TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Alex Antonis 134634IABB

DESIGNING A CONTROL SYSTEM WITH SYSML AND SIMULINK

Bachelor's thesis

Supervisor: Tõnu Näks
MSc

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Alex Antonis 134634IABB

JUHTIMISSÜSTEEMI PROJEKTEERIMINE SYSML'I JA SIMULINK'IGA

Bakalaureuse töö

Juhendaja: Tõnu Näks
Tehnikateaduste magister

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Alex Antonis

20.05.2019

Abstract

The purpose of this thesis is to create a consistent example of designing a control system using SysML modeling language for describing the architecture and requirements of the system and converting the system to Simulink for control algorithm design.

The thesis consists of two parts. In the first part, a model of the control system is created based on the description of the system and in the second part, a realization of the created model of the system is developed using Simulink. The thesis also includes a brief overview of SysML and the tools and methodologies for binding SysML model with Simulink.

The example developed in the thesis shall be suitable for demonstrating control system design with SysML in the real-time software engineering course and to test a methodology developed by AdaCore for transferring the constraints from the SysML model to Simulink and their validity check by using GNATProver.

This thesis is written in English and is 34 pages long, including 6 chapters and 23 figures.

Annotatsioon

Juhtimissüsteemi projekteerimine SysML'i ja Simulink'iga

Käesoleva lõputöö eesmärgiks on koostada sidus näide juhtimissüsteemi projekteerimisest kasutades SysML modelleerimiskeelt süsteemi arhitektuuri ja nõuete kirjeldamiseks ning süsteemi teisendamisest Simulink'i juhtimisalgoritmi konstrueerimiseks.

Töö jaguneb kaheks osaks. Esimeses osas luuakse valitud juhtimissüsteemi kirjelduste põhjal süsteemi mudel, mis sisaldab süsteemi struktuuri ja voogusid süsteemi erinevate osade vahel, funktsionaalseid ja ohutusnõudeid ning juhitava objekti olekumuutjate seoseid parameetritena. Lõputöö teises osas luuakse loodud süsteemi mudeli põhjal realisatsioon Simulinkis, mis hõlmab juhtimisalgoritmi ja mudelis toodud kitsenduste realisatsiooni sünkroonsete monitoridena. Lisaks sisaldab töö ka lühiülevaadet SysML-st ja vahendidest ning metoodikatest SysML-i mudeli sidumiseks Simulink-iga.

Töö tulemus peab võimaldama kasutada koostatud juhtimisüsteemi projekteerimise näidet SysML modelleerimiskeele õpetamiseks reaalaja-tarkavaratehnika kursuses ja testida firmas AdaCore väljatöötatavat metoodikat SysML mudelis toodud kitsenduste ülekandmiseks Simulinki ja nende kehtivuse kontrolli GNATProver'i abil.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 34 leheküljel, 6 peatükki ja 23 joonist.

List of abbreviations and terms

SysML	*Systems Modeling Language*
UML	*Unified Modeling Language*
MBSE	*Model-based Systems Engineering*
UML2	*Unified Modeling Language 2*
IBM	*International Business Machines*
FTMS	*Fuel Thermal Management System*
OMG	*Object Management Group*

Table of contents

List of figures

1 Introduction

As the technology develops more and more complex, so does the systems in different kinds of fields. In order to deal with the growing complexity of designing and managing these systems, a continuous development of different approaches are being made for dealing with these challenges. One of these approaches is called systems engineering.

Systems engineering is a multidisciplinary approach to develop balanced system solutions in response to diverse stakeholder needs [1]. These system solutions, which include the management and technical aspects, must be developed in a way that all stakeholder needs are satisfied with minimal risks that may affect the project. As systems engineering approach is getting more widely used in many fields, a need to develop more enhanced methodologies also includes the need to teach more and more people about this approach.

The purpose of this thesis is to create a design example of a small control system which can later be used for teaching systems modeling with SysML[1]. Another aim of this work is to use it for testing a new methodology for transferring the constraints from the SysML model to Simulink[2] in order to check the validity of these constraints.

This work consists of four parts. First part focuses on giving a brief overview of SysML with an example of a small system model. Next part includes modeling a selected system using this language, which includes modeling of its architecture, requirements, constraints and interconnectivity between its parts. When this is done, the next part gives a brief overview of different methods for binding together SysML and Simulink. The last part focuses on transforming the system model into Simulink in order to model the behaviour of the system and constraints of the system as observers.

[1] https://www.omg.org/spec/SysML

[2] https://www.mathworks.com/products/simulink.html

2 Overview of the methodology

2.1 SysML

SysML is a graphical modeling language used for systems engineering and is intended to support the requirements, analysis, specification, design, verification and validation of systems. SysML enables the application of Model-based systems engineering (MBSE) approach, which supports above-mentioned procedures of these systems. This overview is a short summary of book by S. Fridenthal, A. Moore and R. Steiner [1].

SysML includes nine diagrams, which are used to describe different aspects of a system. The taxonomy of these diagrams is shown in Figure 1 and it includes:

- Package diagram – the organization of a model divided into packages
- Requirement diagram – requirements and relationships with other requirements and model elements.
- Activity diagram – behaviour of the system and its parts.
- Sequence diagram – message-based behaviour of the system and its parts.
- State machine diagram – behaviour of a system's entity and its transition between different states.
- Use case diagram – functionality of a system and how the users interact with it.
- Block definition diagram – the system's structure divided into blocks and including their properties, operations and relationships.
- Internal block diagram – interconnection between system's internal components.
- Parametric diagram – constraints on property values and the relationship among system properties.

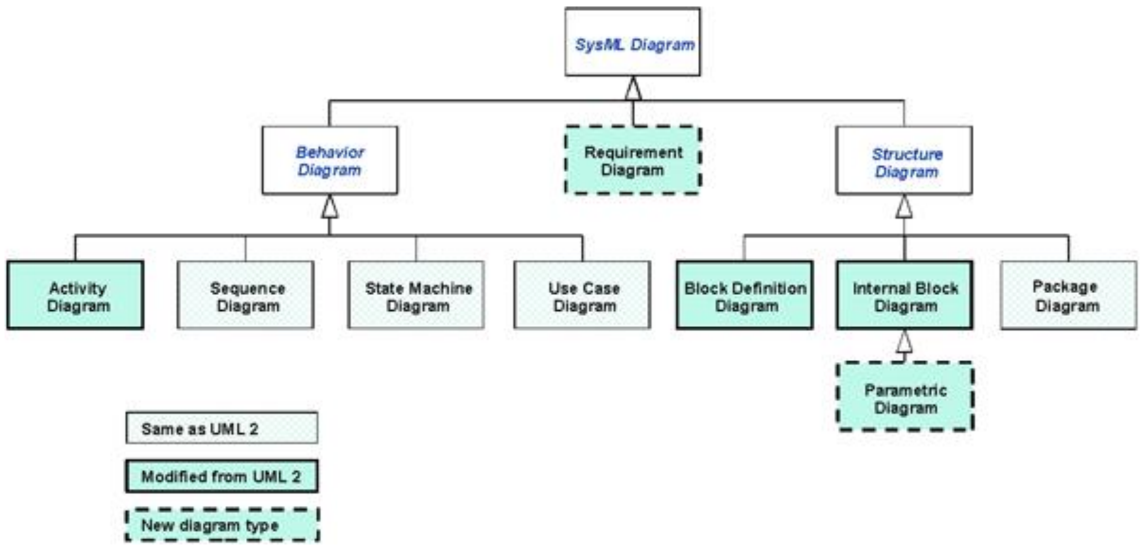


Figure 1. Taxonomy of SysML diagrams [2]

When starting to build a system model the first thing to do is to create the top level package diagram which describes the organization of the model. This diagram may include packages for requirements, behaviour, structure and parametrics. Additional packages may be included in this diagram if necessary.

As the package diagram is created, the modeler can now start to fill these packages with corresponding type of diagrams. The modeler could start with creating the requirements diagram which should be contained in the requirements package. After that, behaviour package can be populated with activity, sequence, state machine and use case diagrams, structure package with block definition and internal block diagrams and parametrics package with all parametric diagrams. There is no strict rule for in which order the diagrams must be modeled as it depends on several factors, including the available information, the preferred MBSE method used or user’s personal preference on how to organize the work. Also, there is no strict rule that one diagram must be fully completed before starting to model another one as the diagrams can complement each other with shared elements, making the whole modeling process iterative.

2.2 Example of a system model developed with SysML

In this paragraph a small system model is being developed using SysML language and a modeling tool called Enterprise Architect[1] by Sparx Systems. The model contains a small subset of diagrams that can be designed with this language and also contains a simulation which is being executed using OpenModelica[2] modeling environment. The model organization and naming of certain blocks are inspired by [1].

2.2.1 Model organization

As it was mentioned above, when starting to build a system model the first thing to do is to create a top level package diagram for describing the organization of the model. In Figure 2 there is shown a package diagram containing packages for describing the model's architecture. Currently, the diagram consists of three packages: *Parametrics*, *Structure* and *Requirements*. Each of the package contains corresponding types of diagrams and entities which are shown in packages. There is also a *SysMLSimConfiguration* artifact represented in the diagram (as well as in the *Parametrics* package) that contains the simulation of a block definition diagram that focuses on the analysis of the system. When the initial organization of the model is done the next step is to start designing the structure of the system.

[1] https://sparxsystems.com/

[2] https://openmodelica.org/

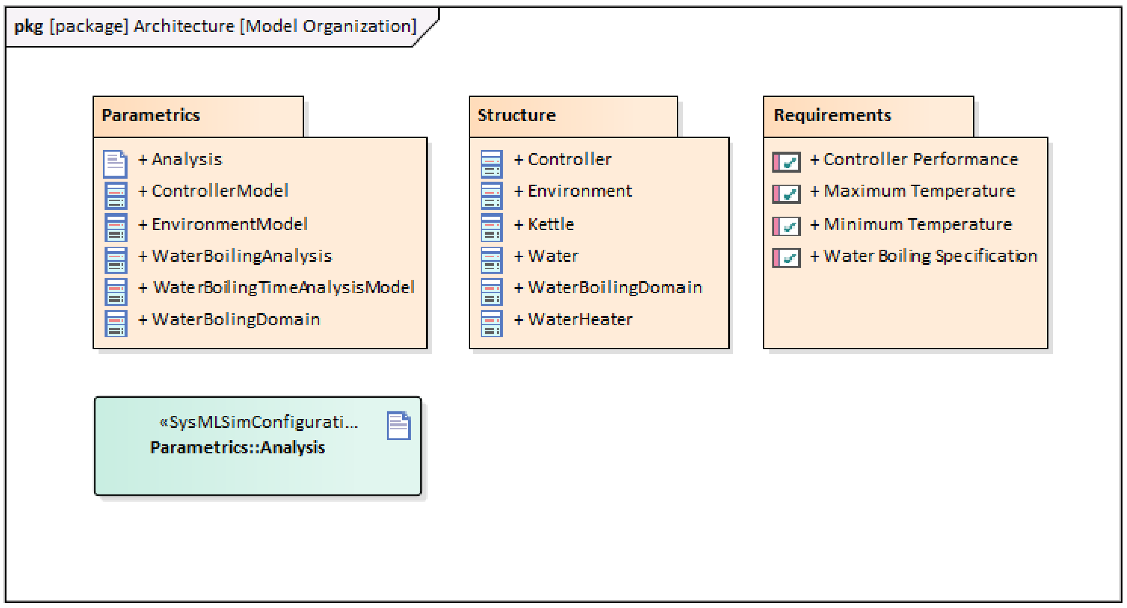


Figure 2. Package diagram for the organization of the model

2.2.2 Structure of the system

The system being modeled depicts a process of boiling water in an electric kettle. Figure 3 features a block definition diagram describing the top level structure of the system with all its parts and their relationships. The *WaterBoilingDomain* block is the top-level block that provides the context for the system. The system consists of *Kettle*, which is a part of the system and is connected with *WaterBoilingDomain* block via a line with a black diamond symbol indicating a part association. *Kettle* block depicts a kettle or any other type of container that contains water and has two properties: *radius* and *height*. *Water* block is connected with *Kettle* block also via part association line as the water is contained in the container, therefore is a part of it. *Water* has a property called *amount* as there is always a certain amount of water being boiled. *Water* is connected to *Environment* block via regular connector as there is no whole-part relationship between them. *Environment* includes properties called *tempEnv* (temperature of the environment surrounding the boiling system) and *heatCoefficient* (The proportionality constant between the heat flux and the thermodynamic driving force for the flow of heat). The *WaterBoilingDomain* block is also associated with a *WaterHeater* block which depicts a hot plate that applies certain amount of power for boiling the water. It has a property called *powerMax* which stands for the maximum amount of power it is capable of applying. *WaterHeater* in turn is attached with a *Controller* block which regulates the power being used.

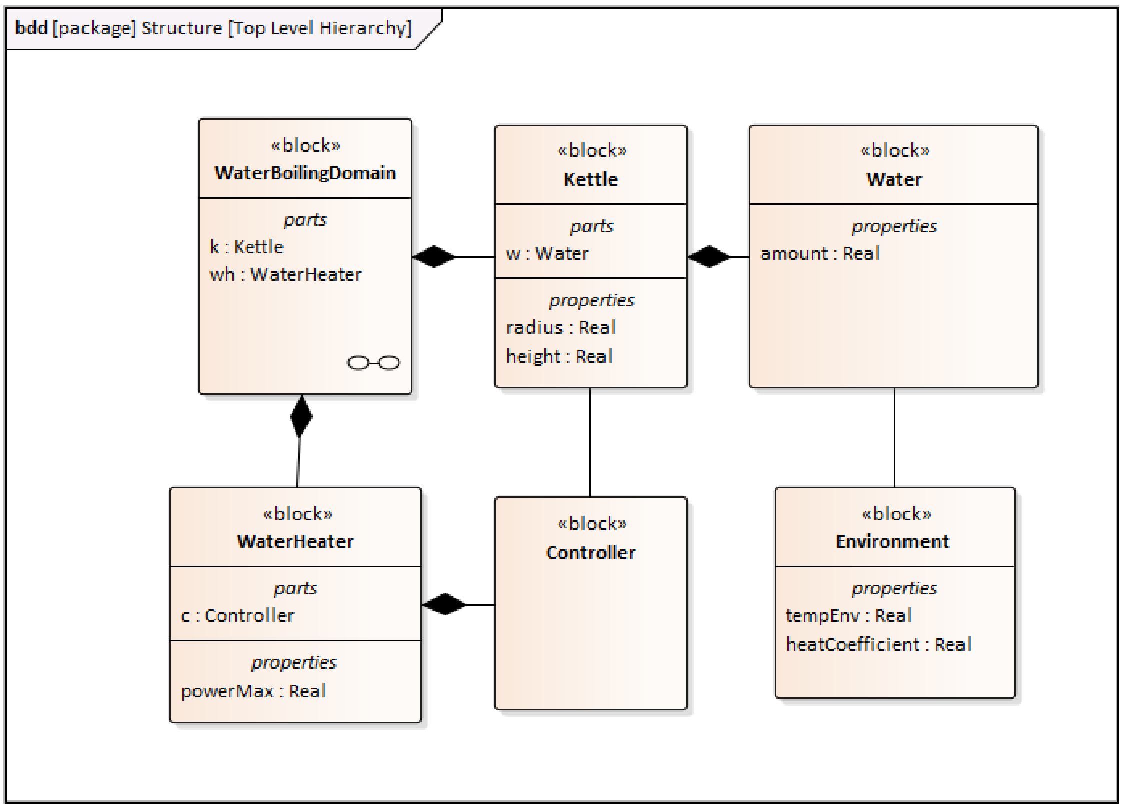


Figure 3. Block definition diagram for structure of the system

Figure 4 represents an internal block diagram that shows how the parts of the *WaterBoilingDomain* block from the last figure are connected to each other. The dashed lines with an arrowhead connecting the parts represent *Item Flow* which means that a physical (e.g. heat) or a non-physical (e.g. information, signal) is sent between the small squares called *ports*. The ports indicate the interfaces of the parts through which the communication takes place. In this particular case, the *Controller* receives the information of the current temperature of the water via *Temperature signal* sent from the *Water*. *Temp out* port indicates that the signal is sent out from it and *temp in* indicates that the signal is sent into it. After receiving the information about the temperature, the *Controller* next sends the *WaterHeater* a *Switch signal* for switching on the heater. The *WaterHeater* now sends the heat out into the *Water* and simultaneously the heat also starts to radiate into the *Environment*. The *Controller* monitors the *Water* as it reaches the boiling point and when the point is reached, a signal is sent to the *Controller* and from there to the *WaterHeater* to switch off the heating.

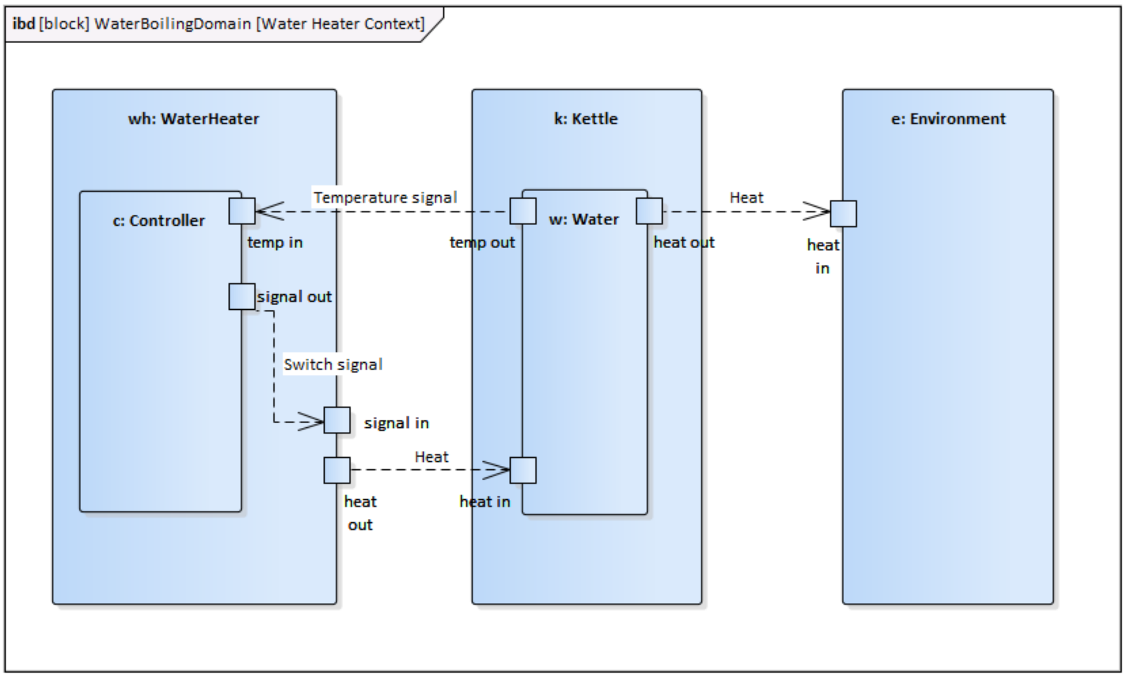


Figure 4. Internal block diagram for water heater context

2.2.3 Safety requirements

As the initial structure of the system is designed, the modeler can now start creating a *requirement diagram* describing the requirements that are set for the system. In Figure 5, the *Water Boiling Specification* indicates a top-level requirement which in turn contains more requirements. The requirements connected to each other via a line with the crosshairs symbol indicates that one requirement contains other requirement(s) and can be seen as a parent-child relationship. The next level requirement called *Controller Performance* depicts the requirements that are precisely imposed on the behaviour of the *Controller*. The *Controller Performance* requirement in turn is composed of two requirements indicating the minimum and maximum temperature of the water between which the *Controller* is allowed to regulate the power for the *WaterHeater*. The both requirements, *Maximum Temperature* and *Minimum Temperature*, contain its *id* and *text* which specifies the requirement.

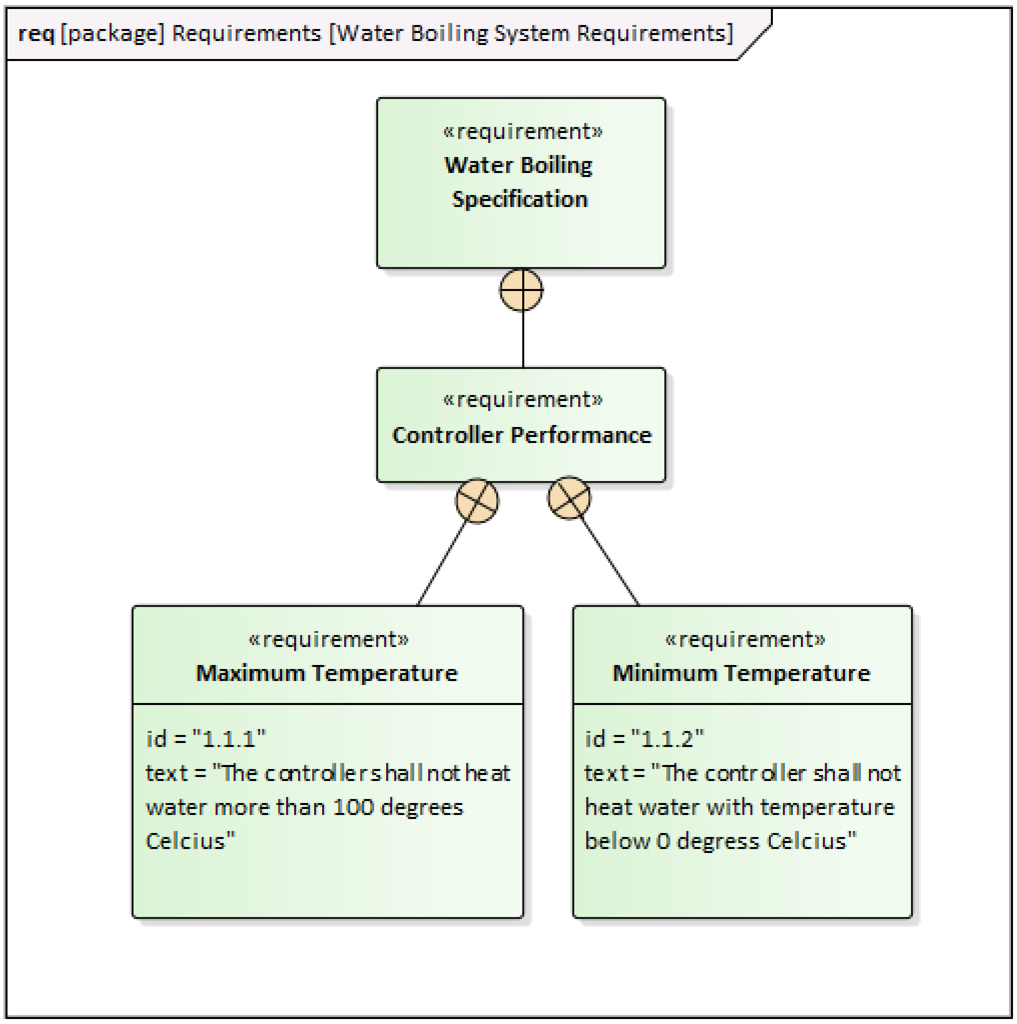

req [package] Requirements [Water Boiling System Requirements]
«requirement»
Water Boiling Specification
«requirement»
Controller Performance
«requirement»
Maximum Temperature
id = "1.1.1"
text = "The controller shall not heat water more than 100 degrees Celcius"
«requirement»
Minimum Temperature
id = "1.1.2"
text = "The controller shall not heat water with temperature below 0 degress Celcius"

Figure 5. Requirements diagram for controller performance

2.2.4 Analysis

Behaviour analysis is also another important part of system modeling as there is a need to verify if the model satisfies the requirements with specified parameters and to find out the best solutions. To carry out the analysis of the system, there are two diagrams which are needed to perform this activity: *block definition diagram* and *parametric diagram*. The parametric diagram contents are later verified in a simulation tool (OpenModelica in our case).

As it was mentioned above, the first thing to do is to create a *block definition diagram* containing all the constraints with equations and parameters. Figure 6 depicts such kind of diagram for the *Analysis Context*. The main block in this diagram is *WaterBolingAnalysis* block which is used for the analysis and it comprises of three constraint blocks: *EnvironmentModel*, *ControllerModel* and *WaterBoilingTimeAnalysisModel*. Each of the constraint blocks contain equations and parameters that are used in them [3][4]. All the parameters from the constraint blocks are also contained in the *WaterBolingAnalysis* block as properties. There is also a block called *WaterBolingDomain* shown on diagram which is a reference from the *Top Level Hierarchy* diagram from Figure 3 to show that it is the subject of the analysis.

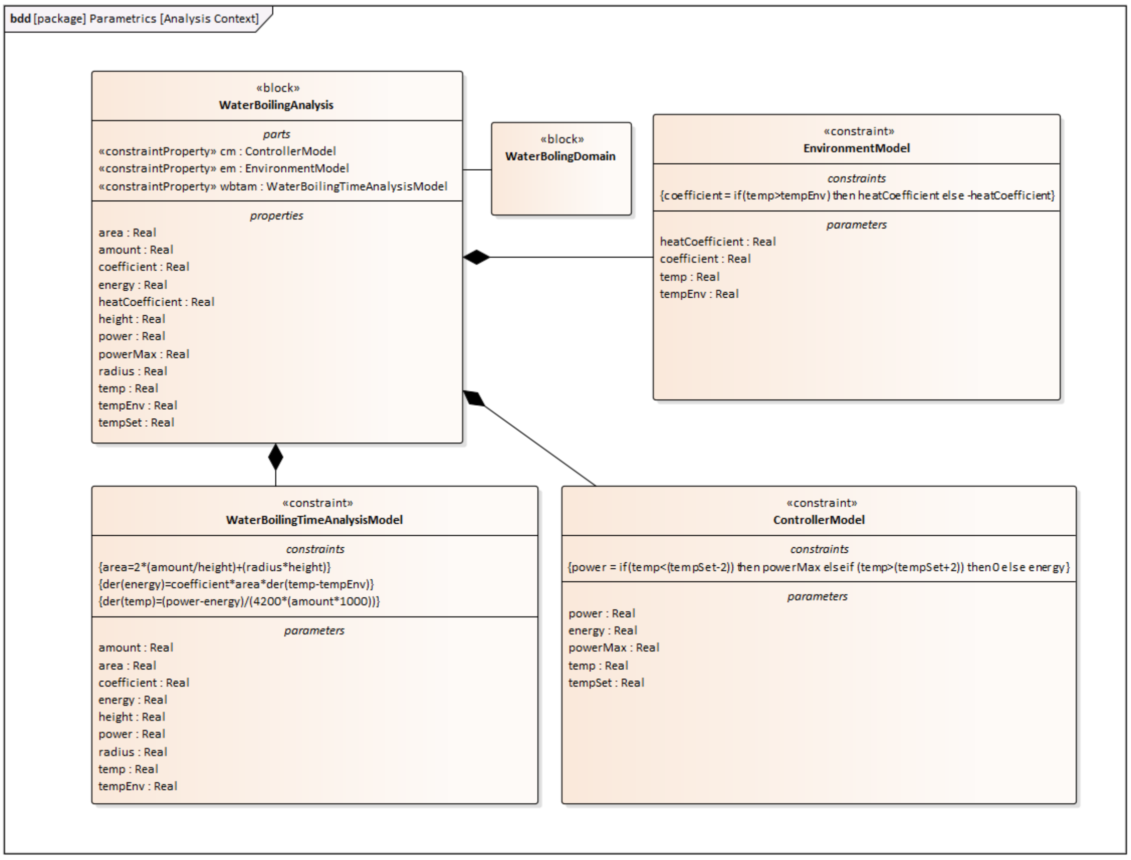


Figure 6. Block definition diagram for analysis context

The *parametric diagram* created based on the *Analysis Context* diagram from Figure 6 is represented in Figure 7. The diagram consists of three *ConstraintProperties* which are defined in block definition diagram from Figure 6 and contain corresponding equations. Each of the *ConstraintProperty* is connected with all the *properties* which are taken from *WaterBoilingAnalysis* block in Figure 6 and are used in corresponding equations. There are three properties that are connected with a directed line from one constraint to another: *power*, *temp* and *coefficient*. The reason for this is that it shows that the value of the property is taken from the solved equation of one constraint and is passed to another constraint to solve its equations.

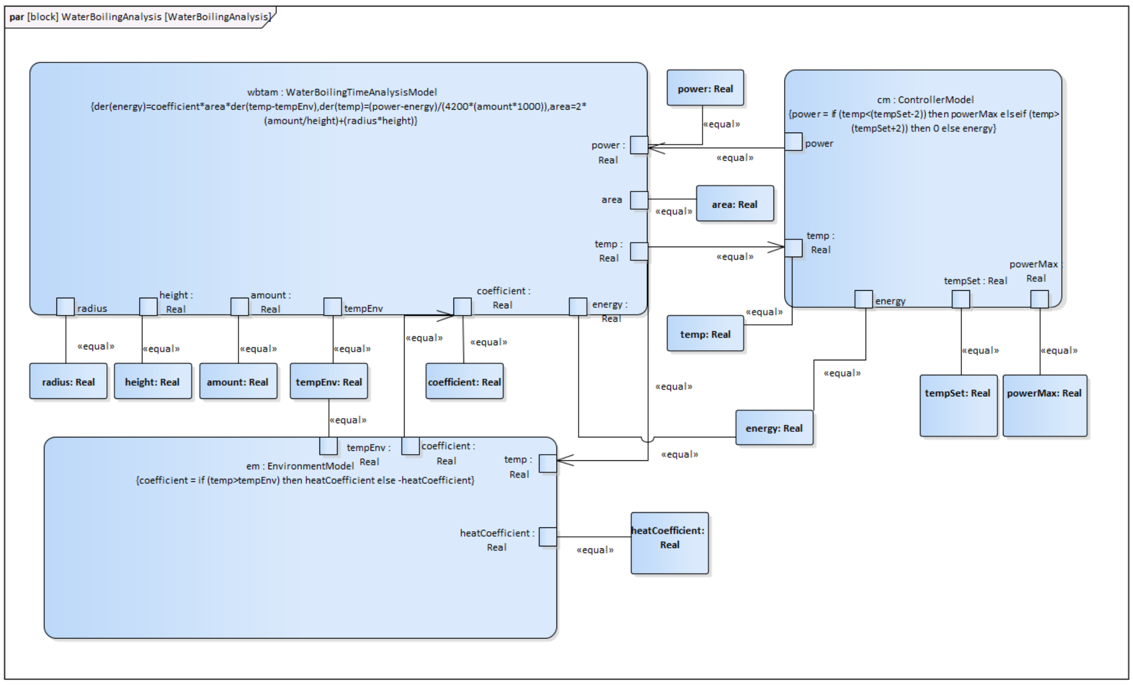
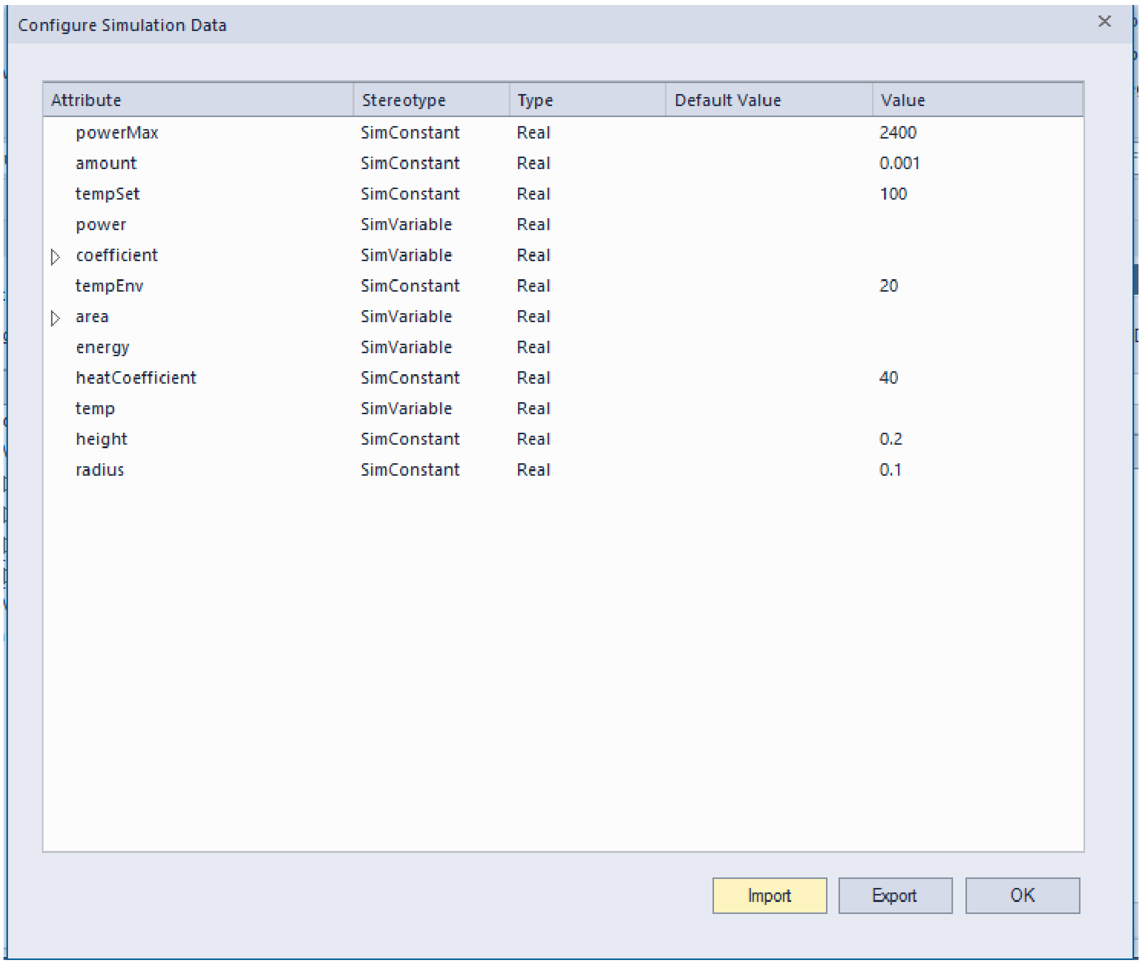


Figure 7. Parametric diagram for analysis

As the parametric diagram is done, now the simulation of it can be carried out by using a separate simulation tool to analyse the results of these equations. In this case, a modeling environment OpenModelica is used for the analysis.

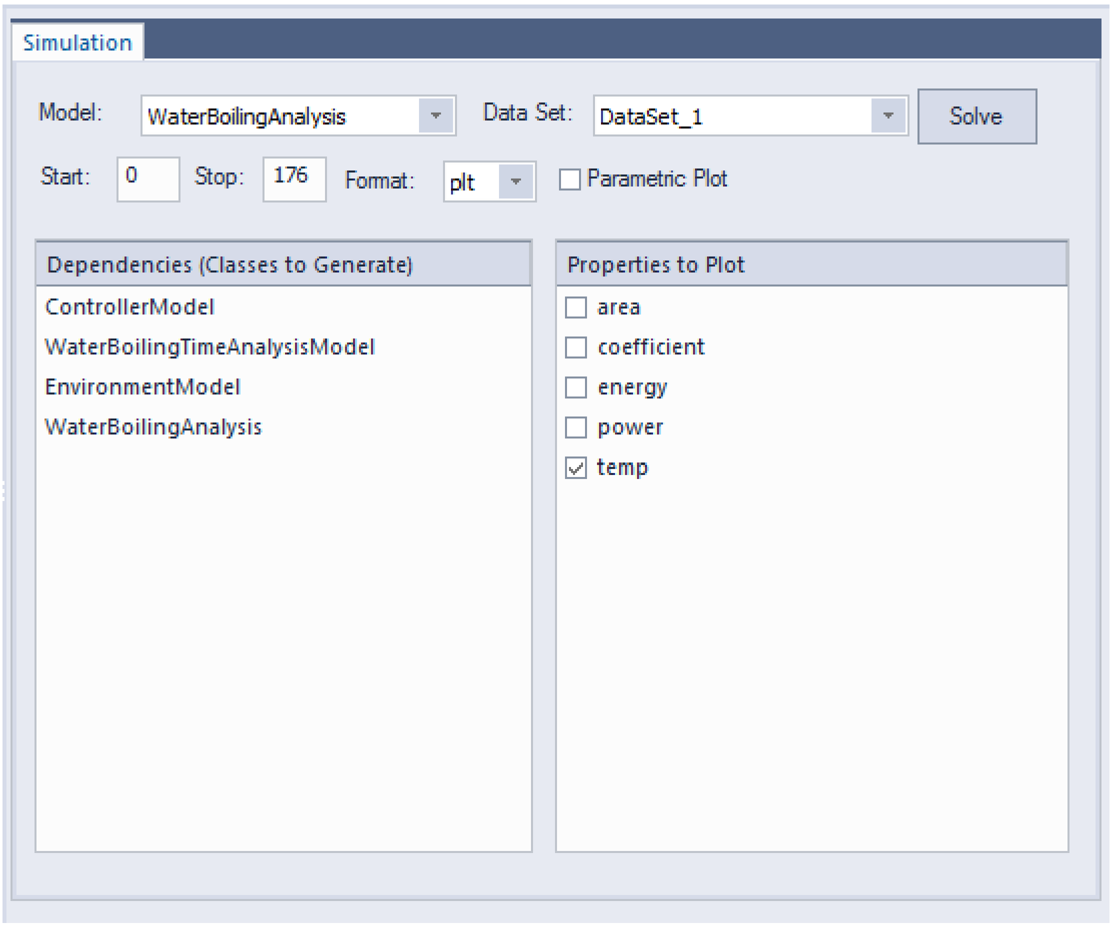
In order to simulate the model, a *SysMLSimConfiguration* artifact needs to be opened. The artifact contains all the blocks, constraints, properties and dependencies between all of them. There is also a need to select which properties are constants and which are variables that need to be plotted in the simulation. Figure 8 represents a list of the properties with their type and initial value [3][5].



Attribute	Stereotype	Type	Default Value	Value
powerMax	SimConstant	Real		2400
amount	SimConstant	Real		0.001
tempSet	SimConstant	Real		100
power	SimVariable	Real		
▷ coefficient	SimVariable	Real		
tempEnv	SimConstant	Real		20
▷ area	SimVariable	Real		
energy	SimVariable	Real		
heatCoefficient	SimConstant	Real		40
temp	SimVariable	Real		
height	SimConstant	Real		0.2
radius	SimConstant	Real		0.1

Figure 8. Simulation data configuration

As there is an interest to know how the temperature of the water changes at a time during the heating process, the property that needs to be plotted is a property named *temp* (Figure 9). The result of the simulation is shown in Figure 10 where x-axis indicates time, y-axis temperature and the red line indicates the change of temperature in time.


Simulation

Model: WaterBoilingAnalysis Data Set: DataSet_1 Solve

Start: 0 Stop: 176 Format: plt ☐ Parametric Plot

Dependencies (Classes to Generate)

- ControllerModel
- WaterBoilingTimeAnalysisModel
- EnvironmentModel
- WaterBoilingAnalysis

Properties to Plot

- ☐ area
- ☐ coefficient
- ☐ energy
- ☐ power
- ☑ temp

Figure 9. Selected properties to plot in the simulation

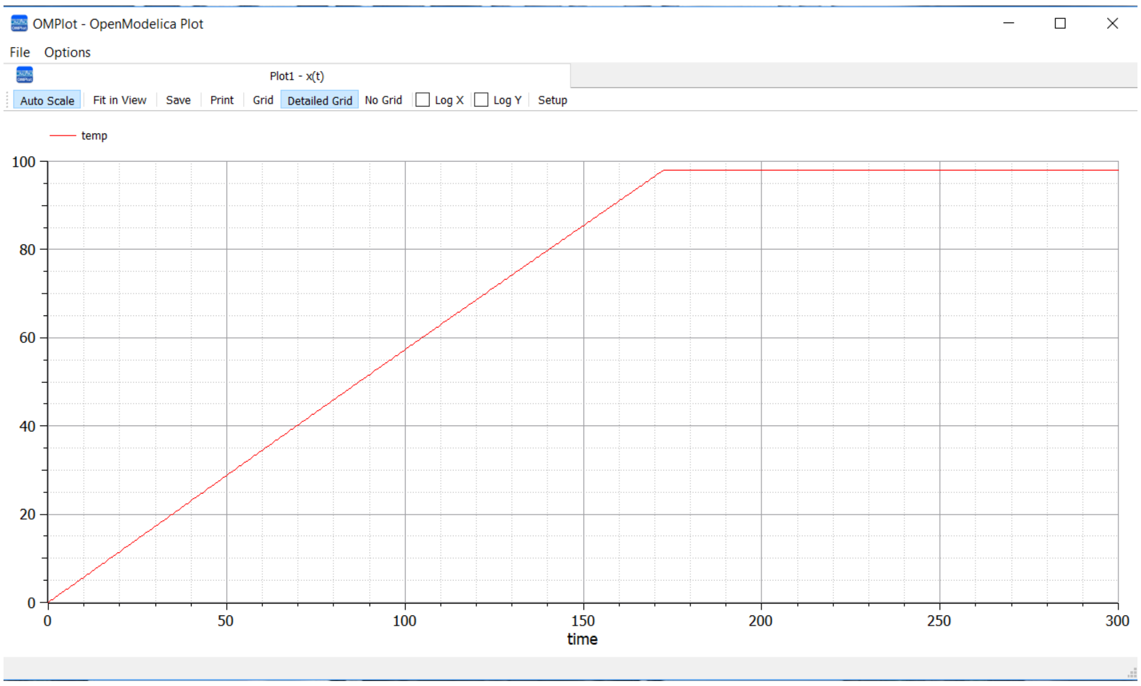


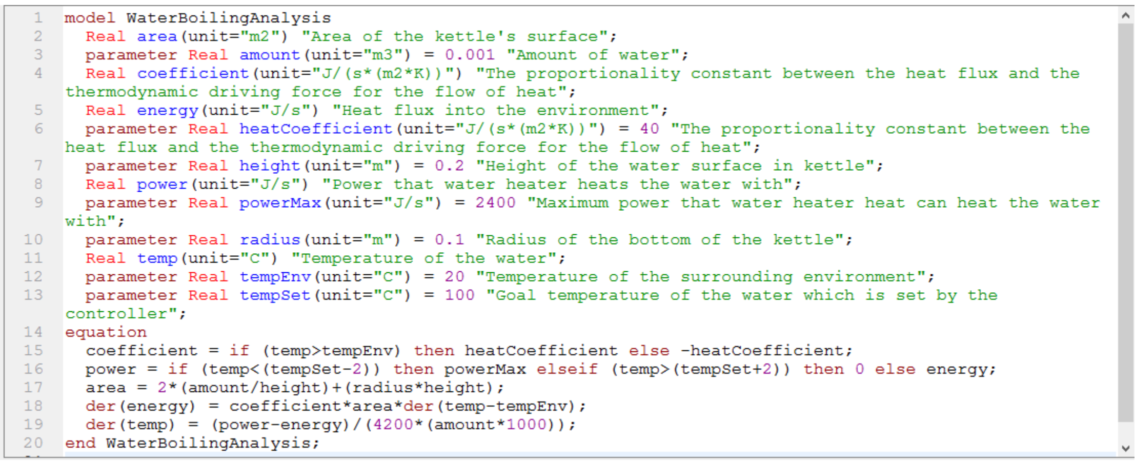
Figure 10. Simulation result

This first example represented the way of how to first model the system with diagrams and then run the simulation. As it was shown, firstly the diagrams describing the system in different aspects were created with Enterprise Architect and then the simulation was run with OpenModelica based on one of the diagrams, a parametric diagram. The next example represents an opposite way to create a system model as in this case the modeling starts with OpenModelica.

2.3 Example of the same system modeled with Modelica

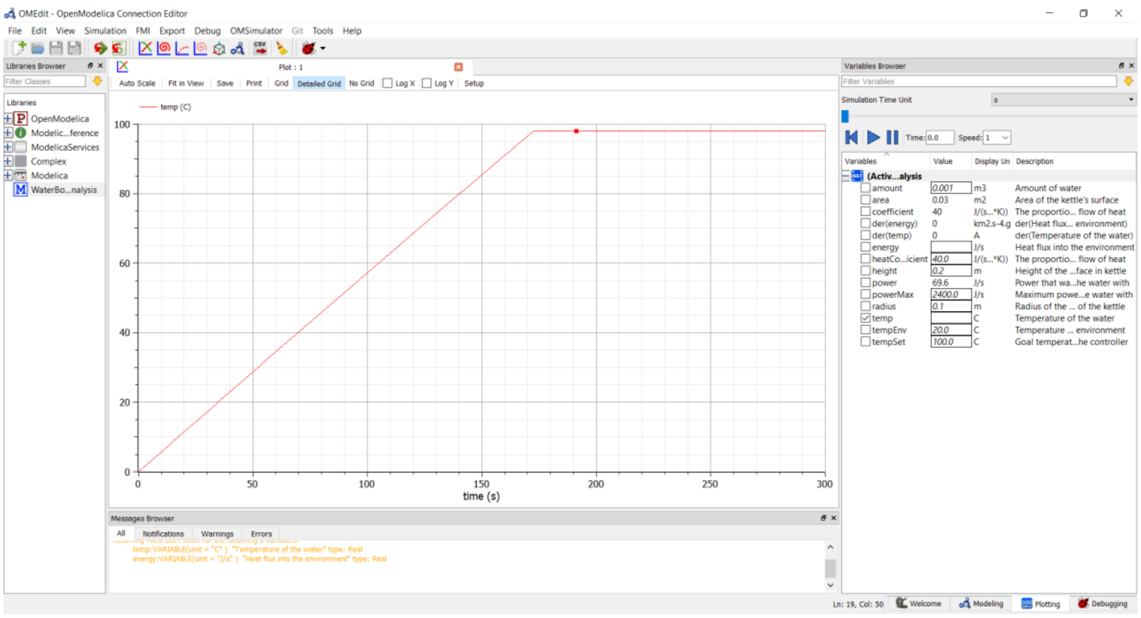
In this section, the same water boiling system model that was represented in the last chapter, is now being developed with Modelica modeling language in OpenModelica modeling environment. Given the simplicity of equation system we chose the flat-model approach as described in chapter 12 of [6].

To start off with this example, the first thing to do is to create a new Modelica class with a name *WaterBoilingAnalysis*. As the model is now created, the model needs to be populated with all the same properties and equations that were used in the previous example. In Figure 11 there is shown a model named *WaterBoilingAnalysis* populated with properties including their unit and brief description in quotation marks. The properties with *parameter* in front of them indicate constant whereas properties without it indicate variables [3][5]. Next to the properties there are all the equations that will be used in the simulation [3][4]. The result of the simulation is depicted in Figure 12 and it is same as in Figure 10.



```
model WaterBoilingAnalysis
  Real area(unit="m2") "Area of the kettle's surface";
  parameter Real amount(unit="m3") = 0.001 "Amount of water";
  Real coefficient(unit="J/(s*(m2*K))") "The proportionality constant between the heat flux and the
thermodynamic driving force for the flow of heat";
  Real energy(unit="J/s") "Heat flux into the environment";
  parameter Real heatCoefficient(unit="J/(s*(m2*K))") = 40 "The proportionality constant between the
heat flux and the thermodynamic driving force for the flow of heat";
  parameter Real height(unit="m") = 0.2 "Height of the water surface in kettle";
  Real power(unit="J/s") "Power that water heater heats the water with";
  parameter Real powerMax(unit="J/s") = 2400 "Maximum power that water heater heat can heat the water
with";
  parameter Real radius(unit="m") = 0.1 "Radius of the bottom of the kettle";
  Real temp(unit="C") "Temperature of the water";
  parameter Real tempEnv(unit="C") = 20 "Temperature of the surrounding environment";
  parameter Real tempSet(unit="C") = 100 "Goal temperature of the water which is set by the
controller";
equation
  coefficient = if (temp>tempEnv) then heatCoefficient else -heatCoefficient;
  power = if (temp<(tempSet-2)) then powerMax elseif (temp>(tempSet+2)) then 0 else energy;
  area = 2*(amount/height)+(radius*height);
  der(energy) = coefficient*area*der(temp-tempEnv);
  der(temp) = (power-energy)/(4200*(amount*1000));
end WaterBoilingAnalysis;
```

Figure 11. Properties and equations in Modelica


OMEdit - OpenModelica Connection Editor
File Edit View Simulation FMI Export Debug OMSimulator Git Tools Help
Libraries Browser
Libraries
OpenModelica
Modelic...ference
ModelicaServices
Complex
Modelica
WaterBo...nalysis
Plot : 1
Auto Scale Fit in View Save Print Grid Detailed Grid No Grid Log X Log Y Setup
temp (C)
time (s)
Variables Browser
Filter Variables
Simulation Time Unit
Time: 0.0
Speed: 1
Messages Browser
All Notifications Warnings Errors
Ln: 19, Col: 50
Welcome Modeling Plotting Debugging

Variables	Value	Display Un	Description
(Activ...alysis			
☐ amount	0.001	m3	Amount of water
☐ area	0.03	m2	Area of the kettle's surface
☐ coefficient	40	J/(s...*K))	The proportio... flow of heat
☐ der(energy)	0	km2.s-4.g	der(Heat flux... environment)
☐ der(temp)	0	A	der(Temperature of the water)
☐ energy		J/s	Heat flux into the environment
☐ heatCo...icient	40.0	J/(s...*K))	The proportio... flow of heat
☐ height	0.2	m	Height of the ...face in kettle
☐ power	69.6	J/s	Power that wa...he water with
☐ powerMax	2400.0	J/s	Maximum powe...e water with
☐ radius	0.1	m	Radius of the ... of the kettle
✓ temp		C	Temperature of the water
☐ tempEnv	20.0	C	Temperature ... environment
☐ tempSet	100.0	C	Goal temperat...he controller

```
    temp:VARIABLE(unit = "C" )  "Temperature of the water" type: Real
    energy:VARIABLE(unit = "J/s" )  "Heat flux into the environment" type: Real
```

Figure 12. Simulation result

3 The case study

In this chapter, a model for a tactical aircraft's dual tank fuel thermal management system is being designed and described through different diagrams. These diagrams are designed to describe the system from different aspects and include diagrams focused on system's structure and interconnectivity between its parts, functional and safety requirements and the relationships between the status variables. The model of the system is created based on the article written by N. Jain and B.M. Hencey [7] which describes how the system works and is included with the schema of the system, equations and parameters with some initial values and graphs based on different analyses. The goal of [7] is to compare efficiency of different control algorithms. Our intention here is simply to show how SysML model can be used to describe the controlled system and controller so we ignore the cost calculation part in the paper.

3.1 Model organization

In Figure 13, a top level package diagram called *Model organization* is depicted to describe the organization of the model. The diagram includes packages named *Parametrics* and *Structure*, which in turn include corresponding diagrams and entities.

There is also a *FuelThermalManagementSystemSimulation* artifact represented in the diagram (as well as in the *Parametrics* package) that contains the simulation. The diagrams contained in these packages are shown and described in next subchapters.

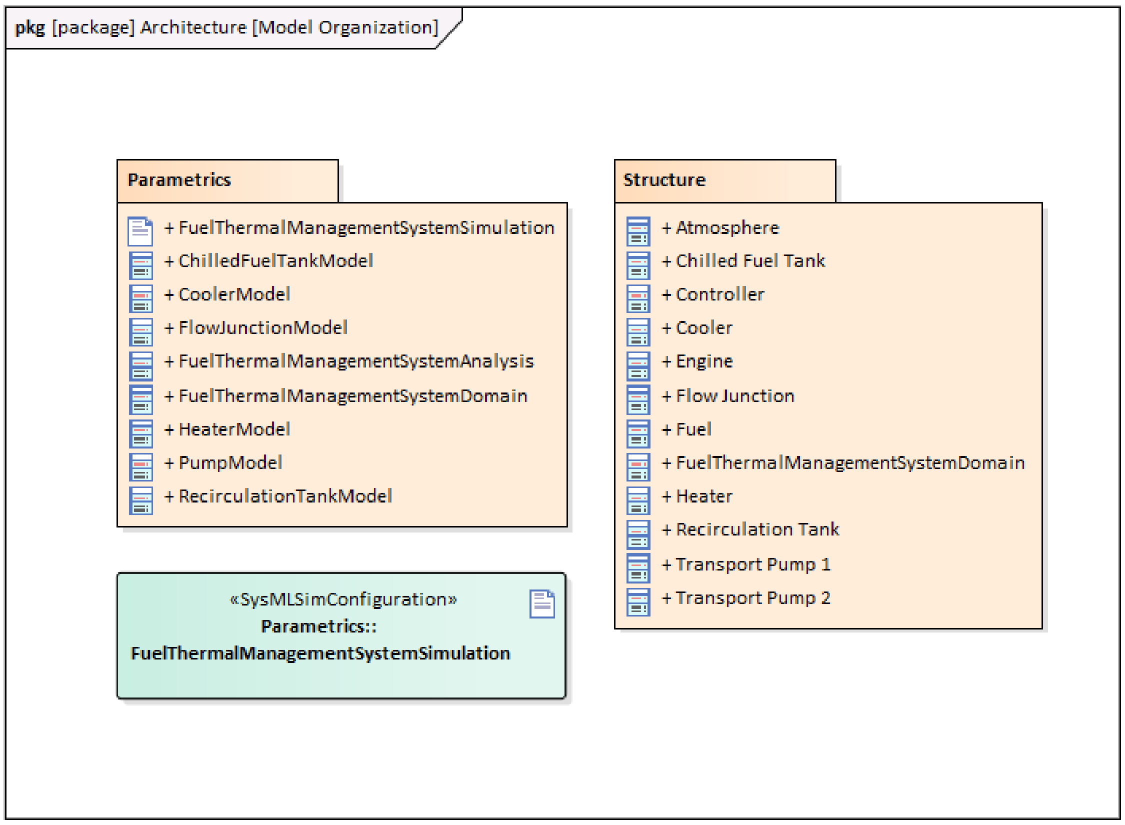


Figure 13. The organization of the model

3.2 Structure of the system

A block definition diagram called *Top Level Hierarchy* is depicted in Figure 14 to describe the structure of the fuel thermal management system. The *FuelThermalManagementSystemDomain* block is the top-level block that provides the context for the system. The top-level block is connected with ten other blocks that are parts of the system. Each partitioning block is included with properties that are related to the block and whose values are constant. The units and meanings of the properties are included in figures in Appendix 1 and Appendix 2. The interconnection between the system's parts is described in the next section.

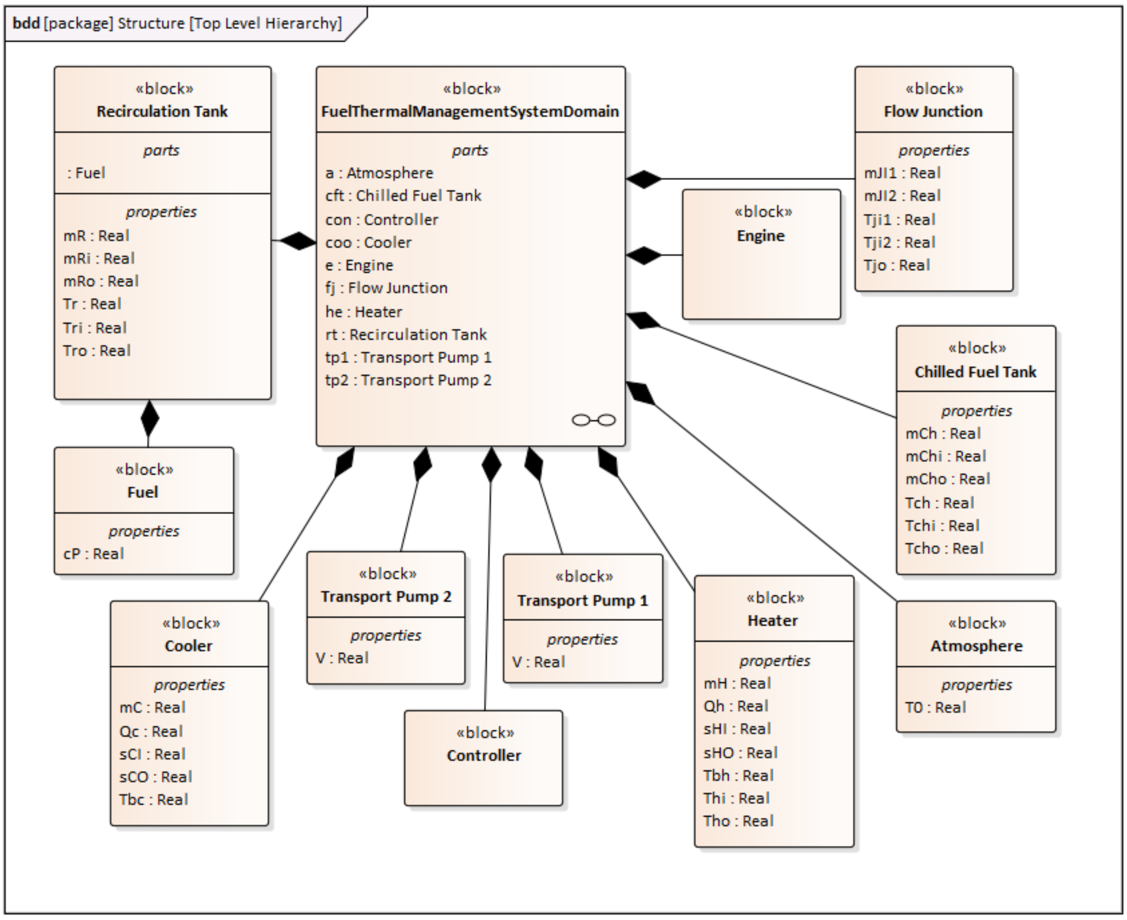


Figure 14. Structure of the system

3.3 Flows between the system's parts

In Figure 15, an internal block diagram called *FuelThermalManagementSystemDomain* is depicted to describe how the parts of the system interact with each other. Both *Recirculation Tank* and *Chilled Fuel Tank* are contained with a certain amount of fuel and chilled fuel, respectively. A certain amount of fuel exits the *Recirculation Tank* and flows into the *Flow Junction*. The same process occurs with the *Chilled Fuel Tank*, however, the chilled fuel first flows through the *Transport Pump 1*. Here, the *Controller* receives information about the fuel's temperature and sends a switch signal in order to regulate the fuel mass flow rate if needed. From the first pump, the fuel next reaches the *Flow Junction*. In the *Flow Junction*, the both fuels are mixed together and the mixed fuel next reaches the *Heater*. In the *Heater*, the temperature of the fuel is being risen with the waste heat that is absorbed from the other flight critical subsystems (mainly the engine) of the aircraft that are not part of this particular system being described. After the heating process, the fuel reaches the engine, where a certain amount of fuel is inserted into the engine and the excessive fuel flows into the *Transport Pump 2*. In here, the *Controller* receives information about the fuel's temperature and sends a switch signal if needed to regulate the mass flow rate through the pump. From the second pump, the fuel reaches the *Cooler*, which is used to cool down the fuel back to its initial temperature. As the cooling takes place, the excessive heat in the *Cooler* is released into the *Atmosphere*. The cooled fuel finally flows back into the *Recirculation Tank* and the same process continues as a loop.

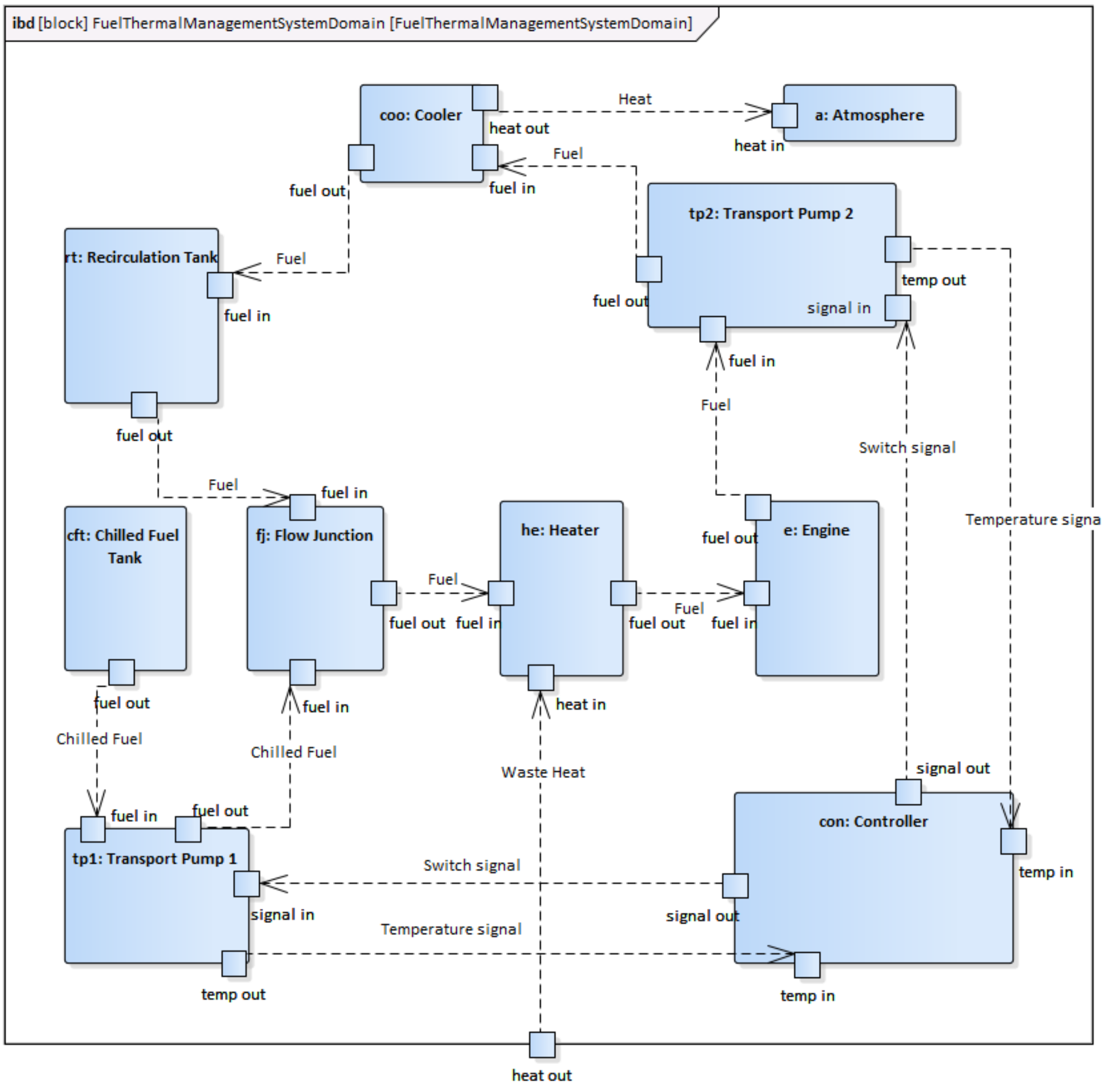


Figure 15. Interconnection between the system's components

3.4 Safety requirements as constraint blocks

Safety requirements for the controller are described in Figure 16 [8]. The diagram includes five safety requirements setting expectations for the controller (connected with <<satisfy>> relationship) and five constraint blocks (<<designConstraint>>) that depict constraints as equations (connected with <<refine>> relationship). These equations show what properties are inputs for the controller with their comparison and how the result of the comparison is solved. In all of the constraint blocks in this diagram, it is shown, that when the input properties satisfy the requirement, then the fuel mass flow rate (*Mf*) remains as it is, however, if the properties exceed the limits, the fuel mass flow rate equals 0 as the controller shuts the pumps that the fuel flows through. The implementation of these requirements in Simulink is later shown in paragraph 5.

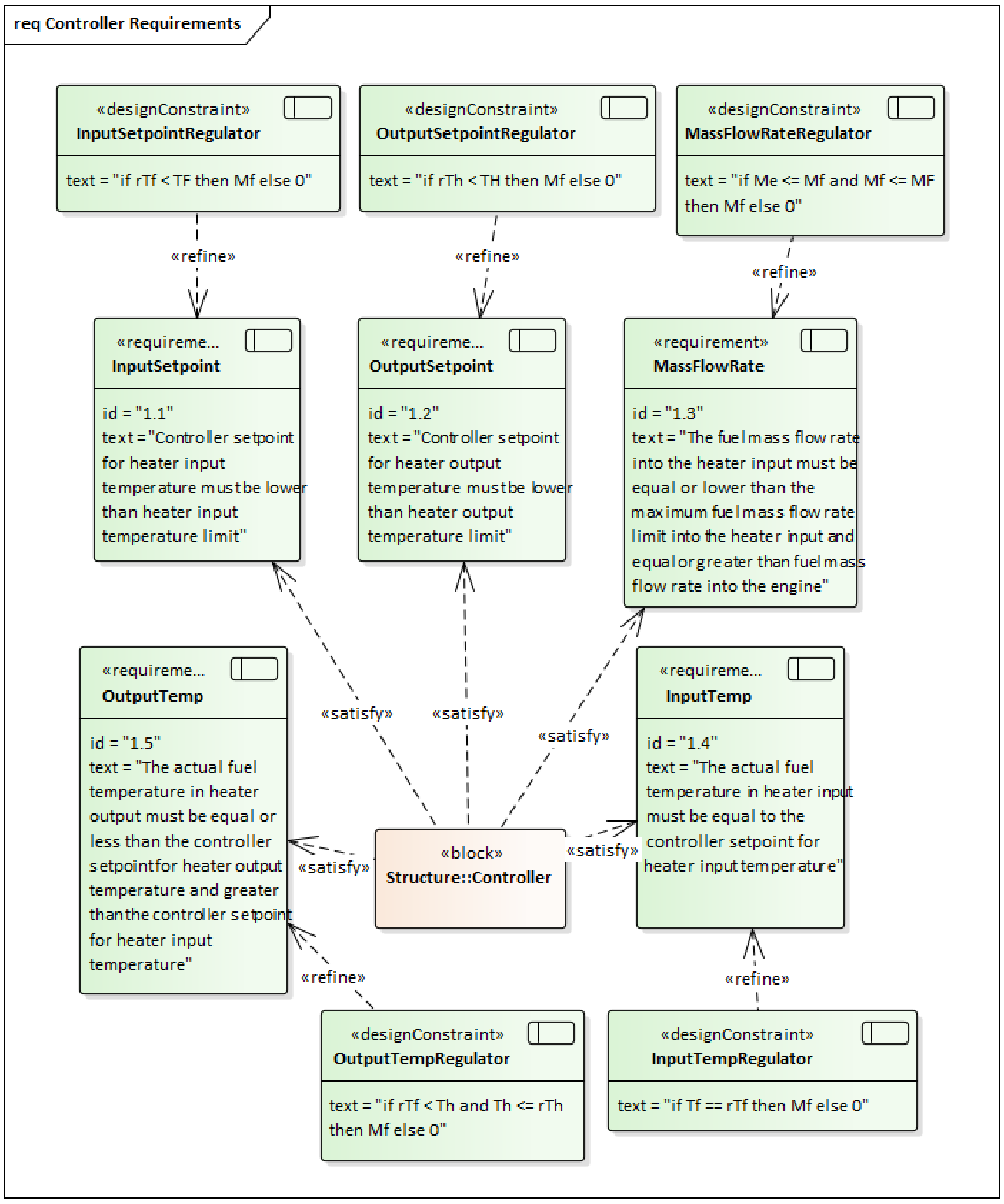

req Controller Requirements
«designConstraint» InputSetpointRegulator
text = "if rTf < TF then Mf else 0"
«designConstraint» OutputSetpointRegulator
text = "if rTh < TH then Mf else 0"
«designConstraint» MassFlowRateRegulator
text = "if Me <= Mf and Mf <= MF then Mf else 0"
«refine»
«refine»
«refine»
«requireme... InputSetpoint
id = "1.1"
text = "Controller setpoint for heater input temperature must be lower than heater input temperature limit"
«requireme... OutputSetpoint
id = "1.2"
text = "Controller setpoint for heater output temperature must be lower than heater output temperature limit"
«requirement» MassFlowRate
id = "1.3"
text = "The fuel mass flow rate into the heater input must be equal or lower than the maximum fuel mass flow rate limit into the heater input and equal or greater than fuel mass flow rate into the engine"
«requireme... OutputTemp
id = "1.5"
text = "The actual fuel temperature in heater output must be equal or less than the controller setpointfor heater output temperature and greater than the controller setpoint for heater input temperature"
«satisfy»
«satisfy»
«satisfy»
«satisfy»
«satisfy»
«block» Structure::Controller
«requireme... InputTemp
id = "1.4"
text = "The actual fuel temperature in heater input must be equal to the controller setpoint for heater input temperature"
«refine»
«refine»
«designConstraint» OutputTempRegulator
text = "if rTf < Th and Th <= rTh then Mf else 0"
«designConstraint» InputTempRegulator
text = "if Tf == rTf then Mf else 0"

Figure 16. Safety requirements as constraint blocks

3.5 Relationships between the status variables of the controlled object as parameters

A block definition diagram in Figure 17 contains two regular blocks and six constraint blocks that are used to carry out a simulation to analyse the rate of exergy destruction for the different parts of the system. The main block in this diagram is the

FuelThermalManagementSystemAnalysis block which comprises of all the properties from the linking constraint blocks that are used for the simulation. A *FuelThermalManagementSystemDomain* block is used here as a reference block from the *Top Level Hierarchy* block to show the subject of the analysis. As it was mentioned above, the analysis consists of six constraint blocks, each containing equations that are used to calculate certain measures of corresponding parts. As all of the six parts contain an equation to calculate the rate of exergy destruction per time, the *PumpModel* also contains three additional equations that are needed for the exergy destruction equation to be solved.

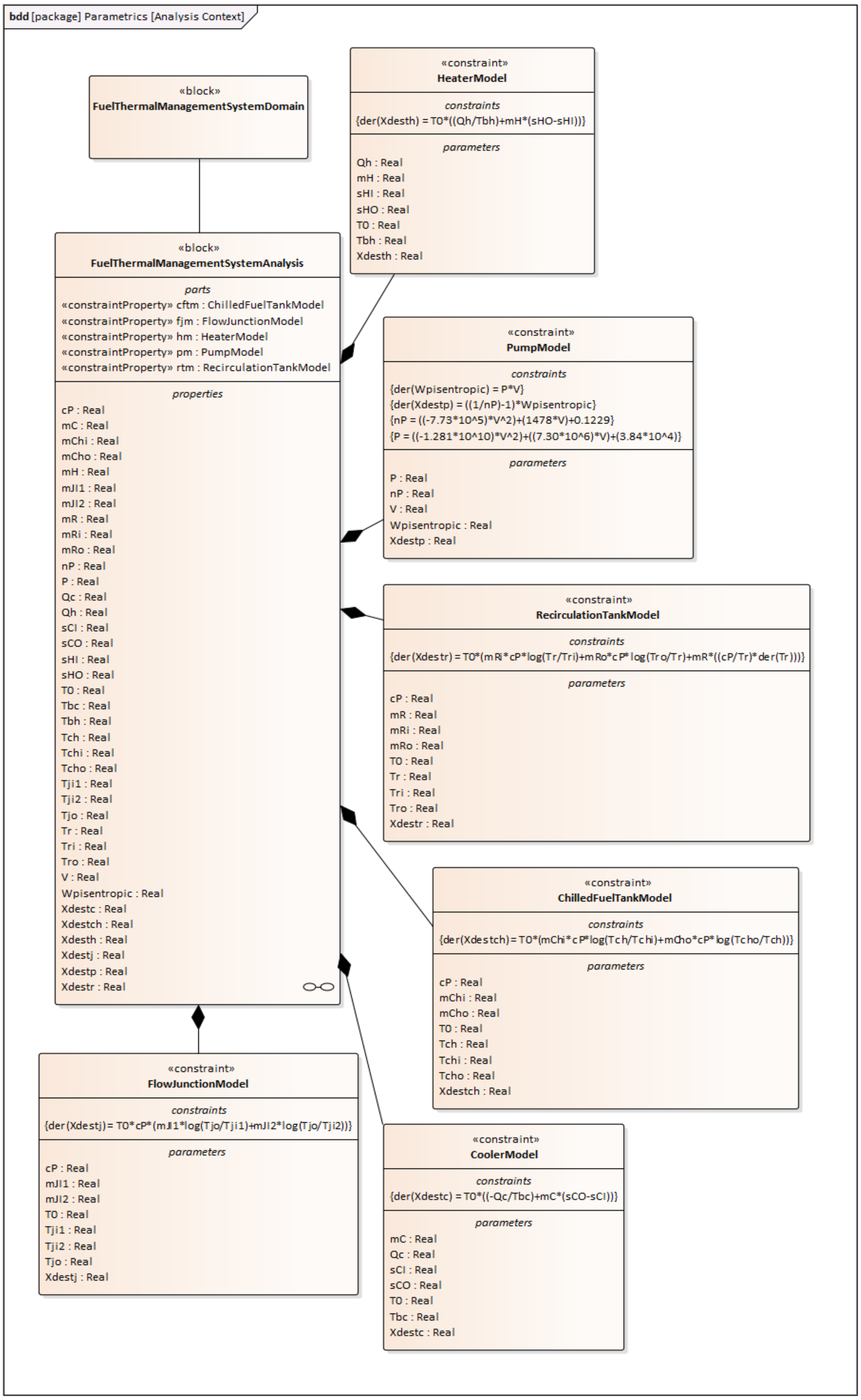


Figure 17. Analysis context

The parametric diagram on Figure 18 is created based on the *Analysis Context* diagram on Figure 17 and is included with all the constraint blocks and its parameters from the previous diagram. There are two properties, *T0* and *cP*, which describe an ambient temperature and specific heat of fuel respectively, that are linked with multiple blocks while all the other properties that are exclusively used only in the equation of the certain block are connected with a corresponding parameter. The simulation based on one of the equations from one of the blocks is carried out in the next figure and the results are also explained briefly.

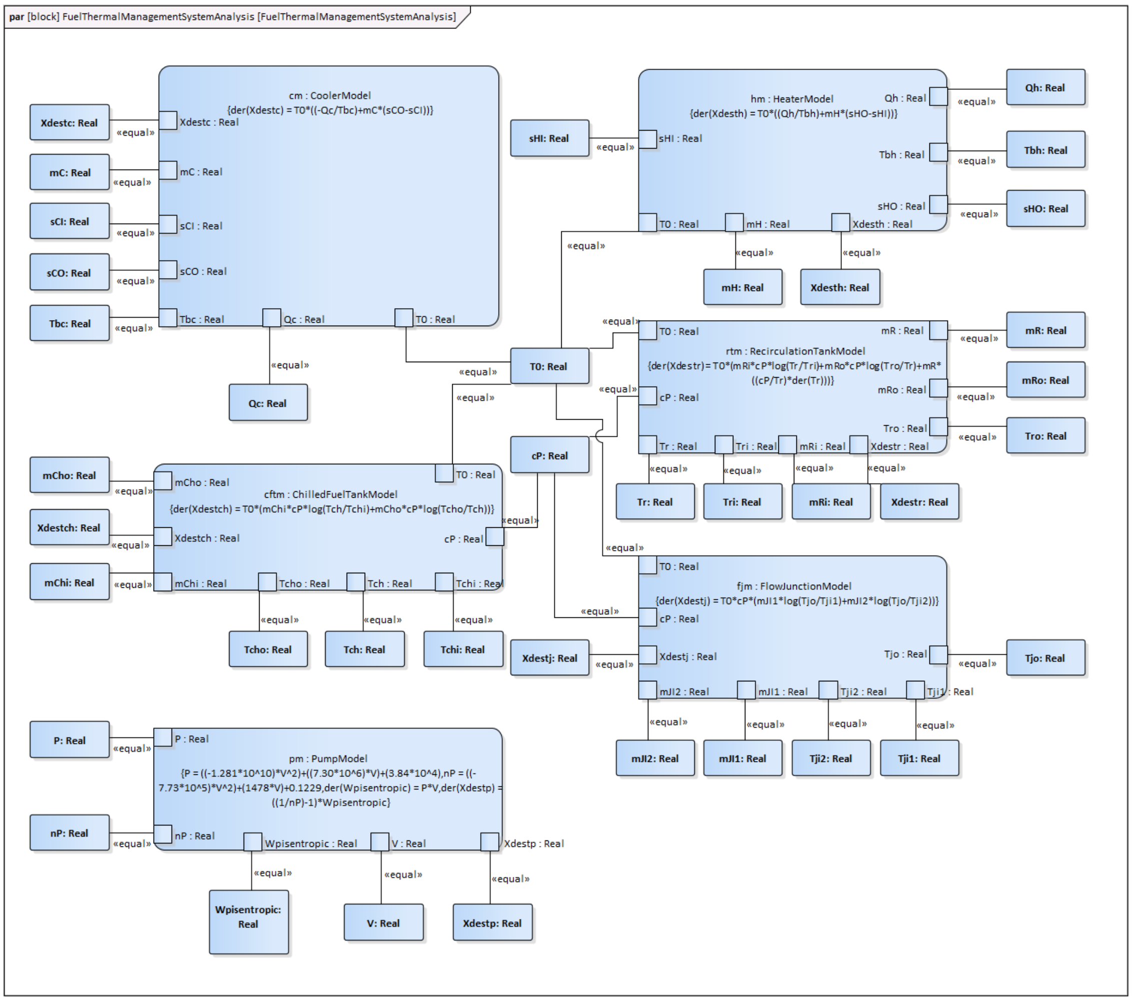
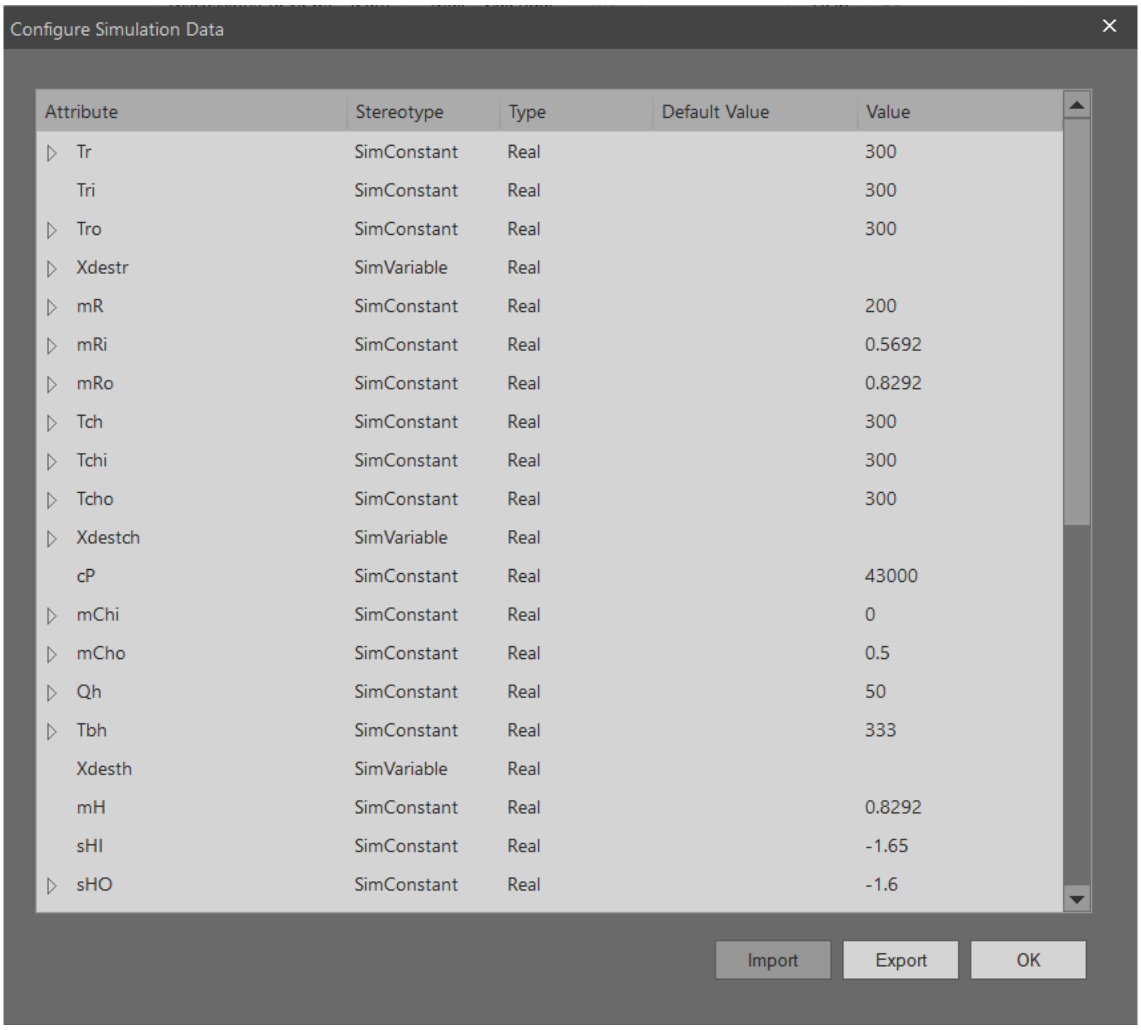


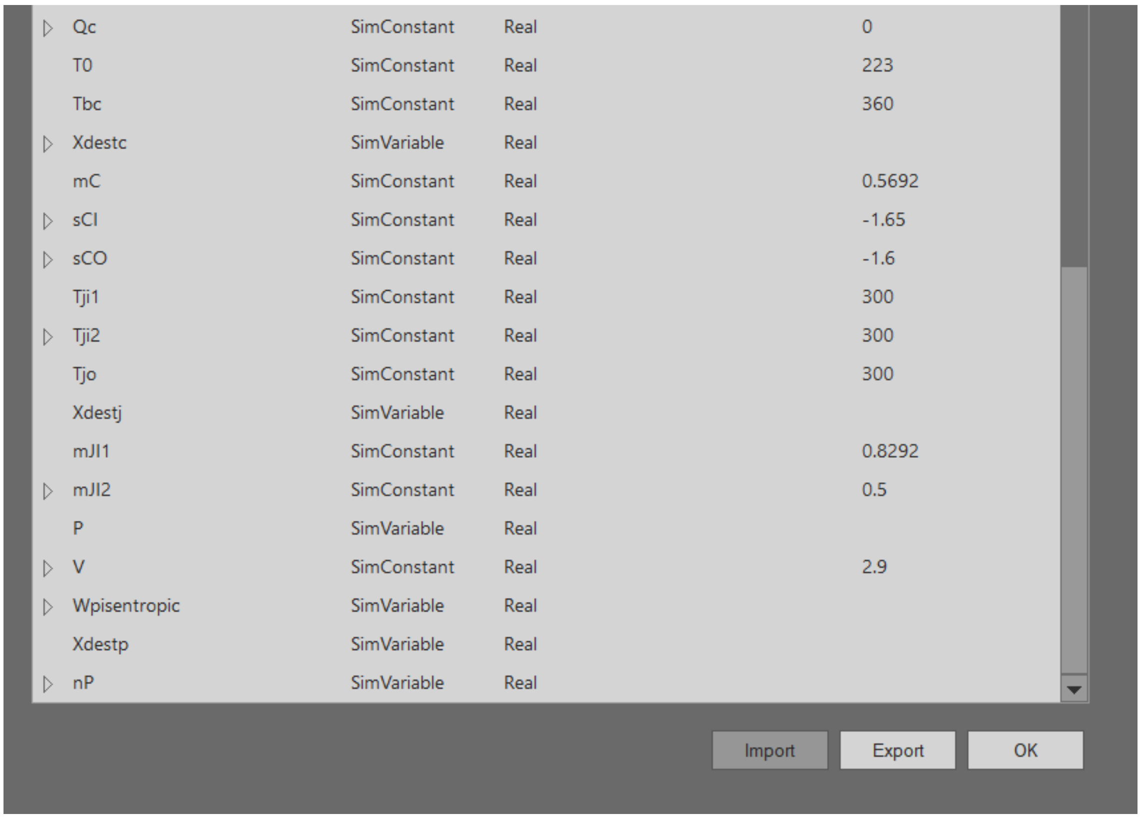
Figure 18. Relationships between constraints and parameters used for the analysis

The dataset for the simulation is depicted in Figure 19 and 20 to show which properties are constant and which variables. As it was mentioned above, all the properties describing the rate of exergy destruction, including *Xdestr*, *Xdestch*, *Xdesth*, *Xdestc*, *Xdestj*, *Xdestp* are variables in this simulation along with pump's pressure differential (*P*), pump's pressure efficiency (*nP*) and the isentropic power consumption of the pump (*Wpisentropic*). All the other properties are used as constants in the simulation. While some of the values of the properties are taken from [7], there are several values taken from other sources as well [9][10] and these values are approximate and do not reflect the actual indicators of the properties.



Attribute	Stereotype	Type	Default Value	Value
Tr	SimConstant	Real		300
Tri	SimConstant	Real		300
Tro	SimConstant	Real		300
Xdestr	SimVariable	Real		
mR	SimConstant	Real		200
mRi	SimConstant	Real		0.5692
mRo	SimConstant	Real		0.8292
Tch	SimConstant	Real		300
Tchi	SimConstant	Real		300
Tcho	SimConstant	Real		300
Xdestch	SimVariable	Real		
cP	SimConstant	Real		43000
mChi	SimConstant	Real		0
mCho	SimConstant	Real		0.5
Qh	SimConstant	Real		50
Tbh	SimConstant	Real		333
Xdesth	SimVariable	Real		
mH	SimConstant	Real		0.8292
sHI	SimConstant	Real		-1.65
sHO	SimConstant	Real		-1.6

Figure 19. Dataset for the simulation



Qc	SimConstant	Real	0
T0	SimConstant	Real	223
Tbc	SimConstant	Real	360
Xdestc	SimVariable	Real	
mC	SimConstant	Real	0.5692
sCI	SimConstant	Real	-1.65
sCO	SimConstant	Real	-1.6
Tji1	SimConstant	Real	300
Tji2	SimConstant	Real	300
Tjo	SimConstant	Real	300
Xdestj	SimVariable	Real	
mJI1	SimConstant	Real	0.8292
mJI2	SimConstant	Real	0.5
P	SimVariable	Real	
V	SimConstant	Real	2.9
Wpisentropic	SimVariable	Real	
Xdestp	SimVariable	Real	
nP	SimVariable	Real	

Figure 20. Dataset for the simulation (cont.)

The simulation result is shown in Figure 21 and it depicts the rate of exergy destruction in the heater as an example. X-axis indicates time in seconds while y-axis indicates the exergy destruction rate in kilojoules. The chart shows how the total rate of exergy destruction in the heater grows in each second and by the 100th second, the total exergy destruction is approximately 4273 kJ. The rate itself is constant and is about 42.73 kJ per second.

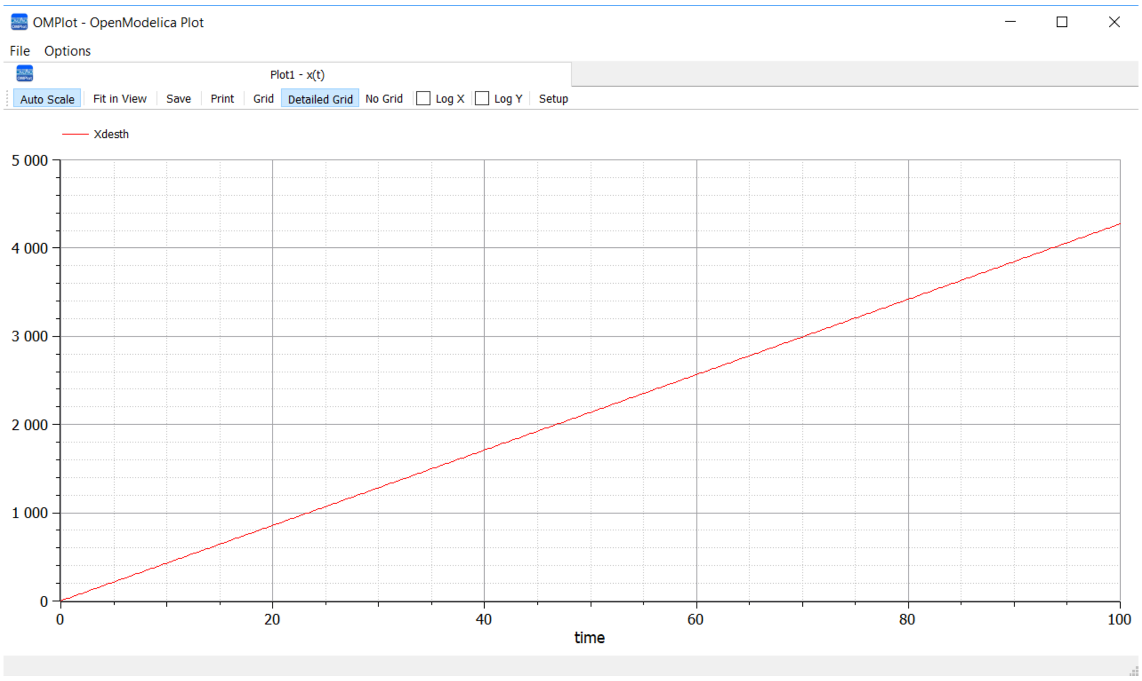


Figure 21. Simulation result

4 Literature review of the means/methods about binding SysML model with Simulink

In order to verify the requirements that are set for the modeled system to satisfy, a number of simulations are needed to be conducted. There are several different simulation environments made for doing that job. In this thesis, a simulation environment called MATLAB/Simulink is used to integrate SysML diagrams with it in order to create new models that can be used to verify and validate the system's requirements. However, there are also several methods and tools in order to integrate these two together. Some of these are briefly descripted in the following paragraphs containing summaries of different research articles.

4.1 Model Transformation approach

This transformation approach is based on a MATLAB/Simulink code generator that generates a MATLAB code from SysML models [11]. This approach consists of SysML models, Acceleo[1] code generation tool templates and SysML4Simulink profile. SysML4Simulink is a model transformation profile that is used to generate Simulink models from SysML source models. This profile consists of three additional stereotypes that extend the existing SysML stereotypes and is used to ensure a better mapping between SysML and Simulink. These three stereotypes are:

“**SimulinkConstant”** – used for indicating that a value remains constant throughout each simulation.

“**SimulinkBlock”** – used for describing the components (blocks) and interactions between them.

“**SimulinkContentBlock”** – used for describing the whole system.

The transformation starts with defining the mapping between SysML and Simulink elements. The Acceleo templates are then created based on the mapping implementation. The SysML model, which now corresponds to the SysML4Simulink profile, is used as an input for the MATLAB code generator. The generator forms script files which can be used to generate a Simulink model.

4.2 Rhapsody

Another way to integrate SysML model with Simulink is to use UML/SysML modeling tool called Rational Rhapsody[2] by IBM [12]. This tool has many capabilities that includes not only modeling the systems, but also integrating the models with Simulink and therefore simulating them. There are three ways for integration:

1. The first way is to import Simulink components into Rhapsody as some of Simulink components are designated in the Rhapsody model. To start with, a C code has to be generated from the corresponding Simulink models. As each Simulink block in Rhapsody includes references to both generated C code and the Simulink model, Rhapsody can infer the Simulink interface by reading the Simulink model and compiling the C code into an executable code for simulating

[1] https://www.eclipse.org/acceleo/

[2] https://www.ibm.com/us-en/marketplace/systems-design-rhapsody

the model. One of the problems with this integration way is that the user can't use variable step solvers as the simulation results from using fixed step solvers may not be accurate or may take a lot of time.

2. Another way is to create an S-function with Rhapsody. The user has to model a part of a system in Rhapsody with SysML so an S-function could be generated from it and then model the whole system in Simulink so the generated S-function could be used in this model. Now the Simulink model can be executed. The problem with this method is that only a part of the system can be modeled with SysML and the rest has to be developed with another modeling language, Simulink in particular, making it harder for the users to use Rhapsody for systems engineering.

3. There is also a third way for integration in order to complement the two existing ways. Firstly, the user has to model the whole system with SysML in Rhapsody and then generate a Simulink model from it. S-functions and Model blocks are converted respectively from defined behaviours in SysML diagrams and Simulink models. Now the Simulink model can be simulated and the user can use variable step solvers if needed.

4.3 From Simulink to Eclipse UML2

The integration from SysML to Simulink doesn't necessarily have to start with designing the system model in SysML first and then transform it into Simulink model, but could also be done in the opposite way [13].

In this approach, a Simulink model of a realization of a function is first created. The Simulink application then parses the model into *mdl* file and converts it into the Eclipse[1] UML2 Framework. While the new model serves as a meta-model and includes Simulink model hierarchy and the relationships between its elements, the model is imported into the SysML model. Now all the components and realized functions in Simulink can be

[1] https://www.eclipse.org/modeling/mdt/?project=uml2

modeled in various SysML structure and behaviour diagrams to represent their behaviour and interconnection with each other.

4.4 SysPhS

In this approach, a standard called SysPhS[1] is used to integrate SysML models with Simulink [14]. The SysPhS is a standardized SysML extension for physical interaction and signal flow simulation, created by the Object Management Group[2] (OMG). This standard describes the ways of integrating SysML models with simulation platforms including Modelica[3], Simulink and Simscape[4], which is an extension of Simulink. As the Simulink can be used only for signal flow modeling, the Simscape is needed for physical interaction modeling. In the following section, a summarized overview of all the elements that are used in the transitioning process is given to show how this standard works. The overview describes transition only from SysML to Simulink and Simscape, but not Modelica as it is not the interest of this topic.

1. **Root element**. All systems and simulation models include root elements that are used to organize the model. SysML root elements are packages contained with diagrams and their elements. A SysML package corresponds to a Simulink library included with a model, both containing a system and to Simscape library.

2. **Blocks and properties**. Blocks in SysML represent classes of systems or components and describe objects with shared features that can be structural or behavioural. Properties are structural features of blocks that may include values or usages of other blocks. SysML blocks with constraint properties correspond to Simulink subsystem blocks and Simscape components, depending if Simscape is included, while SysML blocks without constraint properties only correspond to Simulink subsystem blocks.

[1] https://www.omg.org/spec/SysPhS

[2] https://www.omg.org/

[3] https://www.modelica.org/modelicalanguage

[4] https://www.mathworks.com/products/simscape.html

3. **Generalization**. SysML includes generalization relationship which enables one block reuse the other block’s features as the first block inherits all the properties of the other. While SysML supports multiple generalizations of one block, Simulink on the other hand does not support generalization as Simulink blocks cannot inherit features from other blocks. To fix this, the features need to be redefined in Simulink blocks. However, Simscape does support generalization but it supports only single generalization of components, unlike SysML.

4. **Property redefinition**. In SysML, inherited properties by generalization of the blocks can be modified by redefinition. Simulink on the other hand does not provide redefinition because it does not support generalization of the blocks. To solve this problem, Simulink’s elements that correspond to those properties that redefine the inherited ones can be used to achieve the similar effect as SysML’s redefinition. As with Simscape, it supports generalization, but not redefinition. In this case, Simscape’s corresponding elements for multiple generalization or inherited SysML properties can be used to achieve the redefinition effect.

5. **PhSVariables and PhSConstants**. There are three types of properties in SysML: continuous, discrete and constant properties. Two of them, continuous and discrete properties, are both stereotyped by PhSVariable. The difference between these two property types is that continuous properties are included with isContinuous=true and discrete properties with isContinuous=false while PhSConstant stereotypes constant properties. In Simscape, continuous variables correspond to PhSVariables and constant parameters to PhSConstants while discrete variables are not supported by Simscape. Simulink, on the other hand, does not correspond to PhSVariables nor PhSConstants but corresponds to SysML value properties.

6. **Ports and Flow Properties**. In SysML, parts of the system are included with ports and the connectors are drawn between the ports to describe the interactions between the parts. These ports describe the flow of the properties in the way of what type of properties and which way they flow. In order to model signal flow with SysML, flow properties need to be stereotyped by a non-conserved PhSVariable, typed by Real, Integer or Boolean and must flow either in or out of the port. Simulink has three types of ports: inports, outports and connection ports.

SysML ports included with flow properties on port types modeled in above-mentioned way correspond to Simulink inports or outports and to Simscape inputs or outputs. Modeling physical interactions between system's parts require flow properties to be with inout flow property. Simulink provides connection ports for representing bidirectional flows between ports and they need to be linked to Simscape nodes, which support Simulink ports in order to achieve the correspondence to SysML ports.

7. **Connectors**. In SysML, connectors are used to link blocks together via ports. SysML connectors can correspond to Simulink lines in both ways, either Simscape is included with Simulink or not. When Simscape is used with Simulink and SysML connectors that run from a block without constraints to a block with constraints via ports, the connector corresponds to a Simulink line called *connection*. When a block with constraints is in turn connected to other blocks without constraints, an additional block between them is needed to convert a regular Simulink signal to a Simscape signal. As Simulink connection connects a block with constraints to a converter block, a Simulink line connects a converter block to a block without constraints. As in Simscape, the connectors that run from a block with constraints in SysML correspond to connections in Simscape.

8. **Blocks with constraints**. Constraint blocks in SysML include constraint properties, which are included with constraints that act as equations, and parameters, which are being used in these equations. When signal flow is considered, SysML constraint blocks correspond to Simulink S-functions in a way that each S-function corresponds to a particular parameter in a SysML constraint block. Variable names in S-functions are named with same names as SysML parameters and PhSConstants, which are linked to constant parameters in SysML, are replaced in S-functions with same or a different value. For signal flow, Simscape provides a way to specify input and output signals for its components. When physical interaction is considered, constraints in SysML blocks correspond to the equations in Simscape components.

9. **Default values and initial values**. SysML has two ways to specify values for properties: *default values* and *initial values*. While default values are only set when instances are created, initial values are set when an instance has already

been created and this enables initial values to override default values. SysML default values correspond to initial values of S-function variables in Simulink and to initial values of Simscape variables and parameters.

10. **Data types and units**. In SysML, data types are called value types and they can be linked to units, which are modeled with SysML Unit Block. Simulink inports and outports can also contain units and some of the symbols for these units are defined in Simulink and modelers can also create symbols on their own. These newly defined symbols can also be used in Simscape where these unit symbols can be used for variables and parameters.

11. **State machines**. SysML state machines are used to describe the behaviour of the blocks and the SysML capabilities for concern to simulation include triggering the transitions between states and the values that are sent from one object to another through ports depending on the state. Simulink is provided with an extension called StateFlow which is used for state machines and includes some features of SysML state machines. StateFlow describes transitions and actions performed depending on the state of an object. While state machines act as separate behaviours in SysML, in StateFlow they are depicted as blocks. Even that StateFlow is an extension for Simulink it does not extend for Simscape.

4.5 QGen

This approach consists of several steps in order to transform SysML models to Simulink models using the QGen[1] tool developed by AdaCore [15].

The first step is to describe the structure and the requirements of the system in SysML. The structure is depicted in an *internal block diagram* and the requirements in a *requirement diagram.* Next, the requirements are being formalized as the system-level properties are rewritten as constraints in SPARK language. After this, all the blocks and flows from the internal block diagram are converted into Simulink models using the QGen tool. This transformation produces a subsystem hierarchy where the blocks from SysML internal block diagram are converted into Simulink Subsystems. Additionally, SysML

[1] https://www.adacore.com/qgen

ports are transformed into subsystem ports and the constraints into observers. As the conversion from SysML to Simulink is done, the Simulink models can be populated with algorithms that can be validated by simulation, as well as generate the models into SPARK code and into C or Ada code in order to verify the models and to use the codes for final implementation. These actions can be also performed by using the QGen tools like QGen Verifier Tool and QGen code generator.

5 System behaviour and constraints in Simulink

The transformation of the internal block diagram from Figure 15 to a Simulink model is shown in Figure 22. The Simulink model consists of system components and data flows between them. The difference with SysML internal block diagram is that the model also contains observers that are composed from the constraints in a requirement diagram in Figure 16. The observers are depicted here as subsystems, that include the comparison of the inputs that the observers monitor. If the inputs meet the requirements, the simulation works as it should, but if it does not meet the requirements, then an Assertion block inside the observer stops the simulation.

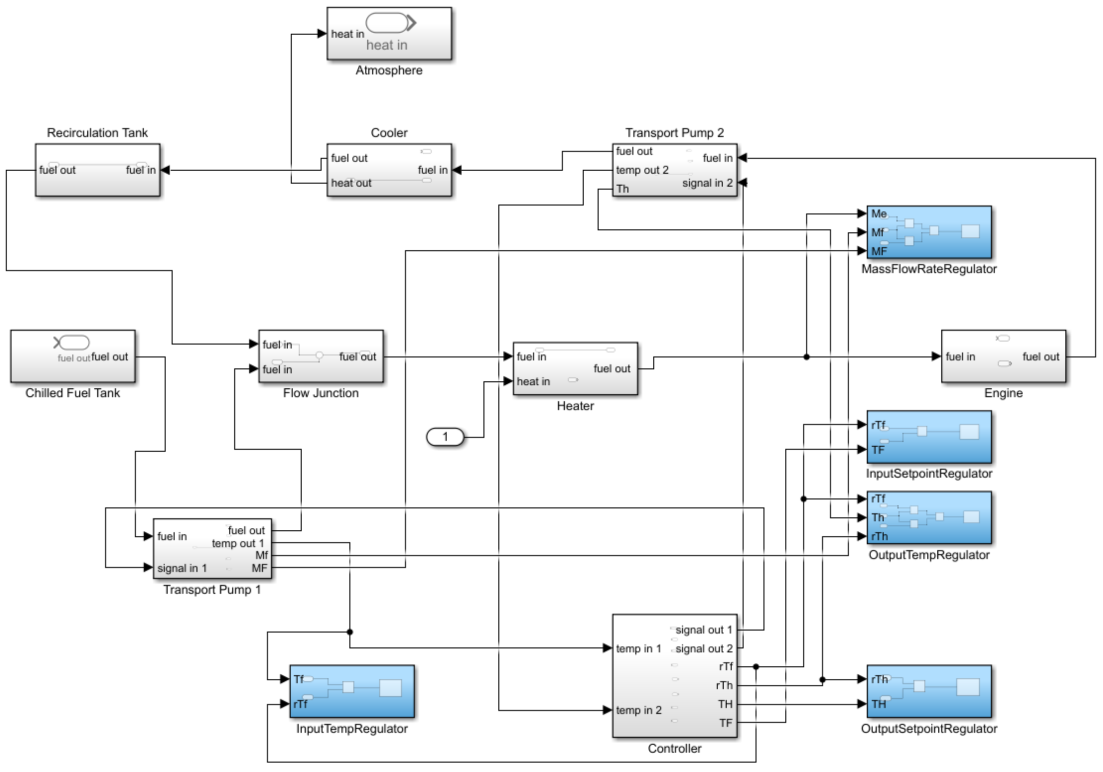


Figure 22. FTMS internal block diagram as a Simulink model

As an example, one of the observer's, named *OutputTempRegulator*, internal composition is shown in Figure 23. Three inputs that are monitored, named *rTf*, *Th* and *rTh* are being compared if they meet the requirements with *Relational Operator* blocks and the *Logical Operator* block named *AND* means that both of the comparisons results have to positive so the simulation can work properly. All observers consist of a diagram implementing the comparison logic and ending with an Assertion block.

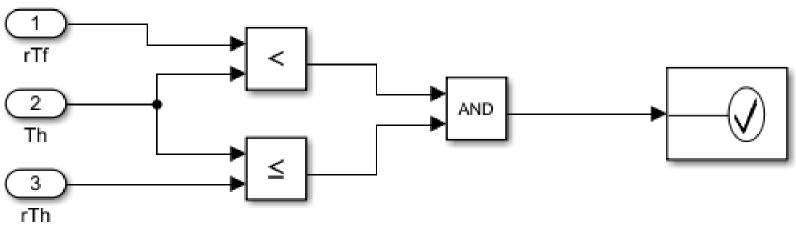


Figure 23. OutputTempRegulator observer

6 Summary

As the amount of workload in this thesis shows that system modeling can be very time-consuming activity and even more when the model is intended to be converted to Simulink in order to design a control algorithm for the system. Ideally, the system modeling should have continued with modeling several other aspects of the system with use case diagram(s), activity diagram(s), sequence diagram(s), state machined diagram(s) and maybe even with more diagrams of the diagram types that were used in this thesis to get the whole system fully modeled, but that was not necessary for the purpose of this work. As the required diagrams were composed in SysML, the next step was to transform the system model into Simulink in order to model the behaviour of the system and constraints of the system as observers that monitor the critical properties of the system. Finalised Simulink models contain the observers and a skeleton of system architecture without behaviour. Control algorithm design, which was initially also planned, was excluded as the part of modeling system's physical properties and simulating in Modelica grew too big. This was considered sufficient in the scope of given work as the goal was to demonstrate workflow.

The next steps in this research will be to complete the control algorithm descriptions, use QGen to obtain Ada code annotated with pre- and postconditions and demonstrate the proof of the properties on this code.

References

[1] Fridenthal, S., Moore, A. and Steiner, R., “A Practical Guide to SysML: The Systems Modeling Language”, Morgan Kaufmann, 2015

[2] “What is SysML?” [Online] Available: http://www.omgsysml.org/what-is-sysml.htm [27.02.2019]

[3] “Electric kettles” [Online] Available: https://www.explainthatstuff.com/how-electric-kettles-work.html [20.02.2019]

[4] “Newton’s Law of Cooling” [Online] Available: https://en.wikipedia.org/wiki/Newton%27s_law_of_cooling [06.03.2019]

[5] “Overall Heat Transfer Coefficient” [Online] Available: https://www.engineeringtoolbox.com/overall-heat-transfer-coefficient-d_434.html [06.03.2019]

[6] Fritzson, P., “Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach”, Wiley-IEEE Press, 2015

[7] Jain, N., Hencey, B. M., „Increasing Fuel Thermal Management System Capability via Objective Function Design“, 2016 American Control Conference (ACC), Boston Marriott Copley Place, July 2016. Boston, MA, USA, pp. 549-556

[8] Doman, D. B., “Fuel Flow Control for Extending Aircraft Thermal Endurance Part II: Closed Loop Control”, American Institute of Aeronautics and Astronautics, January 2016, pp. 1-23

[9] “Exergy Flow Rate” [Online] Available: https://www.sciencedirect.com/topics/engineering/exergy-flow-rate [17.04.2019]

[10] “DS600-24A Type 6902 Fuel Transfer Pump Jaguar” [Online] Available: https://www.eaton.com/ecm/idcplg?IdcService=GET_FILE&allowInterrupt=1&RevisionSelectionMethod=LatestReleased&noSaveAs=0&Rendition=Primary&dDocName=CT_196067 [17.04.2019]

[11] Chabibi, B., Douche, A., Anwar, A., Nassar, M., “Integrating SysML with simulation environments (Simulink) by model transformation approach”, 25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), June 2016, pp. 148-150

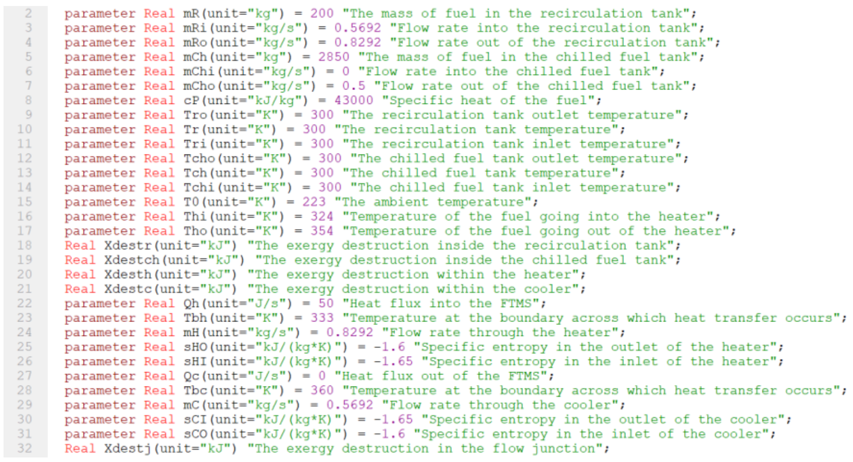
[12] Sakairi, T., Palachi, E., Cohen, C., Hatsutori, Y., Shimizu, J., Miyashita, H., “Designing a Control System using SysML and Simulink”, SICE Annual Conference (SICE), August 2012, Akita University, Akita, Japan, pp. 2011-2017

[13] Qamar, A., During, C., Wikander, J., “Designing Mechatronic Systems, a Model-based Perspective, an Attempt to Achieve SysML-Matlab/Simulink Model Integration“, 2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Suntec Convention and Exhibition Center, Singapore, July 2009, pp. 1306-1311

[14] “SysML Extension for Physical Interaction and Signal Flow Simulation” [Online] Available: https://www.omg.org/spec/SysPhS/About-SysPhS/ [05.04.2019]

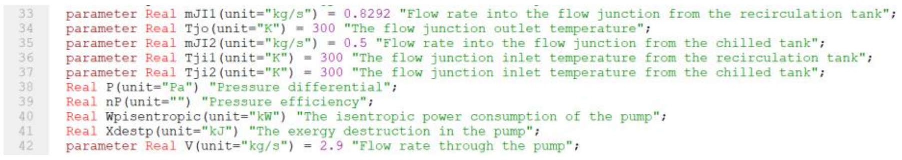
[15] Naks, T., Aiello, M. A., Taft, S. T., “Using SPARK to Ensure System to Software Integrity: A Case Study”, submitted to DeCPS 2019 - Workshop on Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering, Ada-Europe, 11-14 June 2019, Warsaw, Poland.

Appendix 1 – FTMS properties with units and meanings



```
parameter Real mR(unit="kg") = 200 "The mass of fuel in the recirculation tank";
parameter Real mRi(unit="kg/s") = 0.5692 "Flow rate into the recirculation tank";
parameter Real mRo(unit="kg/s") = 0.8292 "Flow rate out of the recirculation tank";
parameter Real mCh(unit="kg") = 2850 "The mass of fuel in the chilled fuel tank";
parameter Real mChi(unit="kg/s") = 0 "Flow rate into the chilled fuel tank";
parameter Real mCho(unit="kg/s") = 0.5 "Flow rate out of the chilled fuel tank";
parameter Real cP(unit="kJ/kg") = 43000 "Specific heat of the fuel";
parameter Real Tro(unit="K") = 300 "The recirculation tank outlet temperature";
parameter Real Tr(unit="K") = 300 "The recirculation tank temperature";
parameter Real Tri(unit="K") = 300 "The recirculation tank inlet temperature";
parameter Real Tcho(unit="K") = 300 "The chilled fuel tank outlet temperature";
parameter Real Tch(unit="K") = 300 "The chilled fuel tank temperature";
parameter Real Tchi(unit="K") = 300 "The chilled fuel tank inlet temperature";
parameter Real T0(unit="K") = 223 "The ambient temperature";
parameter Real Thi(unit="K") = 324 "Temperature of the fuel going into the heater";
parameter Real Tho(unit="K") = 354 "Temperature of the fuel going out of the heater";
Real Xdestr(unit="kJ") "The exergy destruction inside the recirculation tank";
Real Xdestch(unit="kJ") "The exergy destruction inside the chilled fuel tank";
Real Xdesth(unit="kJ") "The exergy destruction within the heater";
Real Xdestc(unit="kJ") "The exergy destruction within the cooler";
parameter Real Qh(unit="J/s") = 50 "Heat flux into the FTMS";
parameter Real Tbh(unit="K") = 333 "Temperature at the boundary across which heat transfer occurs";
parameter Real mH(unit="kg/s") = 0.8292 "Flow rate through the heater";
parameter Real sHO(unit="kJ/(kg*K)") = -1.6 "Specific entropy in the outlet of the heater";
parameter Real sHI(unit="kJ/(kg*K)") = -1.65 "Specific entropy in the inlet of the heater";
parameter Real Qc(unit="J/s") = 0 "Heat flux out of the FTMS";
parameter Real Tbc(unit="K") = 360 "Temperature at the boundary across which heat transfer occurs";
parameter Real mC(unit="kg/s") = 0.5692 "Flow rate through the cooler";
parameter Real sCI(unit="kJ/(kg*K)") = -1.65 "Specific entropy in the outlet of the cooler";
parameter Real sCO(unit="kJ/(kg*K)") = -1.6 "Specific entropy in the inlet of the cooler";
Real Xdestj(unit="kJ") "The exergy destruction in the flow junction";
```

Appendix 2 – FTMS properties with units and meanings (cont.)



```
parameter Real mJI1(unit="kg/s") = 0.8292 "Flow rate into the flow junction from the recirculation tank";
parameter Real Tjo(unit="K") = 300 "The flow junction outlet temperature";
parameter Real mJI2(unit="kg/s") = 0.5 "Flow rate into the flow junction from the chilled tank";
parameter Real Tji1(unit="K") = 300 "The flow junction inlet temperature from the recirculation tank";
parameter Real Tji2(unit="K") = 300 "The flow junction inlet temperature from the chilled tank";
Real P(unit="Pa") "Pressure differential";
Real nP(unit="") "Pressure efficiency";
Real Wpisentropic(unit="kW") "The isentropic power consumption of the pump";
Real Xdestp(unit="kJ") "The exergy destruction in the pump";
parameter Real V(unit="kg/s") = 2.9 "Flow rate through the pump";
```