adevo Consulting AB, Lund, and marks the end of Sofia Mejvik's and Håkan Olin's Master of Science degree in Automatic Control.

We would like to thank our supervisor, Mattias Wallinius, for helping us through this project and Anders Widov at FR Pharma AB for his help in explaining the reverse osmosis process. A special thanks to Maja Arvehammar at Adevo Consulting AB for her much appreciated support with B&R Automation Studio. We also appreciate the help from Prof. Bernt Nilsson and his colleagues at the department of Chemical Engineering, Lund Institute of Technology, for helping us to derive the mathematical models for the plant.

Additionally, we would like to thank Prof. Karl-Erik Årzén at the department of Automatic Control at Lund Institute of Technology, for his support and feedback throughout the project.

Sofia Mejvik and Håkan Olin - Lund, August 2012

Contents

1 Introduction

1.1 Background

Water is a vital resource to all life. For humans the access to clean and safe water is vital not only for drinking, but in many other forms and processes. In many applications, such as in the production of drugs or sterilization of instruments, standard potable water is not considered clean enough, and has to be further purified.

There are various methods to purify water, with everything from filtration to phase shifting and chemical treatment. The main purification method used and described in this thesis is a process called reverse osmosis. Reverse osmosis (RO), is as the name implies, an osmosis process run backwards. The osmosis process was discovered during the 18th century, but the industrial use of its reverse counterpart started first in the middle of the 20th century [1] [2]. It is nowadays widely used in several areas, for example in desalination plants in arid regions, in food processing industry, waste water treatment and as a general drinking water improver.

The process is based on the selective transport through a semipermeable membrane further described in Section 2. During its early days, implementations suffered from a very low flux through the membrane, which made the process inefficient and not commercially viable. As the understanding of the actual processes involved in separating solvent from solute grew, new membranes were developed, greatly improving RO's competitiveness to the point where we are today.

This thesis is done at Adevo Consulting AB for a client company, FR Pharma AB. The client has the agency for the European market of a Chinese owned company, Watertown Pharmaceutical Equipment CO.LTD (吉林省华通制药设备有限公司). Watertown constructs and designs many different types of purification plants but the one considered in this thesis is a pure water (PW) system. Pure water is for example used at hospitals to clean surgical instruments and in the pharmaceutic industry to manufacture tablets.

1.2 Motivation

Today many complex technical systems are developed where many areas of knowledge have to cooperate to achieve a system that fulfills its requirements. To accomplish this it is of great importance to have good communication between the different working areas and to create documentation of the work. Even if the documentation for each area is satisfying, the lack of communication may cause problems to the final system. Each subparts of the system may work flawlessly individually but when put together the entire system may contain fatal errors. One way to solve this issue is to create a documentation that contains information about the entire system including all areas of knowledge. Even if the RO purification plant is not a very complex system there is still an interest to generate a complete documentation.

When purifying water for pharmaceutical purposes, the water quality is essential and strictly regulated by different national and international conventions. To reduce bacteria growth on the reverse osmosis membranes, water should as far as possible keep circulating in the system, thereby preventing colony forming units to cling on to the surfaces. The demand of pure water may vary over the course of the day, so in order to fulfill this requirement the output of the membrane is fed back, reentering the membrane if production is unnecessary. However, running this loop for longer periods of time also give rise to bacteria

growth, so to prevent this some of the water is constantly drained during this circulation mode. The plants considered in this thesis actually have a very low output demand of pure water and are therefore in this circulation mode most of the time. This gives rise to large amount of potable water being wasted. One way to improve this procedure is to continuously measure the water quality and only drain the amount of water necessary to keep a good water quality. In this manner the quantities going to waste is reduced while a water flow and a high product water quality is still sustained.

1.3 Goal

This master thesis has two major goals. The first one is to create a documentation of a RO water purification plant. To get a good overview of the different requirements, both from manufacturers, costumers and other stakeholders, a model based system-engineering approach will be used, where all the necessary information about the system is collected. SysML, a modeling language originating from UML is used to generate the documentation, see Section 3, since this was stated in the thesis description. This documentation includes for example blueprints of the system and graphical pictures of how the software communication works in order to control the plant.

The second goal of the thesis is to develop a new control system for the RO purification plant. This includes presenting a control strategy for feeding retentate water back into the system, and thereby reducing the water consumption of the whole plant. Since there will not be a real process to use during the course of the project, a simple simulation model will be developed to model and simulate the purification process. To control the plant, several state machines will be implemented in Matlab/Simulink using its extension Stateflow. The state machines are then implemented in C code and run on a PLC.

1.4 Thesis outline

The thesis outline follows to a large extent the workflow throughout the whole project. In Section 2 the background and theory behind the purification procedure is presented, with the main emphasis on the reverse osmosis part. Section 3 presents the basic of the modeling language, SysML. In Section 4 the actual SysML model for the system is discussed and developed. All parts, such as requirements, use cases and state machines are described with some examples from the model and at the end on this section some diagrams from the final SysML model are presented. In Section 5 a mathematical model of the reverse osmosis part is developed and used to simulate a control model of the plant. Also simulations of the events in the system are performed by implementing the state machines in Matlab/Simulink Stateflow. In Section 6 the implementation of the control system is described, including the approach taken to program the state machines in C. Last in section 7 conclusions drawn from the project are presented as well as a discussion about further improvements.

1.5 Individual contributions

Sofia's main focus has been on the construction of the state machines, first in Topcased and later using Matlab/Simulink and Stateflow.

Håkan has focused on the derivation of the mathematical model for the plant as well as the testing of the different control approaches.

The SysML model as well as the PLC implementation was done by both Sofia and Håkan.

2 Description of a pure water plant with reverse osmosis

As the name indicates the reverse osmosis purification plant uses the process of reverse osmosis to purify water. Both the basic osmosis process and the reverse osmosis process will be presented in this section. Apart from the reverse osmosis part, the plant also consists of other blocks. The water is first pretreated by filters and softeners. After the reverse osmosis process the water is deionized before the water is distributed. This section will describe those parts in more detail.

2.1 Osmosis

Osmosis is the flow of solvent arising when a semipermeable membrane is separating two solutions with different concentrations [3]. The semipermeable membrane only allows some molecules, usually water, to pass through it, while others, such as ions and larger molecules, are retained. Due to the concentration differences, an osmotic pressure arises, generating a net movement of solvent from the high solvent concentration (i.e. low solute concentration) side to the low solvent concentration (high solute concentration) side by passing the membrane, see Figure 2.1. Since the membrane only allows solvent to pass through it, there is no solute transport across the membrane and in this manner the concentration gradient is leveled out.

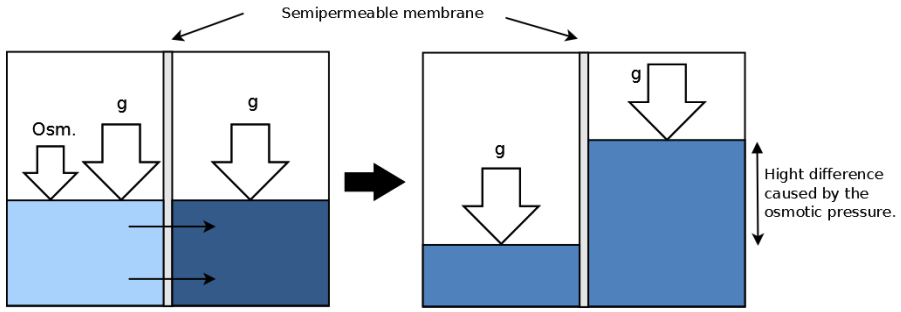

Figure 2.1: Two solutions with different molar concentrations separated by a semipermeable membrane. The differences in concentrations cause the solvent to flow from the high solvent concentration side to the lower, a process called osmosis. The gravitational force is denoted with 'g' and the osmotic pressure with 'Osm'.

2.2 Reverse osmosis

The flow of solvent through the membrane can be stopped if one applies an external pressure equal to the osmotic pressure on the lower solvent concentration side of the membrane. If the applied external pressure is greater than the natural osmotic pressure, one can actually force the osmotic flow to run backwards, i.e. solvent is forced to flow from a low solvent concentration to a higher, see Figure 2.2, and it is this process that has given rise to the name "reverse osmosis" [2].

The requirements on the product water specify the type of membrane needed. Different materials and structure of the membrane influence the selectiveness of the membrane. The first membranes developed had a uniform structure with

equal density throughout the whole membrane layer. However, it turned out that these membranes suffered from very low production rates. During early research, a few drops of permeate water per day were seen. Instead, the first commercial reverse osmosis membranes typically had a very thin dense surface layer, constituting the selectiveness of the membrane, while underlaying porous layers provided structural support, greatly reducing the total flow resistance and increasing permeate flow.

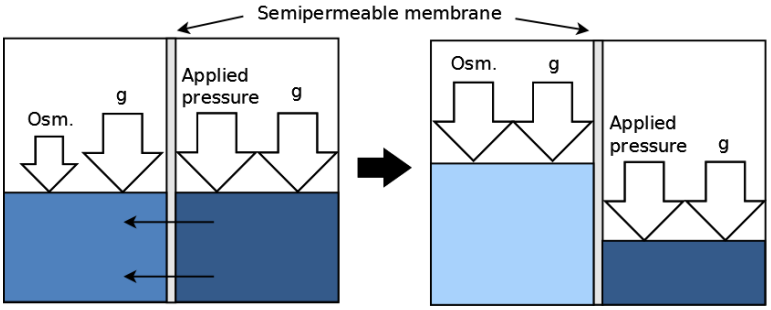

Figure 2.2: By applying an external pressure to the right side of the semipermeable membrane, greater than the osmotic pressure, solvent is forced from the low concentration solvent side to the high one. In this way, the water in the left compartment becomes more diluted, while the right one gets more concentrated.

2.3 Plant structure

The reverse osmosis process is often not used alone, but works in series with several other modules, all with their specific role in the purification process. The plant can be regarded as two parts. The first part mainly consists of a pretreatment part with prefilters and softeners while the second part contains the reverse osmosis part and deionization. Those parts will further on be referred to as Part 1 and Part 2 respectively.

2.3.1 Part 1: Prefilters and softeners

Upstream of the actual pretreatment there is a feed tank used for buffering, marked as number 1 in Figure 2.3. This is directly connected to the main water supply, and is used both for ordinary buffering, but also as a tank for feedback from the Part 1 output. As will be described later, water from other modules could be recirculated back to this initial buffer tank. After the feed tank the water is run through a pump, number 2 in Figure 2.3. This pump is used to push the water through the pretreatment block.

The reverse osmosis membranes are sensitive to fouling, and pretreatment of the water is often necessary. The plant model used in this project has a pretreatment block consisting of a multimedia filter, an active carbon filter and a softening block, all shown in Figure 2.3. The multimedia filter, marked as number 3, is used to filter out big particles that might be present in the incoming potable water. If the purification unit is taking its water directly from a lake or a river, then the presence of a multimedia filter is crucial for the long term durability of the plant. The filtration idea behind the filter is to let the incoming water pass through several layers of sand, each with different granularity and in this manner, incoming particles get stuck, while the water is allowed through. After some time of production the multimedia filter requires a cleaning. This is

performed by running the system backwards, a so called backwash. Dirt is then removed and drained.

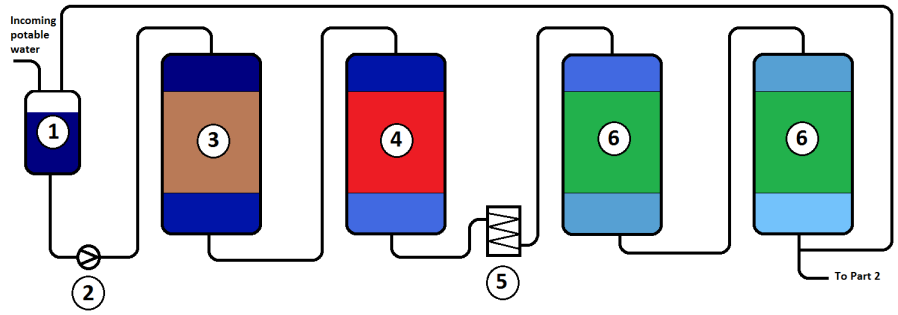

Figure 2.3: The main components of Part 1 is a feed tank (1), a multimedia filter (3), a carbon filter (4) and two softeners (6). To get pressure in the system a pump (2) is needed. Often also a heat exchanger (5) is present in order to perform hot water sanitization. If the incoming water quality is good, the filters might not be needed in the pretreatment. However, softeners are always part of the pretreatment to protect the RO membrane.

The next step in the pretreatment chain is the active carbon filter, number 4 in Figure 2.3. Its purpose is to remove organic chemicals and chlorine. Using surface water from for example a lake or a pond as input water, concentrations of organic acids arising from the breakdown of organic materials are often high. If allowed through the system, these acids can cause damage to the reverse osmosis membranes. If instead a local municipal source is used as the incoming water source, the water often contains chlorine added to control microbial growth. If not removed by the carbon filter, the chlorine will rapidly start degrading the reverse osmosis membranes and thereby reducing productivity. The functionality of the active carbon filter is similar to the multimedia filter. Except for feeding water through the filter during production the filter also once in a while needs to be cleaned by performing a backwash. Both the multimedia filter and the carbon filter can usually be omitted in Sweden where the water quality is good.

The last module in the pretreatment block is the softening block, number 6 in Figure 2.3. The purpose of the softener is the removal of scale forming calcium- and magnesium ions in hard water. If not removed, these ions will form deposits on the inside of the steel piping, as well as cause excessive fouling of the reverse osmosis membranes leading to their rapid degradation and clogging [4]. The scale is usually caused by calcium- or magnesium carbonate. The reaction between Ca^{2+} and two bicarbonate ions, $2HCO_3^-$ is shown in the following equation:

$$Ca^{2+} + 2HCO_3^- \rightleftharpoons CaCO_3 + CO_2 + H_2O \tag{1}$$

Here the calcium bind to the carbonate, creating the common insoluble compound calcium carbonate, maybe more commonly known as chalk.

To prevent the scale from building up, sodium chloride (salt) is added to the water. This causes the following reaction:

$$Ca^{2+} + 2HCO_3^- + 2Na^+ \rightleftharpoons Na_2CO_3 + CO_2 + H_20 + Ca^{2+} \tag{2}$$

The dissolved sodium ions from the salt bind to bicarbonate ions, which otherwise the calcium ions would have bound to and thereby the formation of calcium-carbonate is prevented.

To actually perform those reactions the softener contains an ion-exchange resin. During production, hard water containing magnesium and calcium ions are fed into the softener. The ions are attracted to the resin and exchanged for its sodium ions. After some time of production the resin is full of magnesium- and calcium ions and the system needs to be regenerated. First the system is backwashed to remove dirt from the system. A high concentration salt solution is then circulated in the system. Due to the high concentration of sodium ions the calcium- and magnesium ions are once again exchanged even though a single magnesium or calcium ion has a stronger attraction than a sodium ion. When the system is full of sodium the regeneration is finished by draining the calcium and magnesium ions and refilling of the salt tank. The softener can then go back into production.

The softening block normally consists of two softeners. When none of them regenerate or contain any errors the softeners are connected in series, but if one needs to regenerate or some malfunction occurs the softeners are switched to be connected in parallel. Due to this backup construction the production does not have to stop when a regeneration of a softener is required. However, when cleaning the multimedia filter and active carbon filter the production must stop since they are placed in the production line and can not be bypassed.

In order to clean the plant there may also be a heat exchanger present. By heating the water and letting it circulate through the system the hot water sanitization process is performed. During this process the external input to the feed tank and the output to Part 2 are both closed, forcing the water to return to the feed tank. When the process is finished the dirty water is drained and the system is filled up again by opening the input.

Together those six parts described above and shown in Figure 2.3 creates what is called Part 1. The circulation of Part 1 is not only used during hot water sanitization but also when the water for some reason is not good enough to let in to Part 2 or when the initial buffer tank in Part 2 is full. By stopping water of poor quality from entering Part 2 the RO membrane is protected and can be used for a longer periods of time between cleaning and replacement.

2.3.2 Part 2: Reverse osmosis and continuous electrodeionization

The main purification is, as mentioned before, performed by the reverse osmosis membranes. Before the water reaches the membranes, it is stored in an middle buffer tank, number 1 in Figure 2.4. The tank is used to buffer the incoming water from Part 1, but also as to provide a buffering tank for feedback from downstream components. After the tank, two pumps, numbered 2 and 4, are placed with the purpose of building up the high pressure needed to force water through the membranes. Normally the pressure drop over the membranes varies from 10 to 12 bar depending of plant size and the membrane type used [5] [6].

The reverse osmosis process itself takes place in long pressure vessels, number 5 in Figure 2.4, where the semipermeable membranes are placed. To maximize the area exposed to the pressurized water, the membranes are rolled up around a permeate channel, see Figure 2.5. Two or more membranes are placed back to back with a spacing material in between. Three out of the four sides of the membranes are glued together, thereby creating an envelope structure. The remaining open side is connected to the permeate tube, upon which the membrane construction is rolled. Water penetrating one membrane layer is trapped within the space layer and channeled to the permeate tube. Water not passing through the membrane surface is led out of the pressure tube as retentate. To reduce the waste water generated from the plant, several membrane units are often connected in series, where the retentate from the first unit is connected to the feed input of the second and so on [2]. The membrane units may also be

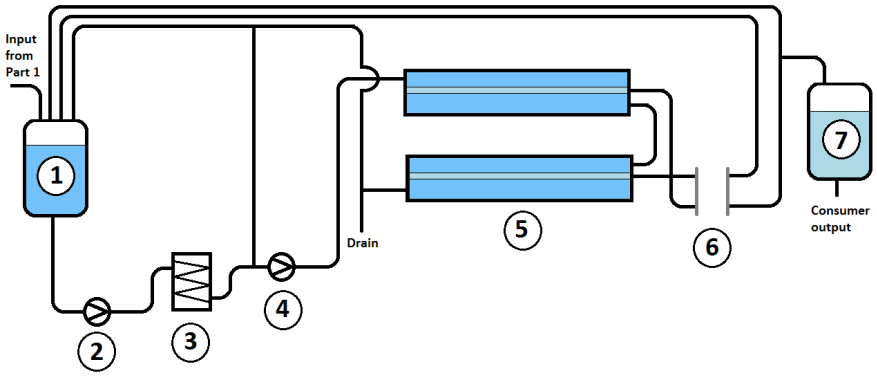

Figure 2.4: Part 2, like Part 1, also starts with a buffer tank (1), called the middle tank, where the output from Part 1 is connected and where water fed back from Part 2 is reentered. Water is led from the buffer tank, to the reverse osmosis membranes (5) via two pressurizing pumps, (2) and (4). It also passes a heat exchanger (3) which is used during hot water sanitization. The pumps force the feed water through the semipermeable membranes. Last, the permeate is led through a continuous electrodeionization (CEDI) module (6), and is at last collected in a holding tank (7).

connected in parallel. Retentate exiting the pressure vessels is then either led to drain or recirculated in the system back to the middle buffer tank. Normally, there is a correlation between the quality of the permeate and the retentate. In this project one of the goals is to automatically adjust the amount of retentate sent back to the middle tank, based on the quality of the permeate water, thereby reducing the water consumption of the plant.

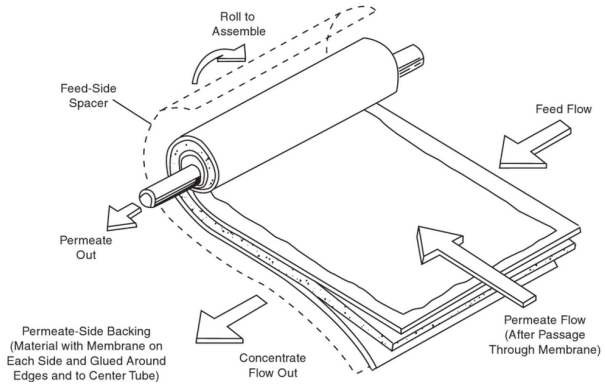

Figure 2.5: The semipermeable membranes are often rolled up in pressure vessels, where the reverse osmosis process takes place. Water is fed to one end of the tube, and water permeating through the membrane is collected in the permeate tube in the middle of the spiral-wound membrane element. Picture taken from [2].

After the reverse osmosis membranes, the water is led through a continuous electrodeionization (CEDI) unit. This module consist of an anode attracting the negatively charged ions and a cathode attracting positively charged ions. Between the anode and cathode, anode-selective membranes and cathode-selective membranes form channels. Those membranes are alternated and closest to the

anode there is a cathode-selective membrane while closest to the cathode there is an anode-selective membrane. This construction attracts the positively charged ions to one side and negatively charged ions to the other side but due to the placement of the selective membranes the ions get caught in every other channel. This also results in deionized water remaining in the other channels [7]. This process is graphically presented in Figure 2.6. After the CEDI unit the purified water is collected in a holding tank, numbered 7 in Figure 2.4. Like Part 1, Part2 also contains a heat exchanger, numbered 3, used during hot water sanitization. In warm countries the heat exchanger may also be used to chill the water before entering the RO unit.

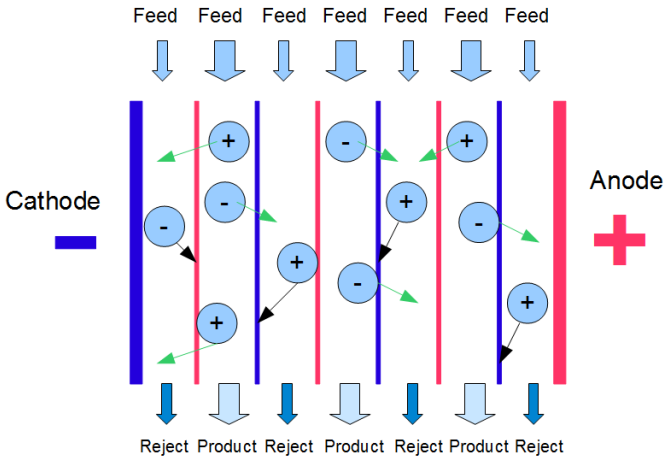

Figure 2.6: Principle used in the CEDI part. Water is fed through channels with selective membrane walls. The thin blue lines indicate cathode-selective membranes which let negatively charged ions pass while the thin red lines indicate anode-selective membranes which in turn let positively charged ions pass. The attraction to the anode, the thick red line, and cathode, the thick blue line, causes a movement of ions which is indicated by the arrows, green indicates penetration while black indicates rejection. Due to the construction of alternating the selective membranes the ions get caught in every other channel resulting in that deionized water remains in the other channels.

2.4 Related work

A lot of work and research have been put into developing better membranes for use in the RO-process. When it comes to modeling the membranes, several different approaches have historically been taken. One common approach is to derive a physical model for the different flows through the membrane. This is also the approach taken later in Section 5.1.1. Depending on the purpose of the model, the details and simplifications made can vary a lot from case to case. If the purpose of the model is to just get a basic idea of the different flows, a simple model is often enough. However, if the model is to be used with for example a model predictive controller (MPC), the demands for an accurate model will be much higher. There is much literature and numerous articles about modeling

the membrane modules themself, with several different approaches taken.

To get an accurate model, one approach is to divide the membrane modules into small thin segments, with each segment having its own balance equation for the different flows and concentrations [8] [9]. As the feed water enters one side of the membrane module, this side has a lower salt concentration than the retentate exit side, giving rise to an increasing osmotic pressure throughout the membrane roll. Also, physical phenomena such as concentration polarization are often also taken into account. Concentration polarization means that the concentration of solute is higher close to the membrane surface, which leads to a higher osmotic pressure. This is often important to consider when designing the plant, as the pump must be able to produce the required pressure. If not taken into account, the applied pressure from the pump might not be sufficient to produce the desired permeate flow. In the model presented later in Section 5.1.1, none of these aspect will be considered due to their complexity.

Apart from the physical modeling of the membranes, another approach is to derive transfer functions for the membranes by system identification [10] [11] . The change of flow in response to step changes of pressure and pH can be used to setup a multi-variable transfer function for the system. The drawback of this method is of course that the transfer functions derived will only be valid for that particular plant. Also, the dynamics of the membranes change with time, an aspect not always taken into consideration. Comparing transfer functions that model the same input/output relationship but from different plants can show very varying results [10].

In this thesis, a physical modeling approach is taken. Mainly because there was no real physical process available to perform a system identification on. If there was, it would be interesting to apply a pseudo random binary sequence (PRBS) signal to the pump, and observe the resulting permeate flow and concentration. As there will not be any real process data to verify the derived model, the models produced will be a rough approximation of the real behavior. Nonetheless, the models will be useful when verifying the various control approaches later considered.

3 Systems Modeling Language (SysML)

The Systems Modeling Language (SysML) is a graphical modeling language used to model and design various kinds of systems. Due to the complexity of today's systems, many different disciplines are involved in the development of a system, for example software, hardware, mechanical, electrical or chemical engineering. SysML is intended to help in bridging the space between the various system areas and help in the specification and architectural design of the system and its components. This section will present some background to SysML, its purpose as well as some basic semantics of the language.

3.1 Background

The society is constantly developing and more and more complex systems are required to meet the demands of the future. A system may consist of other subsystems and may also be a mixture of disciplines. Separate systems may work well on their own, but when put together, the overall behavior can be completely different. To get the complete system to work the surrounding systems need to interact with each other, regardless of the individual system discipline. To gather all disciplines, the term *system engineering* was established. A system can be defined as artificial parts put together and thereby achieving some kind of function that the individual parts can not achieve alone. The second part, engineering, can be seen as the method to develop systems [12]. Even though systems have been developed over hundreds of years the term system engineering emerged during the 1950s. The competition between world leading nations in areas such as space and military development created many new complex systems and the need of system engineering was established. It does not only focus on technical issues but economical and environmental aspects are also taken into consideration. The analysis can also include the entire life cycle of the system, from the idea that triggered the system development, to the realization of the system and finally to the disposal of the system.

Since the complexity of systems increase, the tools for handling system engineering also have to improve. From the standardized and widely used software oriented modeling language Unified Modeling Language, UML, the systems engineering modeling language, SysML was developed. Both languages are graphical and developed by The Object Management Group, OMG, with cooperating companies all over the world. SysML version 1.0 is based on UML 2.1.1 and was accepted by OMG as a standard in April 2006 and was made official in September 2007. Seven of UML's thirteen diagrams are used in either original or edited versions. Also two new diagram types are added, requirement and parametric diagram, to extend the modeling perspective [12].

3.2 Motivation for a model based engineering approach

There are mainly two approaches when modeling a system, document-based and model-based engineering. The most widely used is the document-based system engineering. Here all specifications are stored either as printed hard copy documents or electronically. Those documents are then exchanged between developers, testers, costumers and project managers. Although this method obviously has worked historically, it suffers from some drawbacks and limitations. Graphical software might be used to model software behavior and schematic block diagrams. These are all stored in separate files later included in the documentation. Traceability between system requirements and the final design choice can be hard to follow when the specifications are spread out in this manner. Ad-

ditionally, as the system specifications change and evolve, the documentation may be hard to keep up to date for everyone involved.

The model based system engineering approach on the other hand gathers all documentation in the same digital documentation. First, the basic system model is set up, where all the requirements, blueprints, use cases and desired overall behavior can be included. As the project then evolves new documentation is added, such as the software or hardware model, allowing traceability between different parts of the model. By having all the documentation digitally in one place and synchronized to a repository, one can be sure of always having the latest specification at hand [13].

3.3 Diagram types

SysML has a total of nine diagrams types that are divide into three categories: structural, behavioral and requirements, see Figure 3.1.

3.3.1 Structure Diagrams

To model the structure, Block Definition diagrams, Internal Block diagrams, Package diagrams and Parametric diagrams are used. The Block Definition diagram can be used to show the components in the system, such as parts and connectors, and their reciprocal relationship. These also show the properties of the block and the operations that a building block is intended to perform. The internal block diagrams on the other hand is explicitly used to show the internal structure of the system and its components. One can say that the Block definition diagrams are used to specify the building blocks available to the developer, while the internal block definition diagrams are the blueprint where the building blocks are put together in a context. The Package diagram is used for model management and the diagram shows how packages are organized in the model. This is also used in UML [14]. The parametric diagram is, however, a newly added diagram type that graphically shows the parametric relationship between block properties. An example from the RO-plant may for example be how pressure relates to permeate flow.

3.3.2 Behavior Diagrams

To describe the behavior and dynamics, four types of diagrams are used: activity diagram, sequence diagram, state machine diagram and use case diagram. All of these diagram types are also used in UML. The activity diagram is slightly modified compared to UML's but the usage is still the same. It shows the control and data flow and is used to describe functionality. The sequence diagram shows the interactions between system components, for example the communication between programs. The state machine diagram is used to see what sequences of states that a component or interaction do when an event occurs. When designing a program this type of diagram may be useful. The last behavioral diagram is the use case diagram which shows the relationship between the system user and the functional requirements. These diagrams are further discussed in Section 4.6-4.8 where examples are presented.

3.3.3 Requirement Diagrams

The requirement diagram is a new type of diagram that shows the system requirements and their relationships with other elements. Requirements are usually divided into functional and non-functional requirements. During the development process, the first thing to do is to state the most basic requirements.

From these basic requirements, new ones are derived and referenced. In this way, one can trace every requirements down to the most basic needs of the system. This diagram type gives the opportunity to perform requirement verification and validation which can be very useful. [13].

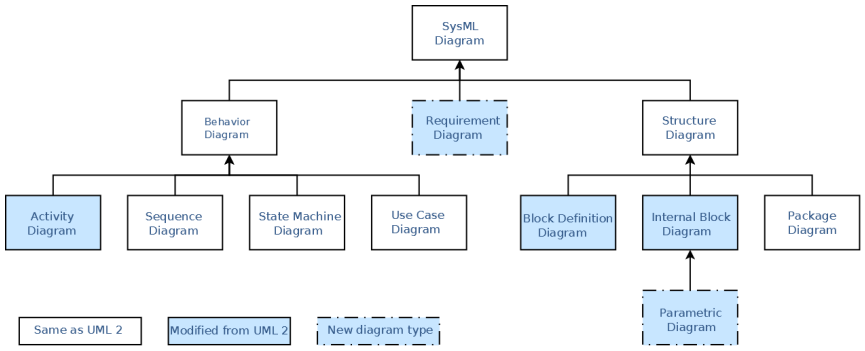

Figure 3.1: Overview of SysML diagrams.

3.4 Structural elements

Before starting to build something, one has to have access to some building blocks. In SysML there are several important structural elements used in different diagrams. In the building block diagram the main constructing component is simply called a block, and is quite similar to the class block in UML. Blocks are modular units of system description used to represent a physical or logical decomposition of the system or the specification of software, hardware or human elements [14]. Every block can include a series of features specific for the model element of interest and provide a general purpose capability to create hierarchies of system components.

In the building block diagram, a block make up the structural backbone of the model. Connected to the block are a series of properties: *parts*, *references* and *value*. Parts describe the decomposition of the block into its smaller elements, while references indicate that there is some sort of weaker relationship between the block at hand and some other block. The value properties can describe certain characteristics of a block, for example the component's weight. Figure 3.2 is an example of a building block diagram showing a valve represented as a block and its properties. The composite association is shown as a line ending with a black diamond. Numbers next to the associations show the multiplicities of associations. In the figure one can see that that valve is composed of one handwheel but one to two packings. Also, the valve's reference association with the RO-plant is shown as a line ending with a white diamond. The blocks can also contain a behavior. For example, the valve block has the operation controlFlow().

To model a requirement, a requirement stereotype is use. The most basic requirement stereotype consists of a string field describing the requirements, as well as an ID number. When documenting the requirements for a system, one usually start out with the most fundamental and as work progresses, new requirements are derived. Figure 3.3 shows a simple requirement hierarchy, with the most fundamental requirement at the top. In order to organize the model, complex requirements are often divided into simpler once using the *containment* relationship, depicted as an line ending with a crosshair. Contained requirements should not add any meaning to the model, but just add differentiation

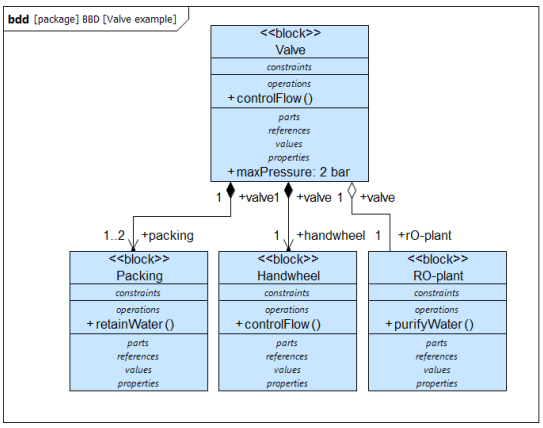

Figure 3.2: Example of a block and its relationship with other blocks. Here one can see that the valve has a composite association with a packing and a handwheel block. The numbers next to the associations show the multiplicities of associations, i.e. in this case the valve contains one to two packings and one handwheel. Also, there is a reference association with the RO-plant.

of requirements into different branches. Another important relationship is the *derive* relationship. This is fundamentally different from the containment relationship. Where a containment relationship is just a differentiation, a derived relationship is based on an analysis, and is depicted as a dashed arrow with the keyword «deriveReqt». The arrow points at the parent requirement. To clarify the reason for a particular design decision a *rationale* can be associated to either a requirement or a relationship between requirements. To provide traceability in the model, one can use a *satisfy* relationship to assert that a model element satisfies a particular requirement, or a *trace* relationship to show the reason for the requirement. In the figure the valve block is said to satisfy the requirement of using a valve for flow control.

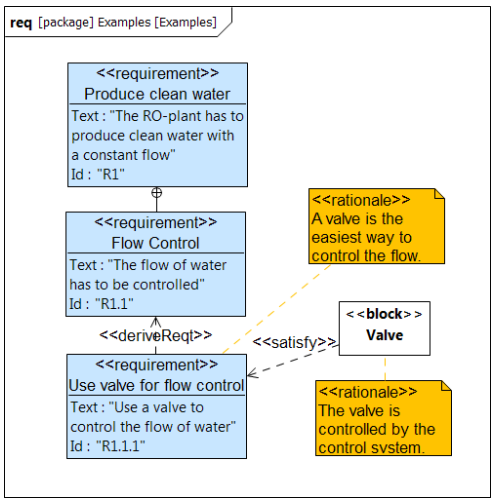

Figure 3.3: Basic example of requirement semantics. Requirements are differentiated using the *containment* relationship, while new requirements are related to their parent requirement using the *derive* relationship.

3.5 Workflow

In system engineering it is important to clearly state the current situation and what the goal is. When first developing the system model, the first thing to do is to determine the purpose and the scope of the model. The purpose of the model could for example be to document and model an already existing system or to specify and design a new system to be developed. When doing the later, emphasis will be more on defining the requirements and general system design, than on for example specifying the software implementation.

When having defined the purpose of the model, the next thing to consider is the scope. This is done in terms of model breadth, depth and fidelity. Model breadth means the limits of the model, i.e. which parts that should be included in the system and which parts should not. An example from the RO-plant could for example be to focus on elements involved in the purification process, but for example not include maintenance routines. Model depth on its hand means how detailed the model should be. Generally the model starts quite basic, just sufficient for the basic model to be set up and as development continues, further details are added to the different building blocks. To again use an example from the RO-plant, one would for example start to model a valve as a black box controlling water flow, but later add the type of valve and its controller interface. Finally, the fidelity of the model means that the required level of detail has to be in line with purpose of the model. A low fidelity diagram might suffice to model the ordering of actions in an activity diagram, but not enough for a communication protocol [13].

As hinted above, the SysML workflow is an iterative process, where the model is improved gradually throughout the development process. As requirements are gathered and use cases are created, new functional ideas may come up, making it necessary to add new contents. Also, you may notice use cases that are not met by any requirement. Additionally, during the structural or behavioral development new ideas may be created forcing you to go back and change in the model. This iterative nature of the process is illustrated as a SysML activity diagram in Figure 3.4. Finally, in the modeling stage, test cases can be created to verify that requirements are satisfied. All of this makes the modeling a bit exhausting but when finished a complete analysis and documentation of the system is done and hours and money will be saved during later stages.

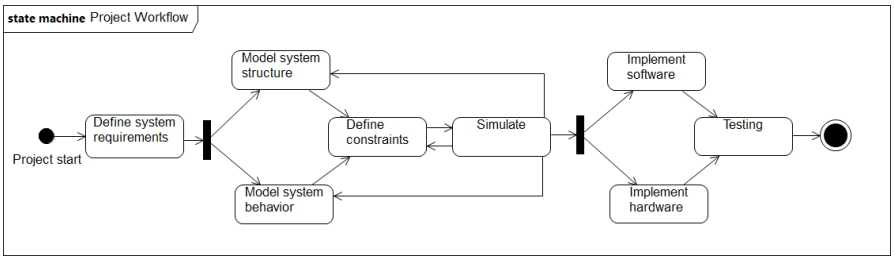

Figure 3.4: The general workflow throughout a project, starting with the collection of requirements and ending with testing.

3.6 Modeling tool

In this thesis, an Eclipse based open source program called Topcased is used to create the SysML model [15]. This program is still under development and unfortunately all of SysML's ideas are not yet supported. In Figure 3.5 a screen

shot from the development environment is shown. A complete documentation of the model can be generated but it is easier to get a good overview by browsing the project digitally. This can be done directly in Topcased or by generating an html page. Topcased also contains a simulation engine to run the state machines, but in this project Matlab will be used instead, since its Stateflow environment is very well tested. At the moment Topcased is in a phase of merging with Papyrus which is a open source tool for graphical UML2 modeling.

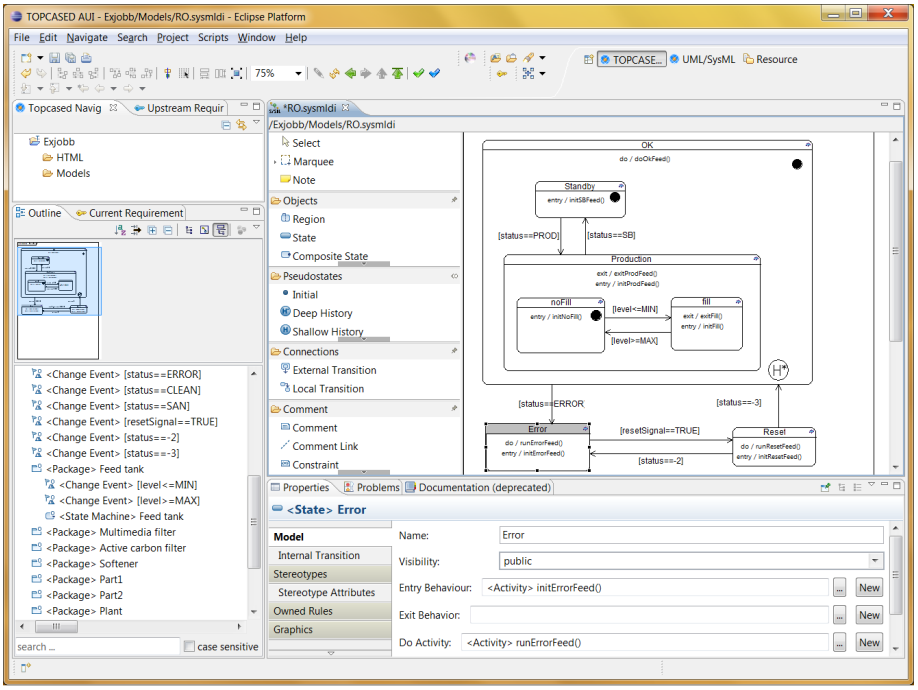

Figure 3.5: A screen shot from the development environment in Topcased.

4 SysML model building

In this section the purification plant will be modeled. The SysML model is based on the approach presented in Section 3. Remember that the model development is iterative so a model step is not completely finished when the next step is started. During the development of later steps you may realize that new information needs to be added to earlier steps. When modeling the behavior, focus will be on use case diagrams and state machine diagrams. Only some examples of sequence diagrams and activity diagrams are introduced. In the last part of this section some diagrams from the final model are presented.

4.1 Model purpose

As described in Section 3.5, the first thing to do when starting to model the system is to define the purpose and scope of the model. The purpose of the model developed in this section is to get a good documentation of the system, and give a good overview of the different requirements stated by the different stakeholders, with the later also defined. Additionally, in order to develop a control system, a complete description of the system components is modeled in building block diagrams. Further, internal building block diagram are built to describe the physical relation between the components.

To model the control system, several state machines are developed. Here, only the general design of the state machine will be drawn, since dynamic modeling is better done in for example Matlab/Stateflow. To show the communication between the different state machines, some sequence diagrams are created. Activity diagrams are also constructed to show some of the functionality sequences used in the program.

4.2 Requirements

After defining the model purpose, requirements on the system were collected. The requirements were divided into five categories: functionality, usability, reliability, performance and supportability. This grouping is called FURPS and is developed by Robert Grady at Hewlett-Packard [12]. This section will only show some examples of requirement diagrams. For the full documentation, please see Appendix A.1. The first category describes the functional requirements while the other four categories describes the non-functional requirements. The functional requirements describe the capabilities that the system must have. For a RO-plant the most basic capability is of course to produce clean, pure water. To accomplish this, the plant has to be controlled by a control system, providing the logic for the control of the plant. The identified requirements on the control system are summarized in a requirement diagram in Figure 4.1. In addition to this, due to legislation, data from the water quality has to be logged to provide traceability if the water later is discovered to have been of bad quality [16].

One important requirement that will be essential to the outline of the SysML model itself and the control system is that the plant should be modular. With modularity, one means that parts should be easy to bring into and out of the system without the need to remake the whole control system or graphical interface. The need for modularity exists because not all of the prefilters are present in every plant configuration, for instance, as mentioned in Section 2.3.1, in Sweden the multimedia and active carbon filter are usually omitted.

Included in the functionality group are also features that are not themselves completely necessary for the overall plant function, but are requirements to ensure the safety and security of the plant. i.e. alarms and alerts. The most

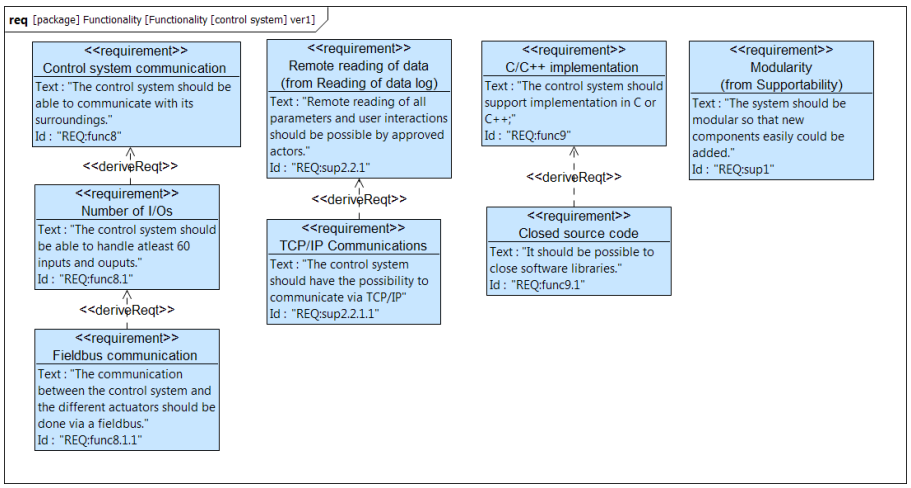

Figure 4.1: Identified minimum requirements on the control system. This will later be the foundation upon which an appropriate control system is selected.

important of the two is the alarm category. An alarm is a signal that something fundamental is wrong with the process. A pump might have stopped working or an essential sensor for control is no longer sending signals. The alarms will automatically put the plant in a fail safe state, where the safety of the personnel working with the plant is the number one priority. An alert on the other hand is just a notification to the operator and does not require any direct action. However, it is still an important message to the operator. An example might be that a pressure transmitter, not essential to the overall plant performance, is not responding. Some alerts may also trigger an alarm after a certain amount of time if not taken care of.

The usability requirement can be seen as the customer needs. This is mainly the requirements on the user interface. Two main requirements are an easy to use interface and four different access levels. The usability requirement diagram can be seen in Figure 4.2. The reliability is connected to the frequency and probability of failure. No such requirement has been stated but of course the failure probability and frequency should be very low. Response time, resource usage and speed are examples of performance requirements. Since the purification plant is a very slow system speed requirements are not an issue. A resource requirement is, however, to waste as little water as possible. Finally, the supportability requirements concern things such as maintainability, configurability, testability, compatibility, installability and portability. Some of the requirements in this category are for example that it should be possible to control an actuator manually, to override sensor values, access data remotely and that many other parameters should be configurable.

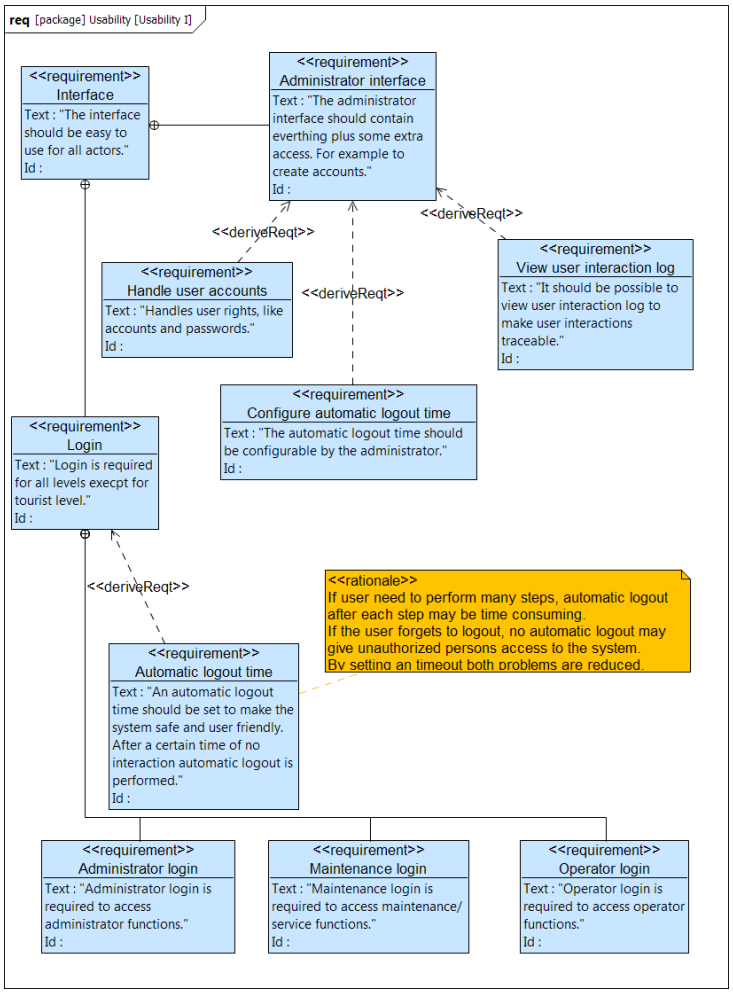

Figure 4.2: Requirements concerning the usability of the system, often the costumer needs.

4.3 Stakeholders

Parallel to the requirement collection, stakeholders were identified. A stakeholder is someone who has an interest in the system. This includes for example costumers, manufacturers and legislation. It is important to state the stakeholders in an early phase of the process since they may have requirements on the system. If a stakeholder is forgotten it may cause serious problems when detected. In order to find all stakeholders it could be a good idea to ask an already identified stakeholder for other stakeholders. Identification of the stakeholders of this project was performed by asking a domain expert who not only knew the system very well but also cooperates closely with both costumers and manufacturers [5]. The obtained stakeholders can be seen in Figure 4.3.

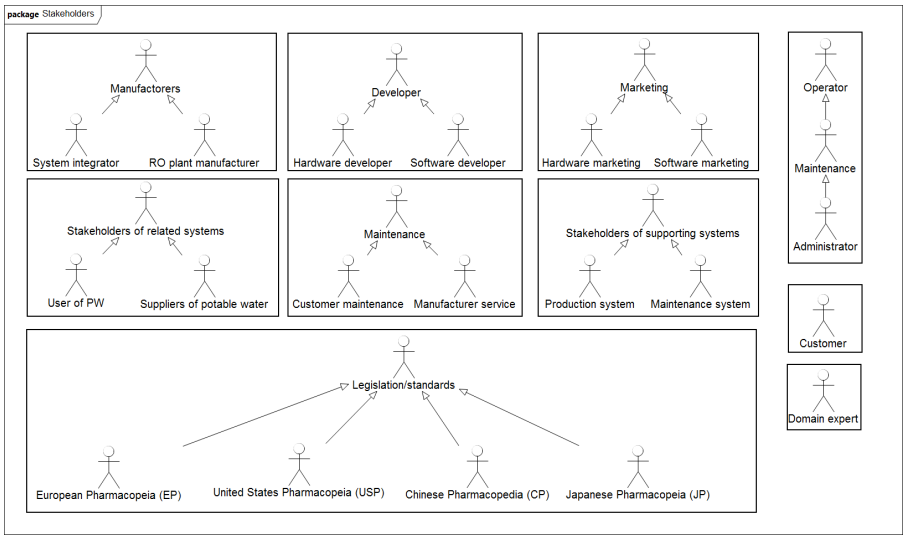

Figure 4.3: Stake holders identified during the modeling process.

Many of the stakeholders will have requirements outside the scope of the model such as a stakeholder of a related system, but they are still included in the model. The most important stakeholders to consider when later developing a control system, are the direct user of the system including an operator, maintenance and administrative personnel. Other important stakeholders are the developers of the system, both software and hardware. The software developers will for example have an interest in a well documented software structure and an 'easy to read' code for debugging.

4.4 Building blocks

Building blocks, which were introduced in Section 3, are the parts that the system contain. As described in Section 2.3 a RO-purification plant can be divided into Part 1 and Part 2. Each of these blocks can then be divided into several other blocks which gives a more detailed description. For example Part 1 can be divided into a feed tank, a multimedia filter, an active carbon filter and a softening block. These blocks, in turn, can be divided into even more detailed blocks. The lowest level described consists of valves, pumps, tanks, power supply, CEDI unit, RO unit etc. The plant parts are thus not modeled into every screw and nut. Frequently used blocks were collected in a library that later was used to build blocks at a higher level. The building blocks do not show how the blocks are connected to each other, as this is handled by the internal

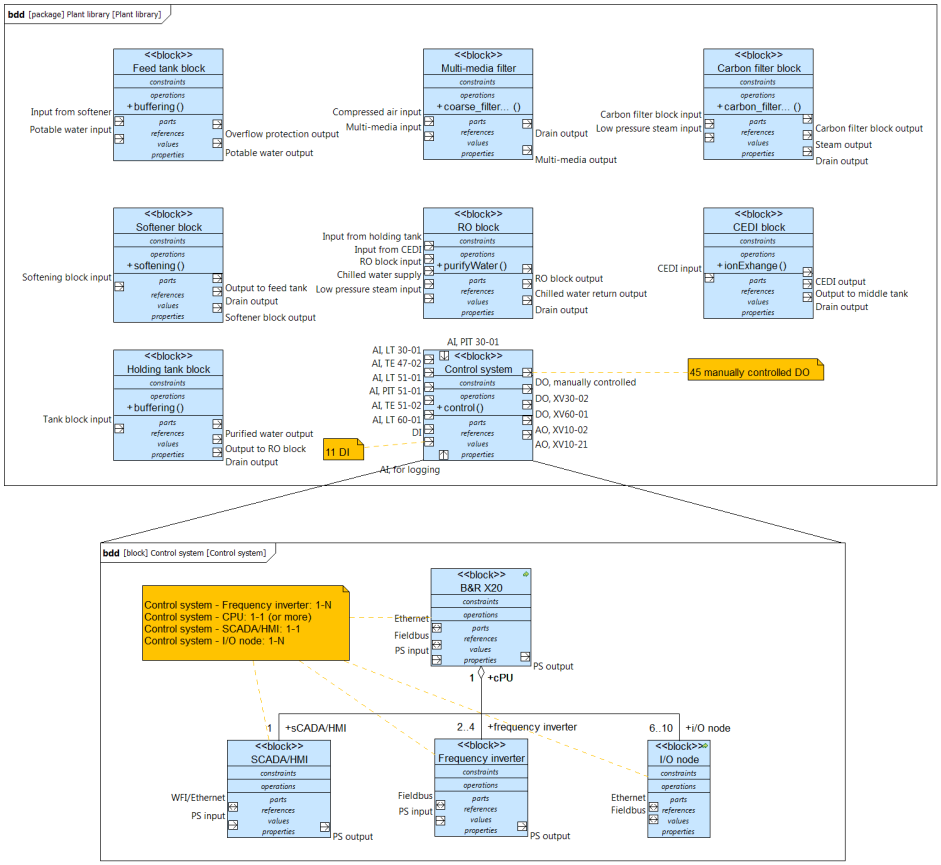

Figure 4.4: Building block diagram showing the decomposition of the plant into its individual modules together with the ports connected to these modules. These modules are then further modeled. As an example the control system modeling is shown.

building blocks described in the next section. However, to identify the building blocks the company's construction drawings were studied. Figure 4.4 shows a building block diagram where the modules of the plant are depicted. Each one of these module blocks are linked to other building block diagrams, showing the internal decomposition of the modules. The figure also illustrates the building block diagram for the control system. Furthermore, the figure shows the ports connected to each and every module. See Appendix A.4 for selected building blocks from the documentation.

4.5 Internal building blocks

As mention in Section 3, the internal building blocks show how the building blocks are connected. By combining different building blocks different models of the RO-purification plant can be created. For example if the ingoing water is of high quality all the pretreatment blocks does not have to be included. Different blocks with equivalent basic functions may also be created but when constructing a specific plant model only one type is chosen. For example a valve may have different numbers of digital outputs depending on the required status feedback. By studying the company's construction drawings for different models internal building blocks diagram were created. See Figure 4.5 for an example

and Appendix A.5 for parts of the documentation.

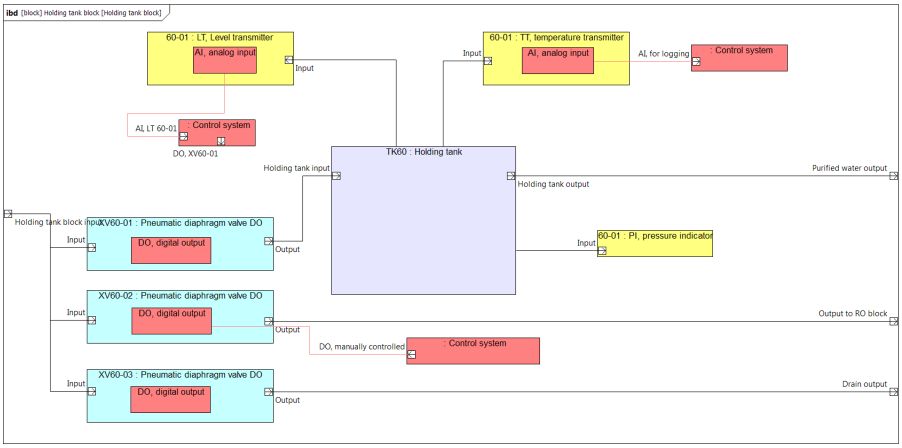

Figure 4.5: Internal building block diagram showing a drawing for the RO-part holding tank. By color coding different kinds of parts, one can more easily interpret the structure. This diagram is used to show a building block in its physical context.

4.6 Use cases

Use cases describe the functionality of the system in terms of user interactions. It is used to model the relationship between the system at hand, its actors and user cases [14]. The different actors acting on the system are most often taken from the already defined stakeholders. Use cases can be seen as a way to capture system requirements in terms of the use of the system. Therefore use cases come to include the direct users of the system, i.e. the operator, maintenance personnel and administrators. By asking key stakeholders, for example the system manufacturer or domain expert, the various use cases can be identified. From these, new requirements are added to the requirement diagrams, further underlining the iterative nature of the modeling process.

The most basic actor acting on the system is the observer. This is a person not allowed to affect the system in any way, but only view the current status of the plant. Important to notice is that the observer is not a stakeholder. In Figure 4.6 the observer use case diagram is depicted. The users of the system can be organized in a tree where the observer is the topmost parent. The child to the observer is the operator, who is the person in charge of the everyday operation on the plant. Often this person is not qualified to change any plant parameters such as PID-parameters or ratios between flows, but instead just monitors the plant and refills salt and other additives when needed. As the operator monitors the plant, reseting and checking of alarm and alerts are part of the routines. Accompanied with every change done to the system, a comment about the reason for the changes should be included. All of this is depicted in the figure 4.7.

The user with the most advanced use cases is the maintenance person. This is the person with the most knowledge about the process itself. The maintenance user should have access to all the actions described in the observer and operator use case diagrams, as well as wide range of actions to configure and tune the plant. An example could be to manually open and close valves, override sensors in order to simulate failing system components or configure alarm and alert

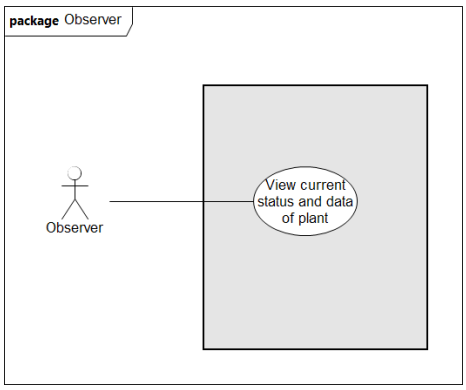

Figure 4.6: Use case for an observer.

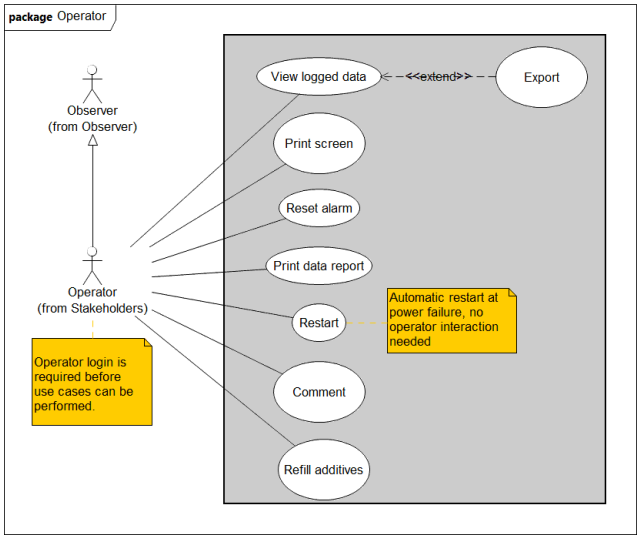

Figure 4.7: Use cases for the operator. The operator inherits the use cases from the observer, shown in Figure 4.6.

levels. All identified actions are summarized in Figure 4.8. The maintenance use cases are the ones that give rise to the most requirements on the control system and on the human machine interface (HMI).

The administrator inherits all use cases from the maintenance user and is the bottommost user. The detailed use case diagram can be seen in Appendix A.3.

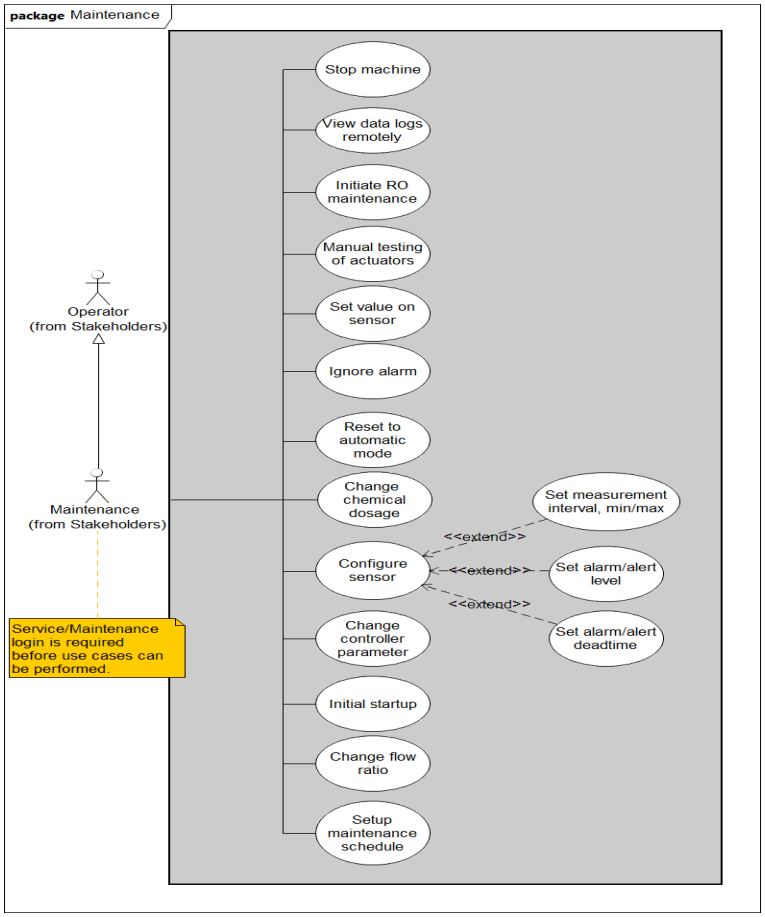

Figure 4.8: Use cases for the maintenance actor. These activities give rise to many configuration requirements in the requirements diagrams.

4.7 State machines

The RO-plant has several different operation modes. For example the plant could be in production mode where the plant produce pure water or it could be in circulation mode where the plant is running but the water is not distributed to the consumers. The production mode is simply used to satisfy the consumer and give them access to purified water. If the water quality becomes too poor or if the last buffering tank is full due to low demand of purified water the production needs to be stopped. Still, to prevent bacteria growth, the water through the membranes has to keep circulating and the plant is therefore set to circulation mode. To model this dynamic process of producing pure water the plant was considered as a state machine where each state indicates an operation mode, see Figure 4.9.

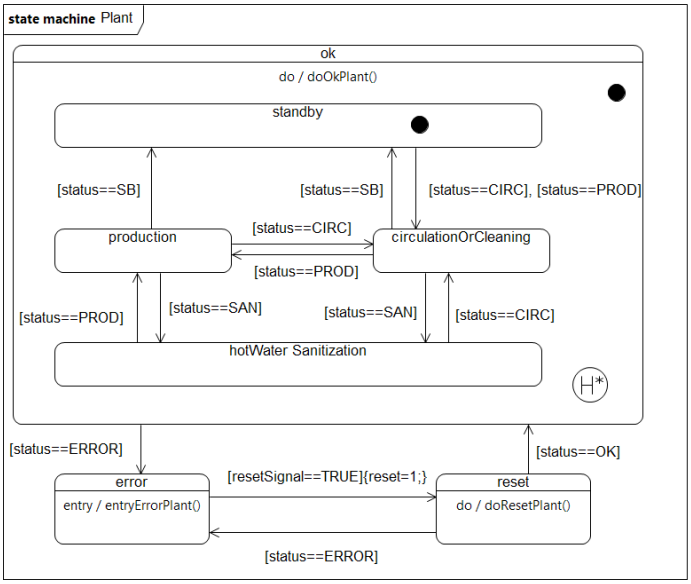

Figure 4.9: The state machine of the plant.

Instead of an ordinary flat state machine, where all the states are on the same level with no inner states, a hierarchical state machine is used. A state machine consists of states that are graphically presented as a rectangular box with rounded corners. In each state different actions can take place. The actions are divided into *entry*, *exit* and *do* actions. The entry and exit actions are executed every time the state is entered or exited respectively, while the do actions are executed when the state is active. To transfer between different states, transitions are used. They consists of a condition, embraced by square brackets, that should be fulfilled. The transition could also include an action which is embraced by curly brackets. To indicate the initial state a black dot is placed in the specific state. The state chart can also remember the history if the state contains a history junction, denoted as an encircled H or H*. When a state containing a history junction is exited, the last active state is stored and the next time the state is reactivated it returns to this stored state. Using an ordinary encircled H the history is shallow and only one level of state is remembered but by using the deep history notation with an additional * the same state as was left is revisited, regardless of the depth of the state. Also, a transition can be split using a diamond symbol, offering a choice for the final destination state. This is just some basic information about state machines

but for now this knowledge is enough. In Section 5.2.2 the state machines are simulated using the Matlab/Simulink extension Stateflow and then some further functionality is presented.

Not only the entire plant can be modeled as a state machine, but also the separate parts of the plant. For example Part 1, which consists of the feed tank, the multi-media filter, the active carbon filter and the softening block, can be regarded as a part with its own state machine. Part 1 can also be in production or circulation mode since the output of Part 1 could be either closed or opened. If it is open, the part is in production mode and if it is closed the water is led back to the feed tank and the part is in circulation mode. The state machine modeling does not stop at this level but the feed tank, the filters and softening block can in turn be regarded as individual state machines. However, their states do not have to be the same. For example the filters have production and error modes but no circulation mode. This division of the separate parts of the plant into their own state machines, further underlines the modular approach stated by the SysML model.

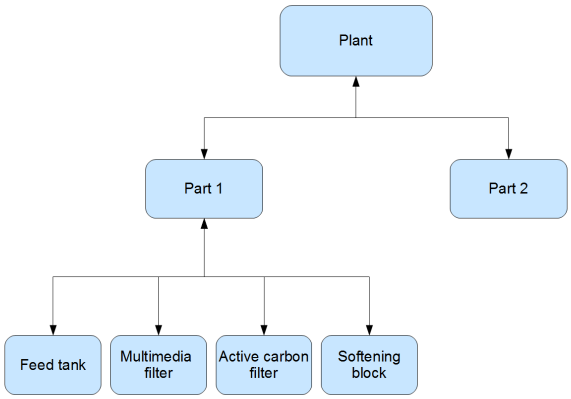

Figure 4.10: The state machine of the plant can be split into several state machines which are arranged in a hierarchical way.

The state machines are also arranged in a hierarchical way, see Figure 4.10. The plant is at the top with two children, Part 1 and Part 2. Part 1 in turn has four children, feed tank, multi-media filter, active carbon filter and softening block. Part 2 does not have any children. It consists of the middle tank, RO, CEDI and holding tank but those parts are not modeled individually since they are all vital parts and can not be excluded in the construction. The communication between the state machines is arranged in the manner that lower level parts report to its parent when a state change has occurred. The parent then handles the event and sends signals to the other parts that are affected by the event. For example, if the active carbon filter needs to do a backwash, it sends a cleaning request to Part 1. If the cleaning procedure is possible, Part 1 sends a cleaning confirmation back to the active carbon filter and appropriate signals to the other children and the Plant.

All state machines are presented in Appendix A.10 but as an example the state machine of Part 1, which has both a parent and children, is presented in Figure 4.11. Even though the states may differ somewhat, the main design features are the same. All state machines consist of three outer states: ok, error

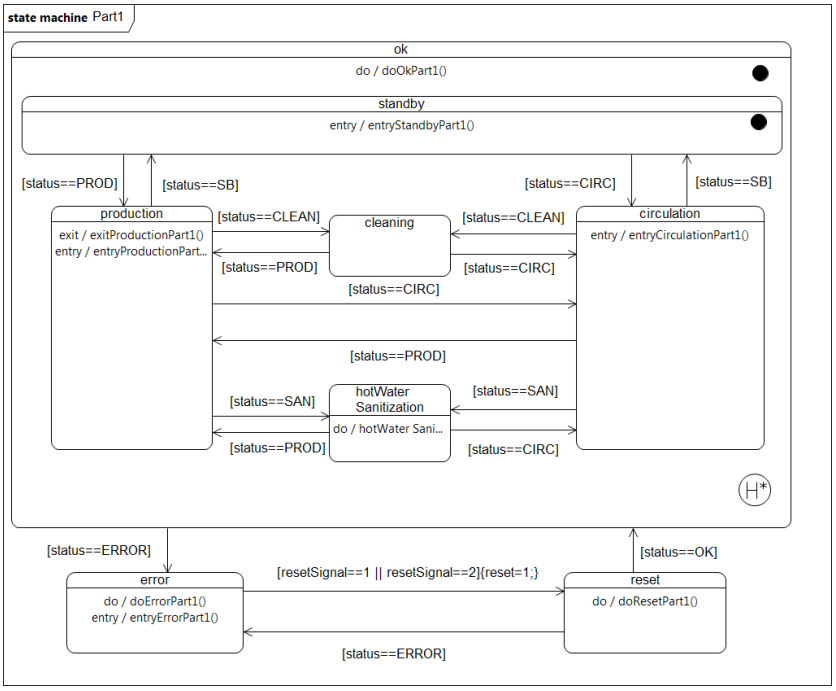

Figure 4.11: State machine for Part 1 which includes the feed tank, multi-media filter, active carbon filter and softeners.

and reset. When the ok state is active the part is ready to run, or is in some way running already. The ok state of Part 1 contains five inner states: standby, circulation, production, cleaning and hot water sanitization. In standby mode, the feed tank input is closed, which could also be considered as the input to Part 1, the output to Part 2 is closed and instead the valve to circulate Part 1 is opened. Also all pumps are turned off. The production state indicates that all children are producing, the input is controlled by the level in the feed tank and the output to Part 2 is open. The same settings are used in circulation mode except that the output to Part 2 now is closed and the water is fed back to the feed tank. When the Part 1 cleaning state is active it indicates that some of the children are cleaning. It could either be backwashing a filter or regenerating a softener. This state is, however, not activated when only one softener is regenerating since the double configuration of softeners then switches from serial to parallel mode. In this manner the production or circulation is not affected by a softener regeneration. During the hot water sanitization the setting are the same as for circulation but the heat exchanger is also turned on. The error state is simply used to indicate that something in the plant is not working properly. This state is activated for all parts regardless of where the actual error occurs. Due to the directive 2006/42/EC on machinery [17], the state machine has to pass though a reset state before the ok state can be reentered. If the error is handled when an reset signal occurs the process returns to the ok state and by the deep history it returns to the specific inner state that was active when the error occurred. However, if the error is still present, the error state is reactivated.

As mentioned before the state machines will be discussed more in Section 5.2.2 where the simulation is presented.

4.8 Activities

The actions that occurs in the state machines can be further modeled by the usage of activity diagrams. In an activity diagram the initial start point is, as in the state machine diagram, shown as a black dot. The activity is presented in a box and transitions between parts are show by arrows. Other parts that are frequently used in activity diagrams are the fork node and join node. These are illustrated as bars and as the names indicate, the fork node splits the flow while the join node combines it. Those nodes can be used to synchronize the flow. The merge node and decision node also combine and split flows but lack the synchronization. A dashed region with a hourglass indicates that the region might be interrupted. To show that an activity has come to an end the exit point is used. It is illustrated as a black dot encircled with an extra circle. An example is presented in Figure 4.12 where the production state of Part 1 is just entered. The output valve to Part 2 is opened and the recirculation valve in Part 1 is closed. In more detail this can also be modeled as in Figure 4.13. Here the response times for the valves are limited and depending on the reply the activity diagram reports different outcomes. The last activity diagram is a bit more detailed but for the modeling purpose the first is enough.

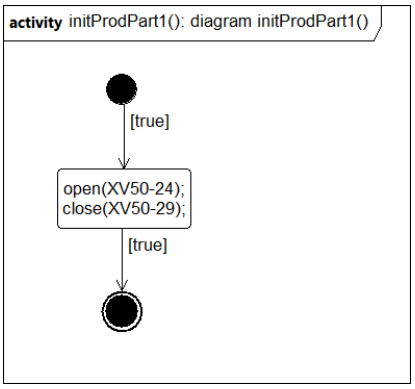

Figure 4.12: This activity diagram shows the actions that occur when the production state of Part 1 is entered.

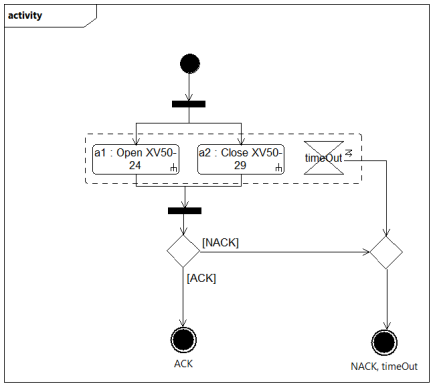

Figure 4.13: In this activity diagram the outcome of the activities are taken into consideration. If the valves limited response time expires or if the reply is not as expected the activity takes an alternative exit path.

As the activity diagram is well suited to model functionality sequences, the sequence for cleaning is shown in Figure 4.14. When performing a cleaning the following sequence is executed:

1. Fill holding tank
2. Backwash
3. Afterwash
4. Reset

Some more examples of activity diagrams can be seen in Appendix A.8.

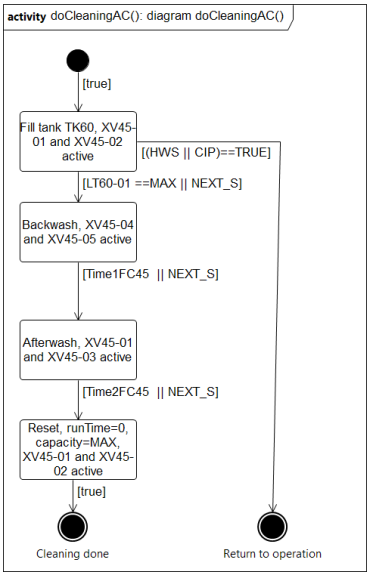

Figure 4.14: Activity diagram illustrating the cleaning sequence of the active carbon filter.

4.9 Sequence diagram

A sequence diagram shows how different parts in the system interact over time. Figure 4.15 illustrates the sequence of the communication, when the power is turned on. Each part has a timeline and the communication between the parts is presented by arrows. A call is a solid line while a reply is dashed.

More sequence diagrams are shown in Section 6.4 and in Appendix A.9.

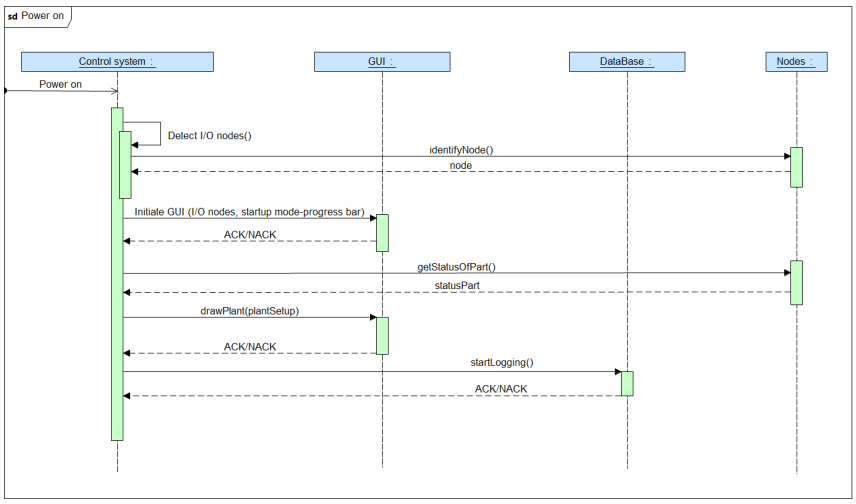

Figure 4.15: A sequence diagram showing the communication between the system parts when the system is powered on.

4.10 Final model

As mentioned before, the modeling procedure is an iterative process, where new content is added as the project proceeds. Initial basic diagrams are continuously updated until they reach the desired depth of the model. To show this, Figure 4.16 shows the final version of Figure 4.2, where the reason for the different requirements are added. As many of the usability requirements originate from use cases, most of the requirements are also traced back to these use cases. But the *trace* relationship only shows the reason for the requirement. To show that a requirement has been meet, the *satisfy* relationship is used. In the figure, the whole login requirement is said to be satisfied by the control system.

Diagrams modeling the intended dynamic behavior of the plant are especially prone to be changed as the project progresses. Often, the first idea of how the software should communicate or how a synchronization phase should be performed has to be redesigned during the implementation phase. Still the first ideas are used as a foundation for the continued development. Figure 4.17 and 4.18 show the initial and final state machine for the softener. For example one sees that the hierarchical approach was not initially considered and that the reset state was not included. Also, to illustrate the functionality sequence, an activity diagram was added in the final version, instead of creating states for each function, see Figure 4.19.

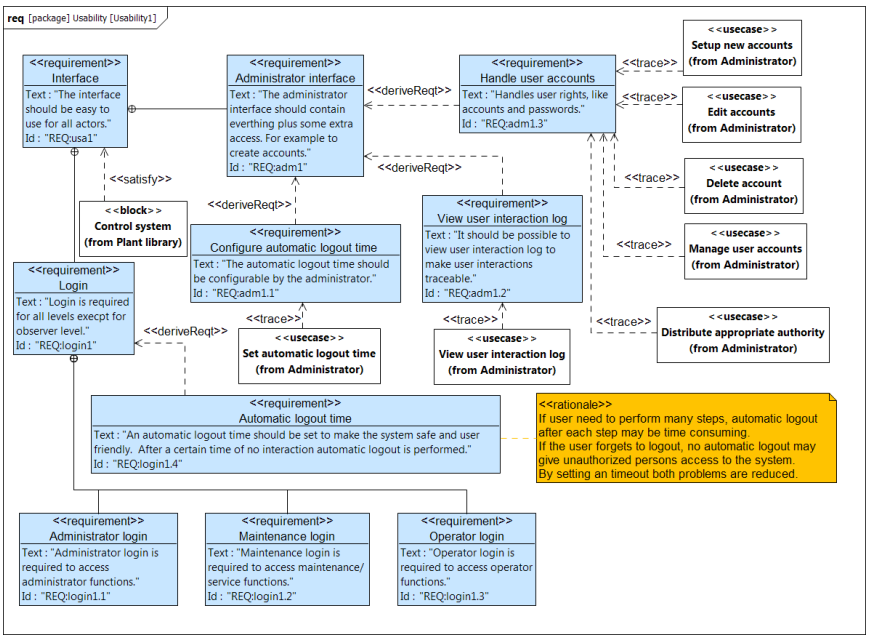

Figure 4.16: Updated version of Figure 4.2, where the reason for the particular requirements are depicted by the `trace` relationship. Also, the whole interface requirement is satisfied by the control system.

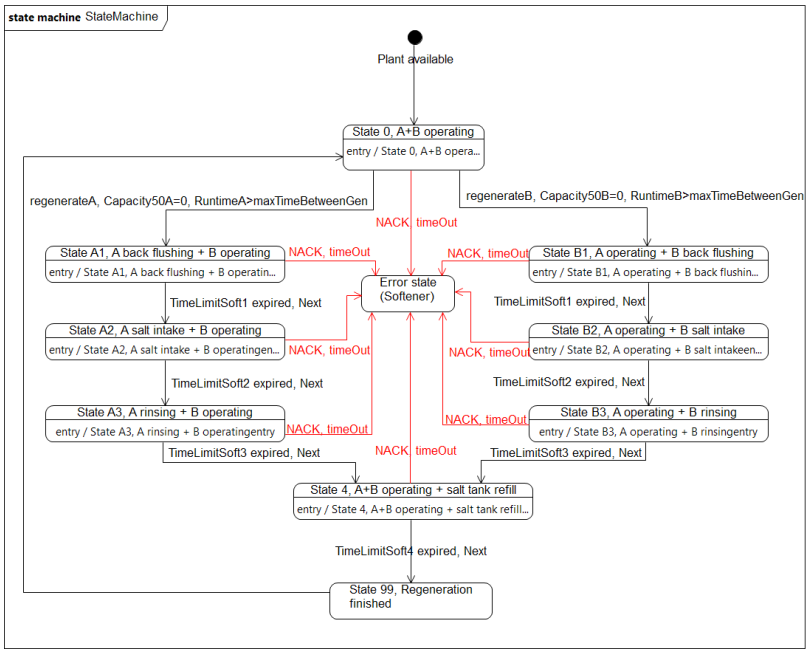

Figure 4.17: Initial state machine for the softener.

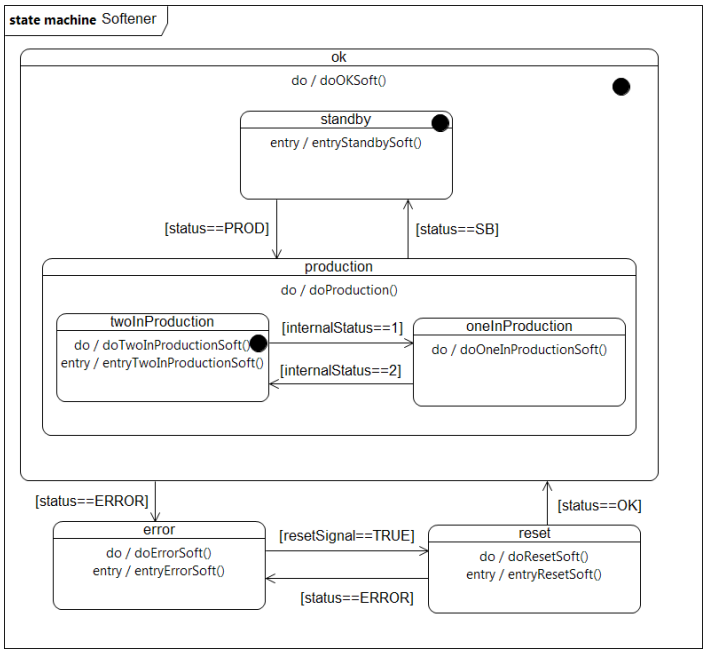

Figure 4.18: Final state machine for the softener.

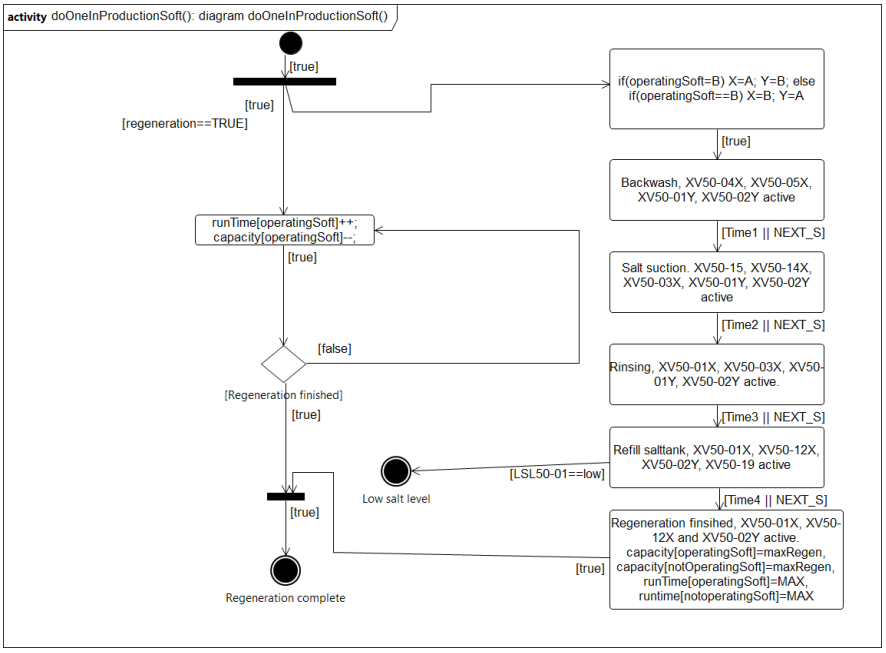

Figure 4.19: Activity diagram showing the regeneration sequence for the softening block.

5 Modeling and simulations

First a mathematical model for the RO part was derived. The model was then transfered to Matlab/Simulink and simulated and control strategies were developed. The state machines developed in the SysML model were also implemented in Matlab/Simulink using Stateflow.

5.1 Mathematical model of the reverse osmosis process

To be able to simulate and evaluate the process in Simulink, a mathematical model of the plant is needed. With this model, it is possible to test and evaluate different control strategies and to check for errors and inconsistencies in the developed state machines. The modeling approach is divided into two parts, one where a simple model for the membrane is derived, and one where balance equations for the RO-step are set up [18]. The balance equations are based on the concept of conservation of mass.

Also and important to notice is that to simplify the simulations, the incoming water is assumed to hold a constant salt concentration (g/ml). In the real water purification unit, the water not only holds unwanted dissolved salts, but also bacteria. Normally there is a correlation between the amount of bacteria present in the water and the conductivity in the water measured by the conductivity transmitters. Therefore, to simplify, the dissolved salt concentration is used instead of the bacteria as a measure of the overall water quality. Nonetheless, for control purposes this should have no significant impact.

5.1.1 Model for the membrane

Several models are proposed in the literature in the attempt to model the flow of fluid and salt through the membrane. The model used throughout this project is based on the so called Solution-Diffusion model (SD) [19]. It was proposed in the 1960's and, as the name indicates, is based on the idea that both fluid and dissolved solute diffuse through the membrane.

According to the SD-model, the flux through the membrane can be approximated as

$$J = \frac{\Delta P - \Delta\pi}{\mu(R_m + R_c)}. \tag{3}$$

Here ΔP is the average pressure drop over the membrane and $\Delta\pi$ is the osmotic pressure over the membrane. R_m is the membrane resistance and R_c is the resistance due to fouling and scaling and μ is the viscosity for the liquid at hand. Please see Figure 5.1 and Table 5.1 for an overview of the variables used.

Since flux is defined as the flow per membrane area, one can approximate the total permeate flow as

$$\begin{aligned} F_p = JA_m &= A_m \frac{(\Delta P - \Delta\pi)}{\mu(R_m + R_c)} \\ &= K_1(\Delta P - \Delta\pi), \end{aligned} \tag{4}$$

The term $(R_m + R_c)$ is the overall flow resistance and increases with time due to fouling. As a consequence, in order to keep a constant flow through the membrane, the applied pressure from the high pressure pump upstream from the membrane unit has to increase over time.

ΔP is the average pressure drop over the membrane and is calculated as

$$\Delta P = \frac{P_f + P_r}{2} - P_p, \tag{5}$$

where P_f, P_r and P_p is the pressure on the feed side, retentate side and permeate side respectively.

$\Delta\pi$ is the osmotic pressure over the membrane

$$\begin{aligned} \Delta\pi &= iRT(C_t - C_p) \\ &= K_2(C_t - C_p). \end{aligned} \tag{6}$$

where T is the temperature and R is the gas constant. i is the van't Hoff factor and is approximated to two. In our case we will also consider the temperature T as constant at 25 degrees Celsius or 298 Kelvin.

The salt flux through the membrane can be written as

$$C_p = K_3 C_t, \tag{7}$$

where K_3 is a constant describing how much salt that goes through the membrane. K_3 usually lies between 0.01 and 0.001. In our case, a membrane LE-4040 from Dow Filmtec™ is used, which according to datasheets has a salt rejection of 99 %, i.e K_3 can be assumed to be 0.01.

5.1.2 Modeling the RO-block

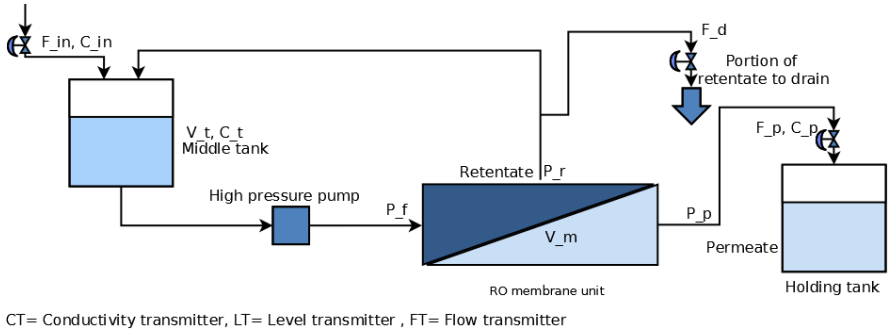

Figure 5.1: Schematic diagram of the reverse osmosis block.

In order to minimize the water going to waste, balance equations have to be set up in order to evaluate the system and controller performance. First the volume balance is set up, where the change in total volume, $\frac{dV_{tot}}{dt}$, is equal to the different flows coming in and out of the system

$$\frac{dV_{tot}}{dt} = F_{in} - F_p - F_d. \tag{8}$$

In order to minimize the water going to waste during circulation, a balance equation for the total salt amount in the system has to be set up

$$V_{tot}\frac{dC_t}{dt} = F_{in}C_{in} - F_pC_p - F_dC_t. \tag{9}$$

where F_{in} and C_{in} are design parameters. This can then be rearranged as

$$\frac{dC_t}{dt} = \frac{F_{in}C_{in} - F_pC_p - F_dC_t.}{V_{tot}}. \tag{10}$$

Table 5.1: Variables used in the reverse osmosis modeling

Parameter	Description
A_m	Total membrane area
C_{in}	Salt concentration for incoming water
C_p	Permeate concentration
C_t	Concentration in the tank
F_{in}	Incoming flow from Part 1
F_d	Retentate flow to drain
F_p	Permeate flow
i	Van 't Hoff factor
J	Membrane flux
K_1	Variable used in equation 4
K_2	Constant used in equation 6
K_3	Constant used in equation 7
P_f	Pressure on the membrane feed side
P_p	Pressure on the permeate side
P_r	Pressure on the retentate side
R	Gas constant
T	Temperature in Kelvin
R_m	Membrane resistance
R_c	Membrane resistance due to fouling
μ	Viscosity
V_{tot}	Estimated volume for the whole system
ΔP	Mean pressure difference over the membrane
$\Delta\pi$	Osmotic pressure difference over the membrane

5.2 Simulink simulation

From the simple model for the membrane and the balance equations described above in Section 5.1, a dynamic model for the system is setup in Simulink, see Figure 5.2. By connecting a simple control system to these equations, it is possible to evaluate different approaches for controlling the plant. The final objective is to control a total of four variables in Part 2, see Figure 5.3. The objectives for the four control loops are listed below.

- Water tank levels. The water levels in the tanks, i.e. the middle tank and the holding tank, should be controlled thereby guaranteeing that there always is enough water in the system.
- Permeate flow. The permeate flow should be kept constant over time. This is done by controlling the high pressure pump in front of the RO-membranes. In the already existing control system, a pressure transmitter in front of the membranes is used to ensure that the pump output gives rise to a high enough pressure right in front of the membranes. Due to fouling, the term R_c in equation 4 will increase. In order to be able to keep the permeate flow constant it is not enough to just keep the input pressure constant, one also has to compensate for this increasing resistance by increasing the pressure from the pump. In order to do that, the permeate flow has to be measured, either directly with a flow indicator, or indirectly with pressure transmitters.
- Retentate flow to drain. The fourth and last control loop is the conductivity feedback from the permeate side to the retentate drain. This is done in order to be able to control the amount of retentate let to drain.

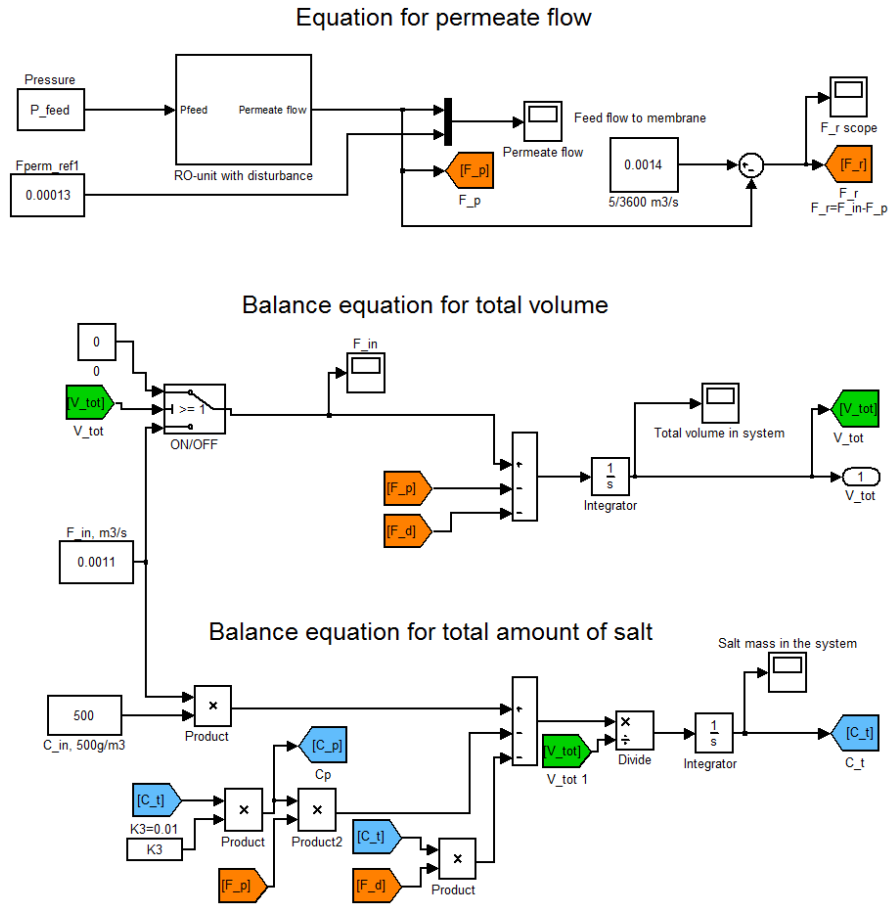

Figure 5.2: Balance equations setup in Simulink to evaluate the different controller approaches. At the top one can see the membrane equation. In the middle the balance equation for the water in the system is shown and at the bottom the equation for the total salt concentration is shown.

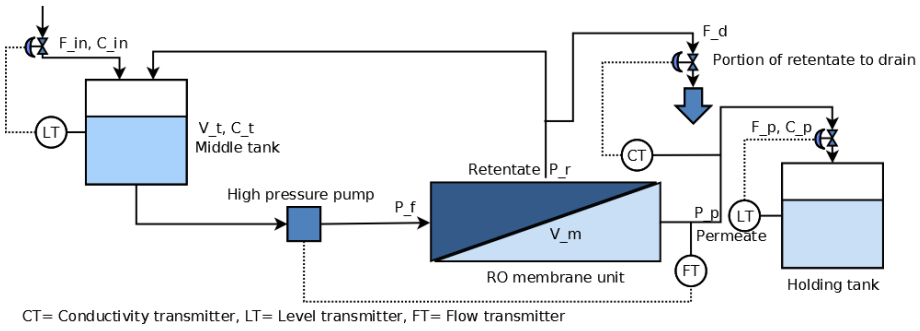

Figure 5.3: Figure showing the RO-unit and the four control loops involved.

5.2.1 Control approach

The simplest possible controller is the on/off controller, where one simply just switch between two modes depending on the control error. The valves controlling the flow into the two tanks are binary valves, and this gives no other option

but to use a on/off control approach for controlling the level in the tanks. The water level could be controlled in a PWM manner, but this would rapidly wear the valves out. Instead, the water levels will be allowed to fluctuate between two predetermined levels, say $level_{min}$ and $level_{max}$.

As mentioned above in Section 5.2, the fouling of the membrane requires the high pressure pump to increase its output pressure with time. As this fouling and scaling process is a very slow process, the first approach taken is to evaluate the use of a simple PID-controller for ramping up the reference pressure. The controller is implemented as a simple Simulink block with appropriate inputs and outputs, see Figure 5.4. A thing to keep in mind is that the membranes have a maximum allowed pressure limit. If exceeding this pressure, there is a risk of pushing the membranes out of their containers, an issue called *telescoping* in membrane terminology [20]. When the pressure exceeds this limit, the plant should automatically shut down in wait for the membranes to either be cleaned or replaced. To prevent this a saturation block is added to the control signal output.

In the simulation the permeate flow is directly measured with a flow transmitter which is fed back to the control system. In the simulation scenario, the total membrane resistance is assumed to increase with 50 % over the course of one month. Figure 5.5 shows the behavior of the flow both without and with the flow feedback. Obviously, as seen in the topmost plot, without the flow feedback, the pressure from the pump will remain constant and the flow will decrease over time. The bottom figure shows the same flow but with the controlled pressure. Not surprisingly, a PID-controller seems to cope well with the slow dynamics of the process.

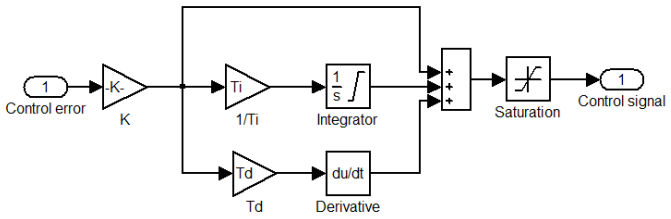

Figure 5.4: Simulink representation of the PID-controller.

Out of the four control objectives, the most complicated is the conductivity feedback from the permeate side. Depending on the size of the tank and the overall water amount left in the piping, the time constant for the process can be quite big, giving rise to a slow step response. On the other hand, the water conductivity will most certainly not make any sudden changes, and hopefully this time constant will not give rise to any serious problems. Another PID-controller, with the same design outline as in the pressure feedback control loop shown in Figure 5.4, is used as controller.

In the simulations, the output signal from the controller is a ratio expressed in percentage, determining how much of the water that should be fed back to the holding tank, and how much should be allowed to go to drain. In this PID, there are some constraints that have to be included. One is that the control signal can not be allowed to be greater than 1, i.e 100 %, since one can not let more water to drain than physically possible. Actually, the portion let to drain would normally not exceed 10 % of the retentate flow. Another important thing is to add an anti-windup constraint to the I-part in the controller. The slow dynamics of the system together with the constraint on the control signal could otherwise lead to problems during big reference changes. The anti windup

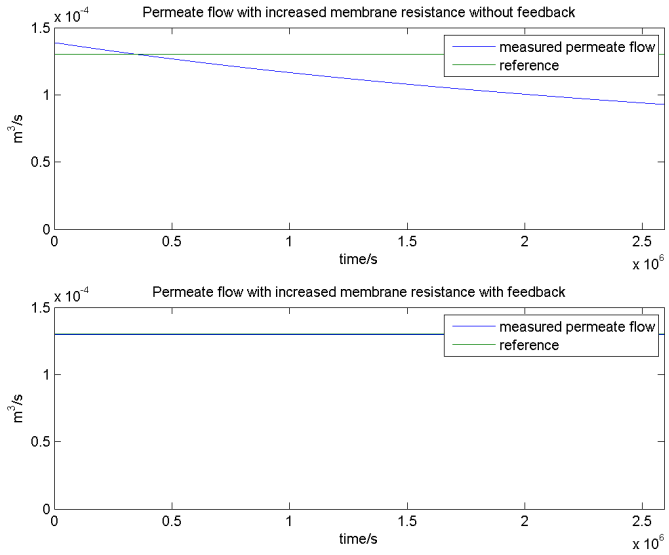

Figure 5.5: Difference between the uncontrolled and controlled permeate flow. Upper figure shows the decline of permeate flow with a membrane resistance increasing with 50 % over a course of one month. The bottom figure shows the same scenario but with the PID plugged in. The reference flow is set to 0.00013 m^3/s or approximately 500 l/h

is simply implemented as a upper saturation limit in the integrator block of the controller. In Figure 5.6 three negative step responses are shown for the RO-process. During theses simulations an input water with a concentration of $0.5kg/m^3$ is used, which corresponds to the salt concentration of drinking water. From the figure, one see that the permeate concentration slowly drops towards the new reference. Due to the constraints on the drain flow, the concentration drops fairly slow towards the new value. Tuning the PID-controller will not improve the performance as the control signal is saturated during the decline. The bottommost figure shows the response when the reference changes to an unreachable value of $4g/m^3$. As the membrane has a salt rejection of 99 %, one can only expect to reach a permeate concentration of $5g/m^3$.

One thing to keep in mind is that the simulations done is just a way to test different controller approaches and validate their performance. The control parameters used in the simulations can not be used on the real process due to the many assumptions done. Also, as the real process comes in many different sizes, the dynamics of the system will vary from model to model. This is shown in Figure 5.7. By increasing the total water volume in the system, the response to a change in reference concentration takes longer. The corresponding flow ratios for the step responses in Figure 5.7 is shown in Figure 5.8. In the figure, the drain flow makes a step to zero as the salt concentration reference changes at t=500. By not letting any salt go to drain on the retentate side, the concentration quickly increases in the system. When the salt concentration reaches the desired reference, the drain flow stabilizes around 30 %, i.e. 30 % of the retentate is let to drain, while the rest is fed back to the buffer tank. To find controller parameters for a real plant, methods such as the Ziegler-Nichols method could be used.

To conclude the control approach section, PID-controllers seem to be able

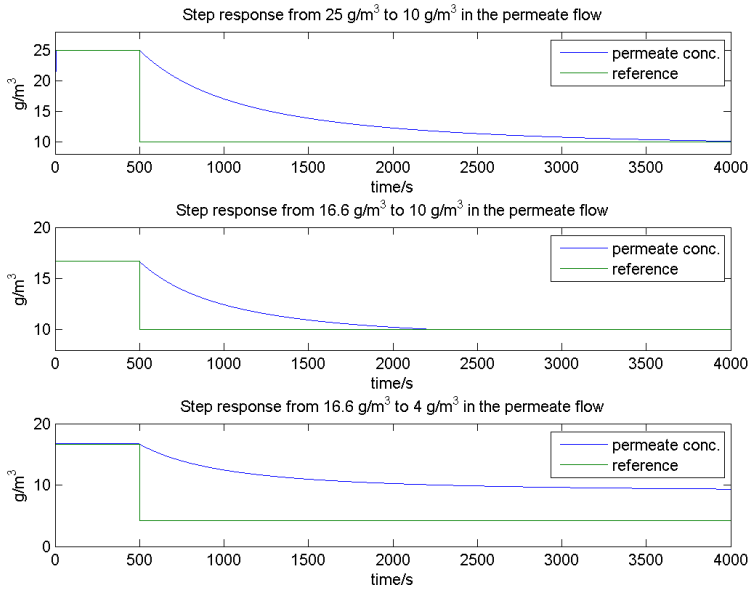

Figure 5.6: Figure showing three inverted step responses when changing the permeate concentration reference in the RO-process. The simulations are done with an input water with a salt concentration of $0.5kg/m^3$, corresponding to normal drinking water. Due to the constraints on the drain flow, the concentration drops slowly towards the new reference. In the bottom figure the reference takes a step to a reference not possible to reach.

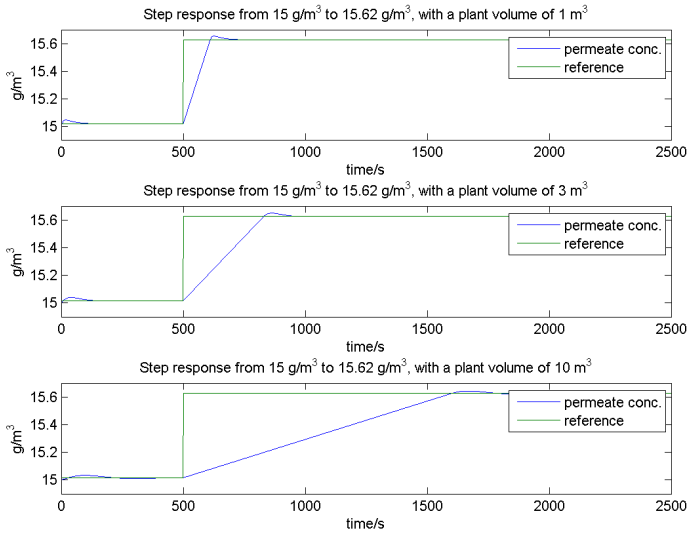

Figure 5.7: By increasing the total water volume in the system, one gets a slower response to a concentration reference change.

to handle both the upramping of pressure over the membrane as well as the conductivity feedback.

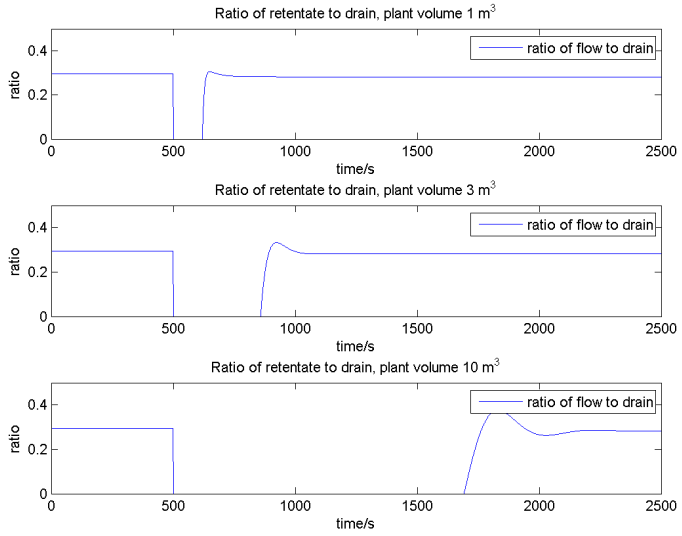

Figure 5.8: Ratio of flows ($\frac{flow_{drain}}{flow_{retentate}}$) for the step responses shown in Figure 5.7. As the salt reference change at t=500, the drain valve closes, and all retentate is fed back to the buffertank. As the salt concentration reaches the reference, the flow to drain stabilizes around 30 % of the total retentate flow.

5.2.2 State machine

In this section simulations of the state machines that were modeled in Section 4.7 are performed. In order to simulate the dynamic behavior, the Matlab/Simulink extension Stateflow is used. Later in Section 6 those state machines are implemented in C code.

The syntax to model a state machine in Stateflow is similar to what was presented in Section 4.7, however, some small changes exist and in order to run the state machines it is important that the syntax is correct. As described earlier, different actions types take place in a state. It could for example be an entry action or a exit action. In Stateflow, those keywords are the same but the do-action is instead called *during*. Also to perform the transitions correctly, the syntax of the transition action type is important. Stateflow uses the following syntax: $event\_trigger[condition]\{condition\_action\}\backslash transition\_action$. The event_trigger is the event that triggers the transition. Additionally, the event_trigger may have a condition and/or a condition action. If an event is triggered and the condition is fulfilled the condition action is executed. Finally the transition can also have a transition action that is executed when the transition occurs. Instead of a black dot in the initial state, Stateflow uses a black dot with an arrow pointing at the initial state. The history junction is represented like described in Section 4.7, but the ordinary junction symbol is instead of the diamond represented as a circle. In Stateflow, Matlab function blocks can also be added where ordinary Matlab code can be implemented. As mentioned before this is just the basic of state machines and Stateflow has a lot more to offer. For example states can be executed simultaneously and then the circumference on the rectangle shaped state is dotted instead of a solid line. However, the presented information is enough in order to grasp the state machines in this thesis.

The state machine of Part 1 is once more presented, in Figure 5.9, but now

the implementation from Stateflow is shown. Unfortunately transitions could not handle enumerators so instead of the state names a number is used. The following table gives the translation between numbers and state names used in the state machine of Part 1 :

Number	**Abbreviated state name**	**Formal state name**
0	SB	Standby
1	PROD	Production
2	CIRC	Circulation
3	ERROR	Error
4	CLEAN	Cleaning
5	SAN	Hot water sanitization
-2	ERROR	Error (Only used in Matlab)
-3	OK	Ok

Table 5.2: Table for mapping the status number used in Matlab to the enumerator used in SysML.

The complete translation table is found in Appendix B together with all state machines implemented in Matlab. One particular thing to notice in the Stateflow diagrams is the additional status signal, -2, which is used to return to the error state from the reset state. This is needed since the Stateflow implementation does not store events in queues but instead continuously reads the status ports. Another status signal used but not mentioned in the table is -1. This signal is used to synchronize the transitions from the reset state. Before the -3 signal is sent, all of the children have to send -1 to its parent indicating that no error exists in them. When the plant receives -1 from both Part 1 and Part 2, a -3 signal is sent back down, stating that it is ok to return to production. Likewise, Part 1 sending -1 means that it received -1 from all of its children. This reset function is also implemented on the PLC and will be further discussed in section 6.4 where a sequence diagram diagram of the reset synchronization procedure is presented.

The different StateFlow state machines are then encapsulated in a Simulink state chart block which in turn is connected to its children's state chart blocks. Part 1 and its children can be seen in Figure 5.10. On a higher level this block is connected to the state machine of the plant. Those connections, including the connection to Part 2 is also presented in Appendix B.

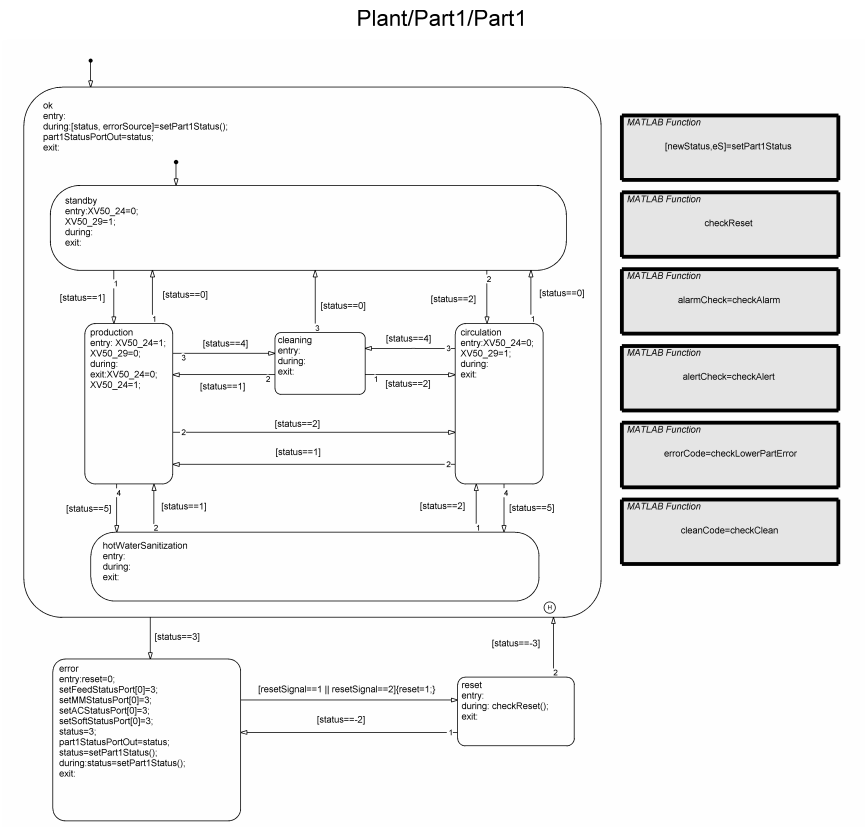

Figure 5.9: Part1 state machine implemented in Stateflow.

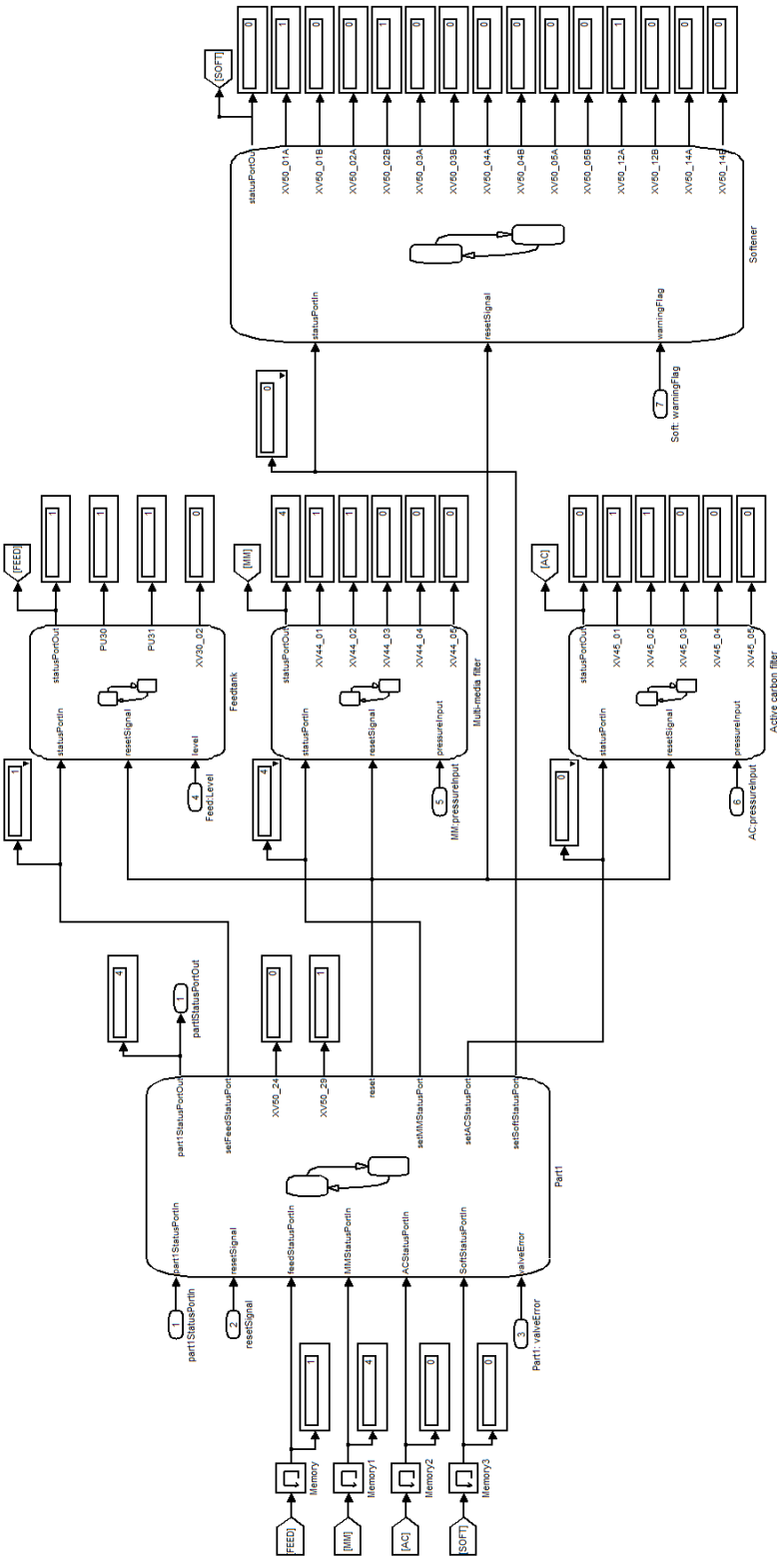

Figure 5.10: Simulink model of Part1 and its children, the feed tank, the multimedia filter, the active carbon filter and the softeners.

6 Implementation

This section covers the implementation on the PLC. First, the system choice is briefly discussed and some general information about the functionality of the PLC. Later the structure of the program is described. Both the state machines and the mathematical model simulated in Simulink are transfered to C code and implemented on the PLC.

6.1 System choice

From the requirements diagrams in the SysML model, one can see a couple of requirements concerning the choice of control system. Some of them are listed below:

- Number of I/Os. The system should be able to handle the required number of input and output ports. For the RO-plant, this varies from model to model, but a minimum of 60 inputs and 60 outputs are required.
- Communication protocols. The communication with actuators should be done via a fieldbus. Also it should support TCP/IP to support remote reading of data from the HMI.
- Modularity. It should be easy to add and remove modules from the system.
- Programming language. The system should support the programming languages defined by IEC 61131-3, but also C.

Although there may be several systems that fulfill these requirements, the control system choice fell on the X20CP1485-1 system from B&R. The X20 supports a total of 250 I/O modules or 3000 channels. Further it supports Ethernet TCP/IP and has a 64 MB DRAM and 1 MB SRAM internal memory [21] . All of this is well enough for the needs of the RO-plant. Also the PLCs from B&R come with their own development environment called B&R Automation Studio. Automation Studio supports development in the languages stated by IEC 61131-3 and C but also C++ with a special package. After the choice of system is done, the SysML model is updated with the appropriate relationship, see Figure 6.1.

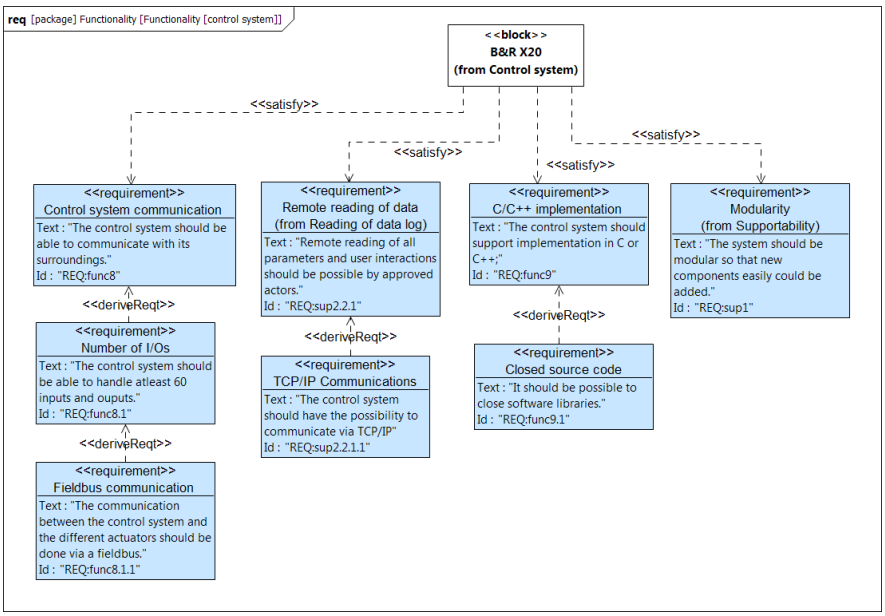

Figure 6.1: Requirement diagram concerning the control system, with the B&R X20 system added to underline that it satisfies the requirements.

6.2 PLC

A programmable logic controller, or PLC, is a low cost computer often used in automation industry. The development of the PLC started in the 60's from the desire to replace old analog controllers with digital ones [22]. It consist of the main logic controller and a number of attached inputs and outputs, as well as some communication interface, see Figure 6.2.

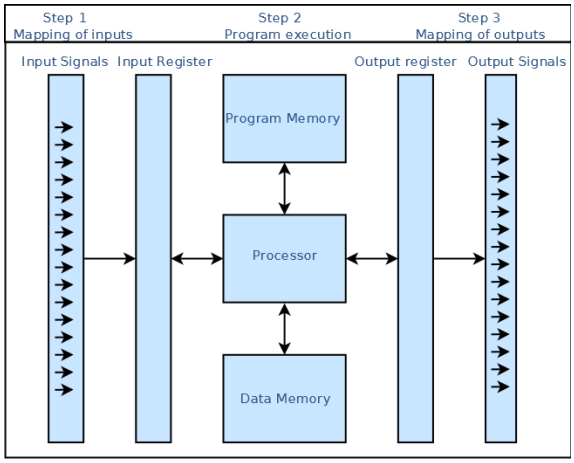

Figure 6.2: The PLC execution cycle consists of three steps. First, the mapping of input signals, followed by the program execution and the mapping to outputs.

When switched on, the PLC executes its programs in a cyclic manner, with each program executed exactly once each cycle [23]. During the start of each cycle, the input ports are scanned and values are transfered to an I/O image table, where every input value is mapped to a software variable in the PLC.

After the scanning, the programs are executed in a predetermined order, some more often than others depending on their priority. When every program has executed, any changed variable is mapped to their corresponding output port. Figure 6.3 shows a priority table from Automation Studio where two tasks, LogActivity and RO_sim are assigned to execute every 50 ms and PLANTSM which is assigned to execute every 200 ms. Figure 6.4 shows how the tasks will execute during runtime. If LogActivity and RO_sim finish their cyclic tasks in 10 ms and 5 ms respectively, there will be 35 ms of idle time during each 50 ms cycle, leaving room for PLANTSM to execute. If any task fails to finish during its assigned cyclic time, a cycle time violation error will be thrown and the system will enter service mode if not handled.

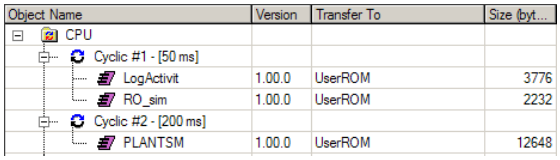

Figure 6.3: Different cyclic tasks can be assign to execute more or less often, depending on their priority. The figure shows the priority assignment table from B&R Automation Studio.

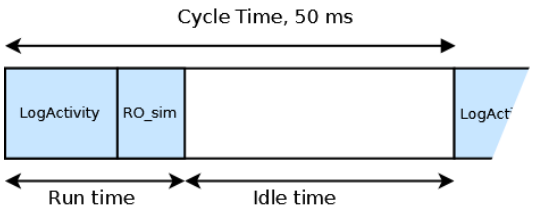

Figure 6.4: Execution order for the 50 ms cyclic tasks LogActivity and RO_sim. Any idle time during the 50 ms cycle can be used to execute the PLANTSM task.

6.3 Libraries

To keep the modular approach, defined as a requirement in the SysML model, the programming task is divided into several libraries, one for each of the physical modules in the plant. Starting out from the simplest components in the plant, i.e. valves and sensors, more and more advanced and complex functions are created. The valve library will for example contain functions to open and close a specific valve, check if the valve is open/closed etc, see below.

```
...
plcbit valveClose(struct Valve *pValve){
  if(pValve==0){
  return FALSE; //nullpointer
  }
  pValve->output=FALSE; //Close valve
  return TRUE;
}
...
```

The feed tank library in its turn, see below, will contain functions to set the tank in its different operation states, using the functions previously defined

in the valve library. The feed tank library also contains different functions for checking possible alarms. Additionally, get and set operations for specific sensor readings and alarm levels are implemented.

```
...
plcbit setStandbyModeFeed(FeedTankBlock *pFeedBlock){
  if(pFeedBlock==0){
    return FALSE; //nullpointer
  }
  setOperationStateFeed(pFeedBlock, STANDBY_S);
  valveClose(&pFeedBlock->feedTank.inputValve); //CLOSE XV30-02
  pumpStop(&pFeedBlock->PU30_01); //STOP PU30-01
  pumpStop(&pFeedBlock->PU30_02); //STOP PU30-02
  return TRUE; //Function OK
}
...
```

6.4 Hierarchal state machine

The most common way to implement a state machine is to use a flat state machine, where all the states are on the same level and with no inner states. This is easily implemented in code using a switch case statement. An example consisting of three states can be seen in the code below and the arrangement of states is depicted in Figure 6.5.

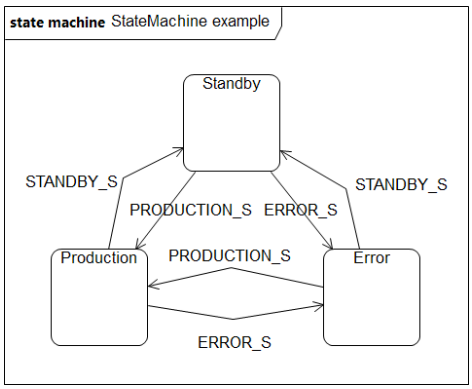

Figure 6.5: Example state machine described in code below.

```
  State myState;

void myStateHandler(unsigned const signal){
  switch(myState){ //Switch on the current state
  case STANDBY:
    switch(signal){ //Switch on incoming signal
    case ERROR_S:
      myState=ERROR; //Change state
      break;

    case PRODUCTION_S:
      myState=PRODUCTION; //Change state
      break;
    }
    break;

  case PRODUCTION:
    switch(signal){ //Switch on incoming signal
    case ERROR_S:
```

```
        myState=ERROR; //Change state
        break;

      case STANDBY_S:
        myState=STANDBY; //Change state
        break;
      }
      break;

    case ERROR:
      switch(signal){ //Switch on incoming signal
      case STANDBY_S:
        myState=STANDBY; //Change state
        break;

      case PRODUCTION_S:
        myState=PRODUCTION; //Change state
        break;
      }
      break;
    }
}
```

This very simple way of implementing a state machine has both its pros and cons. The advantages are that it is simple, one do not have to be a professional programmer to understand what it does and its memory footprint is small as one only needs an enumerator to represent each state. The disadvantages are that the construction does not have any built in support for either entry or exit conditions nor for nested states. Manually coding the entry/exit behavior will soon lead to a hard to read code. Entry behavior to a state will for example be spread out in every state that can make a transition to the state in question [24].

Instead of flat state machines the idea of hierarchical state machines, which consists of nested states, was used. This reduces the transitions since the parent's transitions only have to be declared once and not in all of the inner states. The hierarchical state machine implementation in this thesis is based on the approach by Miro Samek, and a framework called Quantum Leaps [25]. Quantum Leaps uses, in short, instead of long switch/case statements, functions and function pointers to organize the state machines. This framework also contains built in support for entry and exit conditions. The principle is perhaps best illustrated with a code example, with the same three states as before.

```
...

QState STANDBY(feedTank_S *me, QEvent const *e) {
  switch (e->signal) { //Switch on incoming signals
    case Q_ENTRY_SIG: { //Do entry action
    return Q_HANDLED();
          }
    case Q_EXIT_SIG: { //Do exit action
    return Q_HANDLED();
          }
    case PRODUCTION_S: {
    return Q_TRAN(&PRODUCTION); //Make transition to PRODUCTION
          }
    case ERROR_S: {
    return Q_TRAN(&ERROR); //Make transition to ERROR
          }
    case DURING_S: {
      doStandbyStuff();//Perform during action
    return Q_SUPER(&QHsm_top); //Pass event to parent state machine
          }
      }
return Q_SUPER(&QHsm_top); // No match, pass event to pseudo SM.
}
```

```
QState PRODUCTION(feedTank_S *me, QEvent const *e) {
  switch (e->signal) { //Switch on incoming signals
    case Q_ENTRY_SIG: { //Do entry action
    return Q_HANDLED();
    }
    case Q_EXIT_SIG: { //Do exit action
    return Q_HANDLED();
    }
    case STANDBY_S: { //Make transition to STANDBY
    return Q_TRAN(&STANDBY);
    }
    case ERROR_S: { //Make transition to ERROR
    return Q_TRAN(&ERROR);
    }
    case DURING_S: {
      doProductionStuff(); //Perform during action
    return Q_SUPER(&QHsm_top); //Pass  event to parent state machine
    }
  }
return Q_SUPER(&QHsm_top); // No match, pass event to pseudo SM.
}

QState ERROR(feedTank_S *me, QEvent const *e) {
  switch (e->signal) { //Switch on incoming signals
    case Q_ENTRY_SIG: {// Do entry action
    return Q_HANDLED();
    }
    case Q_EXIT_SIG: {// Do exit action
    return Q_HANDLED();
    }
    case STANDBY_S: { //Make transition to STANDBY
    return Q_TRAN(&STANDBY);
    }
    case PRODUCTION_S: { //Make transition to PRODUCTION
    return Q_TRAN(&PRODUCTION);
    }
    case DURING_S: {
      doErrorStuff(); //Perform during action
    return Q_SUPER(&QHsm_top);//Pass event to parent state machine
    }
  }
return Q_SUPER(&QHsm_top); // No match, pass event to pseudo SM.
}
...
```

The code might be hard to understand at first, but once you understand the mechanism behind it, the construction of the state machine is simplified a lot. In the framework, there are three reserved events, `Q_ENTRY_SIG, Q_EXIT_SIG` and `Q_INIT_SIG`. The first two signals are generated by the framework itself, once a state is either entered or exited. The init-event is used to specify where in a hierarchy of states that the state machine should start. In the example there are no nested state, so this signal is not used. Apart from these three enumerators, the developer is free to define his or hers own signals. In the example, the user-defined signals are `STANDBY_S`, `PRODUCTION_S` and `ERROR_S`. Once a signal is generated somewhere in the system, it is sent to a dispatch functions that traverses the state machine and checks if the current innermost active state has any valid transition to pair with the signal. If it does, the transition takes place and the corresponding exit and entry actions are performed and the event is said to be handled. If there is no valid transition, the signal is passed to the closest parent of the current state, as the parent might have a valid transition in response to the event. If no state in the current active state hierarchy handles the event, it is passed to a pseudo state, `QHsm_top` which is the topmost state of the state machine, where the event is automatically handled without any action.

The Quantum-Leap framework has, however, no built in support for a during-behavior, and to solve this, a `DURING_S` signal was created. This signal is generated when no other user-defined event takes place. The `DURING_S` signal is sent to the innermost active state and then passed upwards in the state hierarchy as every active state should perform the during action. The deep history junction is implemented by introducing a state variable and saving the active state every time a nested state is entered. When reseting from an error, the state machine will then return to the state referenced by the state variable.

As mentioned before the PLC runs in a cyclic manner, performing one task after another. There exists one task for each state machine, but also one cyclic task that handles the graphical user interface (GUI). The execution order is listed below:

1. GUI
2. Plant
3. Part 1
4. Part 2
5. Feed tank
6. Multimedia filter
7. Active carbon filter
8. Softening block

Since the GUI only communicates with the Plant state machine, a signal from the GUI is handled during the same cycle as it was sent. An error signal from the feed tank, however, takes one cycle to reach Part 1 and another one to reach the Plant. Since the cycle times are quite short, the delay from child to parent is not a problem. During development the time for each cycle has been 1 second but this is a design parameter that can be changed. The chosen time was appropriate during real-time debugging.

To handle the sending of event in the state machine, a number of event-queues are used, one for every individual state machine. The queues are needed in order to be able to handle numerous events being sent during the same execution cycle. This may only happen on very rare occasions on the real process as the cycle time will be quite small, but it does still need to be handled.

During each execution cycle, every state machine starts by checking for any errors within itself. If an error is detected, an alarm is generated and an event `ERROR_S` is generated. This event sets the whole plant in a fail safe state. However, if no error is detected, the program continuous and checks for any incoming events in its corresponding event queue. If any events are present in the queue, they are sent to an event handler that provides the logic for the communication between the state machines. The procedure is shown in a sequence diagram in Figure 6.6. In the figure, if the feed tank detects an error, it puts itself in a fail safe state and sends an `ERROR_S` event to the Part 1 state machine, which in turn, puts itself in a safe mode while passing the message upwards to the Plant and down to its other children.

Reseting the plant from an error requires some synchronization between the different state machines. When the operator sends a reset signal from the GUI, the signal, `RESET_S`, is sent to the topmost state machine, which in turn distributes the signal. Figure 6.7 shows a SysML sequence diagram, describing the synchronization process. The execution order of the individual state machines determines the number of cycles it takes for the whole plant to reset. It takes

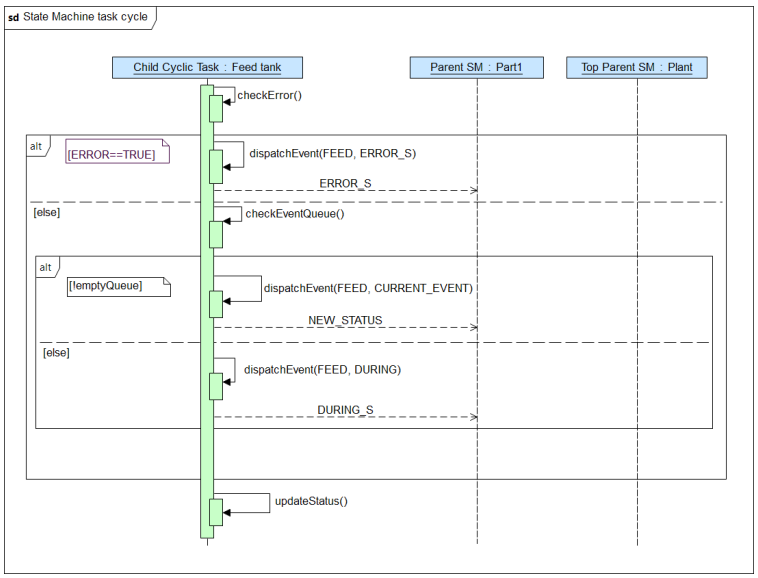

Figure 6.6: Sequence diagram illustrating the procedure during one task cycle. First, the state machine checks for errors. If any, an `ERROR_S` event is dispatched, putting the state machine in a fail safe mode. The `ERROR_S` event is then passed on to the parent state machine. If no errors are detected. the event queue is checked for incoming events. If any, the appropriate action is taken by an event handler. Last the status of the state machine is updated. When the Part1 task executes, it will handle the event from the Feed tank and pass it along to the Plant.

one cycle to tell all state machines to make a transition to their reset states. Once there, the individual parts will send a reset request to their parent. The parent will in turn ask the Plant if it is in order for the sub state machines to return to operation. If so, an `OK_S` signal is dispatched and operation is resumed. All of the signals used in the state machines can be seen in Appendix C.

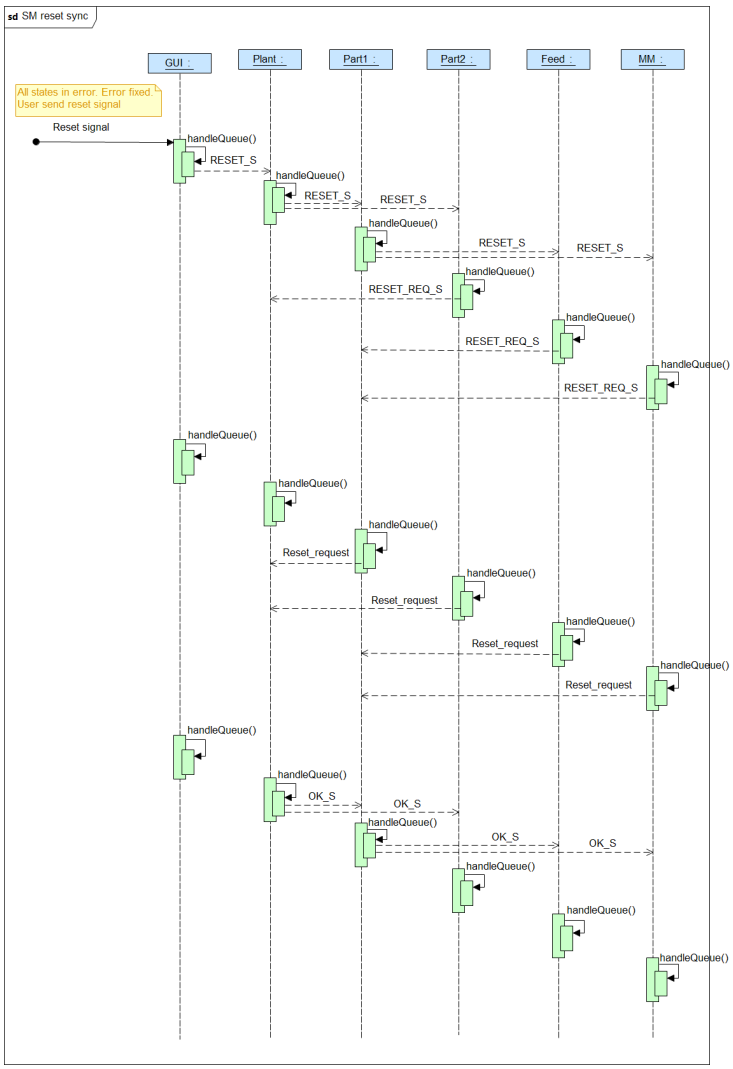

Figure 6.7: SysML sequence diagram illustrating the reset synchronization procedure when reseting from an error.

To show and debug the execution of the state machine, logging functions for the state machines were implemented. At the start of every cycle, the main logger writes the current cycle number, as well as the current status for every individual state machine to a file. Also individual loggers are created for each of the state machines. In these loggers more information about the transitions are logged. However, to get a better overview, only comments from those logs are posted in the example of the main logger below. The example shows the state of events that take place when the multi-media filter wants to do a backwash.

```
Cycle # State Machine Current State Comment
Cycle 48:
        Plant:          PROD
        Part1:          PROD
        Part2:          PROD
        Feed:           FILL
        MM:             PROD        //Multimedia filter wants to clean.
                                    //Sends CLEANING_S to Part 1
        AC:             PROD
        Soft:           PROD
Cycle 49:
        Plant:          PROD
        Part1:          CLEAN       //Enters clean mode. Sends CLEANING_S
                                    //back to MM, and STANDBY_S to AC
                                    // and Soft. Sends CIRCULATION_S to Plant
        Part2:          PROD
        Feed:           NOFILL
        MM:             CLEAN       // Multimedia enters cleaning mode
        AC:             SB          // Enters standby mode
        Soft:           SB          // Enters standby mode
Cycle 50:
        Plant:          CIRC        // Enters circulation mode
        Part1:          CLEAN
        Part2:          PROD
        Feed:           NOFILL
        MM:             CLEAN
        AC:             SB
        Soft:           SB
......
......
......
Cycle 56:
        Plant:          CIRC
        Part1:          CLEAN
        Part2:          PROD
        Feed:           NOFILL
        MM:             CLEAN
        AC:             SB
        Soft:           SB
Cycle 57:
        Plant:          CIRC
        Part1:          CLEAN
        Part2:          PROD
        Feed:           NOFILL
        MM:             PROD        //Done cleaning. Sends DONE_CLEANING_S
                                    //to Part 1
```

```
        AC:         SB
        Soft:       SB
Cycle 58:
        Plant:      CIRC
        Part1:      PROD        // Returns to production
                                //sends DONE_CLEANING_S to Plant
                                //and PROD_S to AC and SOFT
        Part2:      PROD
        Feed:       NOFILL
        MM:         PROD
        AC:         PROD
        Soft:       PROD
Cycle 59:
        Plant:      PROD        // Returns to production
        Part1:      PROD
        Part2:      PROD
        Feed:       NOFILL
        MM:         PROD
        AC:         PROD
        Soft:       PROD
```

From the logger output, one can see that the plant is running normally during execution cycle 48, but that the Multimedia filter wants to clean, and sends a `CLEAN_REQUEST_S` clean-request to Part 1. At cycle 49, Part 1 responds to the clean-request and sends `CLEANING_S` back to the Multimedia filter and puts itself in its clean state. It also sends `STANDBY_S` to the active carbon filter and the softening block. When the multimedia filter executes, it checks its incoming event queue, sees that Part 1 has accepted the clean-request and the cleaning of the filter can start. The active carbon filter and the softening block go to their standby states from the event passed down from Part 1. Due to the task order between the state machines, the Plant enters its circulation state during cycle 50. The cleaning procedure takes place during cycle 49-56, a very quick clean of course, but the overall functionality is still the same as if the cleaning would have taken an hour or so. When the multimedia filter returns to its production state during cycle 57, it takes one cycle for Part 1 to return to production due to the execution order. When Part 1 returns to production during cycle 58, it also sends a `DONE_CLEANING_S` signal to the Plant, which in turn, returns to production during cycle 59.

Since the implementation is not run on a real process some simple functions simulating the plant had to be implemented in order to execute the state machines. For example functions to get a measurement value from a sensor or feed back from a valve about its current position.

6.5 Controller

Since there is no real plant to verify the PLC implementation on, the membrane and balance equations derived in Section 5 will be implemented as a simulation on the PLC itself, in order to have something to run the state machines against. For this C implementation to coincide with the simulations done in Matlab/Simulink, the cyclic RO-simulation task has to be run fast enough to ensure that the small error done when approximating a continuous integration with a Riemann sum will be small. The integration function used ships with Automation Studio, and recognizes the scan time between the function invocations. Figure 6.8 shows two plots where the simulation done on the PLC is

compared to the Matlab/Simulink simulation. As one can see, they agree quite well with each other, concluding that the C code can be used as a simulation. The reason for the small discrepancy in the topmost plot is because the collection of data from the PLC only starts after a few samples, thereby causing a slight dislocation of the plot.

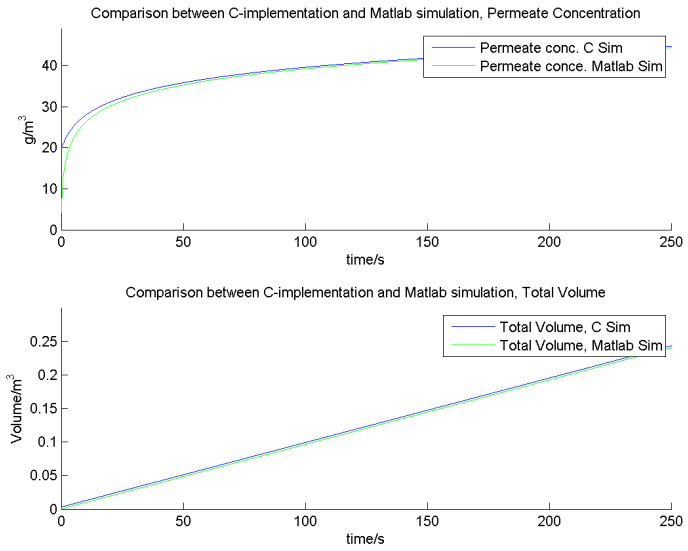

Figure 6.8: Comparison between the Matlab implementation and the simulation of the process implemented in C on the PLC.

Once the RO-simulation is concluded to work well, a PID controller is implemented to control the drain flow on the retentate side of the membrane. Opposite to the simulations in Matlab, simulations using the PLC are done in real-time, making it a quite time consuming process to debug. The PID is implemented using a control library in Automation Studio. The PID supports quite advanced features such as feed forward and limits on the upramping of the control signal, but none of these features will be used. Results from the simulations are shown in Figure 6.9, using the same controller parameters as in the Matlab simulations. The figure illustrates the system at startup, with incoming water filling up the system. The reference for the permeate concentration is initially set to 40 g/m^3. At around 250 s the reference is changed to 20 g/m^3. Disregarding the slight undershot after the reference change, the permeate concentration follows the reference quite well. Also, as the drain flow increases during the step, the total volume in the system decreases for a short period of time. In the figure, there are no constraint on the maximum flow to drain. When introducing a limit on the maximum drain flow, the step response is much slower, as seen in Figure 6.10

The figures conclude that the PID-controllers implemented on the PLC work in the desired way. Still, and as mentioned in Section 5, one can not be sure that the dynamics of the mathematical model are in line with the real process, nor that the controller parameters used will work. This section and Section 6.4 conclude that the implemented state machines, and the overall control system, work in a satisfactory fashion, which was one of the two main objectives of this project.

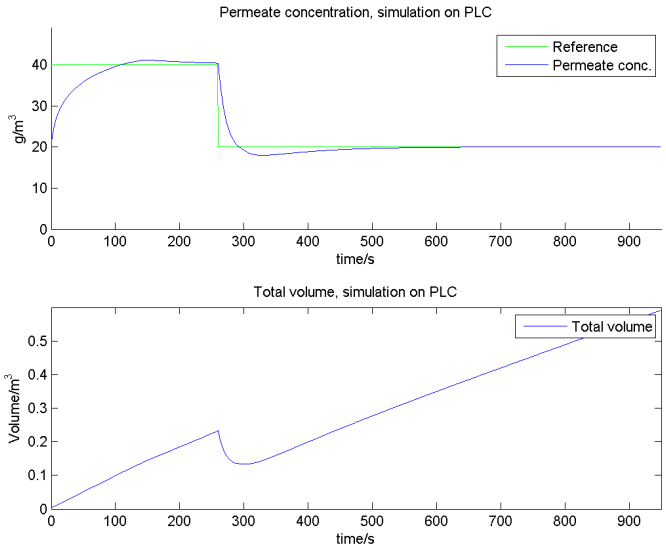

Figure 6.9: Results from the PID controller on the PLC. The permeate concentration follows the reference quite well. When the reference is changed, the total amount of water decreases in the plant, since more water will be let to drain. These simulations had no upper limit on the control signal.

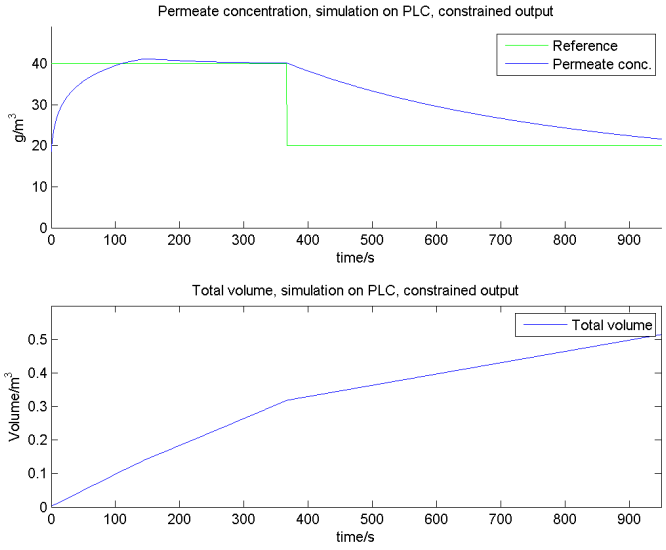

Figure 6.10: Results from the PID controller on the PLC, when there is a maximum allowed flow to drain.

7 Conclusion and further work

The two main objectives of this thesis were to first create a thorough documentation of the reverse osmosis water purification plant using SysML and then use this documentation to create a control system for the process. Included in the SysML model are a set of requirements stated by different stakeholders that the control system should meet. Apart from the requirements, behavioral diagrams such as state machine diagrams and sequence diagrams are created to document and model different aspects of the control system. The state machine diagrams are used to setup a simulation environment using Matlab/Simulink to simulate the different events that can occur. From these simulations, the state machines are implemented in C code and run on a PLC.

The model based engineering approach turned out to be a powerful modeling methodology, once you really got familiar with the development environment. Having all documentation in one place really proved profitable once the project grew. However, the modeling also turned out to be quite time consuming, and for smaller projects, the modeling procedure may turn out to take up considerable amounts of time, time maybe better spent elsewhere in the project.

The C implementation of the state machine on the PLC shows promising results, as it copes well with different events that can occur. The nonstandard approach taken using Quantum Leaps when implementing the state machine, turned fruitful as it eased the construction of the hierarchical state machines greatly. Also, from simulations, the PIDs used to control both the permeate flow and the retentate drain flow seem to be able to handle the dynamics of the system quite well.

The next step would of course be to try the system out on the real process. The models derived in Section 5 are not validated against any data, and the dynamics of the real system can prove to be quite different. With access to the real plant, it would also be possible not only to validate the mathematical model, but also to derive transfer functions by system identification. From these transfer functions it would be possible to use more sophisticated control methods such a model predictive control.

To get a more finished system a complete graphical user interface also has to be implemented. An idea is to implement an Android-based GUI. This would meet the requirements stated in the SysML model, including the advantages of having remote access to data logs and status readings.

References

[1] Wikipedia, "Osmosis — Wikipedia, the free encyclopedia." Website, 2012. http://en.wikipedia.org/wiki/Osmosis.

[2] American Water Works Association and Robert Bergman, Reverse osmosis and nanofiltration. Denver, CO: American Water Works Association, 2007.

[3] Peter Atkins and Julia de Paula, Atkins' Physical Chemistry. Oxford University Press, 2009.

[4] I. S. for Pharmaceutical Engineering, ISPE baseline guide, volume 4: Water and Stream systems. ISPE, 2009.

[5] Anders Widov, FR Pharma AB. Personal communication, 2012.

[6] Wikipedia, "Reverse osmosis — Wikipedia, the free encyclopedia." Website, 2012. http://en.wikipedia.org/wiki/Reverse_osmosis.

[7] CEDI University, "Intro to CEDI." Website, 2012. http://www.cediuniversity.com/.

[8] James Marriott and Eva Sörensen, "A general approach to modelling membrane modules," Chemical Engineering Science, vol. 58, pp. 4975–4990, 2003.

[9] M. Ben Boudinar, W.T. Hanbury and S. Avlonitis, "Numerical simulations and optimiation of spiral-wound modules," Desalination, vol. 86, pp. 273–290, 1992.

[10] A. Gambier, A. Krashnik and E. Badreddin, "Dynamic modeling of a simple reverse osmosis desalination plant for advanced control purposes," in American Control Conference, 2007.

[11] M.W. Robertson, J.C. Watters, P.B. Desphande, J.Z. Assef and I.M. Alitqi, "Model based control for reverse osmosis desalination processes," Desalination, vol. 104, pp. 59–68, 1996.

[12] Tim Weilkiens, SystemEngineering with SysML and UML. Morgan Kaufmann OMG Press, 2006.

[13] Sanford Friedenthal, Alan Moore and Rick Steiner, A Practical Guide to SysML. Morgan Kaufmann OMG Press, 2009.

[14] Object Management Group, "OMG Systems Modeling Language (OMG SYSML)." Website, 2012. http://www.omg.org/spec/SysML/1.3/PDF.

[15] Topcased. Website, 2012. http://www.topcased.org.

[16] U.S. Food and Drug Administration, "21 CFR Part 11 Final Rule." Website, 2012. http://www.21cfrpart11.com/files/library/government/21cfrpart11_final_rule.pdf.

[17] European Commision, "Directive 2006/42/EC on machinery." Website, 2012. http://ec.europa.eu/enterprise/sectors/mechanical/documents/legislation/machinery/index_en.htm.

[18] Bernt Nilsson, Membrancentrum LTH. Personal communication, 2012.

[19] J.G. Wijmans and R.W. Baker, "The solution-diffusion model: a review," Journal of Membrane Science, vol. 107, pp. 1–21, 1995.

[20] Jane Kucera, Reverse osmosis - Industiral Applications and Processes. Schrivener Publishing LLC, 2010.

[21] B&R Datasheet. Website, 2012. http://www.br-automation.com/downloads_br_productcatalogue/BRP4440000000000000121710/X20CPx48x-ENG.pdf.

[22] FESTO, "Programmable Logic Controllers, Basic Level." Website, 2012. http://www.festo-didactic.com/ov3/media/customers/1100/093311_web_leseprobe.pdf.

[23] infoPLC.net, "Programmable controllers Bernecker&Rainer, Basic programming." Website, 2012. http://infoplc.net/files/descargas/br/infoplc_net_br_leccion_2_basic_programing.pdf.

[24] Miro Samek, Practical Statecharts in C/C++. CMP Books, 2002.

[25] Quantum Leaps ®, LLC, "State machines & tools." Website, 2012. http://www.state-machine.com.

A SysML Model

A.1 Requirements

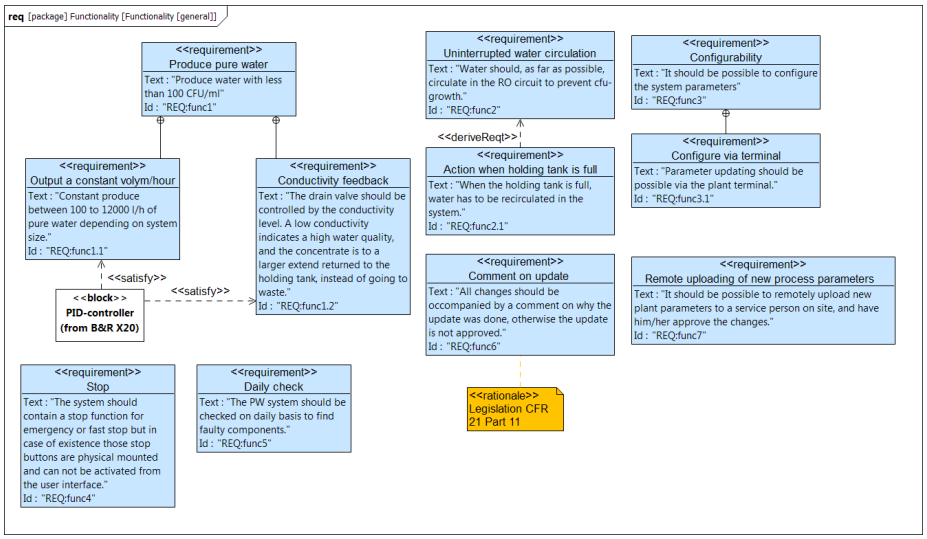

Figure A.1: General functionality diagram

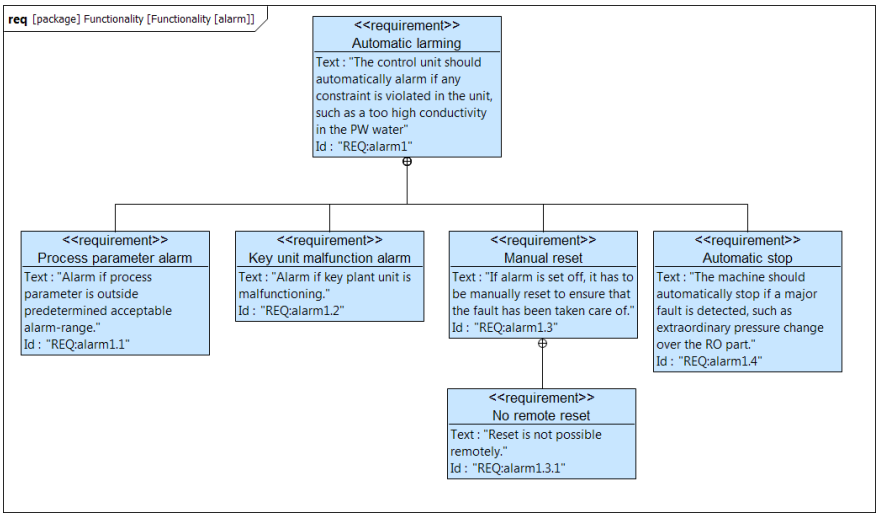

Figure A.2: Functionality diagram concerning alarms.

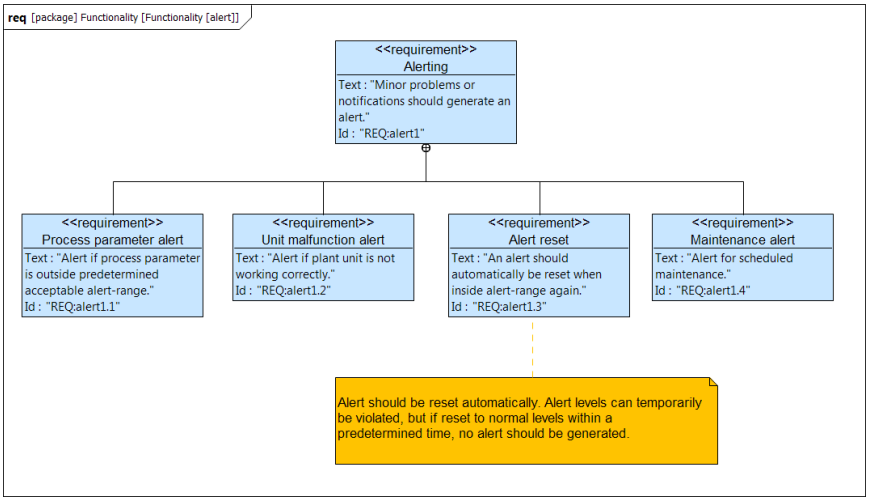

Figure A.3: Functionality diagram concerning alerts

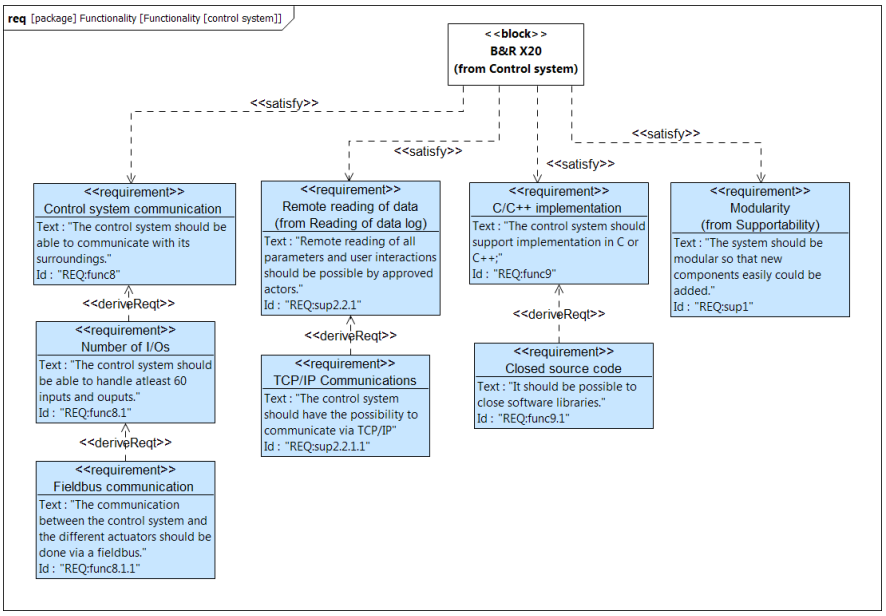

Figure A.4: Functionality diagram for the control system.

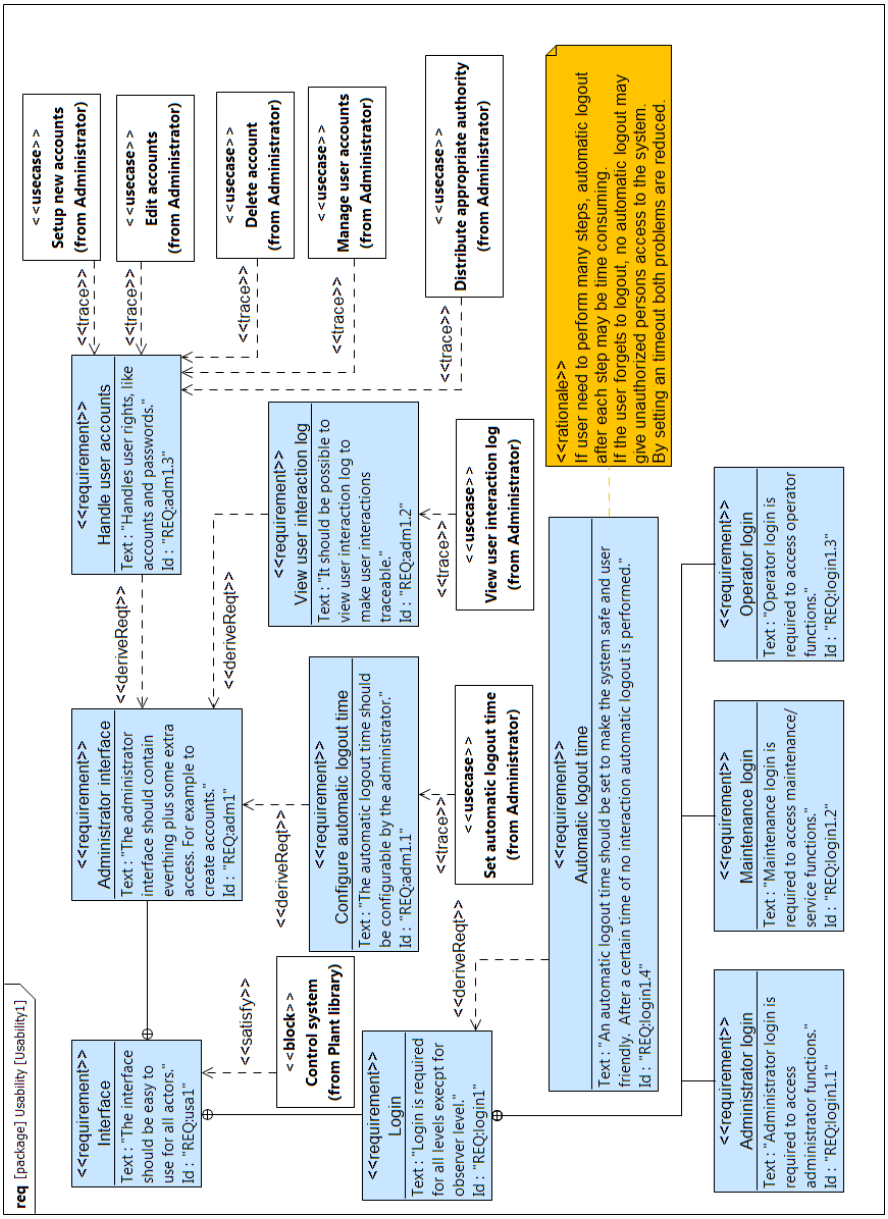

Figure A.5: Usability diagram 1.

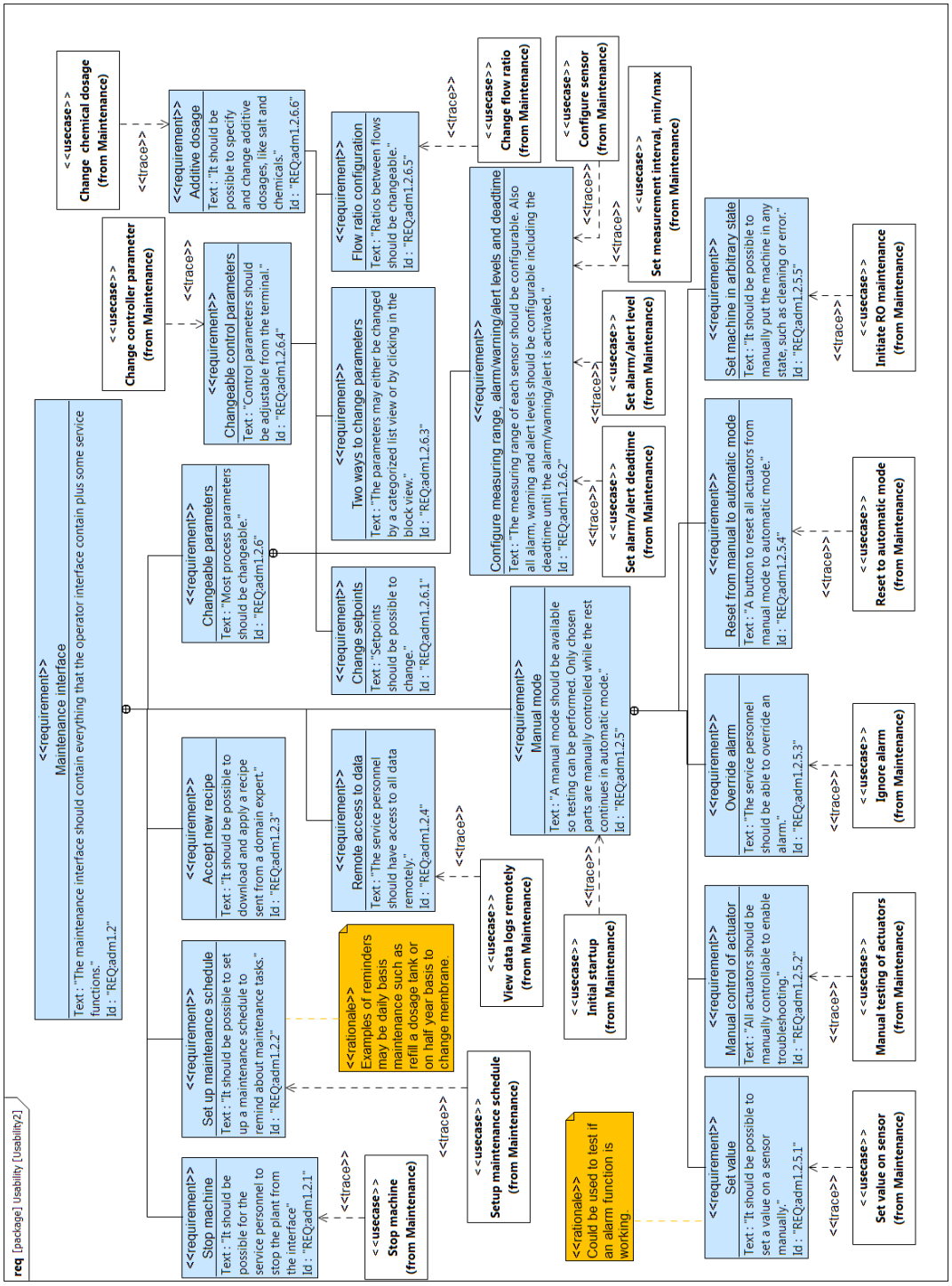

Figure A.6: Usability diagram 2.

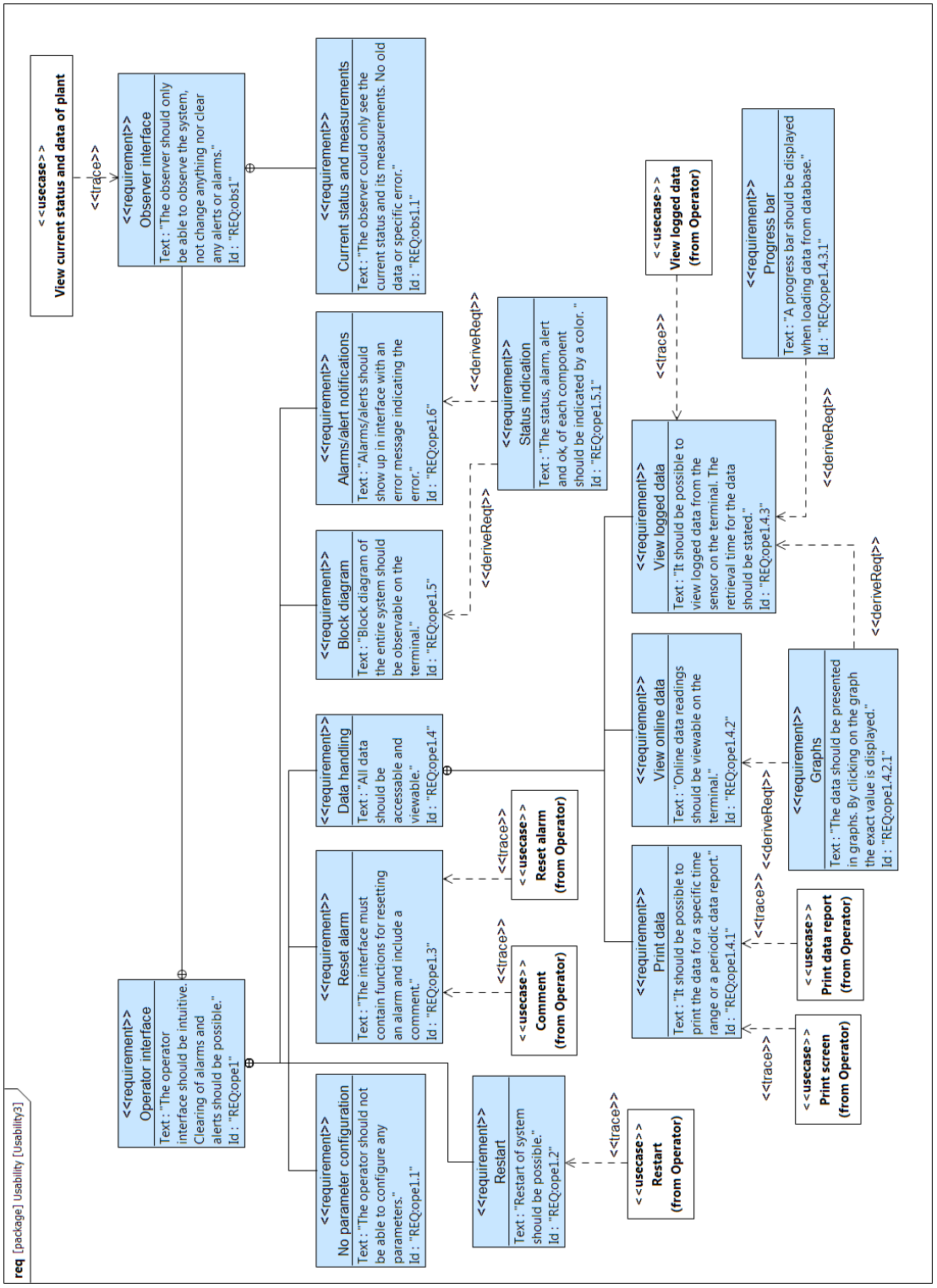

Figure A.7: Usability diagram 3.

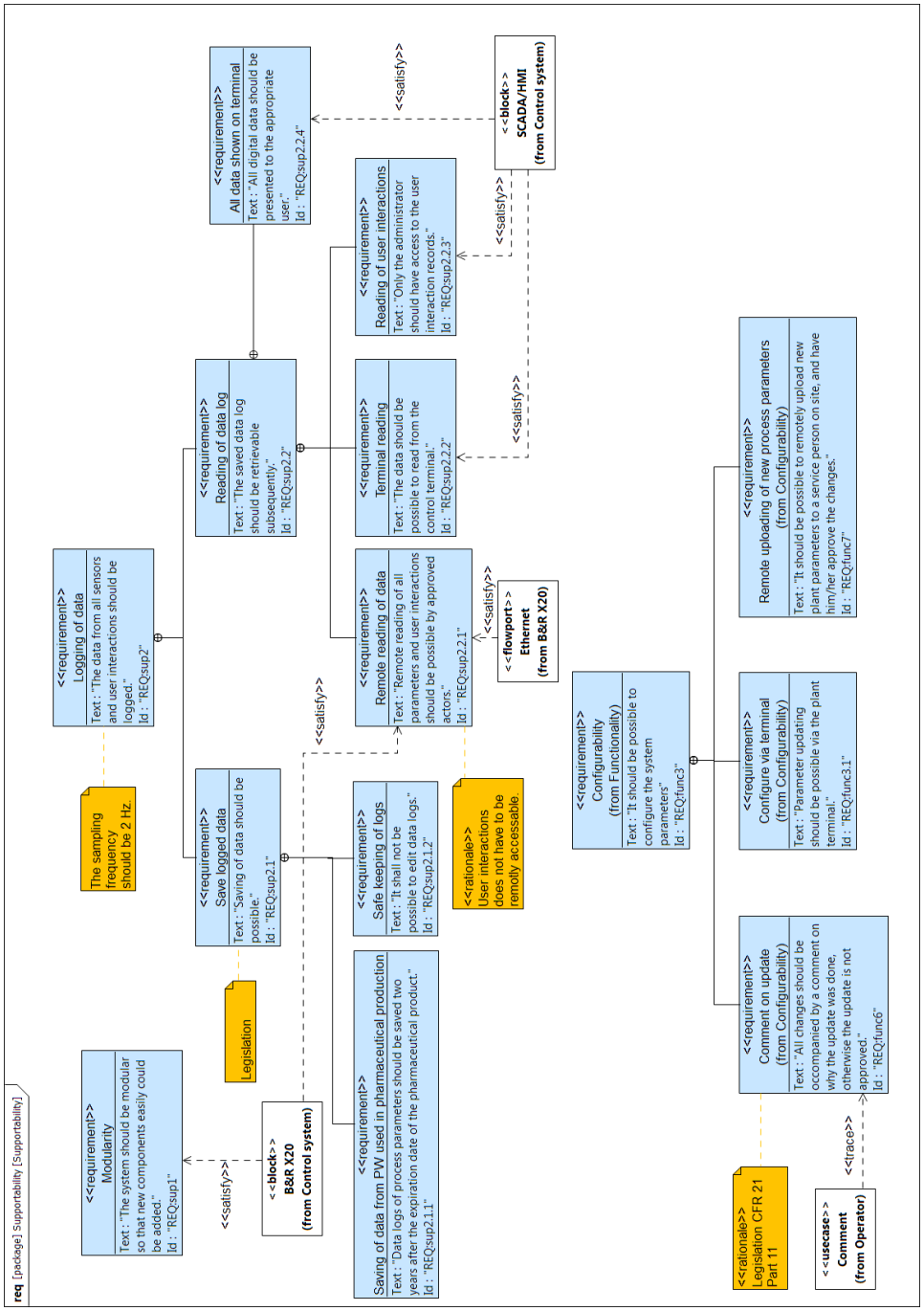

Figure A.8: Supportability

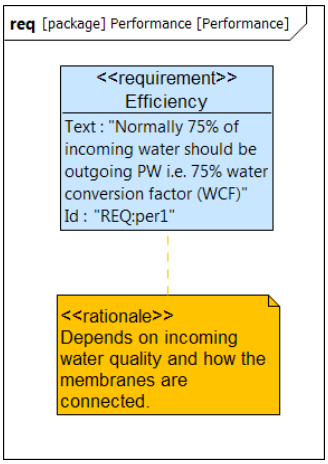

Figure A.9: Performance

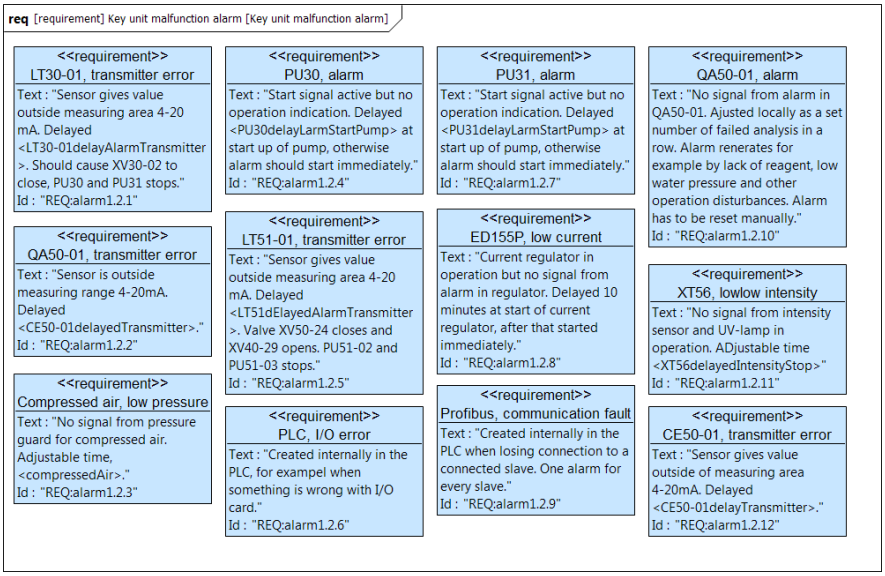

Figure A.10: Key unit malfunction alarm

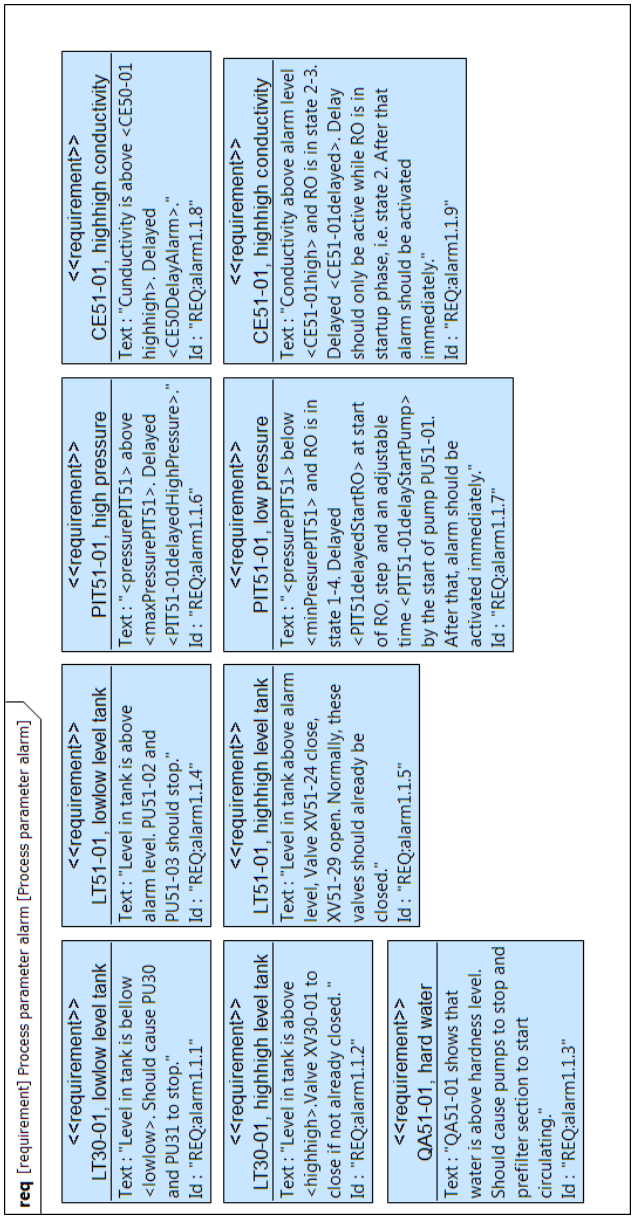

Figure A.11: Process parameter alarm

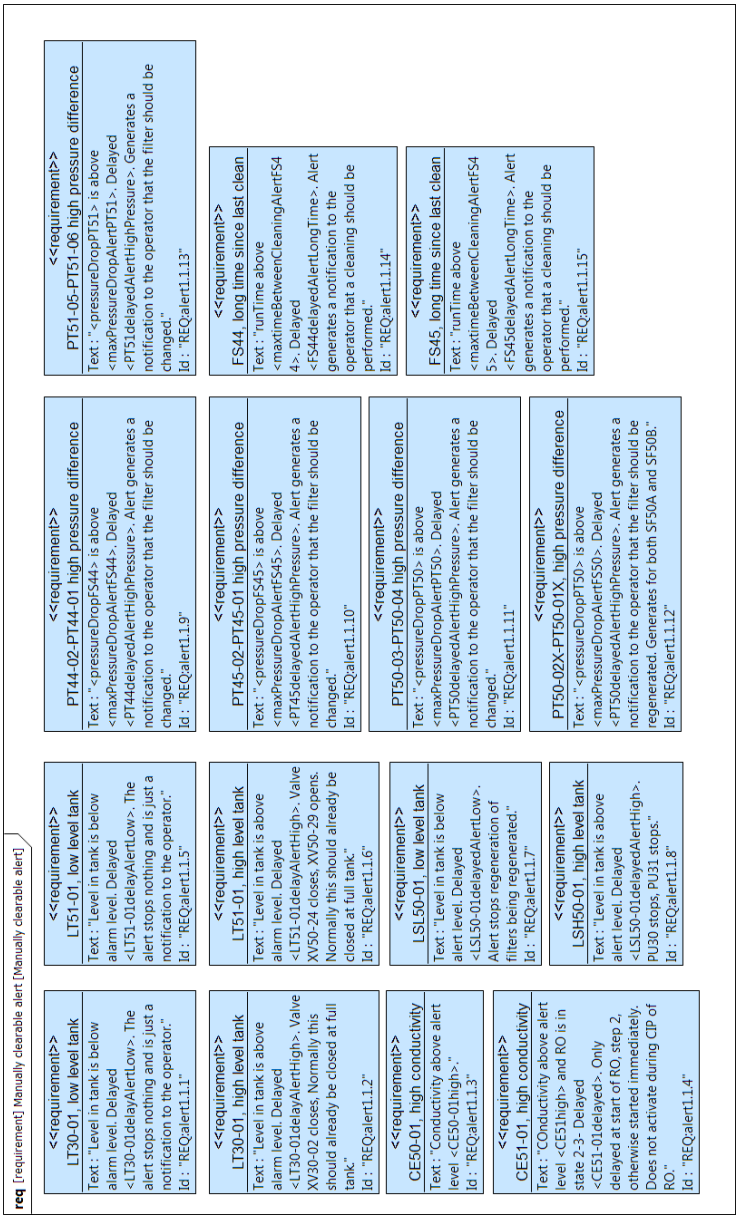

Figure A.12: Manually clearable alerts

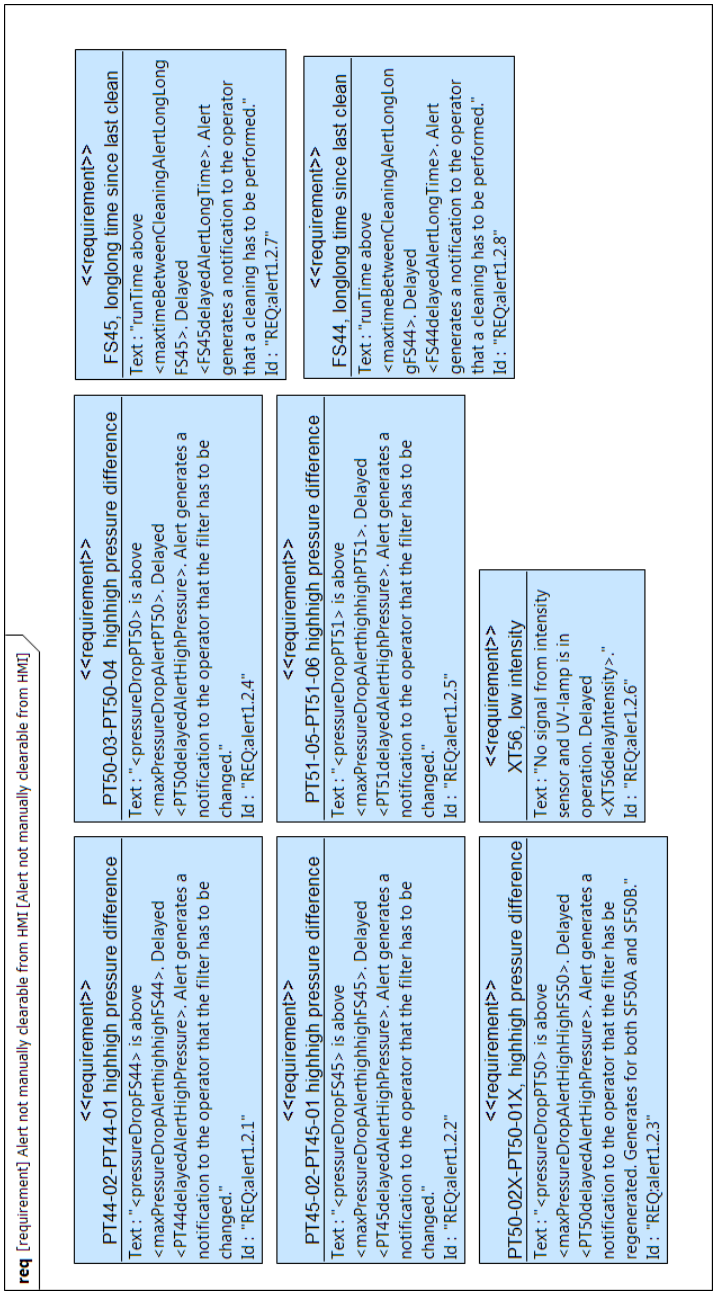

Figure A.13: Alert not manually clearable from HMI

A.2 Stakeholders

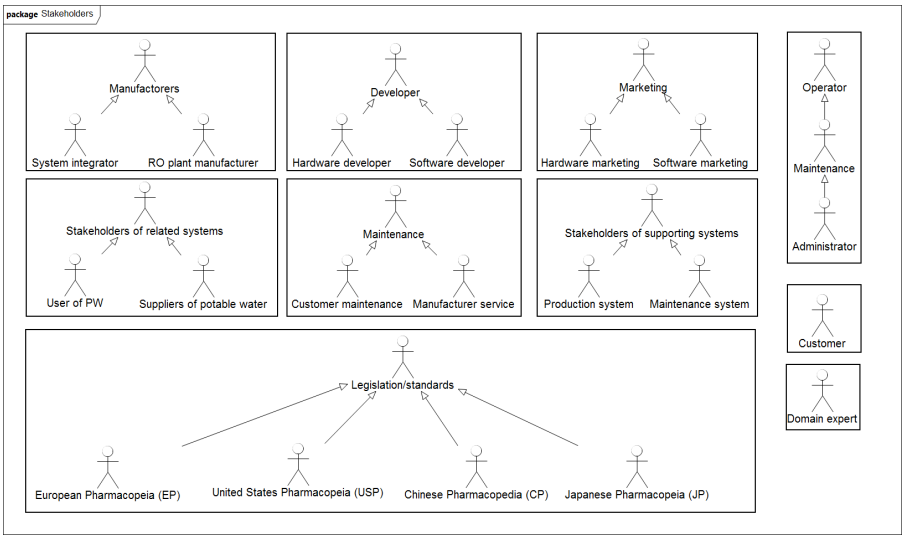

Figure A.14: Stakeholders

A.3 Use cases

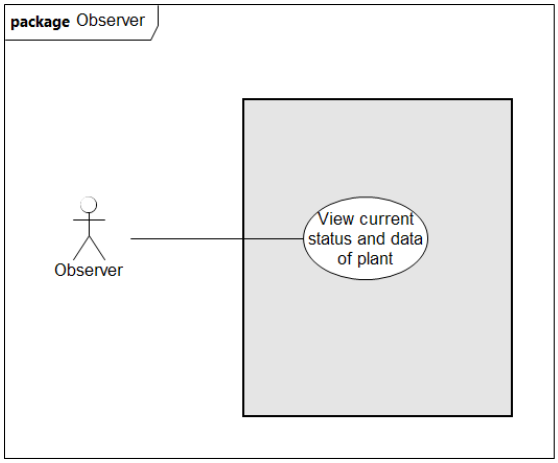

Figure A.15: Viewer

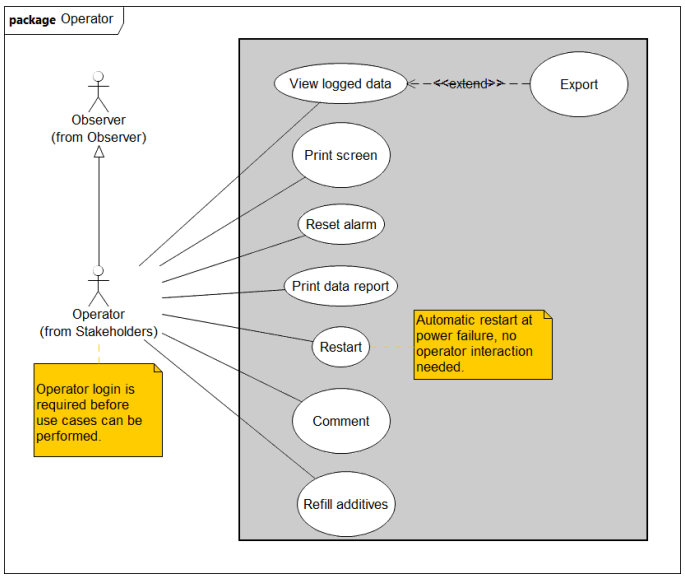

Figure A.16: Operator

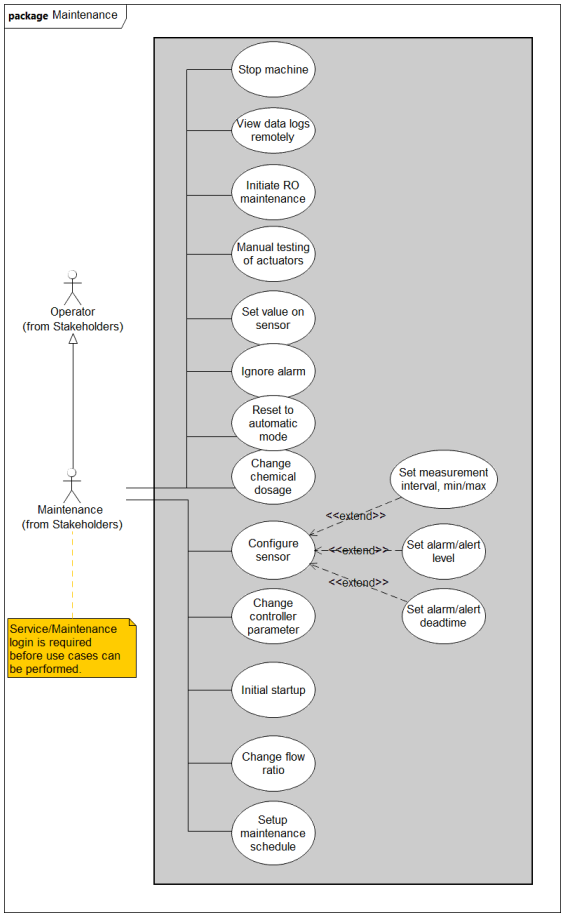

Figure A.17: Maintenance

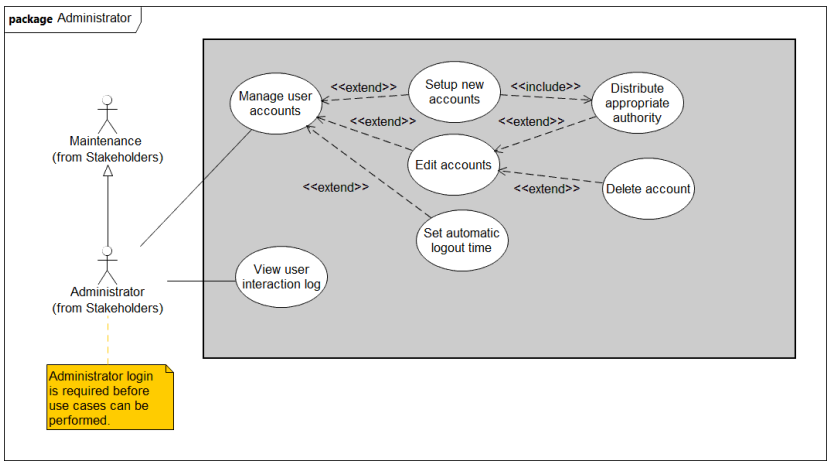

Figure A.18: Administrator

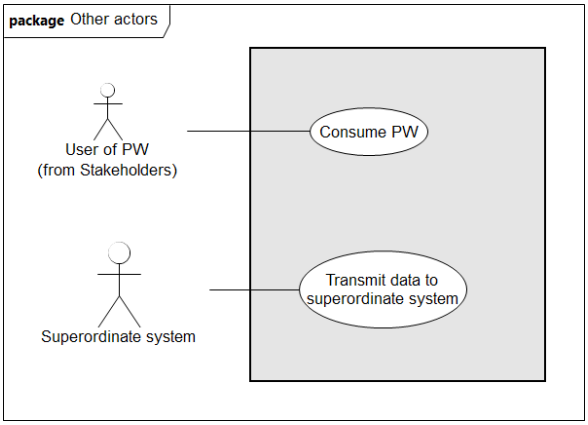

Figure A.19: Other actors

A.4 Building block diagrams

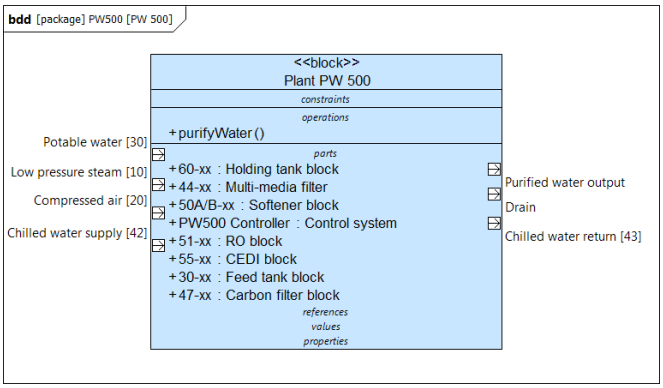

Figure A.20: PW 500

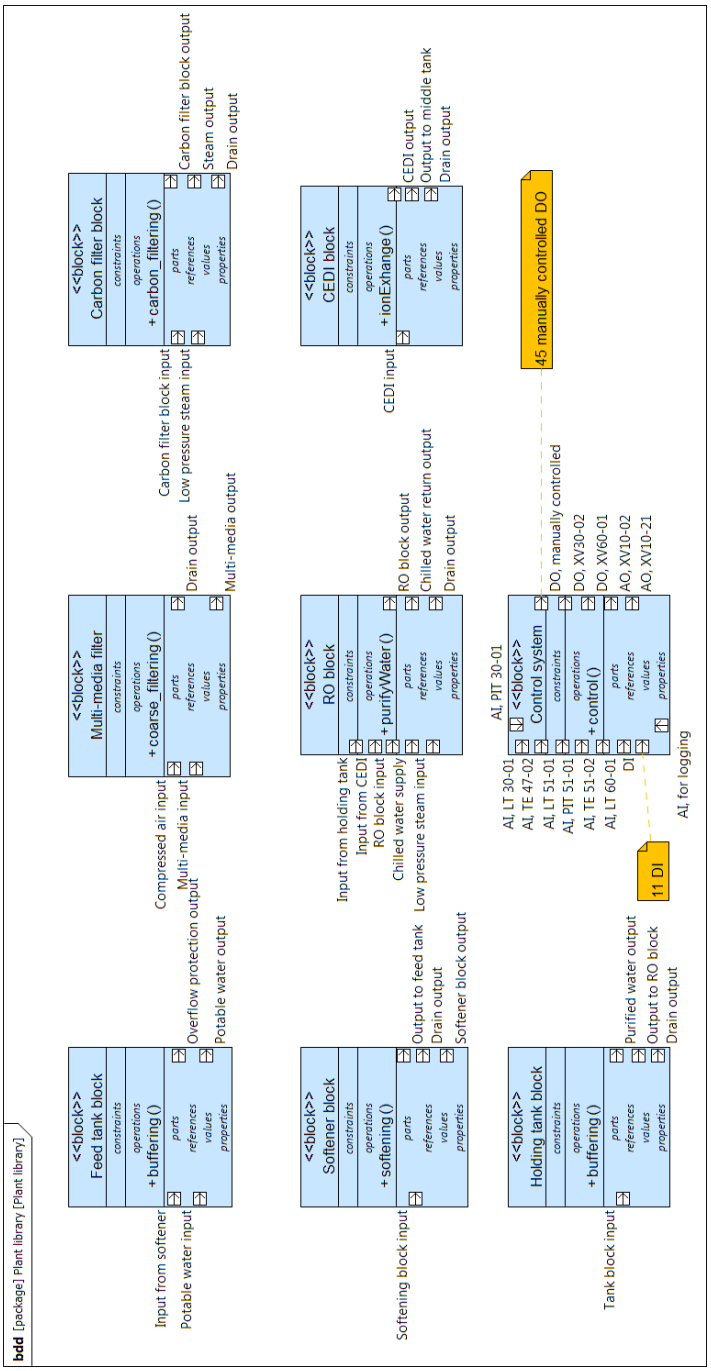

Figure A.21: Plant library

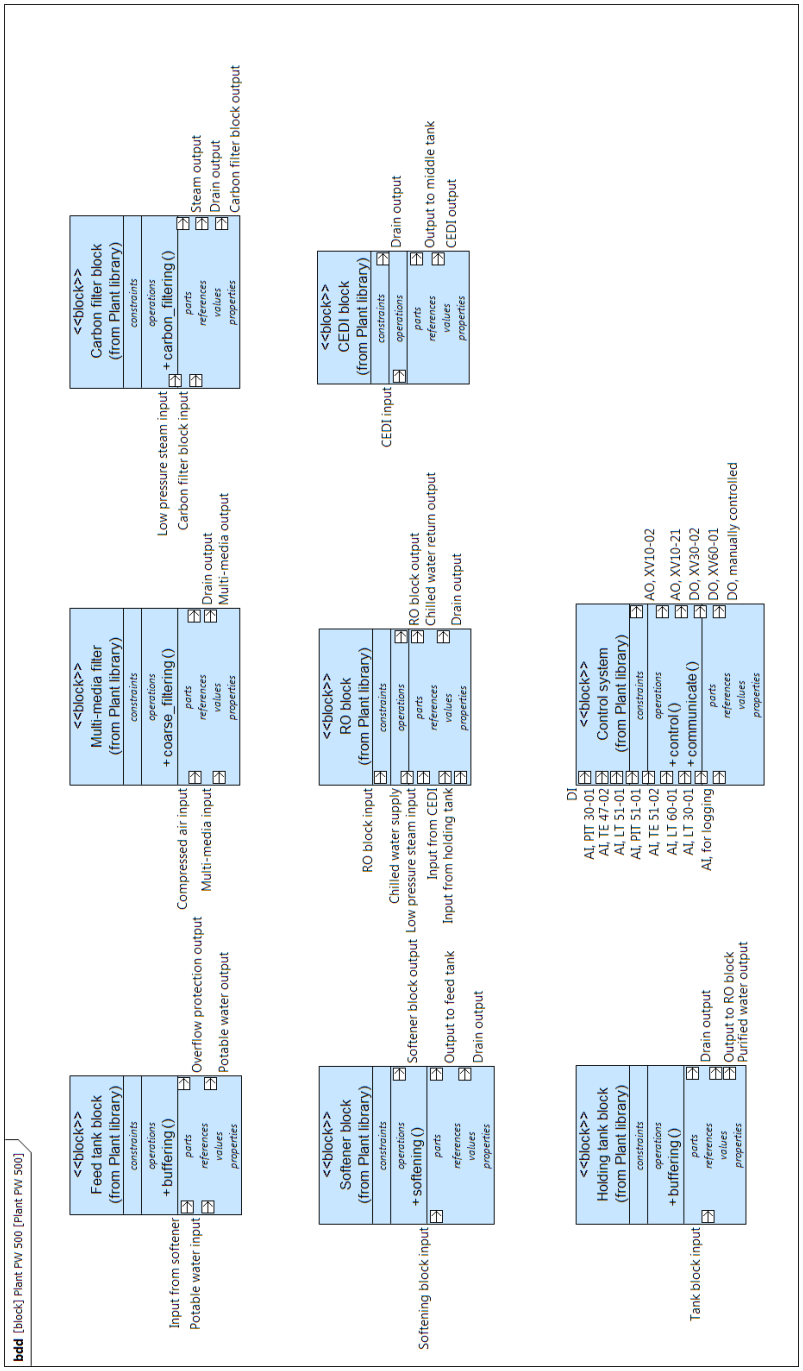

Figure A.22: Plant PW 500

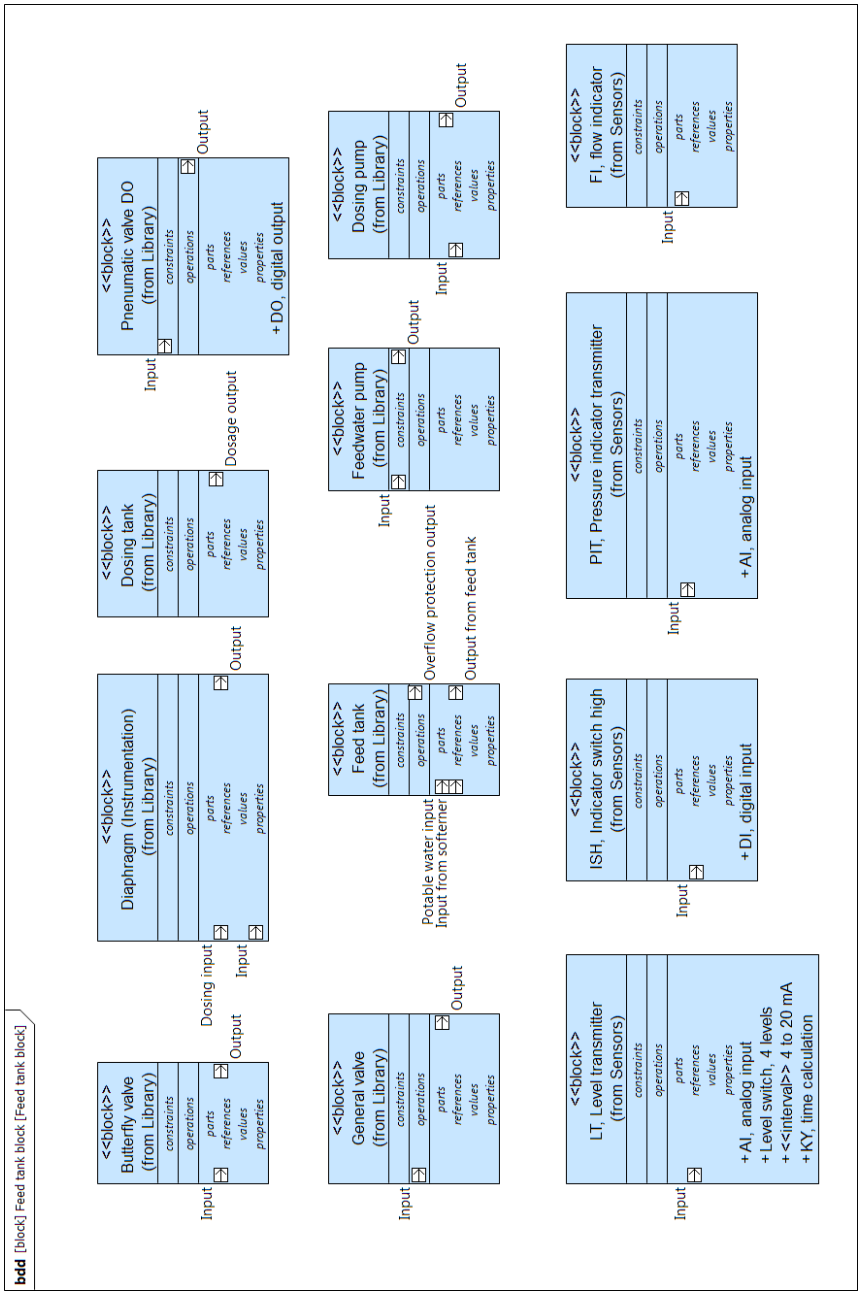

Figure A.23: Feed tank block

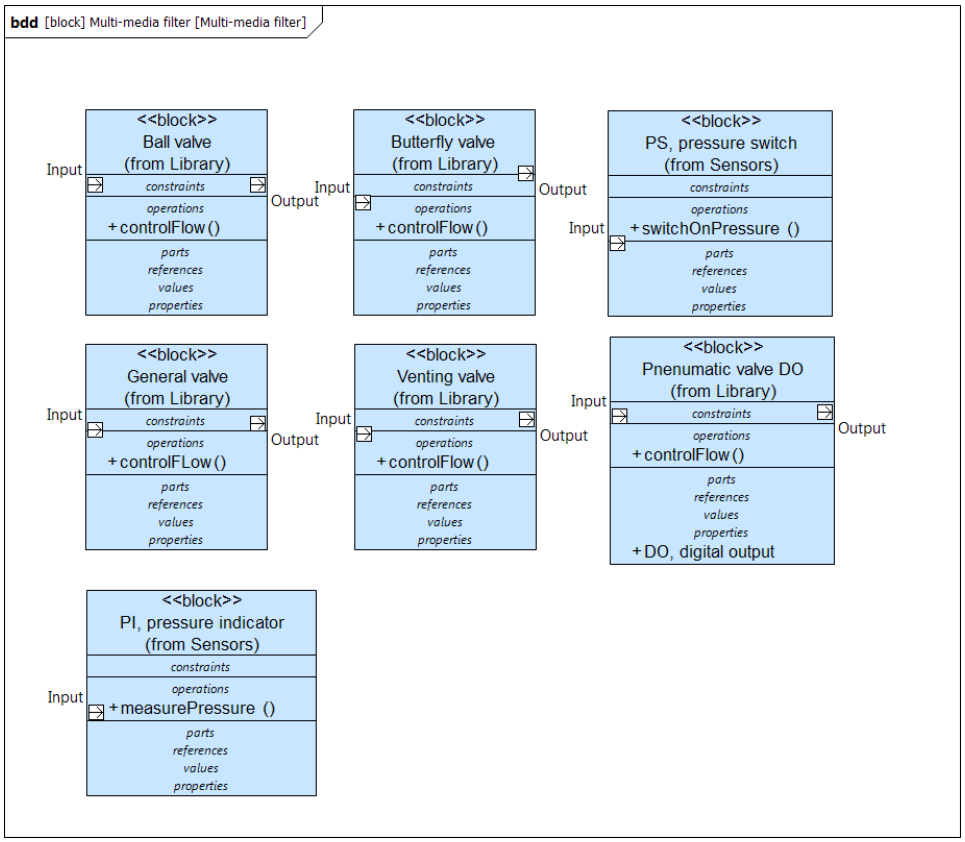

Figure A.24: Multi-media filter

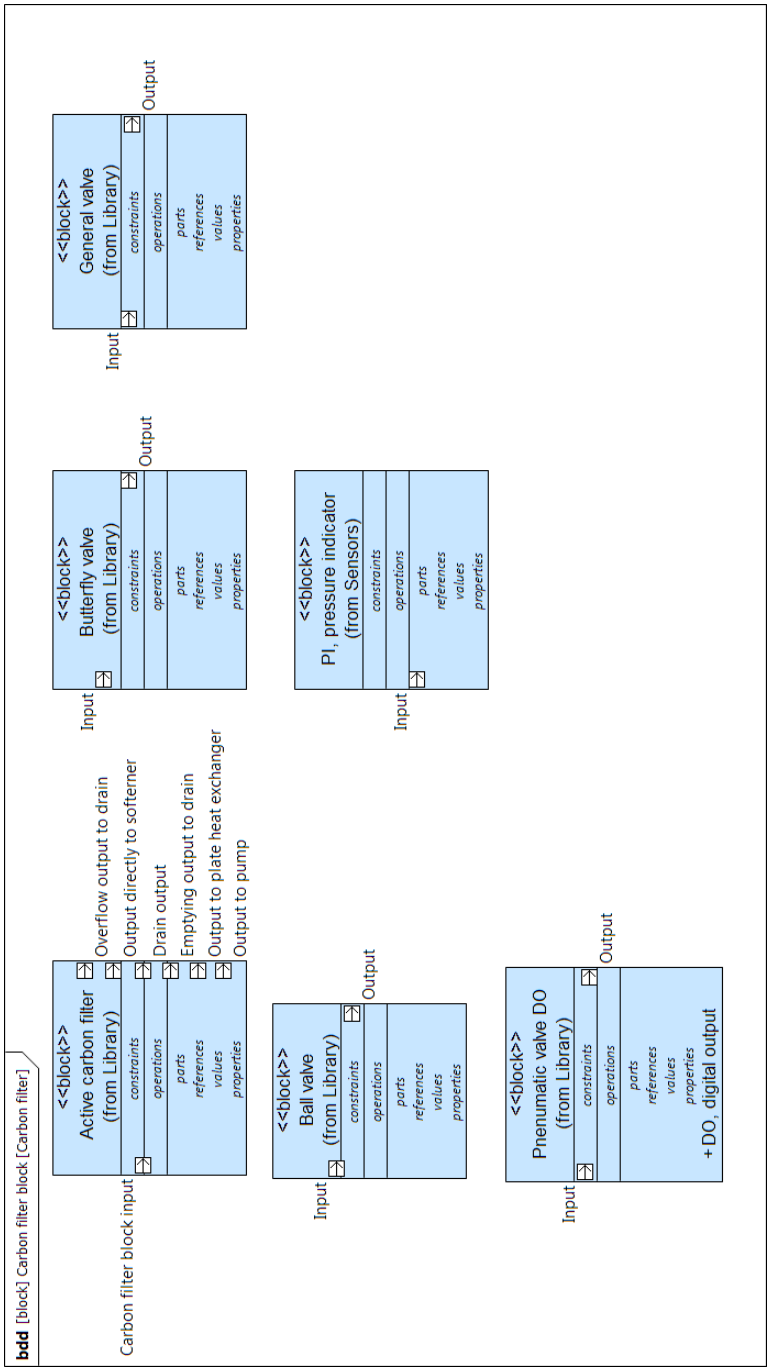

Figure A.25: Activ carbon filter

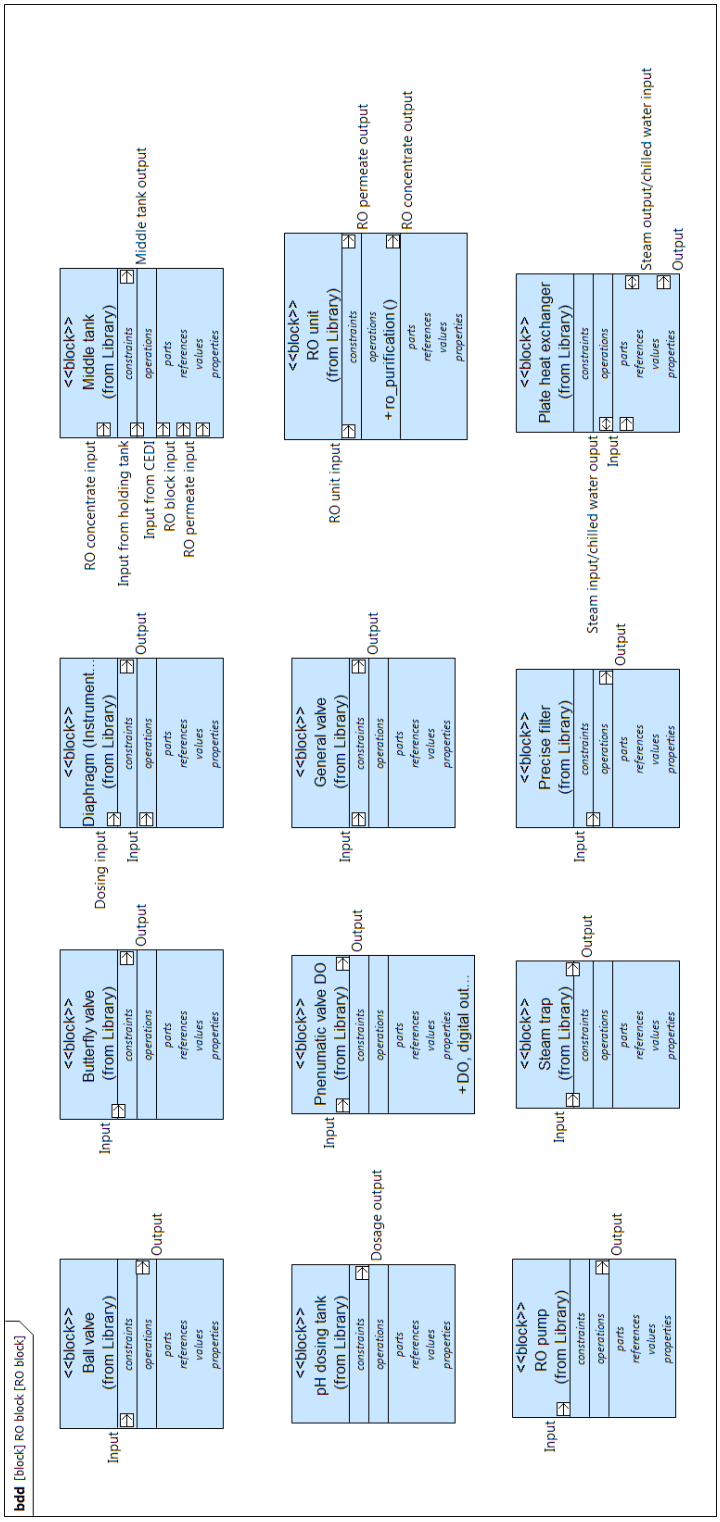

Figure A.26: RO block

A.5 Internal building block diagrams

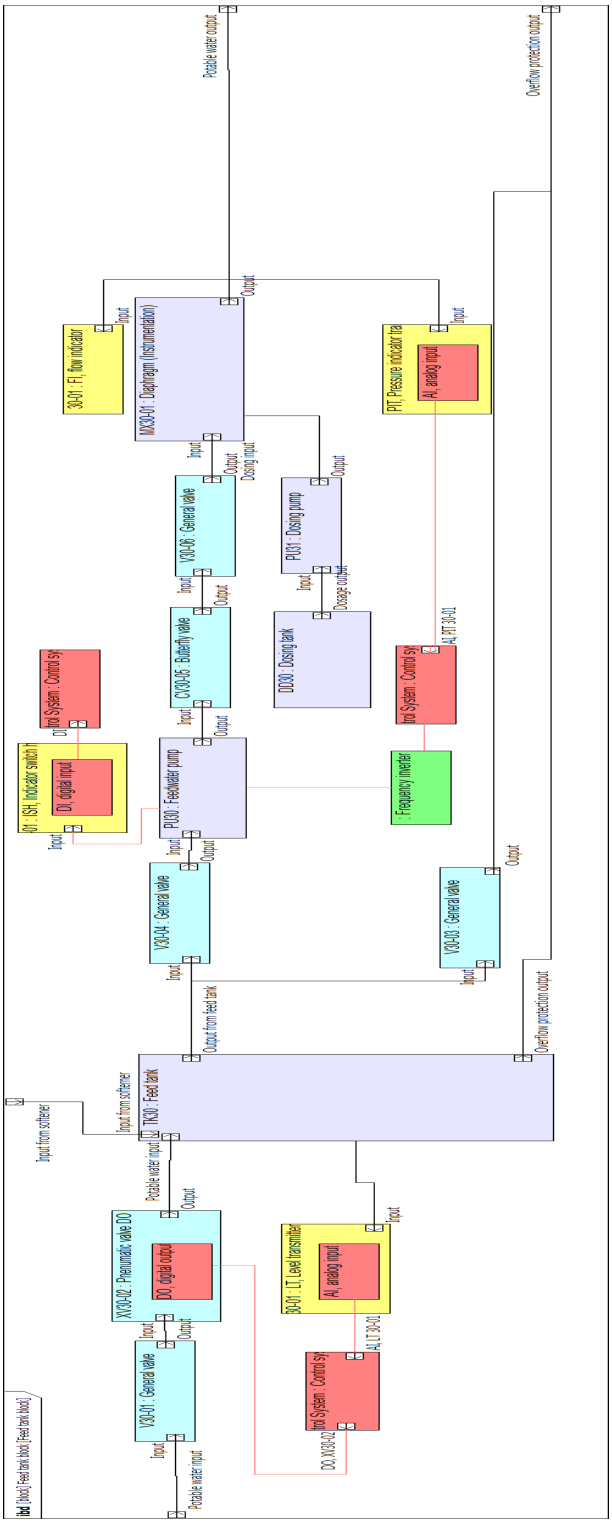

Figure A.27: Internal Building Block diagram for the feed tank

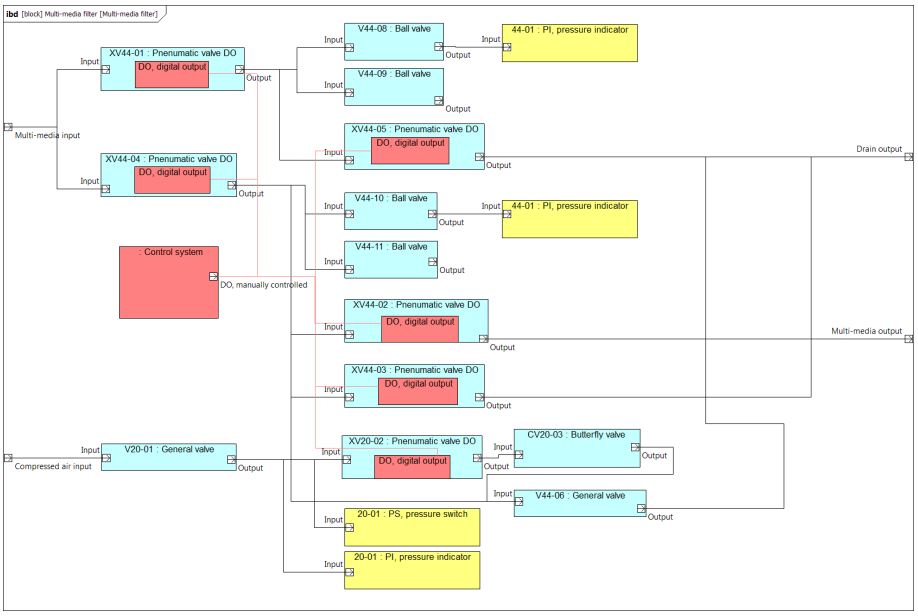

Figure A.28: Internal Building Block diagram for the multi-media filter

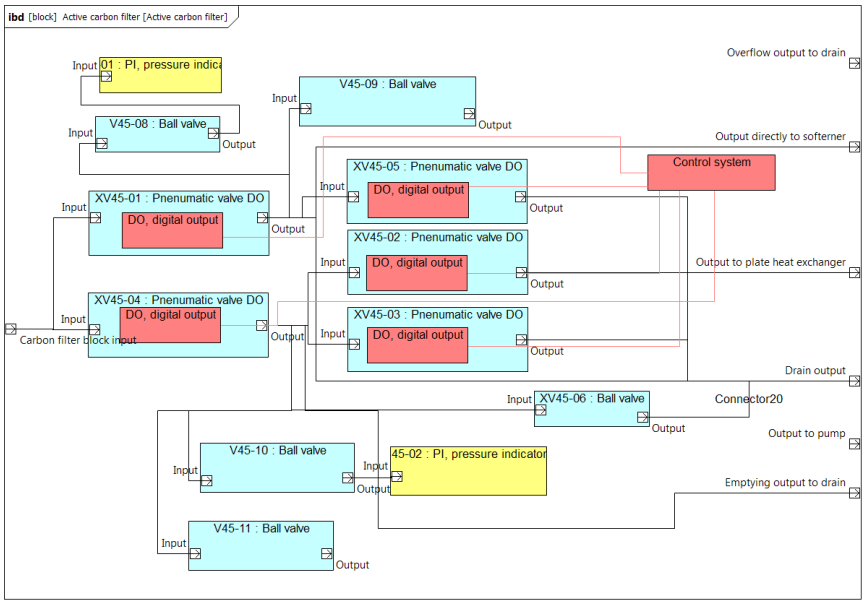

Figure A.29: Internal Building Block diagram for the active carbon filter

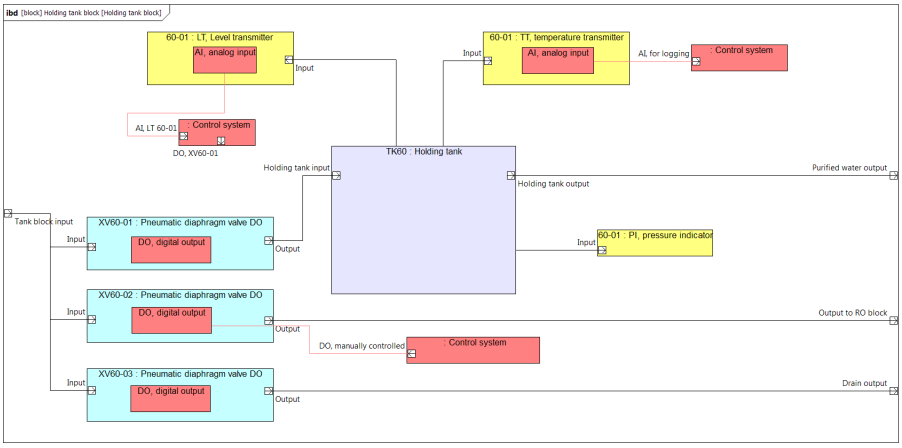

Figure A.30: Internal Building Block for the holding tank

A.6 Parametric diagrams

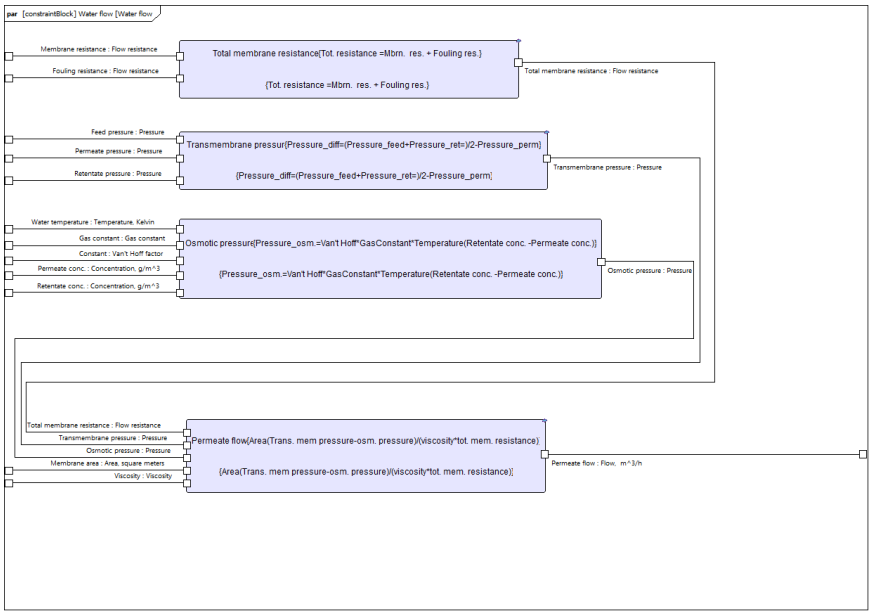

Figure A.31: Parametric diagram describing the water flow through the membrane

A.7 Package diagrams

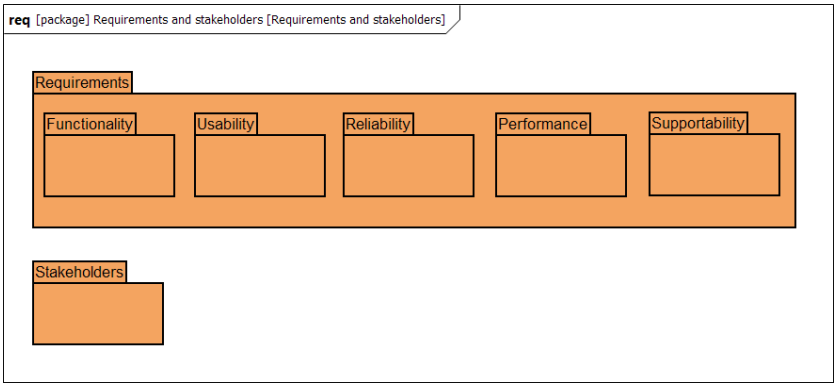

Figure A.32: Package diagram for requirements and stakeholders

A.8 Activity diagrams

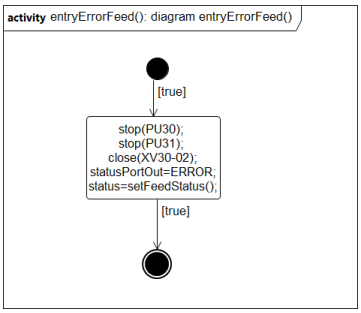

Figure A.33: Action when entering error state in feed state machine

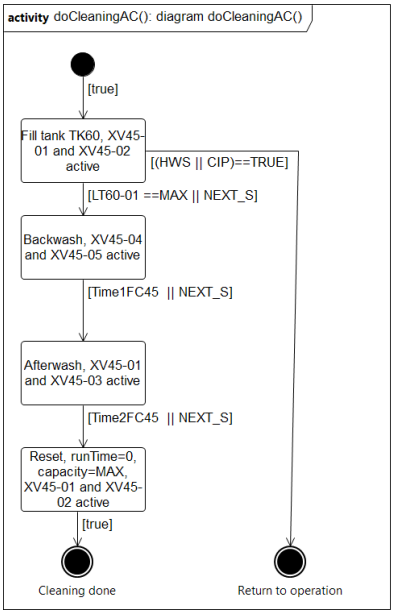

Figure A.34: Action during cleaning state in active carbon state machine

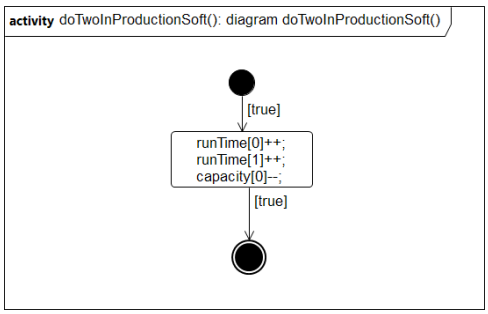

Figure A.35: Action during twoInProduction state in softener state machine

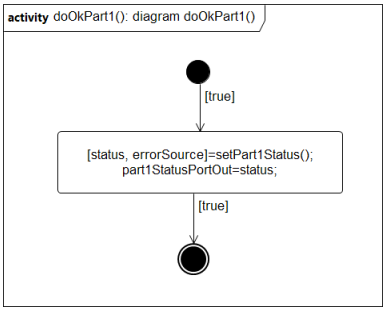

Figure A.36: Action during ok state in Part1 state machine

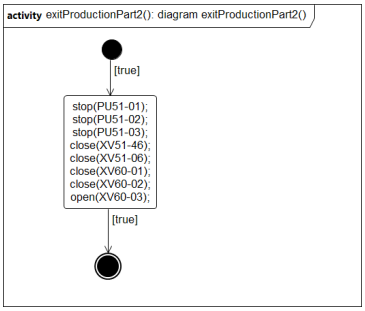

Figure A.37: Action when exiting production state in Part2 state machine

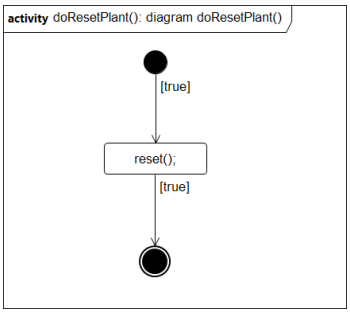

Figure A.38: Action during reset state in Plant state machine

A.9 Sequence diagrams

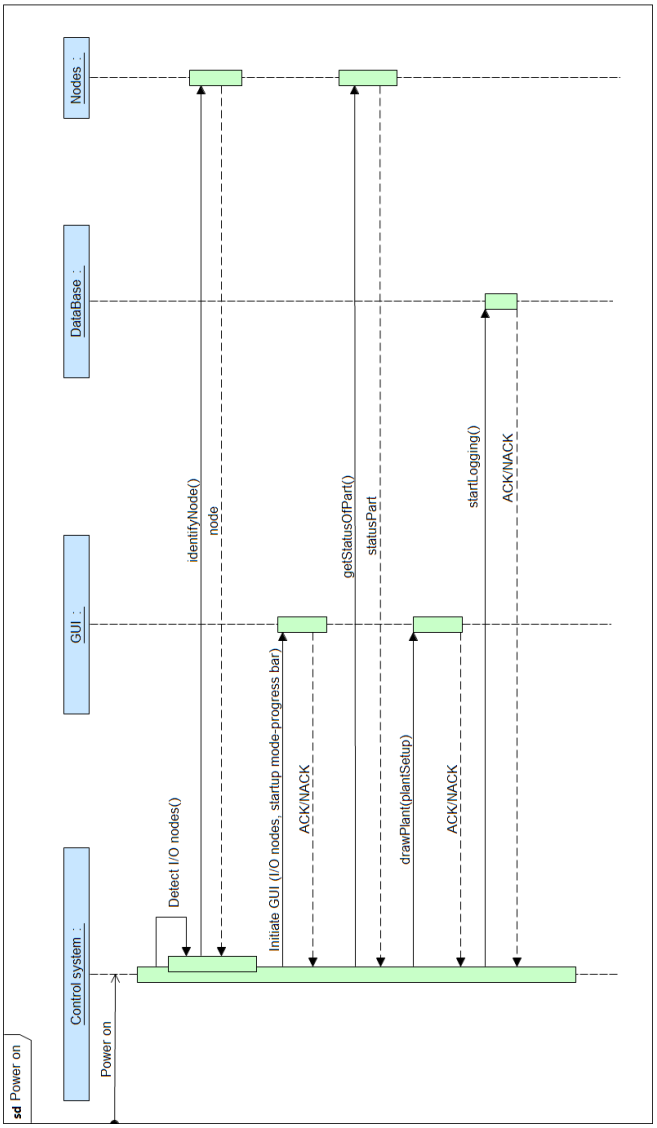

Figure A.39: Initial startup

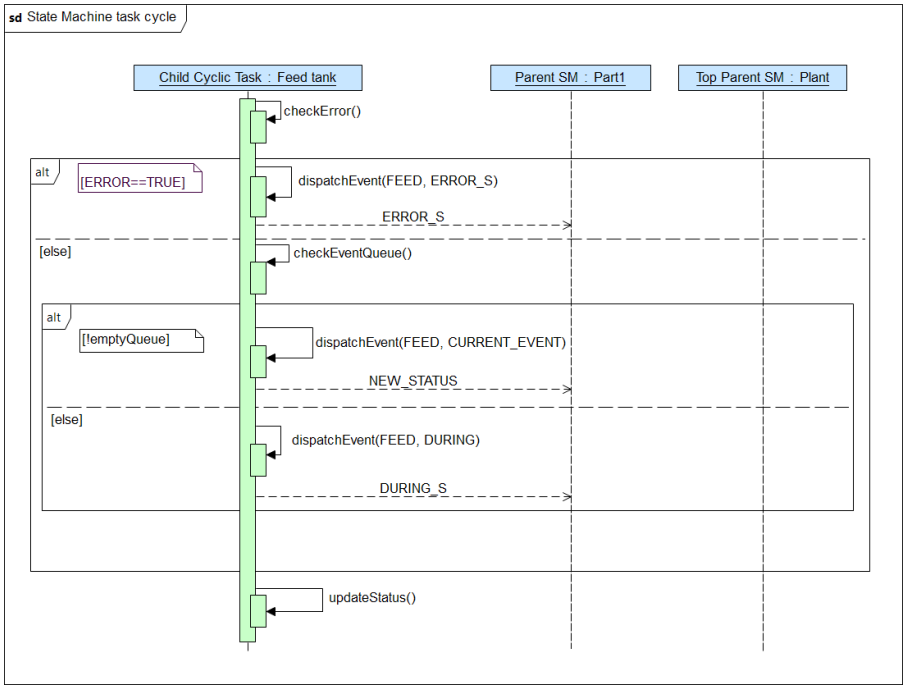

Figure A.40: State machine task cycle

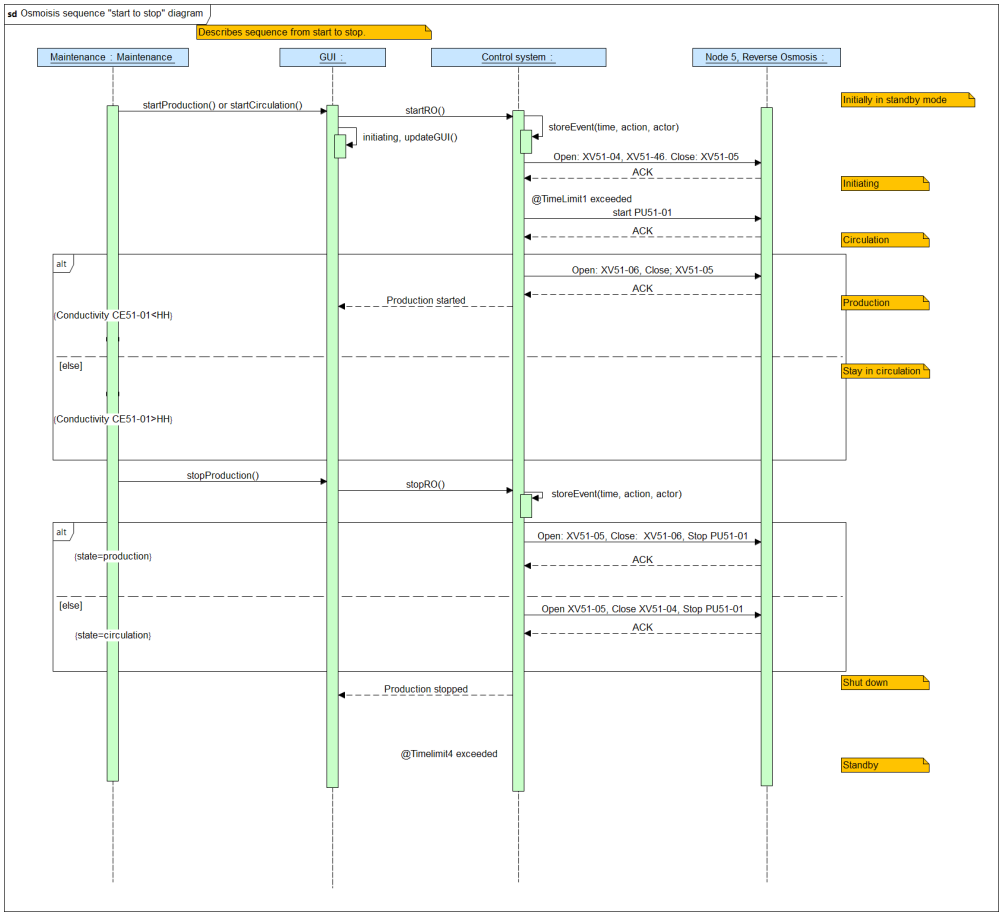

Figure A.41: Osmosis start and stop sequence

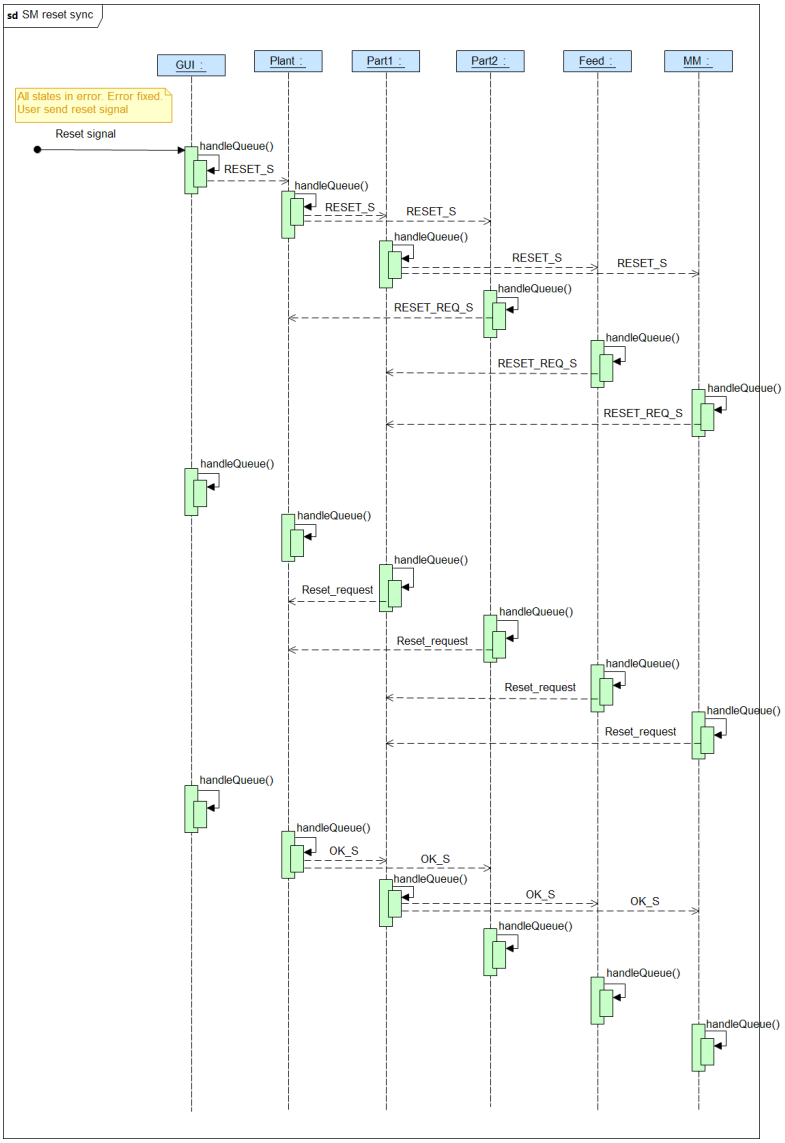

Figure A.42: Synchronization sequence when reseting from an error

A.10 State Machine diagrams

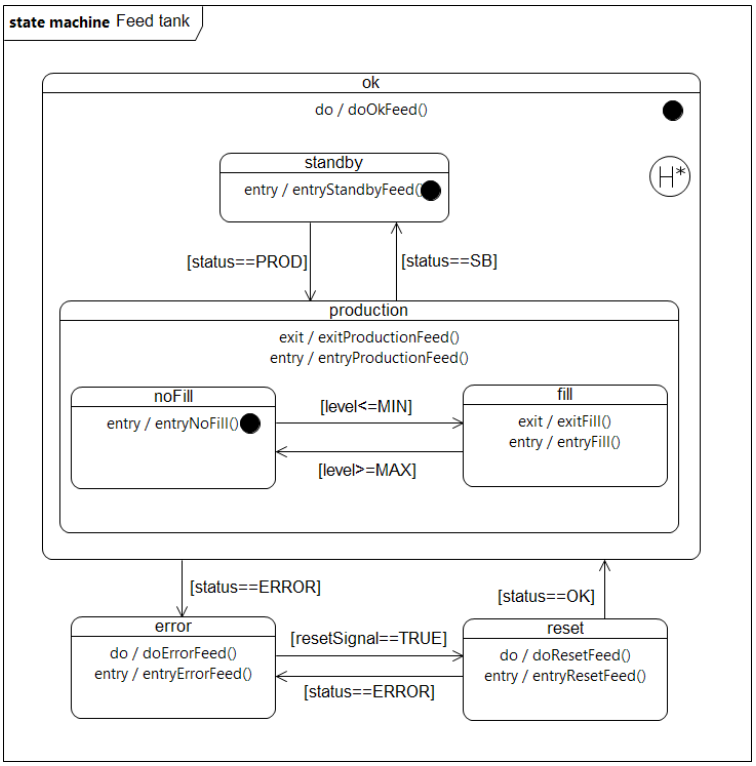

Figure A.43: State machine diagram for the feed tank

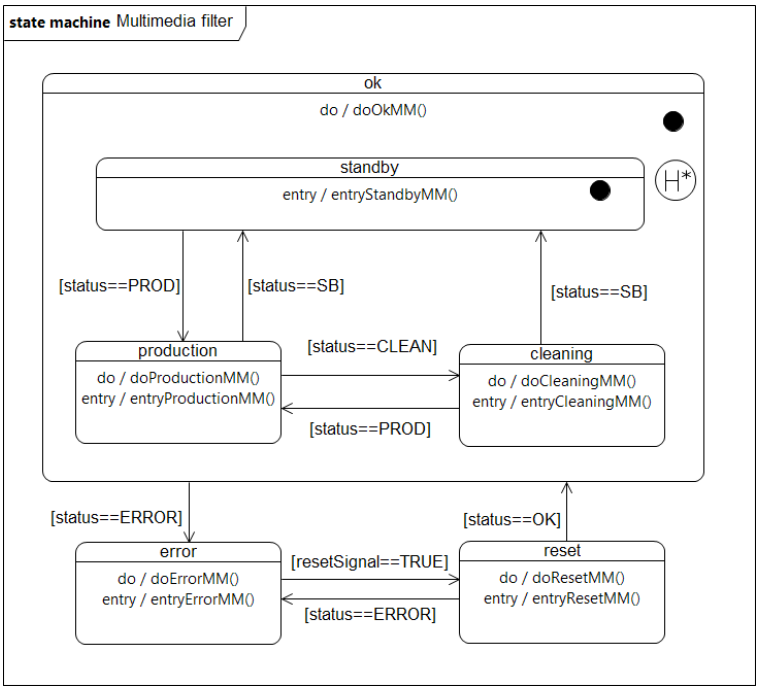

Figure A.44: State machine diagram for the multi media filter

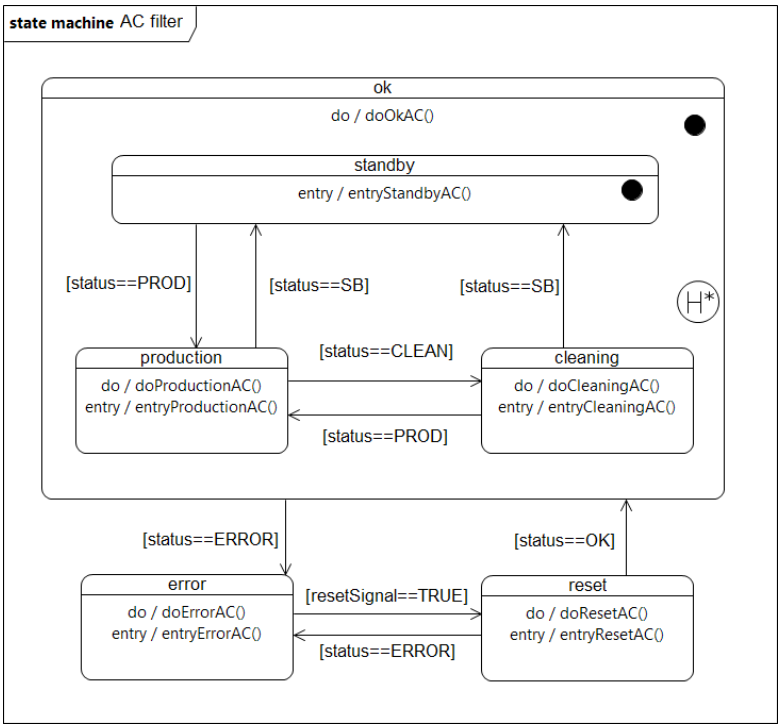

Figure A.45: State machine diagram for the active carbon filter

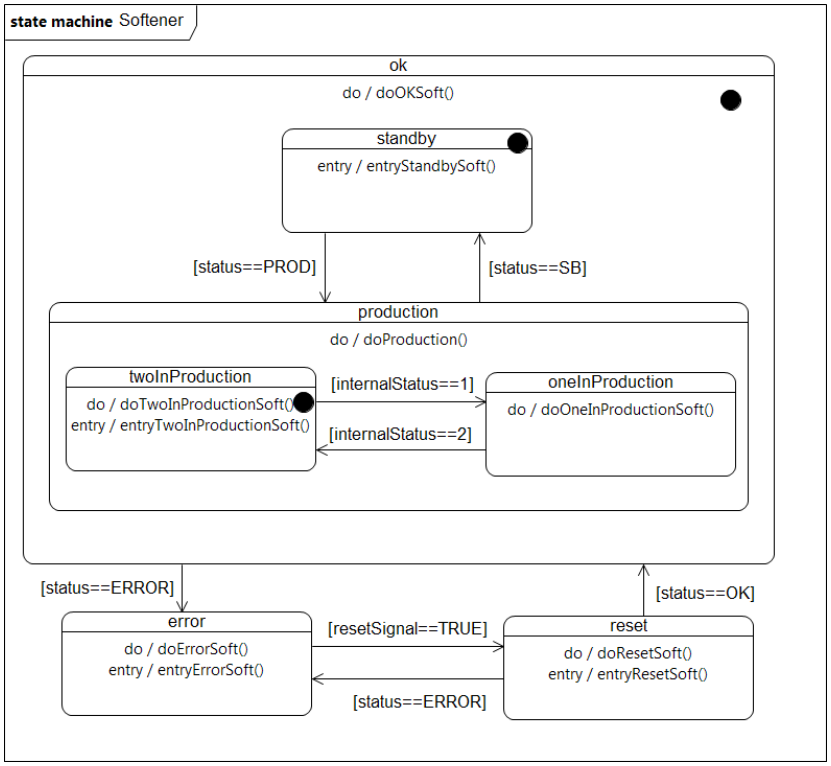

Figure A.46: State machine diagram for the softener

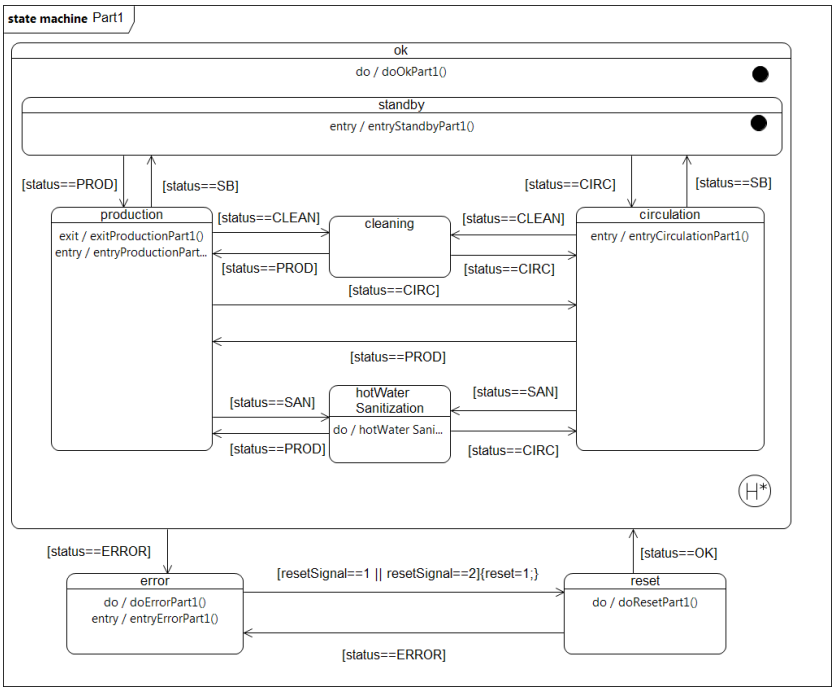

Figure A.47: State machine diagram for Part1

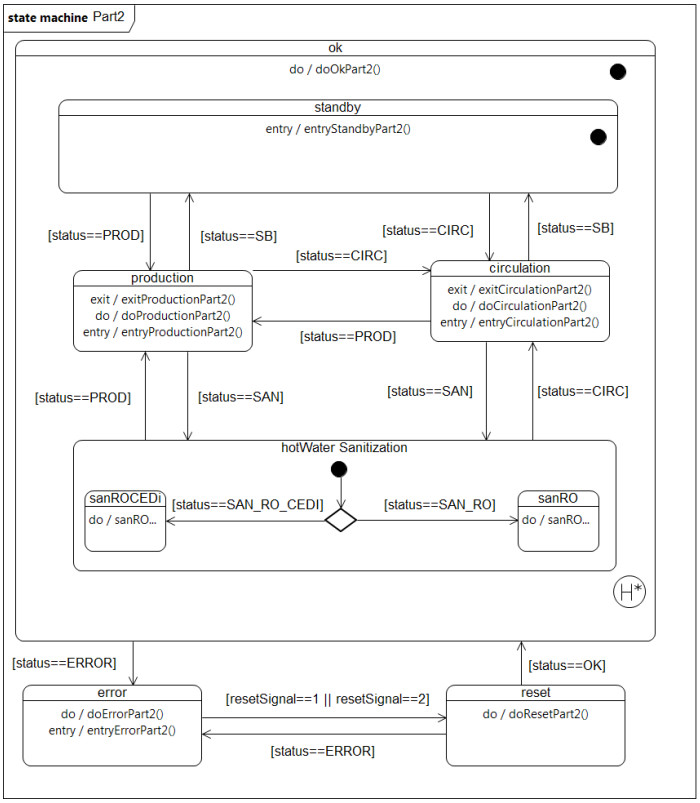

Figure A.48: State machine for Part2

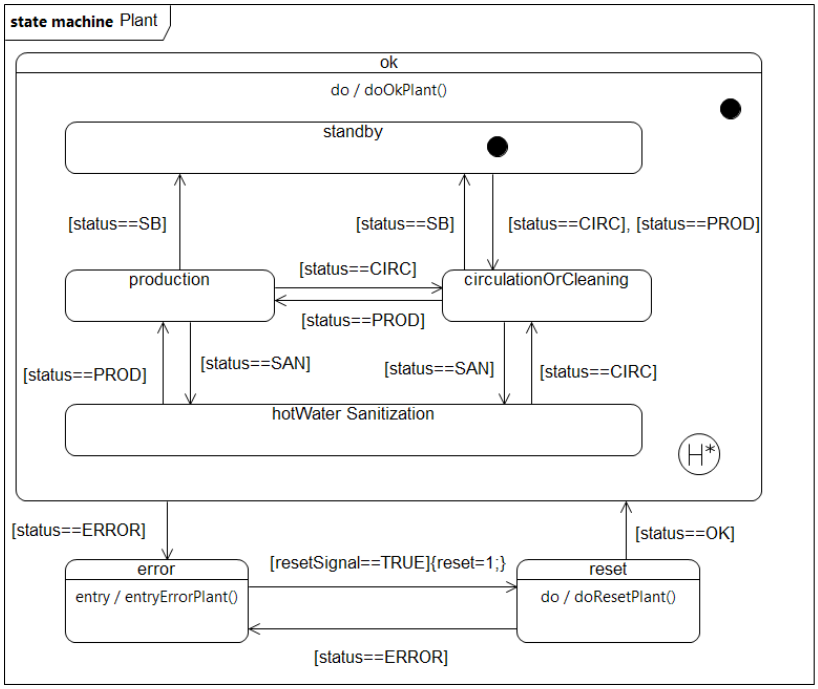

Figure A.49: State machine diagram for the Plant

B Matlab

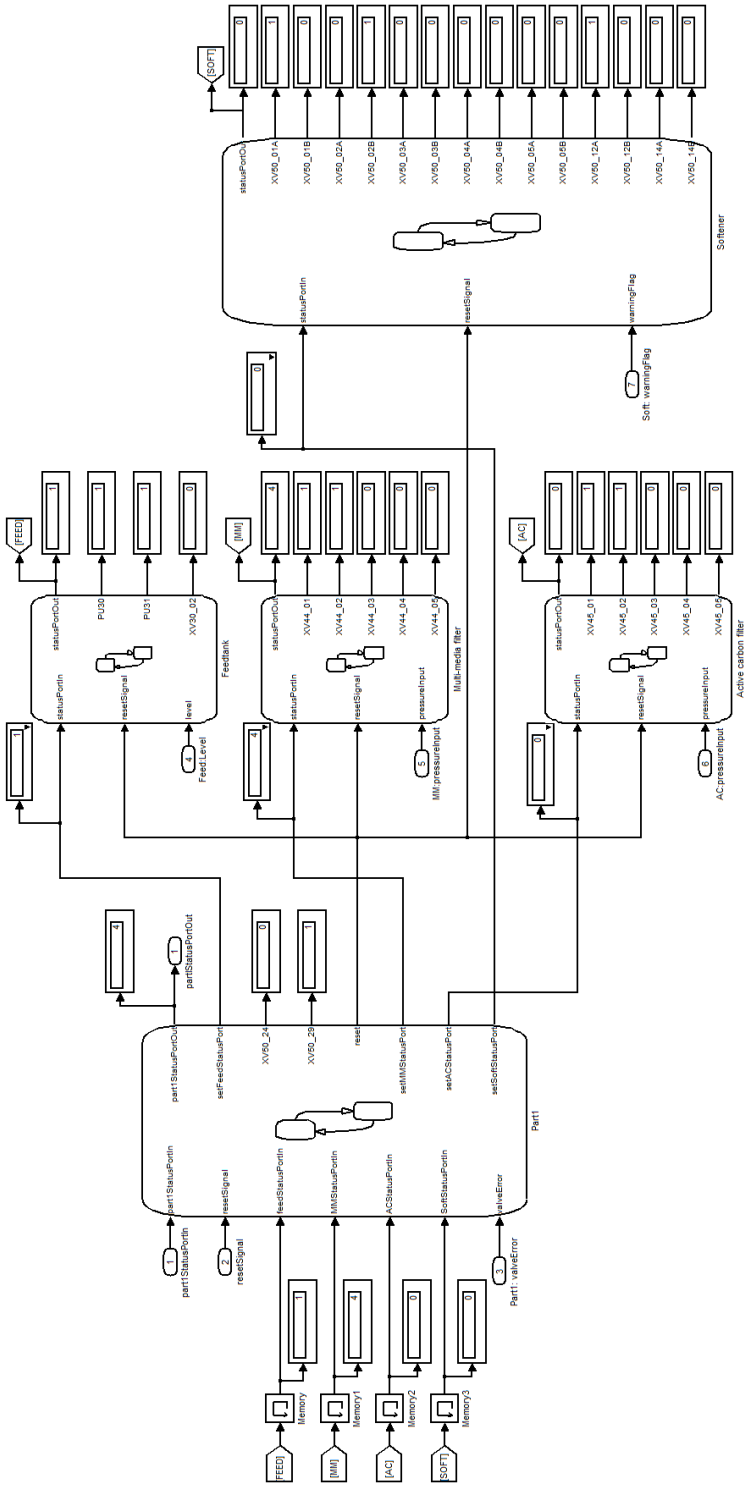

Figure B.1: Simulink setup for Part 1

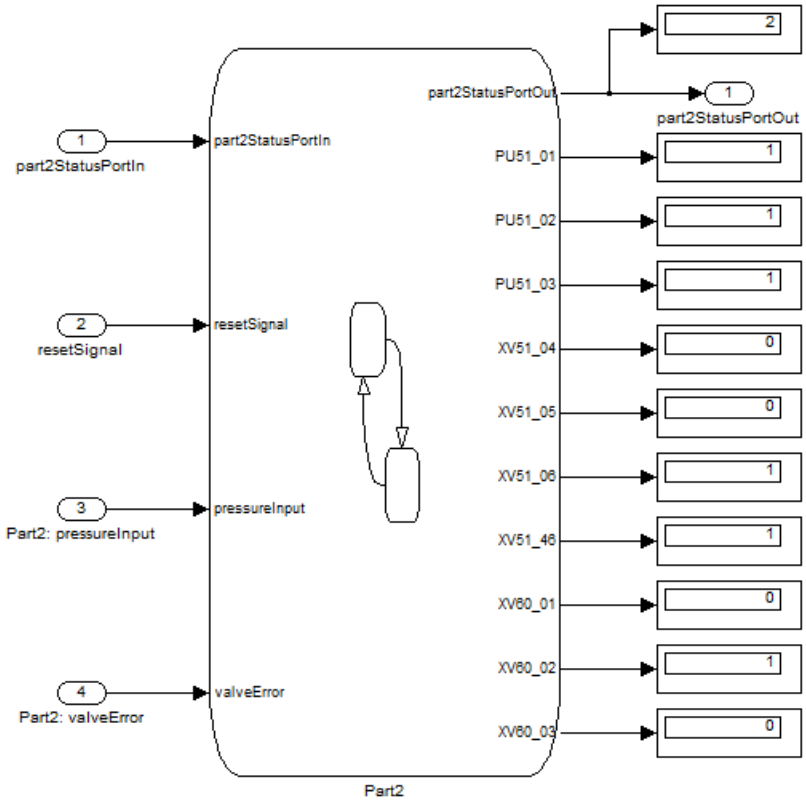

Figure B.2: Simulink setup for Part 2

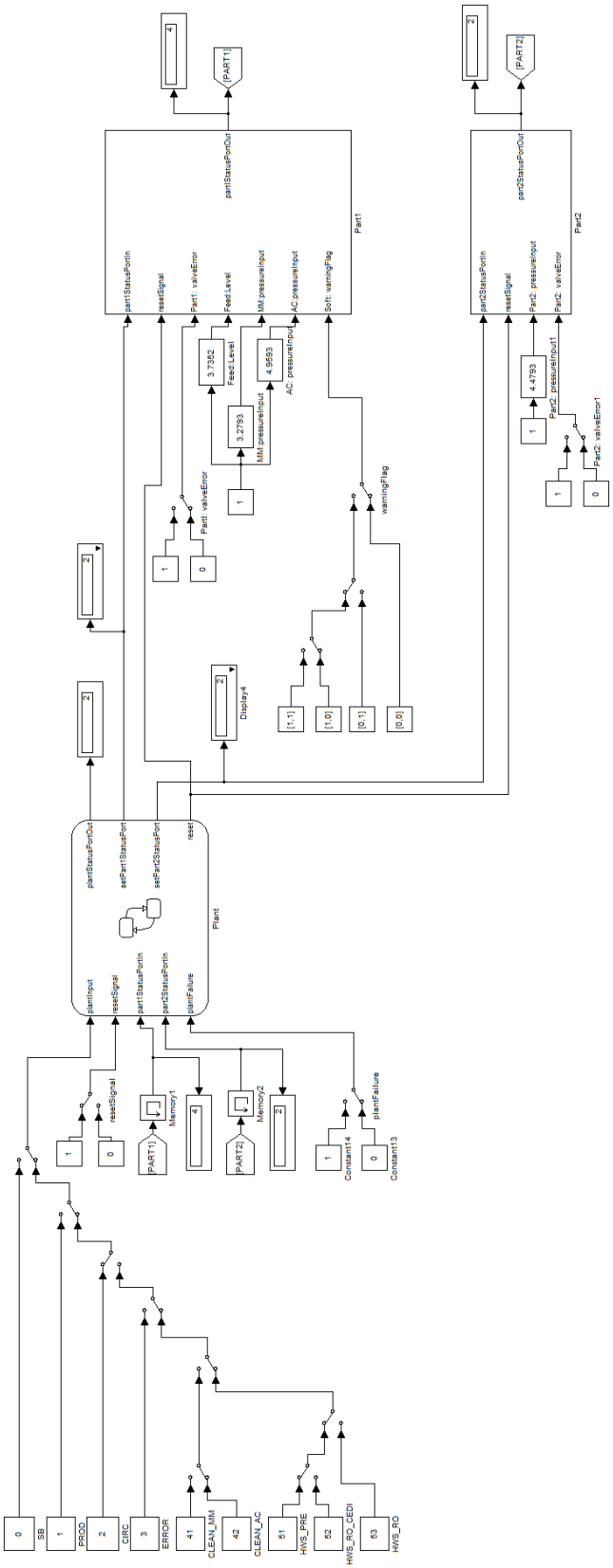

Figure B.3: Simulink setup for the Plant

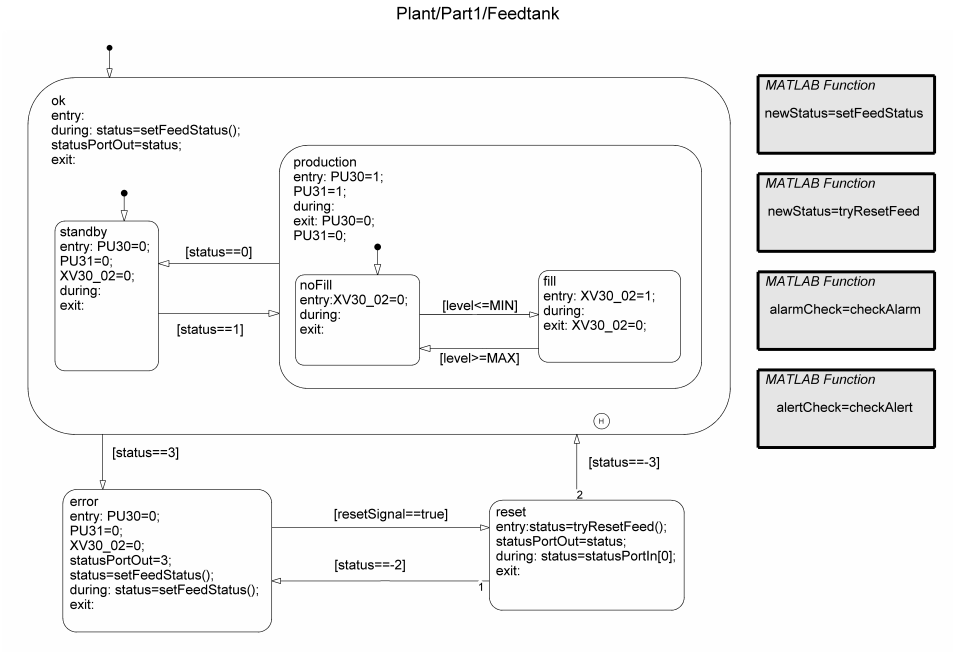

Figure B.4: State machine diagram for feedtank

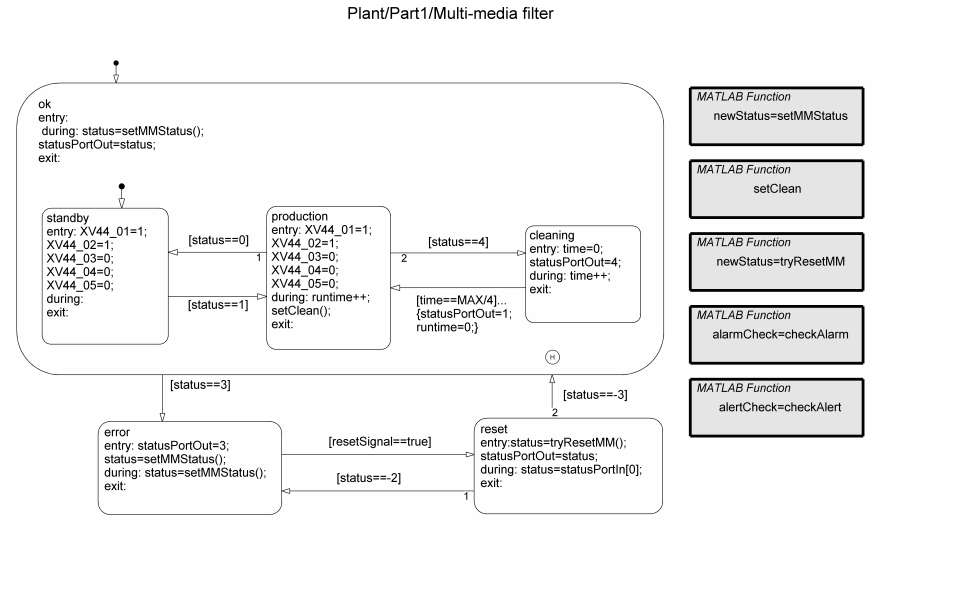

Figure B.5: State machine diagram for multimedia filter

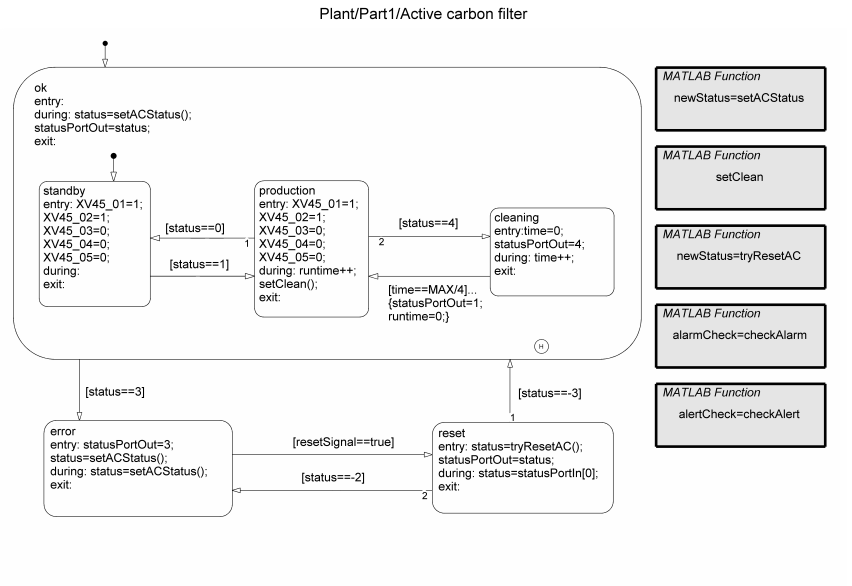

Figure B.6: State machine diagram for active carbon filter

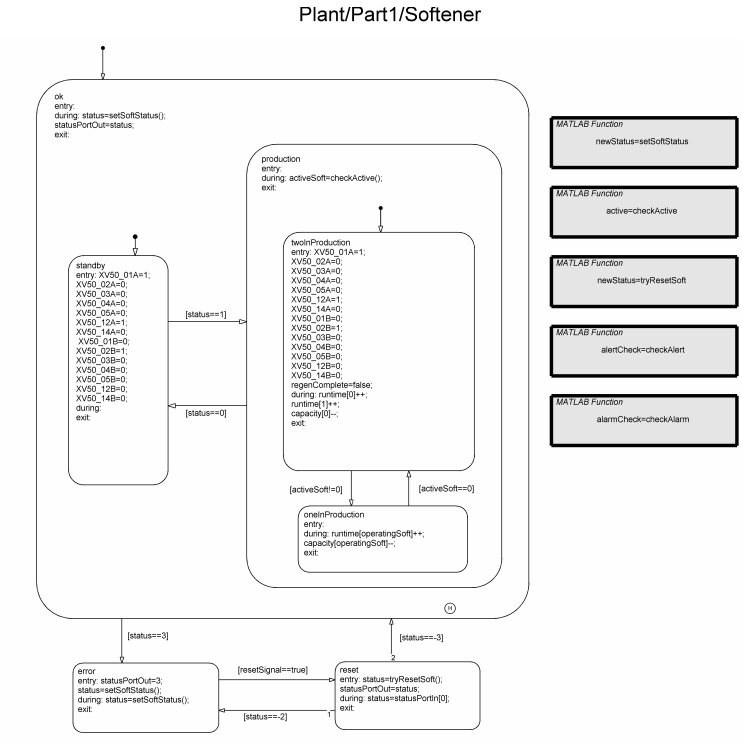

Figure B.7: State machine diagram for softener

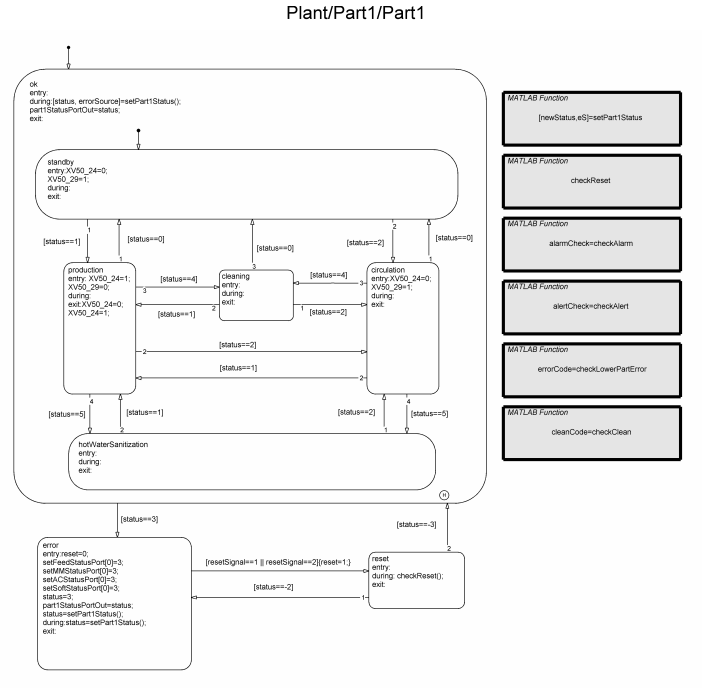

Figure B.8: State machine diagram for Part 1

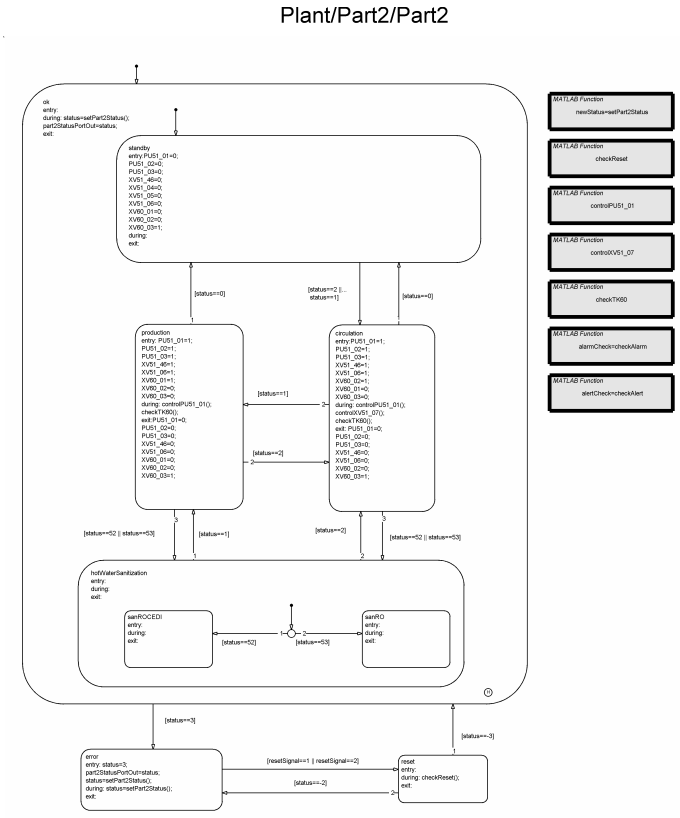

Figure B.9: State machine diagram for Part 2

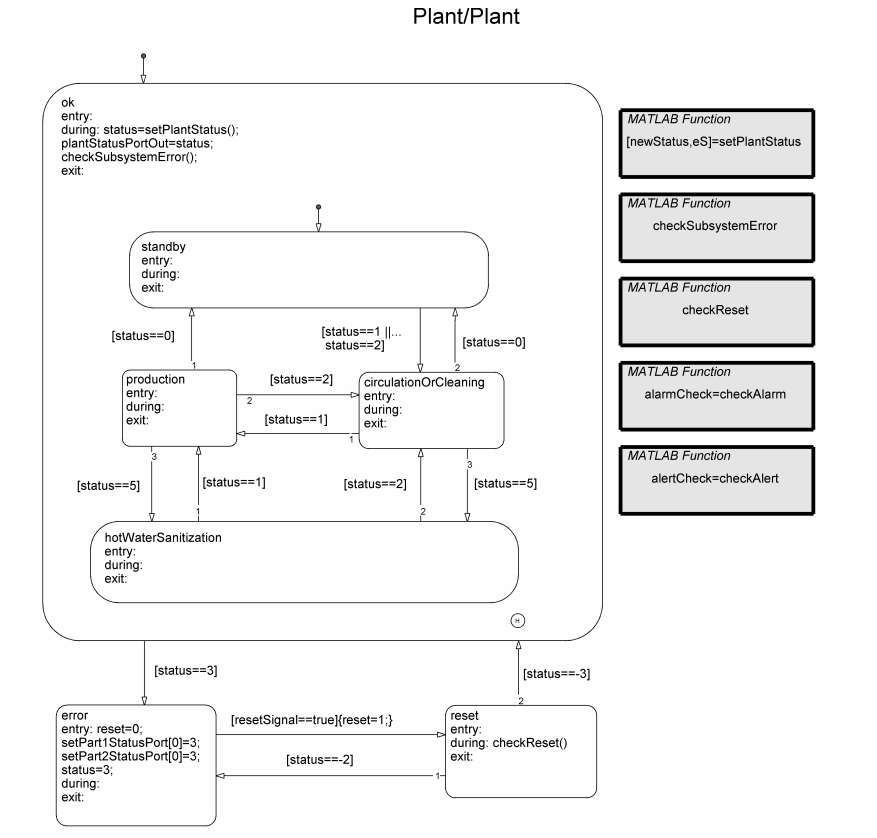

Figure B.10: State machine diagram for the plant

Number	Short state name	Formal state name
0	SB	Standby
1	PROD	Production
2	CIRC	Circulation
3	ERROR	Error
4	CLEAN	Cleaning
5	SAN	Hot water sanitization
-2	ERROR	Error (Only used in Matlab)
-3	OK	Ok
52	SAN_RO_CEDI	Hot water sanitization of RO and CEDI
53	SAN_RO	Hot water sanitization of RO

Table B.1: Table for mapping the status number used in Matlab to the enumerator used in SysML.

C Implementation

Signals used:

- STANDBY_S
- PRODUCTION_S
- ERROR_S
- OK_S
- NOFILL_S
- FILL_S
- CLEANING_S
- CLEANING_MM_S
- CLEANING_AC_S
- CIRCULATION_S
- CIRCULATION_PART1_S
- CIRCULATION_PART2_S
- RESET_S
- RESET_REQUEST_S
- REGEN_SOFTA_S
- REGEN_SOFTB_S
- HWS_S
- HWS_PART1_S
- HWS_RO_S
- HWS_PART2_S
- ONE_IN_PRODUCTION_S
- DONE_CLEANING_S
- DONE_REGEN_S
- DONE_HWS_S
- DURING_S