CODEN:LUTEDX/(TEIE-5313)/1-110/(2013)

Industrial Electrical Engineering and Automation

Modelling and Simulation of Smart Grids using Dymola/Modelica

Jonas Enerbäck
Oscar Nalin Nilsson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Modelling and Simulation of Smart Grids in Dymola/Modelica

J. Enerbäck & O. Nalin Nilsson

June 11, 2013

Abstract

Smart grids have been proposed as a way to increase grid robustness and reduce consumption peaks and at the same time decrease electricity costs for the end users. This thesis aims to develop generic models for smart grids focusing on smart houses which are used to test if it is possible to reduce consumption peaks, decrease electricity cost and at the same time increase grid stability. Models have been developed in *Dymola* using the *Modelica* language and a controller program has been developed using *C++*. Based on simulations of the models it is concluded that the consumption peaks in the grid can be decreased while at the same time grid robustness in terms of withstanding short power outages is improved. Furthermore this effort reduced electricity cost for the end users as well.

Acknowledgements

We would like to thank our supervisor at Lund University Faculty of Engineering Jörgen Svensson and our mentors at Modelon AB Jens Pålsson and Carl Wilhelmsson for their input and comments.

We would also like to thank the people working at Modelon AB for their hospitality towards us when working with this thesis at Modelon AB's office.

Contents

Chapter 1

Introduction

In this chapter the background to smart grids is given as well as the project scope and goals of the thesis.

1.1 Background

Today most electricity is produced mainly with fossil fuels, nuclear and hydro. These energy sources are dispatchable in their nature, i.e. the amount of electricity that is produced is determined exactly and can be increased or decreased within a specific time frame according to changes in demand. Due to the probable negative impact of CO_2 [1] on the environment and the depletion of fossil fuels it is essential that the electrical energy consumption has to decrease and that the consumption is done in a more efficient way.

Smart grids is a concept that tries to solve these issues. The concept is not well defined but it circles around communication between different actors such as power producers, transmission system operators , distribution system operators, and end users. The communication might contain minute wise information about current production, consumption and price of electricity. In the short term this may be used to create incentives to lower consumption of electricity at peaks of high demand. In the longer run households may have automatic systems that will close down low priority loads, e.g. outdoor lights, during hours of high demand (high price) and start resources with low priority, e.g. laundry machine or dishwasher, during hours of low demand.

Small-scale power generation or *micro generation*, such as solar and wind power at household level, might be a cost efficient way of generating electricity locally, both to fulfil the consumers needs but also in order to sell to the electric grid.

The U.S. Department of Energy describes smart grid as: "An automated, widely distributed energy delivery network, the Smart Grid will be characterized by a two-way flow of electricity and information and will be capable of monitoring everything from power plants to customer preferences to individual appliances. It incorporates into the grid the benefits of distributed computing and communications to deliver real-time information and enable the near-instantaneous balance of supply and demand at the device level." [2, p.13] they continue with:

"While supply and demand is a bedrock concept in virtually all other industries, it is one with which the current grid struggles mightily because, as noted, electricity must be consumed the moment it is generated. Without being able to ascertain demand precisely, at a given time, having the 'right' supply available to deal with every contingency is problematic at best. This is particularly true during episodes of peak demand,those times of greatest need for electricity during a particular period." [2, p.13]

1.2 Motivation

The aim of this project was to develop a model of a smart house connected to an external grid using the *Modelica* based tool *Dymola* and to design control laws that control production and consumption on the grid using *C++*. The control structure was designed and implemented in C++ since the simulation in Dymola has no pre-defined causality which makes it difficult to do control

of the system. By using Dymola to simulate the physical parts and using the imperative programming language C++ to make decisions, it is possible to use the strength of Dymola and the strength of C++. The control system continuously gather information from the grid such as power consumption, voltage level and priorities for different loads and acts using this information to optimize the energy use. The grid should in an automated fashion act and gather information to keep the grid operating in an optimal way. Apart from modelling a DC grid on household level a grid connecting several households should also be modelled. This grid connecting several houses will form a model over a smart cell. Control laws for this cell should be able to distribute electricity from local wind farms, solar power plants and over production from households in an efficient way.

To model and control smart grids is important. These grids can be used to distribute variable produced electricity such as wind power, in a more efficient way than the current electricity grid. Furthermore these grids can be used to reduce consumption peaks and improve the reliability of the grid. A DC grid is modelled instead of AC since the basic smart control aspect is still the same with a DC grid and is easier to model. Furthermore some researchers believe that future smart grids will be implemented using DC components and DC grids. Since many micro generating sources are DC in their nature, such as solar power plants, it will be possible to avoid transformations losses using DC. Also an important role in future smart grids will be the ability to store energy, preferable in battery equipped cars, and since energy storage in batteries are done in DC, using DC grids will further avoid transformation losses[3].

1.3 Project Scope

This project is limited to develop a DC model of a household and a medium DC voltage grid. The household controls an arbitrary number of loads which importance is determined according to their priority. Furthermore the household model should be able to have micro production, such as wind power and solar power, and a battery to be able to operate when the external grid fails. The model of the mid voltage grid has the ability to connect wind power plants and batteries. The storage discharged during times of high electricity prices and charged during times of low prices to deliver cheaper electricity to the end user. Also the battery functions acts as a temporary buffer to be used when the transmission network fails. All control logics are developed in C++ and interfaced to Dymola using C.

1.4 Goals

This project seeks to answer the following questions:

- Does a household have the capacity to deliver power for electrical components when there is a blackout on the power grid through the use of micro generation and energy storage?

- Does the distribution network have the ability to do voltage control and provide power to households when the transmission network can not provide any power?
- How large difference in electricity price between daytime and nighttime is necessary in order to have a pay off for installation of "smartness" in a house? What is the reduction of cost for a household (10-30 MWh/year)
- Is it possible to reduce power peaks in the grid with the use of the proposed control strategy and price differentiation?

This is done by developing components such as loads (electronic devices), generators, batteries and connections between voltage levels which are used to model *smart houses* and *smart grids*.

Chapter 2

Smart Grids

Since the beginning of last century transmission grids has been used to distribute centrally produced electricity from a few places to the end user. With the increased usage of renewable energy sources such as wind power and solar power new challenges are put on the grid[4]. A future grid must be able to distribute locally produced electricity from a lot of different sources and coordinate consumption to handle the varying electricity generation from the renewable energy sources. The following chapter will give an introduction to the concepts of smart grids and compare it with the power grid used today.

2.1 Electricity Generation Today

Today the electricity in the world is produced mainly with fossil fuels, hydro and nuclear energy. The *non renewable energy sources* adds up to over 80% of the total electricity production. The global distribution between energy sources can be seen in figure 2.1 [5]. The electricity comes predominantly from a few *Large central power stations*; the nuclear energy is produced from 437 nuclear reactors [6, p. 20], electricity from coal is produced from 2300 coal-fired power stations (7000 individual units)[7] and hydro electricity comes from 2744 hydropower plants [8].

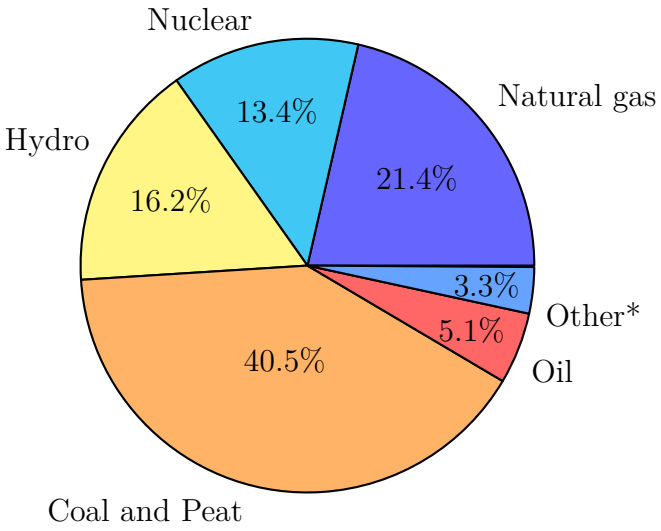


Figure 2.1: World electricity generation by source of energy 2009, a total of 20 052.8 TWh, [5]. *Includes biofuels, wind, geothermal and solar.

In the early ages all energy sources used were renewable: wind, firewood and water from water dams. During the last century the turbines and electricity generators have revolutionized the electricity consumption and changed the way we live. Near all electricity is produced by *electric generators*. They are based on *Faraday's law of induction* which was discovered in the 1930s by the British scientist Michael Faraday. The law describes how a magnetic field will interact with an electric circuit. When a permanent magnet is moved relative to a conductor, an electromotive force *EMF* is created. If the generator is connected to a load a current will flow. Generators will produce a current that alternates, thus *Alternating Current*, AC. The movement of the magnets is created by the flow of a medium though turbines. This medium might be steam (e.g nuclear power), water (hydro power) or wind (wind power).

2.1.1 Supply and Demand

Electricity is consumed the same instant as it is produced, there is no delay and energy is not stored on the grid for later use (except when specific storage is used, e.g. pumped hydro). Therefore there has to be an equilibrium between

the production and consumption at all times. If there is overproduction, the frequency of the grid will increase and if there is overconsumption the frequency will drop. This is similar to how acceleration of a car works, the force acting on the car due to the torque produced by the engine has to be equal to aerodynamic drag plus friction in order for the acceleration of the car to be zero, i.e. constant speed. If the car suddenly is going up hill, the speed will drop if the torque is not increased. The fluctuations in consumption/demand is compensated by changing the production.

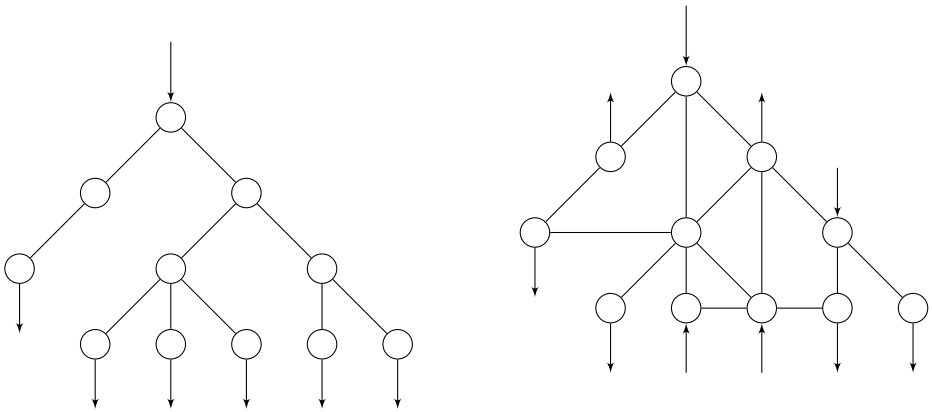
The recent increase of installed renewable energy sources puts new demands on the grid. Apart from wind and solar energy, the electricity generation is deterministic in its nature, i.e. the source of energy, fossil fuels, water, uranium or biomass, is always available and thus the electricity power produced can be set arbitrary within certain limits. Renewable energy sources such as solar and wind power are in that sense very different since the source of energy is intermittent and thus the amount of electricity generated can not be decided by an operator or a control system. To keep the balance between supply and demand the rest of the power sources need to compensate for fluctuating power from the renewable energy sources (RES). For example if the generated power from a wind power plant drops, a coal power would increase its generation to compensate for the power loss. With the increasing installation of RES the power control task is getting more difficult. One possible solution to this would be to not only control the generation of power but also control the consumption. This could be done by delaying non essential loads to run during times of high generations. This would decrease the impact of the fluctuating power generation and enable the installation of more RES.

2.2 Electricity Grid

The electricity grid used today is built by high voltage transmission lines connected to a distribution grid at a lower voltage. The transmission grid is used to transport electricity long distances. Energy loss in a cable is proportional to the square of the current times the resistance in the cable. Therefore high voltage lines are used when transporting electricity long distances. Most commonly used voltage levels in Sweden are 130 kV and 400 kV. The distribution grid is connected to the transmission grid at local transformation stations and provides local houses and industries with electricity. Typical voltage on the distribution grids in Sweden are 10, 20 or 50 kV [9, p. 78-79]

The distribution grid is designed using a radial structure, see figure 2.2a. This means that the grid is only connected to the transmission line at one point and power is only flowing one direction. This structure is cheap but it is very error-prone. If one connection fails all resources connected further down from this point fails. The transmission grid is built to be more robust. Mainly because a blackout on the transmission would impact the whole electricity grid, in worst case whole countries electricity grids could fail. In July 2012 India experienced a major blackout, 700 million people were left without power for two days. This was a so called cascading failure. In this case one transmission

line failed which caused the parallel lines to fail and soon the failure cascaded through the whole northern grid [10]. To make the transmission grid more reliable, redundant connections are introduces, see figure 2.2b. This makes it possible to disconnect failing part of the grid to keep the grid operating normally[9, p. 80-81].



(a) Radial structure

(b) Redundant connections introduced.

Figure 2.2: Grid structures used in electricity grids today.

2.3 Key Concept

The key concept of a smart grid is to integrate and coordinate the action of all grid users, such as *big power plants*, *small micro production units* and *active* as well as *passive households* in an efficient and environmental friendly way. This is done by introducing more interconnections in the grid and more intelligence which can take automatic decision based on measurements from the grid. Active households are costumers that make active decisions based on the current electricity price, e.g. postpone the start of the dishwasher or the washing machine
The main advantages of smart grids can be divided into three parts: lower environmental impacts, higher reliability and lower operational cost.

1. Climate - The main objective of a smart grid is to allow an increased usage of renewable energy sources such as wind power and solar power. In USA wind power is expected to increase from 145 TWh in 2013 to 1160 TWh by 2030 [11] [12]. Renewable energy sources are stochastic in their nature, which is problematic since electricity must be consumed when it is generated. Smart grids can be controlled in such a way that it follows the stochastic production pattern, and can therefore enable an increased usage of renewable energy sources.

2. Reliability - Reliability is one of the most important aspects of a new generation power grid. By introducing more intelligence in the grid and

using measurements from the grid it will be possible to automatically react to and prevent errors in the grid[12].

3. Economy - The next generation smart grids will make it possible for the end consumer to buy, sell and consume electricity more efficient and easily. This is made possible by introducing more intelligence into the power grid and houses. End users will have the possibility to postpone for instance their dishwasher until the night when the electricity price is low and sell micro produced electricity during times of low consumption.

The movement from non-renewable production of electricity puts new strains on the grid. Today electricity is produced centrally at few *large power stations* and then transported to the end user, from high voltage to low voltage. Renewable energy sources such as wind turbines and solar panels will be distributed locally and produce electricity close to the end user. This introduces new issues, since wind and solar power are stochastic energy sources, it is hard to tell in advance when electricity can be generated. It could locally be overproduction of electricity which the grid must be able to address. Traditional grids are designed and built mainly to transport electricity from high voltage (central generation units) to low voltage (end users) and it is therefore not possible to deal with local overproduction in a sufficient way. Smart grids will make it possible to coordinate consumption such that overproductions will be addressed. One idea would be that all electrical cars in the neighbourhood would be charged when there is overproduction in the neighbourhood. It is also possible to charge local energy storage units during times of overproduction. Smart grids are targeted to transport electricity from low voltage to high voltage, which will make it possible to sell locally produced electricity to neighbouring areas when there is overproduction.

Another big problem which must be taken into account when designing an electrical grid is peaks in demand. The grid must be designed such that it copes with the demand peaks even if these peaks only are present during short times a few days a year. This means that the grid is usually operating far from the systems operating limits. By using smart grids it will be possible to reduce the demand peaks by coordinating generation and consumption. This means that the grid will operate at a higher base level but with smaller peaks. The grid can therefore be designed for a lower stress level. [12].

2.4 Current Projects

Even though the concept of smart grids has been around for a couple of decades it is not until recent years that some projects have been carried out. In the following chapter two current projects in Sweden are presented; *Smart Grids Gotland* and *Stockholm Royal Seaport*.

2.4.1 Smart Grids Gotland, Sweden

At the Swedish island Gotland there is an ongoing smart grid project that was initialized in September of 2012 and will continue to 2015. The local energy company *GEAB*[1] is together with *Vattenfall*, ABB[2], *Svenska Kraftnät*, *Schneider Electric* and *KTH*[3] building a smart grid. Its goal is to investigate the possibilities to modernise current electrical grids to include larger amount of renewable energy. This is done with the background of the *20-20-20-goals*, where *EU* shall achieve 20 % renewable energy, 20 % reduction in greenhouse gas emissions and 20% energy efficiency increase by the year 2020. [13] [14]

The project has three overall goals; optimal integration of large amounts of wind power in a current distribution grid, show new technical solutions to increase power quality on a rural power grid with a large part distributed power production and create possibilities for demand side participation in the electricity market. Some of the quantifiable goals are; improve reliability by reducing SAIDI4[4] by 20 % on the grid between the substations *Källunge* and *Bäcks*, active participation of approximately 30 industry customers and to attract approximately 2000 private households to participate in market tests.[13] [14]

The private households will receive electricity price signals and will adapt accordingly. *Energy Service Interface* (ESI) is a product that will enable automatic start of electronic devices, e.g. dishwasher or water heater, when the availability of electricity is high thus the electricity price is low. This active participation will give the end user the opportunity to lower its overall electricity costs. Industries and large farms will also have information about the current electricity price in order to reduce the total cost and to decrease consumption peaks. This might involve turning off processes with low priority when the availability of electricity is low and the price is high. Households, industries and farms that are producing electricity through small scale solar or wind power will be able to sell electricity to the grid during times of overproduction[5], this is called *prosumer* (contraction of producer and consumer).[13] [14]

ABB is developing an energy storage for Gotland which is going to be used to handle fluctuations in demand. The storage will be charged when the supply of wind power is high and discharged when there is high demand of power. The storage will be able to produce 3.6 MW during 5 minutes and at the same time help to stabilize the grid.[13] [14]

[1]Gotlands Energi AB

[2]ASEA Brown Boveri ltd.

[3]Kungliga Tekniska Högskolan (Royal Institute of Technology)

[4]System Average Interruption Duration Index, is a reliability indicator and is calculated as ”the sum of all customer interruption durations divided by the total number of customer served”.

[5]The consumption of power is less than the production.

2.4.2 Stockholm Royal Seaport, Sweden

Stockholm Royal Seaport, Stockholm, Sweden (sv. Norra Djurgårdsstaden) extends from Husarviken in the north, over Värtahamnen and free port, to Loudden the south and is one of the most extensive urban development. The first housing units where built in 2012 and the area is expected to be completed by 2030. Stockholm Royal Seaport will then consist of 10 000 residences and 30 000 workplaces. The project is supported by the *Clinton Climate Initiative* (CCI) and some of the climate and environmental goals involves reducing the emissions of CO_2 from the national average of 4.5 tonnes per person to 1.5 tonnes for habitants in the Stockholm Royal Seaport. The smart grid development is lead by by Fortum together with Ericsson, ABB, Electrolux, HSB, KTH.[15][16]

The goal is to increase the energy efficiency and reduce consumption peaks by visualizing energy usage and make it easier for the users to be active consumers and producers of electricity, thus *prosumers*. This will be achieved with the help from components such as; photovoltaic solar panels, local energy storage, control systems for more effective allocation of energy usage, individual measurements and visualization of consumption for control and monitoring. The power grid development includes research in medium and low voltage switchgear and substations with extensive monitoring and control, cable boxes with control and monitoring, centralized energy storage and integration of charging stations for electric vehicles (EV) to the power grid.[16]

Residents at the Stockholm Royal Seaport will be able to:

- Produce electricity from roof mounted solar panels and through the smart grid the power can be stored locally or sold to the grid. The goal is to have 30 % of the electricity used by a building produced locally from solar panels, wind power or to use excess energy.
- Influence their electricity usage by shifting consumption to periods of the day where supply of power is good and price is low.
- Charge electric vehicles from their own charging post. The smart grid will automatically start charging during periods of low price. In the future the electric vehicles might be used as a power reserve the same way as an energy storage.
- Visualize information about prices, power consumption for each electronic device and CO_2-impact.

[16]

2.5 Issues

The main problem with smart grids is the installation cost. The cost of creating a national smart grid system is projected to be incredibly high. Last year the USA congress invested $4.5 billion in an economical incentive program for

smart grids. A full build up is believed to cost at least a couple of hundred billion dollars more [17].

Even though smart grids are presented as a way to solve a lot of the problems that are associated with the power grid used today, it could potentially introduce new problems to the grid if it would be widely adopted. For a smart grid to operate effectively the need for communication infrastructure in the grid is essential. All grid users would need to know what is going on in the neighbouring cells. To make this possible huge amount of information must be sent through the grid and complex control systems need to gather and send this information. The potential threat of cyber criminals is a big concern since smart grids would need the ability to remotely disconnect power systems. Furthermore the integrity for the end user is a concern. A fully adopted smart system would imply that the power grid operator would have huge amount of information of how households are operating. If this information should be available to the power grid operator is a source of discussion. It could potentially threaten the integrity of the end user [18].

Another problem with smart grids is that the increased complexity of the system will make the average life time of the components in the grid much shorter. A new transformation is believed to have an average life time of 40 to 50 years but introducing more complex control system where the loads vary fast the life time is believed to be reduced to 10 - 15 years. Smart grids is therefore associated with higher maintenance costs than the traditional grids used today [18].

Chapter 3

Modelica/Dymola

Dymola is a commercial modeling and simulation environment that is developed by *Dassault Systémes AB*. The name *Dymola* is a contraction of *Dynamic Modeling Language*. The program is a tool for the modeling language *Modelica*, which is an open source, object oriented modeling language for component oriented modeling of complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents. The modelica language is standardized by the *Modelica Association*, www.modelica.org. In this chapter *Modelica* and *Dymola* is introduced and its interactions with *C* and *C++* are explained. Modelica/Dymola were used to develop the models of the physical components of the smart grid and C++ were used to design the control laws for the system.

3.1 Modelica

Modelica is an object-oriented, *declarative* modeling language which the program *Dymola* is a tool for. It's developed by the non-profit organization *Modelica Association*. The *Modelica Association* provides the popular *Modelica Standard Library* which provides model components for physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents. The standard library is a free library and may be used in commercial products.[19, p. 25]

The declarative way of assigning relations differs from the most common programming languages. In languages like *C/C++*, *Java*, *PHP*, *Python* , *Ada* or *Fortran* declarations are *Imperative*, which means that statements assign values to variables. For example, in an imperative programming language the code below would be interpret as "assign the value 5 to the variable y" followed by "assign the value 9 plus the value of y to the variable x", i.e. the *left hand side* of ":=" is assigned the value of the *right hand side*.

```
1 y:=5;
2 x:=9+y;
```

This might seem like the most obvious way of constructing programs since this is the way that the hardware operates. Changing the order of the lines in the code above would not be possible since y has not been initialized yet.

In a *declarative programming language* or in *acausal modeling* there are no assignments, there are only relations. For example, the code below would be interpreted as "The product of x and y is equal to z".

```
1 x*y = z;
```

Note that there is no difference between the code above and $z = x * y$; This is the usual way we think of equations, as pure relations rather than assignments. More advanced declarations are also possible such as derivatives which is desired from a modeling point of view since most physical relations can be described by differential equations. An example of a simple *Ordinary Differential Equation*, or *ODE*, is seen in equation 3.1. The modelica implementation of the *ODE* is seen in the code below. This relation would not be possible to write in an imperative programming language.

$$T\frac{dy}{dt} + y = u; \tag{3.1}$$

```
1 der(y)+ y/T =u/T;
```

There is however other modeling and simulation tools that can achieve the property of *ODE*. A Simulink implementation of the differential equation above is seen in figure 3.1. In Simulink transfer function blocks are used rather than the differential equation.

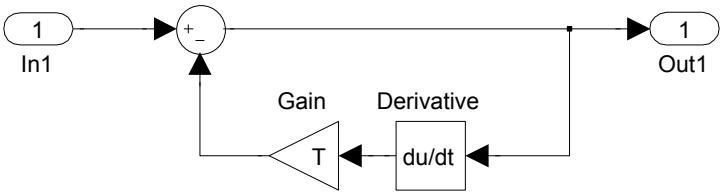


Figure 3.1: Simulink implementation of the *Ordinary Differential Equation* in equation 3.1

Using Simulink for modeling has drawbacks. It is problematic to create non-linear models in Simulink. For example, the differential equation

$$\frac{d^2y}{dt^2} + 2y\frac{dy}{dt} + y = u \tag{3.2}$$

has Simulink implementation seen in figure 3.2.

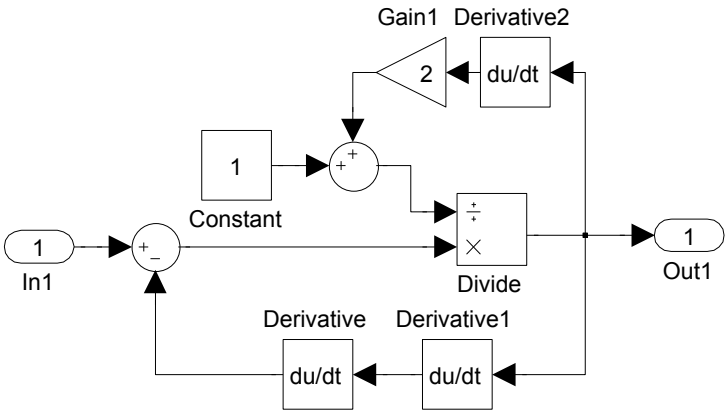


Figure 3.2: Simulink implementation of the non-linear differential equation in equation 3.2

Equation 3.2 can be rewritten by introduction the states $x_1 = \frac{dx}{dt}$ and $x_2 = \frac{d^2x}{dt^2}$ as:

$$\begin{aligned} \frac{dx_1}{dt} &= x_2 \\ \frac{dx_2}{dt} &= u - x_1 - 2x_1x_2 \end{aligned} \tag{3.3}$$

The corresponding Modelica code is seen below.

```
1 der(x1) = x2;
2 der(x2) = u-x1-2*x1*x2;
```

This is a very convenient way of declaring advanced differential equations.

Another problem is the lack of unit handling. In Simulink all signals between blocks are unit less. For example modeling a gear box which has two inputs, the ratio and a rotational speed. Simulink will not notice nor give any warnings if one of the inputs is a Voltage. In Dymola/Modelica variables and parameters have units to avoid this problem.

3.1.1 Electric Power Library

Electrical Power Library, or EPL, is a Modelica library developed by *Modelon AB* that provides a framework for modelling and simulation of power systems in three or one phase AC and DC. EPL provides standard connections and components which makes it possible to easily design electrical systems in a drag and drop manner. All physical modelling in this thesis was done using the Electric Power Library. The library provides components to model power systems in both transient and steady state mode. The EPL has special connectors. All connectors have a flow, current, and a potential, voltage which obey to Kirchoff's laws (sum of currents are zero and the potential is equal). The following setup provides a small example of how EPL could be used.

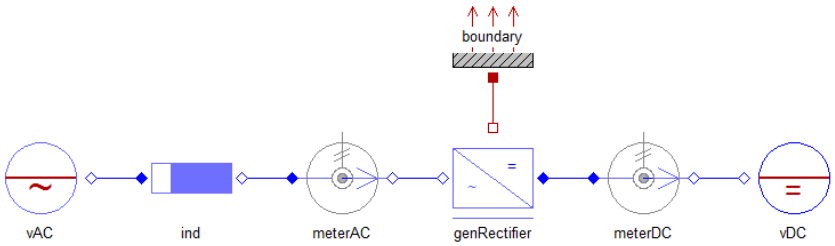


Figure 3.3: EPL rectifier example

In the example from figure 3.3 a rectifier is used to convert AC to DC. This example is set up by an AC voltage source, leftmost position in the figure, that is connected to the rectifier through an inductor and an AC meter. On the right hand side the rectifier is connected to a DC meter and a DC source, left most in the figure, which acts as a boundary condition for the problem. The AC and DC meters are diagnostic component which are used to check the AC and DC level before respectively after the rectifier. The rectifier is also connected to a boundary component through a thermal connector which models heat loss in the rectifier. Connecting two components in EPL means that the current and potential are set to be equal in the common node. This provides an example of how easy it is to design your own power system and test it by dragging and dropping components in *Dymola*.

3.2 C/C++ using External Static Libraries

It is common to use *libraries* for functions or methods that are used often. For example a function that counts the number of characters in a string might be used in a great number of programs and thus it is desirable to have a precompiled library rather than having to include the function code every time a program is counting characters. There are essentially two kinds of libraries; static and shared libraries. A new instance of the static library will be created for every program that uses the library. Several programs might access the same instance of a dynamic library. Thus static libraries will be the choice when interacting with Dymola.

External Static Libraries is a way of accessing functions written outside *Dymola* with memory attached to it. The idea is to have a compiled library of functions which are accessed through an interface in *Dymola*. In the *Modelica* code an interface is written to pass arguments from *Dymola* to the *C* header file which in its turn have access to the library. The methodology is described by a simple example where an External Static Library is used to multiply two numbers outside *Dymola*.

In *modelicaFunction.mo* the modelica function is defined. It takes two real inputs and calls the function product in *product.h* with the two real inputs and returns one real output. The annotation defines the names of the header file and the library. In this case the *product.h* and *product.cpp* are compiled to a library called *ext*.

Listing 3.1: modelicaFunction.mo

```
function product "Multiply two numbers"
        input Real x, y;
        output Real prod;
        external "C" prod = product(x,y);
        annotation(Include="#include <product.h>", Library="ext");
end product;
```

In *interfaceDymola.h* the function *product* is wrapped as a C function [20] and calls the *C++* function product.cpp. The code written in the header file must be in the *C* subset of *C++* since *Dymola* only supports external function calls from *C* and not *C++*. In listing 3.3 the simple product function is seen.

Listing 3.2: interfaceDymola.h

```
#ifndef PRODUCT_H
#define PRODUCT_H

#ifdef __cplusplus
extern "C" {
#endif
  extern double product(double x, double y);
#ifdef __cplusplus
}
#endif
#endif
```

Listing 3.3: product.cpp

```
#include <product.h>
double product(double x, double y)
{
  return x * y;
}
```

This shows a simple example of how functions written outside Dymola can be called using External static libraries. It is also possible to use *External objects* with Dymola, see 3.3 for more information.

3.3 C/C++ using External Objects

External functions sometimes need to access internal memory like objects in object oriented programming.This can be done in *Dymola* by using external

objects. An external object should extend the subclass *ExternalObject* and must have two local external functions: a constructor and a destructor. The constructor is called automatically in *Dymola* once for each object when the simulations starts and outputs the external object. The external object is mapped in *C* as a void pointer. The destruction is automatically called once for each object at the end of the simulation and has exactly one input, the external object. The external object can be passed to other external functions to access the internal memory.
In the following section a small example is described where a block is implemented in *Dymola* that can store and output a vector using external objects. In listing 3.4 the external object is defined as a subclass to *ExternalObject* and the constructor and destructor is defined with inputs and outputs. Other external functions that access the same internal memory has the external object as input, which can be seen in listing 3.5.

Listing 3.4: Resources.mo

```
class MyBlock
  extends ExternalObject;

  function constructor
    input Integer[:] inVec;
    output MyBlock obj;
    external"C" obj = initBlock(inVec, size(inVec,1));
      annotation(Library="inOutBlock", Include="#include \"interfaceDymola.h\"",
      IncludeDirectory="modelica://vector/Resources/Include/",
      LibraryDirectory="modelica://vector/Resources/Library/");
  end constructor;

  function destructor
    input MyBlock obj;
    external"C" closeBlock(obj)
     annotation(Library="inOutBlock", Include="#include \"interfaceDymola.h\"",
      IncludeDirectory="modelica://vector/Resources/Include/",
      LibraryDirectory="modelica://vector/Resources/Library/");
  end destructor;
end MyBlock;
```

Listing 3.5: otherFunctions.mo

```
function getBlock
  input MyBlock obj;            // reference to the external object
  input Integer size;           // size of the output vector
  output Integer[size] outVec;
  external"C" getBlock(obj,outVec,size);
end getBlock;
```

When all the external functions are defined in *Dymola* and the *C* structure that implements the function are written, *Dymola* is ready to use the external functions. In this thesis *C++* has been used. Since *Dymola* only supports external objects written i *C* an interface class has been written to interact with the *C++* code. This interface can been seen seen in listing 3.6 and 3.7.

Listing 3.6: interfaceDymola.h

```
#ifndef __INTERFACEDYMOLA_H
#define __INTERFACEDYMOLA_H

#ifdef __cplusplus
extern "C" {
#endif

  typedef struct Block Block;
  void* initBlock(int* inVec, int size);
  void closeBlock(void* obj);
  void getBlock(void* obj, int* myPointer, int size);

#ifdef __cplusplus
}
```

```
15 #endif
16 #endif
```

Listing 3.7: interfaceDymola.cpp

```
1  #include "Block.h"
2  #include <stdlib.h>
3
4  extern "C" {
5    void* initBlock(int* inVec, int size){
6      Block* myBlock = new Block(inVec,size);
7      return (void*) myBlock;
8    }
9
10   void closeBlock(void* obj){
11     Block* myBlock = (Block*) obj;
12     if(obj == NULL)
13       return;
14
15     myBlock->closeBlock();
16     free(myBlock);
17   }
18
19
20   void getBlock(void* obj, int* myPointer, int sizeIn){
21     Block* myBlock = (Block*) obj;
22     myBlock->get(myPointer, sizeIn);
23   }
24 }
```

The purpose of the interface is to call *C++* function from C code and typecast the objects such that it fits both the *C++* code and the *Dymola* functions. Since the external object is mapped as a void pointer in *C* one must cast the return object from the *initBlock()* function to a void pointer. Furthermore the *closeBlock()* and *getBlock()* function must typecast the input object to be a block pointer to be able to use Block class functions.
The resulting code for the block in *Dymola* that can store and output a vector can be see in listing 3.8. At line 6 an external object is initialized called *myBlock* and the input vector k is stored in *myBlock's* data structure. Using the function *getBlock()* and passing the external Object the vector k is returned and outputted.

Listing 3.8: vectorBlock.mo

```
1  model vectorBlock
2
3  parameter Integer k[:];
4  Integer out[size(k,1)];
5
6  vector.Resources.MyBlock myBlock = vector.Resources.MyBlock(k);
7  Modelica.Blocks.Interfaces.RealOutput y[size(k,1)]
8      annotation (Placement(transformation(extent={{90,-10},{110,10}})));
9
10 equation
11   out = Resources.getBlock(myBlock, size(k,1));
12   y=out;
13
14 end vectorBlock;
```

Chapter 4

Smart Cell

A cell is built up by a number of houses and generators that are connected together using a local medium voltage grid. This grid is controlled by a system controller which main task is to maintain voltage on the grid to provide the connected houses with power. The houses connected to the grid are also controlled by a system controller that manages the micro generation and loads. This chapter introduces the concepts of smart cells and the modelling of these. The models were built up by a number of components which will be presented in this chapter. The process of putting these components together to form a smart cell will also be explained.

4.1 Concepts

A smart house is a house that is fully integrated with the smart grid. It communicates with the grid how much power it generates and consumes. Using this information the house can decide how much power should be sold to the grid during times of overproduction and how much power can be bought from the grid during times of low micro generation in households, without violating the grid conditions. Each smart house is equipped with a *System Controller*, which continuously gathers information from the house[21]. Using this information the system controller makes decisions to make sure that the internal grid conditions are not violated and the micro generation units are used in an effective way. This system can be combined with smart loads such as dishwasher and other electric appliances that can be controlled by the system controller to run during times of over production and/or when the electricity price is low. Smart houses should be able to integrate and control solar panels, wind turbine and batteries on a household level to make sure that these units are used in a cost efficient way and that the internal DC bus is operating without violating the grid conditions. Many believe that battery equipped cars would be the perfect energy storage for the smart house. Using these already available batteries the consumer will not need to invest in extra storage and the system controller would make sure that the car is fully charged when needed. Furthermore when the car is connected to the house it can be used as an energy buffer for the micro generation units to more effectively be able to use these energy sources. A schematic idea of the smart house is seen in figure 4.1. [22]

Connecting these houses to a local medium voltage grid that is controlled by system controller forms a smart cell. The system controller on this level controls a number o generators, such as wind turbines or solar panels, a battery and a number of houses. The main task for the controller on this level is to maintain voltage on the grid and provide the houses with power. A schematic idea of smart grids is seen in figure 4.1.

4.2 Modeling

The houses are modelled using a DC bus which connects all loads, generators and batteries. The local medium voltage grid is modelled using DC as well. Using DC has the advantages that it simplifies the computations for the simulation tool, i.e. Dymola. The modelling of the internal bus and the medium voltage grid could be done in AC as well. But this modelling was concentrated on the control structure of a smart cell and therefore DC chosen. Houses and the medium voltage grid are equipped with a central *system controller* which controls all resources. A resource is a generic description of either a load, generator, battery or an external connection. External connections are used to connect houses to the external medium voltage grid.

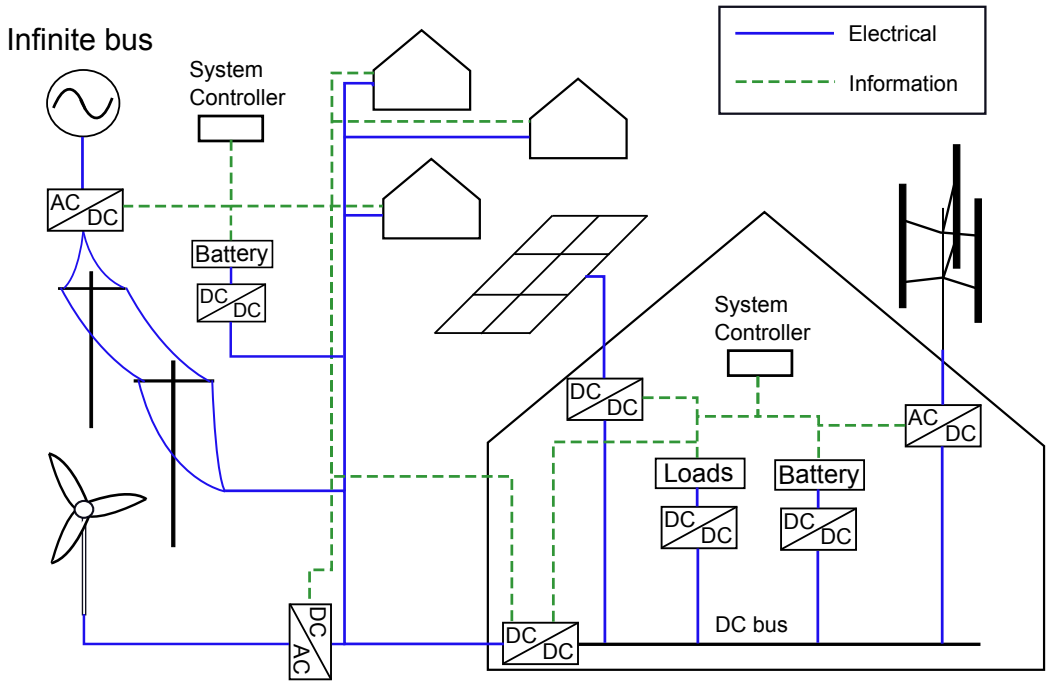


Figure 4.1: Overview of the smart grid. All houses are equipped with a battery, a solar panel, a small scale wind turbine and a number of loads. The external grid is equipped with generators and an energy storage.

4.2.1 Internal DC Bus and medium voltage grid

The internal DC bus and the medium voltage grid were modelled using EPL standard component for DC capacitor and a standard DC ground component. This to model the dynamical behaviour of the grid voltage depending on the current power balance on the grid. The voltage for the DC grid would vary in the following way if the power is not balance on the grid.

$$\begin{aligned} \Delta P(t) &= P_{in}(t) + P_{gen} - P_{con}(t) \\ v &= \frac{1}{C} \int \frac{\Delta P(t)}{v} dt \end{aligned} \tag{4.1}$$

Where the P_{in} is the power in from the external connection, P_{gen} the power generated from the generators, P_{con} the power consumed, C the capacitance of the grid, v the grid voltage and t the time variable. The *Dymola* figure representing this model can be seen in figure 4.2.

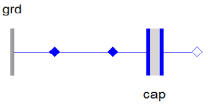


Figure 4.2: External DC grid and internal DC bus icon in EPL.

4.2.2 Converter

All resources uses a converter to connect to the DC grid. The converter is used to be able to connect different voltage levels to the grid. The model uses the power balance equation (equation 4.2), where i_1 and v_1 are the current and voltage from the grid and i_2 and v_2 are the current and voltage on resources side. The power balance is assumed to be perfect, i.e. now power losses. Furthermore the exchange unit has the possibility to set the current on the grid side, since the current that flows through the unit need to be controlled. This current is set to be an external input to the exchange unit. The *Dymola* icon representing this model can be seen in figure 4.3.

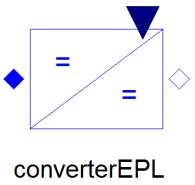


Figure 4.3: Converter figure in EPL

$$v_1 \cdot i_1 = v_2 \cdot i_2 \tag{4.2}$$

4.2.3 Battery

The battery is modelled as a capacitance. The usage of the *Electric Vehicle* during day time is modelled as two switches and two resistances to allow the battery to discharge during daytime. The *Dymola* model of the battery used can be seen in figure 4.4. The pulse is used to control the switches. By pulsing

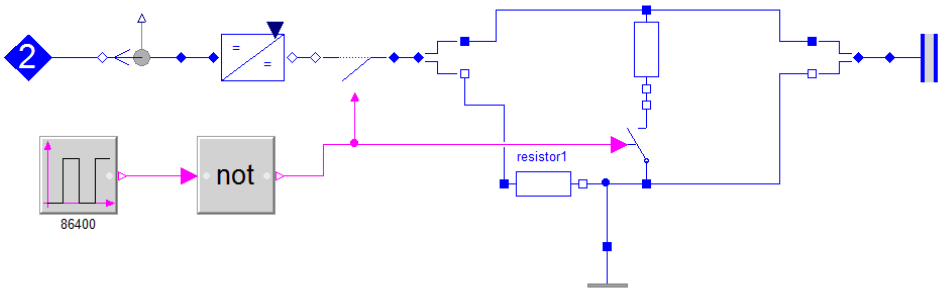


Figure 4.4: Internal DC grid figure in EPL

with a period of a day it is possible to discharge during daytime and connect the battery to the system, to charge, during night. The energy stored in a capacitance is given by equation 4.3.

$$E_{stored} = \frac{1}{2}CV^2 \tag{4.3}$$

Where C is the capacitance and V the voltage. The capacitance is therefore chosen rather big to model a car battery's energy capacitance.

The energy left in the battery is often given as percentage of the maximum energy that can be stored in the battery. This value is called *State of Charge*, or *SOC*, and is calculated with equation 4.4 where $E_{current}$ is the current energy level in the battery and $E_{maximum}$ is the maximum energy that can be stored in the battery.

$$SOC = \frac{E_{current}}{E_{maximum}}, \quad 0 \leq SOC \leq 1 \tag{4.4}$$

4.2.4 Load

The model of a smart load is using standard *EPL* components: a one terminal variable resistor which is connected in parallel to the grid through an *EPL* switch, which makes it possible to turn on and off the load giving an external connection. The one term variable resistor is modelled using *Ohm's law* with a resistance that may varied during the simulation with the input. The switch implements a continuous transition from closed to open with a very low conductivity when the switch is open and a very low resistance when the switch is closed. The *Dymola* figure representing this model can be seen in figure 4.5.

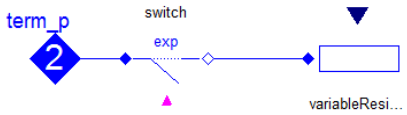


Figure 4.5: Smart load figure in EPL

4.2.5 Generator

The model of a generator uses a converter, from section 4.2.2, and an EPL voltage source. The voltage source is an infinite power source and the converter is used to control the current outputted from the generator. The *Dymola* figure representing this model can be seen in figure 4.6. The control current to the converter can be mapped to power using either equation 4.5 or 4.7.

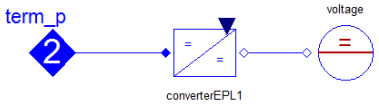


Figure 4.6: Generator figure in EPL

Small Scale Wind Power

The amount of power generated from a wind power plant is given by equation 4.5 [23].

$$P_w(U) = \begin{cases} 0, & U < U_{min} \\ \frac{1}{2}\rho A U^3 C_p(\lambda, \beta), & U_{min} < U < U_{rat} \\ P_R, & U_{rat} < U < U_{max} \\ 0, & U_{max} < U \end{cases} \tag{4.5}$$

where ρ is the air density (kg/m^2), A area of the wind turbine (m^2), U wind speed (m/s), C_P the coefficient of power and P_R is the rated power (W). U_{min} or *Cut-in wind speed* is the minimal wind speed for which the power plant generates electricity, U_{rat} is the *Rated output wind speed*, U_{max} or *Cut-out wind speed* is the maximum wind speed for which the plant generates power . For simplification the following parameter values has been chosen:

$$\begin{aligned} \rho &= 1.225 \ \text{kg/m}^3 \\ C_P(\lambda, \beta) &= 0.33 \end{aligned} \tag{4.6}$$

The modelica implementation of the Wind power plant is seen in section B.2

Photovoltaic System

The amount of *Photovoltaic* generation, P_{PV}, can be calculated by

$$P_{PV} = \eta_{PV} n_{PV} S_{PV} I_{PV} \left(1 - 0.005 \left(t_{CR} - 25\right)\right) \ \ [\text{kW}] \tag{4.7}$$

where η_{PV} is the conversion efficency of the solar cell, n_{PV} is the number of solar panels, S_{PV} is the array area, I_{PV} is the solar radiation (kW/m^2), t_{CR} is the outside air temperature (°C) [4]. The modelica implementation of the Photovoltaic cell is seen in section B.1.

4.3 Functionality

To be able to control the system in an efficient manner the resources are equipped with some functionality, such as priority and delay time. This functionality will help the system controller to keep track of its resources and control them in a satisfactory way. [21] The resources are equipped with the following functionality:

4.3.1 Load

All loads have a consumer signal which determines when the consumer wants to turn on the load. This signal is sent from a random pulse block. When the load receives a pulse the system tries to turn on the load. The time frame for this pulse can be set in the random pulse block and are uniformly probable to pulse during this time frame. Each load also has a parameter called *operation*

time, during this interval the load will remain turned on. It is also possible to configure the loads with a *price limit*. This means that the system controller will not turn on the load until the price has dropped below a certain price limit. A *maximum delay* is also defined. This defines the maximum time the system controller can postpone a consumer start of a load. When the delay is greater then the maximum delay the load will automatically turn on. All loads are also configured with *priorities*, which defines in which order the loads should be turned on.

4.3.2 Battery/Storage

The system controller can define a *charge price limit* and a *discharge price limit*. This will make it possible to charge the car battery during night if price differentiation is used. The battery will be used to control voltage when the house operates in islanding mode.

4.3.3 Generator

The generators are either *photovoltaic solar panels* or *small scale wind turbines*. The power generated is determined by equation 4.5 and 4.7. The power information from each generator is sent to the *system controller* in order to keep track of the total power that is available.

4.3.4 External connection

The *external connection* is a connection from one voltage level to another, eg. from *medium voltage grid* to *internal DC bus* or from *high voltage grid* to *medium voltage grid*. Its objective is to control voltage the lower of the two voltage levels. This is done with a *PI-controller* that controls the current through the external connection. The external connection is able to control the voltage level as long as there is power available on the higher of the two voltage levels. When the available power drops below a given value the external connection is no longer suitable for control and thus the battery will provide power to the house.

4.4 Smart Cell Models

The components developed in *Dymola* and described in sections 4.2 and 4.3 where used to put together models for smart cells. The cell was built up by smart houses and a distribution grid. The models were configured in *Dymola* in a drag and drop manner.

4.4.1 Smart House

The *smart house* contains a combination of *loads*, *battery/storage*, *generators* and *external connections*, see table 4.1. These resources are connected to an

internal DC bus to form a house model. The configuration possibilities can be seen in table 4.1. A house need one external connector to be able to connect to the external grid.

Loads	$0 - \infty$
Battery/storage	$0 - 1$
Generators	$0 - \infty$
External connections	1

Table 4.1: Components and the number that each smart house contains (the upper limit for loads and generators is limited by the computer memory).

4.4.2 Distribution Grid

The *Distribution Grid* were built up by one connection to *high voltage grid* and several connectors that connects *smart houses*. The external grid might optionally contain generators and storage. Using the storage the grid would be able to cope with short blackouts on the main high voltage grid. The external medium voltage grid does not have the ability connect loads. But this could easily be implemented if this functionality is desirable.

Loads	0
Battery/storage	$0 - 1$
Generators	$0 - \infty$
External connections	$2 - \infty$

Table 4.2: Components and the number that each distribution grid contains (the upper limit for generators is limited by the computer memory).

Chapter 5

System Controller

This chapter introduces the *System Controller* and explains how the controller handles resources, blackouts and grid control. The control strategy is explained and the decision making for house level and the medium voltage grid are presented using workflow figures.

5.1 System Controller

The *System Controller* is what makes the system smart. On house level it is essentially a computer connected to all resources in the house such as loads (lamps, TV, dishwasher), possible generators (wind turbines, solar panels, car batteries) and a connector to the external grid. The connections enables information exchange. The *System Controller* takes information about consumption, production and requests from the end user and acts accordingly. The *System Controller* is also connected to the *electricity market* in order to receive current electricity prices. Users configure the *System Controller* in order to achieve the desired behaviour. This configuration should be as simple as possible and could involve priority assignment for loads to assure that vital household component always are active. The configuration is optimally done through some *Graphical User Interface* preferably on a *smart phone* or *smart pad*. The medium voltage grid will essentially be controlled in the same manner. But the controller on this level will control external connections to houses instead of loads.

5.2 Control Strategy

The main objective for the *System Controller* is to always keep the voltage level, v, at the reference value, v_{ref} for the grid it is supervising. The second goal is to minimize the electricity cost. Generators such as solar panels and wind turbines will always provide 100 % of its generated power to the grid. Since net metering[1] is used there is no need to store energy from the micro generators during times of high production and low consumption. The grid can be used as a storage, all energy sold to the grid can later be bought at the same price.

On house level the system controller always tries to connect loads from a load queue. The loads are put in the queue when the consumer wants to turn it on. If more then one load wants to be turned on at the same time the load with the highest priority is considered first. Loads that have been configured with a price limit will be delayed until the price has dropped below the price limit. If the price has not dropped below the price target before the maximum delay time the system controller will force the load to turn on. If the grid conditions changes there might not be enough power available to keep the loads turned on. The strategy is then to turn off the load which has the lowest priority first. If this is not enough more loads will be turned off according to their priorities. The energy storage on both the medium voltage grid and in the house will serve as a buffer during power loss. The houses will use their external connection to the medium voltage grid to provide power and control voltage when its possible. But during blackouts when the external grid fails the energy storage serves as voltage controllers. The system controller

[1]Net metering is when a prosumer (producer and consumer) is allowed to sell power to the grid for the same price as buying.

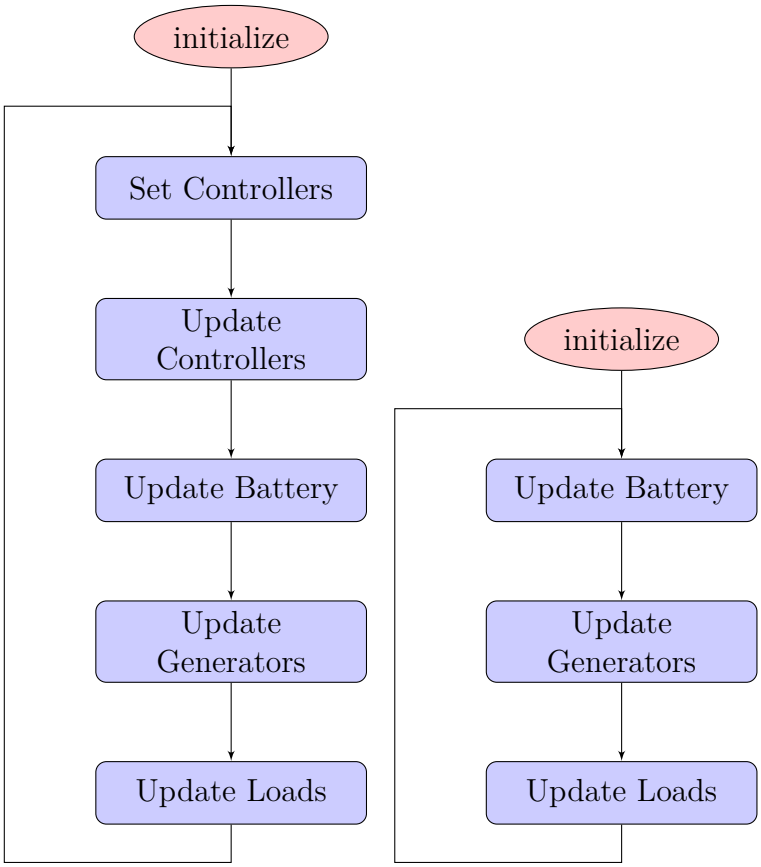
on the medium voltage grid will use its connection to the high voltage grid as voltage controller. But will also use its energy storage during blackouts on the external grid. The system controller on all levels will always strive to have a fully charged battery. If there is enough power the battery will be charged. This is prioritized higher than turning on loads.

5.3 Workflow for Smart House

Each time the *System controller* is updated the functions are called according to the *main workflow*. During this process the System Controller goes through all resources connected to its system and sends instructions to them if needed. The *main workflow* is seen in figure 5.1. For short term simulations the work flow for figure 5.1a is used and for long term simulations the workflow for 5.1b is used. The difference between long term and short term is that for long term the voltage control is not considered. Furthermore behaviour during islanding mode is not of interest. Instead the voltage is determined by an ideal voltage source and not by a *PI-controller*. By not considering the voltage control it is possible to sample the system much slower. This will make it possible to simulate the system for much longer times. Year long simulations was done to determine the systems power performance and cost. This would require a lot of computational power and time if voltage control would be done during these simulations.

When the system controller needs to determine which resource should be used for voltage control the *set controller* function is used. The decision structure for this function is seen in 5.2. Its objective is to find a suitable controller every time the system controller updates the system. The main goal is to always use the external net for voltage control if possible and thus using the battery only when the external net has a power outage. When the system controller needs to find a new suitable controller the *find controller* function is used. The workflow for this function is seen in 5.3a. It is fairly simple: if there is power available on the external net the external connection is chosen as controller otherwise the battery is chosen. Updating the PI-controllers which are used for voltage control are done by calling the *update controller* function. The function calculates the voltage error and calls a PI-controller to calculate the proper control signal.

The system controller gives the energy storage unit instructions by calling the *update battery* function. This function will only instruct the energy storage to charge when its not used for voltage control or is fully charged. Whether loads should be turned on or off is determined by the *update loads* function. Loads will be turned on if the voltage at the internal DC bus is at the correct level, at steady state and if there is power available. If there is a load in the waiting/ready queue and there is power available for this load, it is turned on. Loads that are configured with a price limit will only be placed in the queue if the price is below the price limit. Whether there is power available or not is determined by the difference between the total amount of power (external connection/battery and micro generation) and the current consumption of power



(a) Main workflow short term

(b) Main workflow long term

Figure 5.1: Work flow structure for the main loop used every time the system controller updates the system.

(sum of all individual loads and charging batteries).

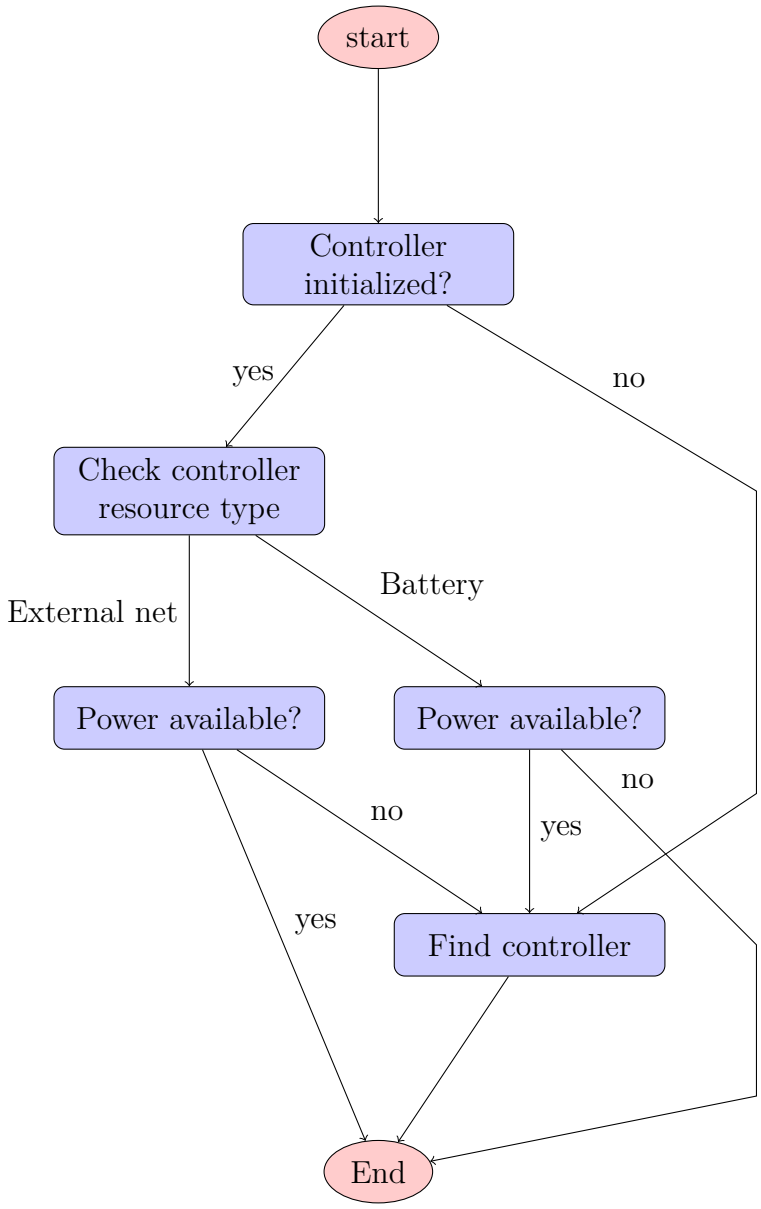
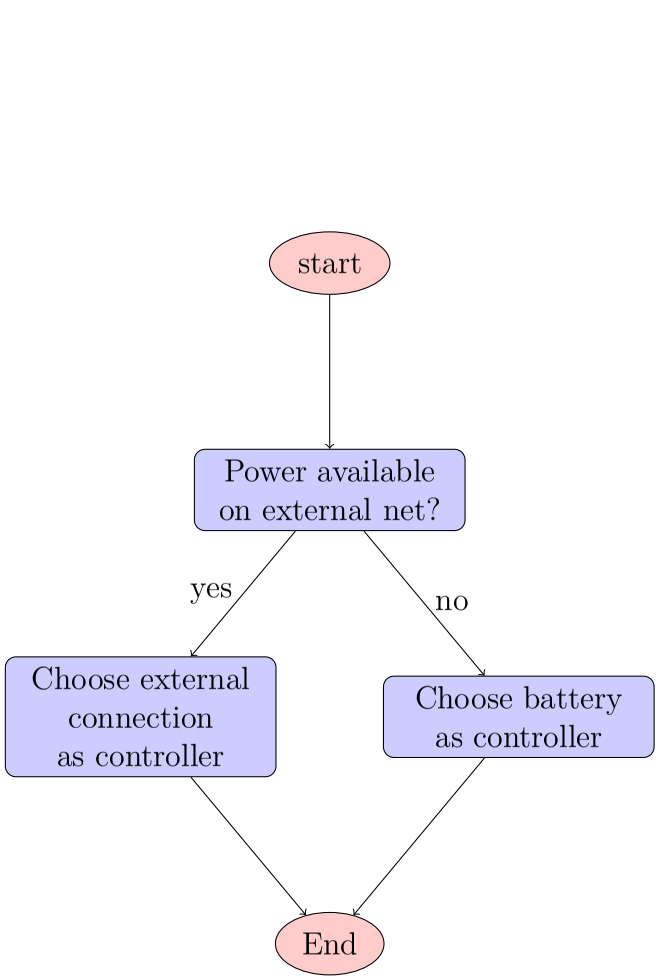
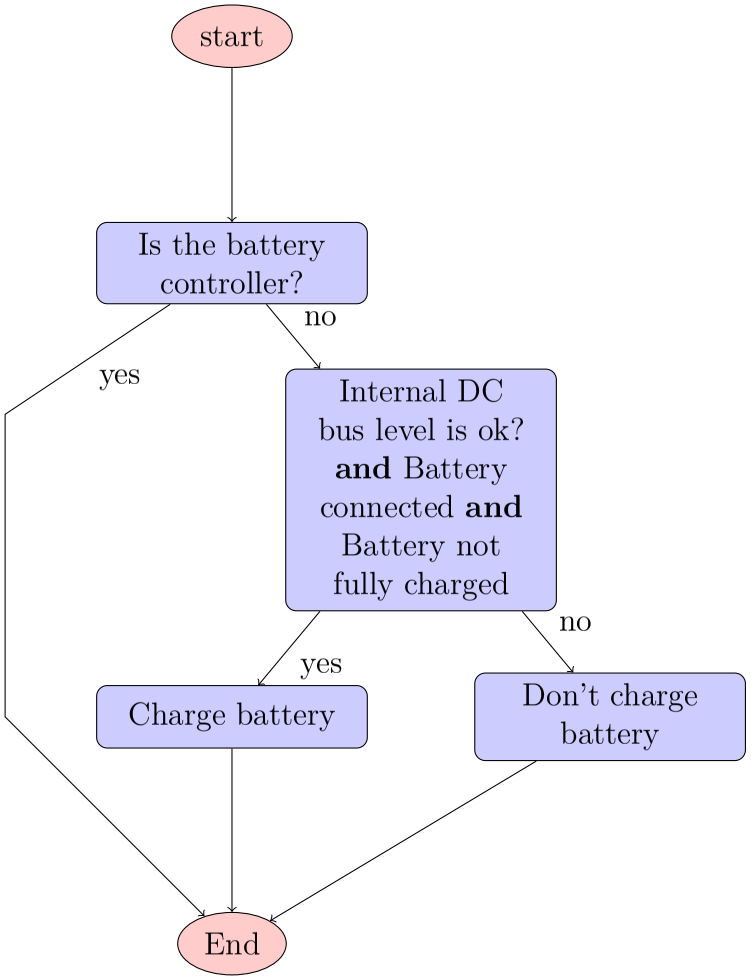


Figure 5.2: Decision structure for the Set controller function. Used by the system controller to determine if the controller used is suitable. This function is called every time the system updates.



(a) Find controller.



(b) Update battery.

Figure 5.3: Decision structure for find controller and update battery. Find controller is called every time the system controller needs to find a new controller. Update battery is called every time the system controller updates the system to determine whether the battery should charge or not.

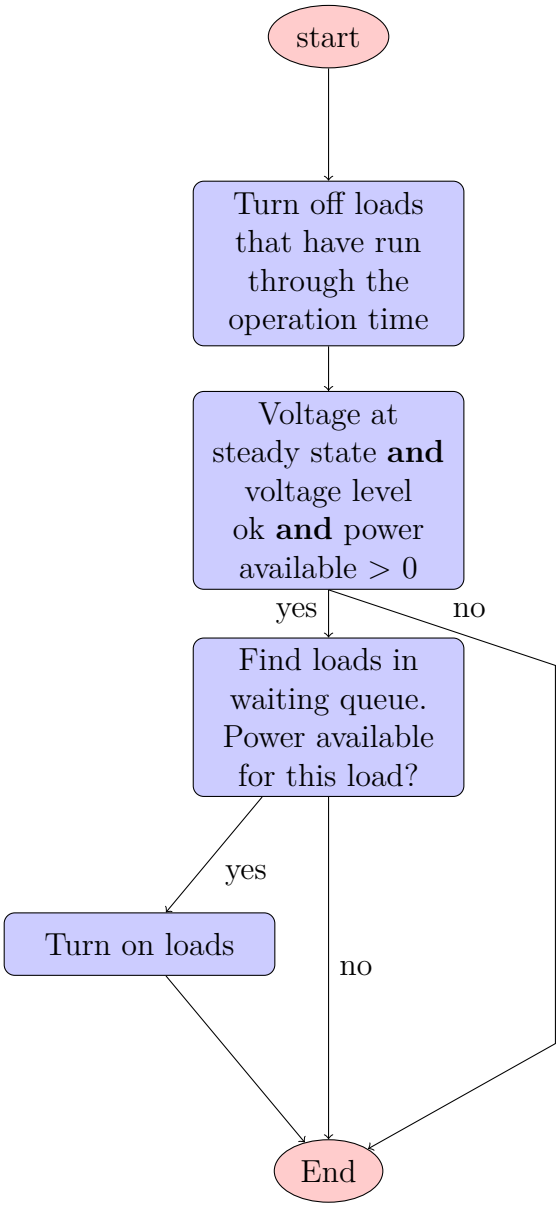


Figure 5.4: Decision structure for update loads. This function is called every time the system controller checks whether a load should be turned on or not.

5.4 Workflow for External grid

The system controller for the external grid is similar to the controller on house level. The difference lies in that no loads are connected to the system controller. Instead the controller needs to manage a number of external connections. These connection are connecting houses and other external grids to the grid. Therefore the *update loads* function are not used by the system controller on this level. Instead a function called *update external connections* is used. This function will find and choose the external connection that is con-

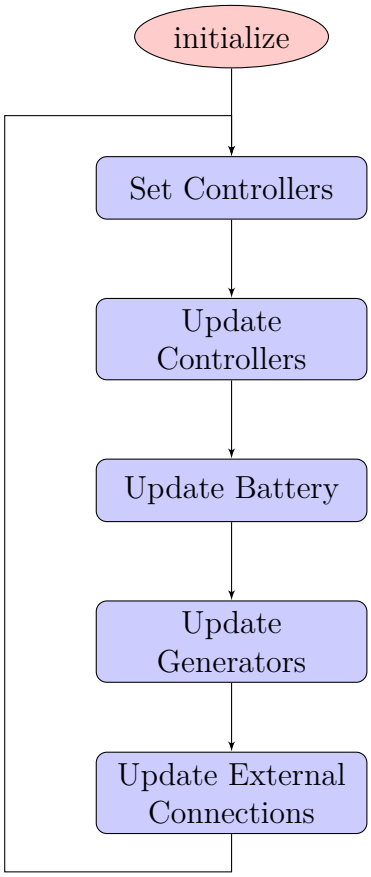


Figure 5.5: Work flow structure for the main loop used on the medium voltage grid every time the system controller updates the system.

nected to the higher voltage grid as controller for the medium voltage grid. It also takes information about the available power from the high voltage grid together with the power from the generators and energy storage (battery) and divides it equally among the rest of the external connections (houses). The main objective for the battery on this level is to act as a buffer. The power storage is charged when the electricity price is below a certain level (usually night time) but can also be configured to discharged during times of high price (usually in the evening). This is done to reduce power peaks and to reduce the overall electricity cost. Furthermore the energy storage is used for voltage control when the high voltage grid fails. The workflow for the energy storage at the external grid is seen in figure 5.6.

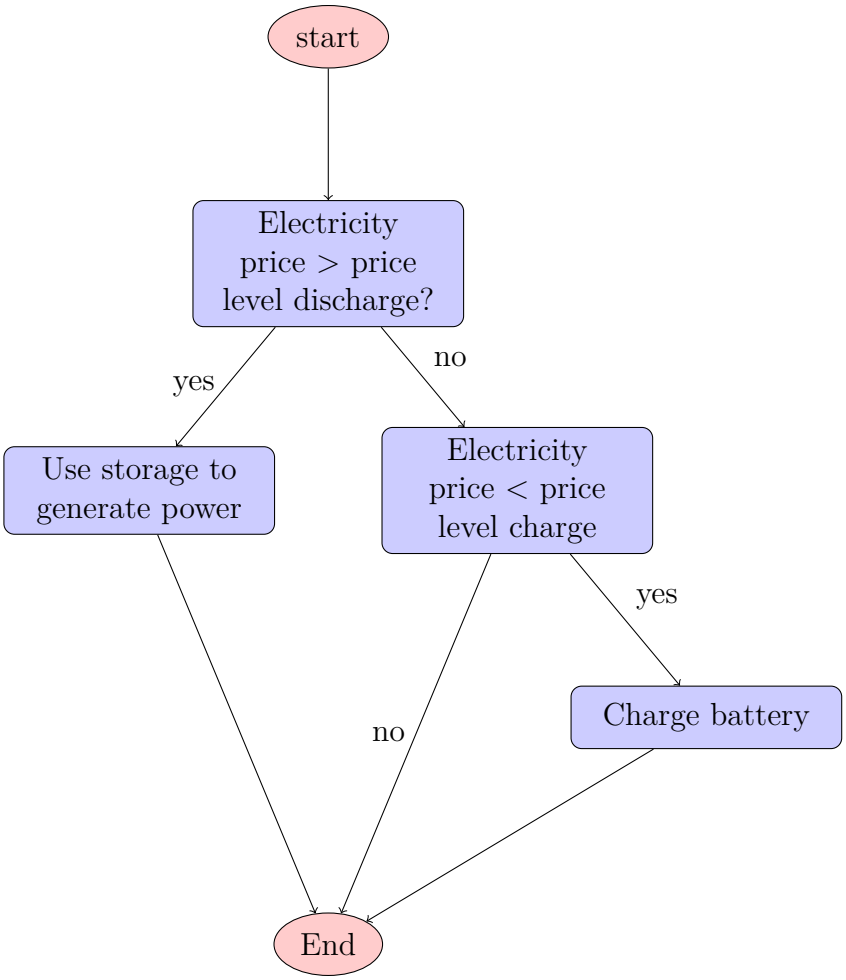


Figure 5.6: Decision structure for update storage external net. This function is called every time by the system controller on the externla grid to determine how the storage should be used.

Chapter 6

Implementation

In this chapter the general idea of how to use Dymola together with an external C++ program is presented. The data structure that was used to manage systems and resources are also presented. Furthermore the Modelica implementation is shown. Apart from this the interactions between Dymola and the control program are explained.

6.1 General Idea

The general idea is to use Dymola to model the physical parts, i.e. the grid and resources, and to use a controller program written in C++ for decision making. All resources are using *smart blocks* to send and receive information and commands from the C++ program. These blocks will be presented in detail in section 6.3. During the simulation the controller program is running in parallel with *Dymola* and is doing all controlling of the system.

6.2 C++

The controller program must keep track of how many systems that are currently running and how many resources each system is connected to. The main program contains a vector which contains all system controllers that are currently running. Each system controller contains a vector of all resources connected to it. All system controllers and resources are associated with an id (identity number) according to their position in the system's vector and the position in the vector that is controlled by the main program, starting from zero. Every resource can therefore be uniquely determined by the system controller id and the resource id. When the program need to instruct a resource to turn on or off, the program knows the resource id and system id and can therefore find the resource in the data structure and can instruct it to output the necessary signals to *Dymola*. This data structure can be seen in figure 6.1.[21]

The data structure is implemented using the following three classes: *SystemControllerInter*, *SCADA* and *Block*. The *SystemControllerInter* class manages the vector with systems and is the only class that implements functions that are called from Dymola. These functions are interfaced with *Dymola* using a *C interface*, see section C.1. Each system is an instance of the *SCADA* class and manages a vector with all the resources connected to it. Futhermore this class implements all the control functions acting on the system. These functions can be seen in section C.2.[21]

All resources that are connected to a system are an instance of the *Block* class. The code for this class can be seen in C.3. Each *Block instance* holds a configuration vector, a get vector and a set vector and some internal variables. The get vector is used to send information from *Dymola* to the controller program and the set vector is used by the controller program to send information to *Dymola*. The internal variables are used to store information that is only necessary for the controller program to know of, i.e. if the current resource is operating in controller mode or not. How Dymola is receiving and sending information is explained in detail in the next section.[21]

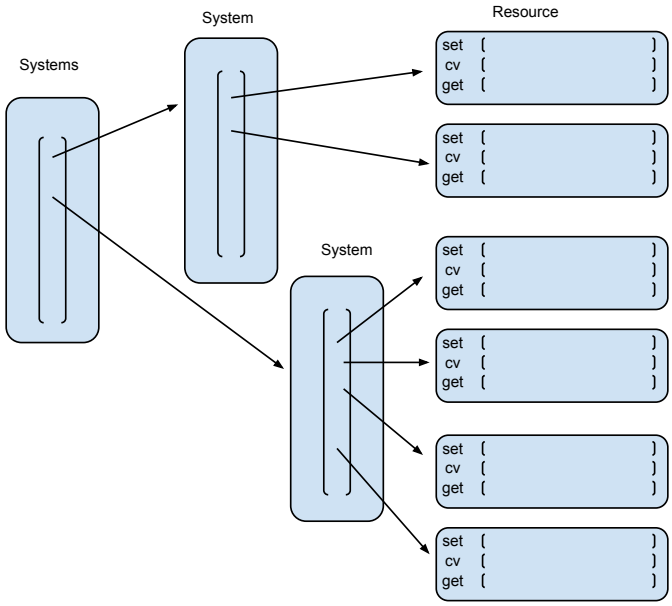


Figure 6.1: Schematic figure over the data structure used by the controller program. The program keeps track of the systems using a vector of systems and each system has a vector of resources to keep track of its resources.

6.3 Dymola implementation

The smart house can be configured in a drag and drop manner with different number of resources, see 4.4. A typical configuration for a smart house in *Dymola* can be seen in figure 6.2. The house is equipped with two generators, one battery, one external connector and three loads. The system controller, is connected to each resource and all resources are connected to the internal *DC bus*. The inputs to the house is the id from the system controller. The system controller controls the medium voltage grid and uses the shared external connector unit and an input for the price information. This could be used by the controller unit to determine when it is time to turn off or on loads. The house also has an *(EPL) DC converter* to connect the house to the medium voltage grid and a price output signal, which gives information of what the total electricity cost has been. The total price is calculated by integrating the product of the price signal and the power going through the external connector.

6.3.1 System Controller

The Modelica code for the system controller can be seen in listing 6.1. From line three to 18 some inputs, outputs, functions and parameters are defined. Inside the algorithm block from line 19 to 28 the system controller operation is defined. At the first sample the system controller block calls the function init_SystemControllerInter which calls a function in the controller program that initializes the system and returns its id to Dymola. This id is

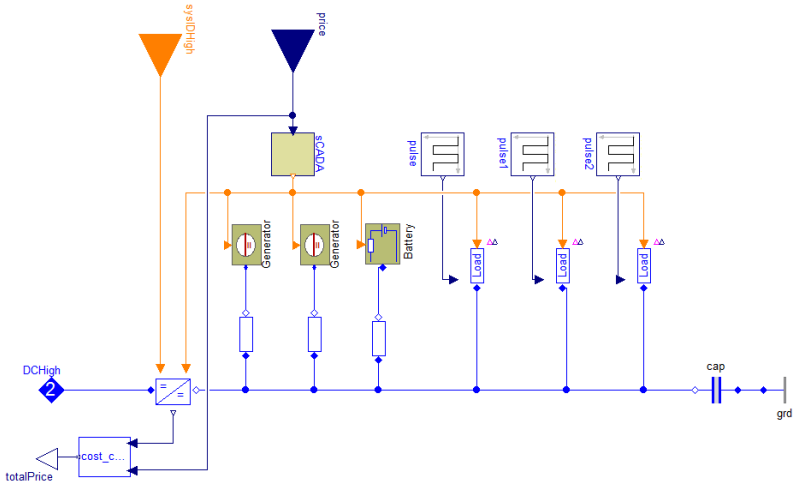


Figure 6.2: Overview of an typical Modelica model of a smart house with two generators, one battery, one external connector and three loads.

then outputted from the block using the integer output sysID. At all other sample points except first and second the scada blocks calls the function *update_SystemControllerInter* which calls an updatefunction to run in the controller program. This causes the system to update.

Listing 6.1: scada.mo

```
1  model SCADA
2  public
3    Modelica.Blocks.Interfaces.IntegerOutput sysID
4    Modelica.Blocks.Interfaces.RealInput price
5
6  extends systemController.Blocks.MyDiscreteBlock;
7
8  import init_SystemControlInter =
       systemController.ExternalFunctions.SystemControlInter_init;
9  import update_SystemControlInter =
       systemController.ExternalFunctions.SystemControlInter_update;
10
11 parameter Real voltageRef=10 "Refernec value for the voltage";
12 parameter Real toleranceVoltage=0.05 "Tolerance for turning on loads";
13 parameter Real tolerancePower=0.05 "Tolerance for power";
14
15 protected
16   Integer id1;
17   Real cv[:] = {voltageRef, toleranceVoltage, tolerancePower};
18   constant Integer signalsSize= 3;
19 algorithm
20   when firstTrigger then  // first sample
21    id1 := init_SystemControlInter(cv, signalsSize);
22   end when;
23
24   when sampleTrigger and  not firstTrigger and not secondTrigger then  // third sample
25     update_SystemControlInter(id1, price);
26   end when;
27
28   sysID :=id1;
29
30 end SCADA;
```

6.3.2 Resources

All resources, generators, batteries and loads, are equipped with a generic smart block. The Modelica model of a generic generator can be seen in figure 6.3. The implementation of the generator will be presented here as an example, loads and batteries are implemented in a very similar way.

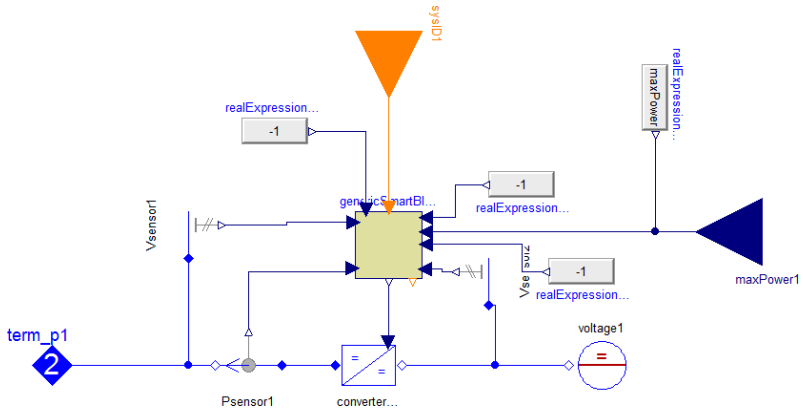


Figure 6.3: Overview of the Modelica model of a generic generator.

All resources are built up around the generic smart block, which can be seen in figure 6.3. The model feeds a number of measurements and signals to the generic smart block. Since the same block is used for all resources some measurements and signals will not be used by the generator and is therefore set to -1, such as the resistance signal. The generic smart block outputs a control current to the DC/DC converter and a breaker signal, which is used to turn off and on loads. The Modelica code for the main algorithm of the generic smart block can be seen in listing 6.2, the complete Modelica implementation can be seen in appendix B.3.

Listing 6.2: The main algorithm for the genericSmartBlock.

```
1  model genericSmartBlock
2
3  ....
4
5  algorithm
6      when initial() then
7        // nothing
8      end when;
9      when sampleTrigger and secondTrigger then
10       resourceID := init_Block(sysID, cv, signalsSizeIN, signalsSizeOUT);
11     end when;
12
13     when sampleTrigger and  not firstTrigger and not secondTrigger then
14       // -- set singnals out
15       signalsOUT := {pre(voltageGrid), pre(powerOUT), pre(voltageExternal), pre(SOC), pre(
             maxPower), pre(resistance), -1, pre(consumerProfile)};
16       set_Block(sysID, resourceID, signalsOUT);
17       // -- set signals in
18       signalsIN       := get_Block(sysID, resourceID, signalsSizeIN);
19       breaker         := integer(signalsIN[1]);
20       controlCurrent := signalsIN[2];
21     end when;
22 end genericSmartBlock;
```

Since the resources need to know the system id before they can initialize it is initialized at the second sample, see line 10, the system controller is initialized at the first sample. For the rest of the simulation time the block gathers the measurements and external signals and calls the set_Block function to send the signalsOUT vector to the controller program. When this is done the block calls the get_Block to receive signals from the controller program and outputs this to the control current signal and the breaker signal.

6.3.3 External Connection

The external connector is used to connect houses with external grids and external grids with with other grids. Therefore this resource is a shared resource. Both systems, the house controller and the grid controller need to be able to send and receive signals and measurements to the external connector. The Modelica model of this can be seen in figure 6.4.

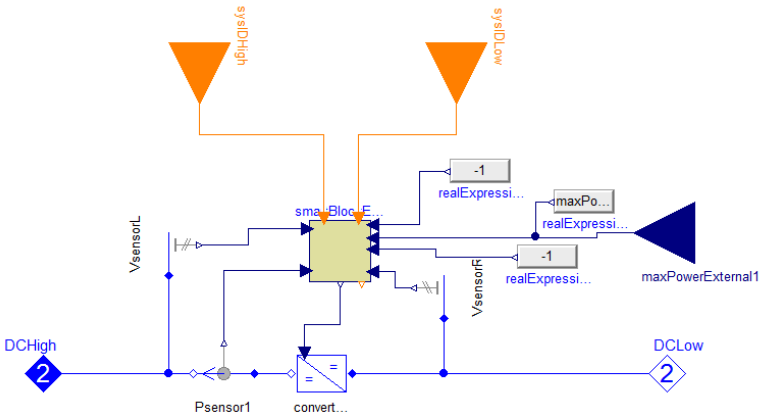


Figure 6.4: Overview of the Modelica model of an external connector.

The Modelica model of the external connector has the same signals and measurements as the resource block but has one additional system id. The external connection is shared between two system and is therefore fed with two system ids. Both systems need to be able access it. Furthermore notice that the block has two *(EPL) DC connectors*, one connecting to the high level grid and one connecting the low level grid. The Modelica code for the main algorithm can be seen in listing 6.3, the complete Modelica implementation can be seen in appendix B.4.

Listing 6.3: The main algorithm for the external connector block.

```
1  model smartBlockExternal22april
2
3  ...
4
5  algorithm
6      when initial() then
7        // nothing
8      end when;
9
10     when sampleTrigger and secondTrigger then
11       resourceIDHigh := init_Block(sysIDHigh, cvHigh, signalsSizeInHigh,
             signalsSizeOutHigh);
12       resourceIDLow  := init_Block(sysIDLow, cvLow, signalsSizeInLow, signalsSizeOutLow);
13
14     end when;
15
16     when sampleTrigger and  not firstTrigger and not secondTrigger then
17     // first get the power blanace information form the high voltage grid
18     signalsINHigh       := get_Block(sysIDHigh, resourceIDHigh, signalsSizeInHigh);
19
20     // set this information to the low voltage side of the grid
21     signalsOUTLow    := {pre(voltageLow), pre(powerOUT), pre(voltageHigh), pre(SOC), min(
           pre(maxPower),signalsINHigh[3]),    pre(resistance), signalsINHigh[3]};
22     signalsOUTHigh   := {0, 0, 0, 0, pre(maxPower),  0};
23
24     set_Block(sysIDLow, resourceIDLow, signalsOUTLow);
25     set_Block(sysIDHigh, resourceIDHigh, signalsOUTHigh);
26
27     // get the signals from the low voltage external block!
28     signalsINLow        := get_Block(sysIDLow, resourceIDLow, 2);
29     breaker             := integer(signalsINLow[1]);
```

```
30      controlCurrent     := signalsINLow[2];
31      end when;
32  end smartBlockExternal22april;
```

At the first sample two instances of the block is created, line 11 and 12. One for the system controller controlling the high voltage grid and one for the system controller controlling the low voltage grid. When the initialization is done, after sample point two, the block will call the function get_Block for the high voltage system to receive data from this system. The high voltage system will provide the low voltage system with information of how much power is available for this system to take off the high voltage grid and still keep the power balance in the grid. This information exchange between the high voltage and low voltage system is done in line 24 and 25 where the blocks uses the set function. The low voltage system then call the get function to get the control signals from the control signals in line 28.

6.4 Dymola and C++ running together

Dymola is using a static library file where all functions that interacts with Dymola are defined. The data flow between the *Dymola* simulation and the controller program can be seen in figure 6.5. At every sample point the smart block calls the get and set function to send respectively get information from the controller program.

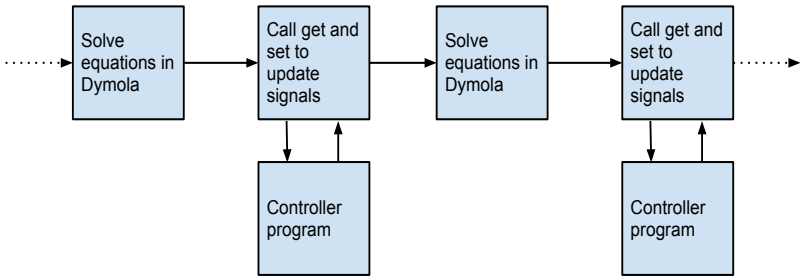


Figure 6.5: Schematic figure over how Dymola and the controller program run together.

Chapter 7

Scenarios

In this chapter the scenarios that were used to test the functionality of the system controller are presented. There are two main scenarios, one short term and one long term. The short term scenario was chosen to test how the system reacts to fast changes in the external high voltage grid. This scenario aims to test the robustness of the system, how it handles grid blackouts. By letting the high voltage grid fail the system is forced into islanding mode and need to rely on its energy storage to maintain voltage and deliver power to the grid. The long term scenario was designed to test the power performance of the system and the economical aspects of this. By introducing price differentiation, lower electricity price during night, it would be cheaper to run loads during night time. By allowing the system controller to delay certain loads to run during the night, power peaks can be reduced and the cost of running these loads will be reduced. This scenario was simulated for four different houses with different micro generation.

7.1 Short term scenario:

This scenario was designed to test if the external medium voltage grid and the smart houses were able to adapt to changing conditions of the external high voltage grid. At $t_0 = 0$ s the medium voltage grid and the houses internal grids start from $v = 0$ V. The voltage at both the medium voltage grid and the houses internal grids were controlled to the desired voltage level. At $t_1 = 80$ s the external high voltage grid fails and the medium voltage grid goes into islanding mode, i.e. operates without any external connection. The system controller at this level is forced to use all its resources to try to maintain the voltage level.

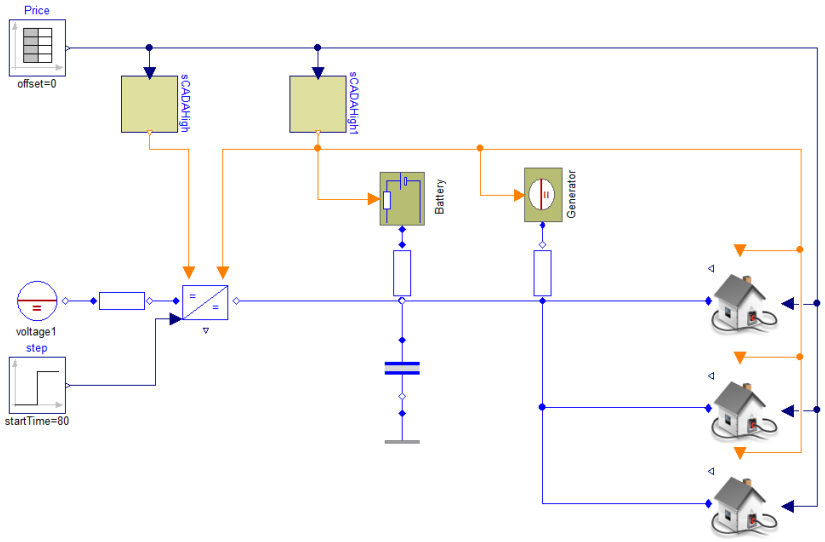


Figure 7.1: Overview of the Modelica model used to simulate the short time scenario.

At the medium level grid there are three houses connected as well as a battery and a small generator, the generator represents a wind power plant. This configuration can be seen in figure 7.1. The houses have two small generators representing a small wind turbine and a solar panel. They are also equipped with a battery i.e. a battery equipped car connected to the house. Each house is also equipped with three loads. The setup can be seen in figure 7.2. The following configuration has been chosen to highlight the control strategy the system controller uses to maintain voltage level and power loads.

- **House1:** Generator 1: 700 W, Generator 2: 700 W, Battery: $P_{max} = 1$ kW, $E_{max} = 0.46$ kWh
- **House2:** Generator 1: 300 W, Generator 2: 400 W, Battery: $P_{max} = 1$ kW, $E_{max} = 0.46$ kWh
- **House3:** Generator 1: 0 W, Generator 2: 0 W, Battery: $P_{max} = 1$ kW, $E_{max} = 0.46$ kWh

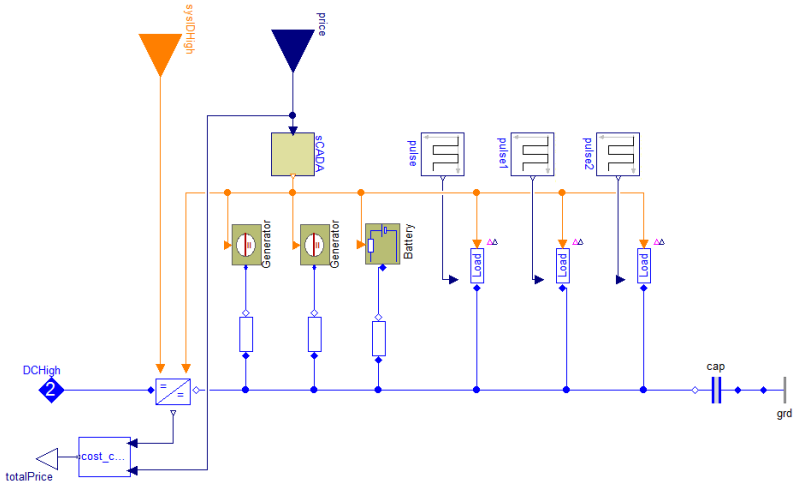


Figure 7.2: Overview of the Modelica model of the houses used to simulate the short time scenario.

The generation is set to be constant since this scenario only tested how the system reacts to fast changes in the transmission grid. The voltage reference value for the external medium voltage grid was set to $v_{ref} = 1000V$ and the battery on this level has a storage capacity of 1.4 kWh. All the houses have voltage reference of $v_{ref} = 230V$. The simulation is running for 1000 seconds. Using this configuration, the simulation will show how the control program handles different micro generations during changing conditions on the high and medium level grid in other words islanding mode.

7.2 Long term scenarios:

In this section the *long term performance* of a smart house is investigated from an economic and a power peak perspective. The main questions are: are there any economic incentives in installing this kind of control in a household and can it decrease power consumption peaks.

Depending on the size of a house and whether electricity is used for heating/cooling the total consumption per year will vary. In this example we have chosen houses with central heating and a total consumption of 13 MWh per year. The Swedish power company *Vattenfall AB* fixed price at 93.25 öre/kWh was used[1]. It is assumed that from 07:00 to 23:00 the cost for buying electricity is given by the *Vattenfall*-price and from 23:00 to 07:00 the price will be 30 %, 40 % or 50 % lower, this will be refereed to as *price differentiation*.

All houses are equipped with the loads from table 7.1 and they have batteries installed that charge during the night and discharges during the daytime, outside the grid. This models a battery equipped car which charges during the night and discharges during the course of day, due to usage of the car. Furthermore the houses are configured with the following micro generation:

[1]Vattenfall 1 year fixed price contract (taken from vattenfall.se the 13th of May 2013)

- **House 1** has photovoltaic solar panels as well as a small scale wind turbine.
- **House 2** has no micro generation at all.
- **House 3** has a small scale wind turbine but no solar panels.
- **House 4** has solar panels but no small scale wind turbine.

It is assumed that net billing/metering is used. Net billing is a concept where a house with micro generation will sell power to the power company during times of overproduction. The house will then be able to swap the exported power to import power when needed.

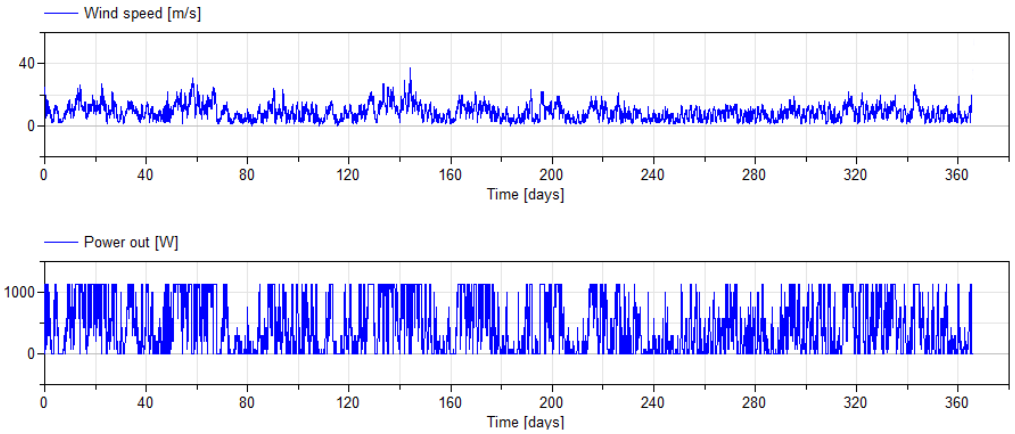


Figure 7.3: Digram of the input data and output power from the wind power plant model used. In the top subplot the wind data is plotted and in the bottom the output power from the wind power is plotted.

For the small scale wind turbine the equation described in section 4.2.5 is used together with the wind data[2] presented in figure 7.3. The wind speed data is given in hourly measurements. The parameters for the wind turbine in equation 4.5 are chosen as:

$$\begin{aligned} \rho &= 1.225 \text{ kg/m}^2 \\ A &= \left(\frac{2}{2}\right)^2 \cdot \pi = 3.141 \text{ m}^2 \end{aligned} \tag{7.1}$$

The photovoltaic panels are using the equation described in section 4.2.5 together with the radiance data[3] and temperature data[4]. Temperature, sun

[2]The wind data is taken from the course *EIEN10- Wind Power Systems* (2012) given at *The Faculty of Engineering* at *Lund University*

[3]Radiance data for Norrköping for the year 2012 is provided by the database *STRÅNG*, which is a *mesoscale model for solar radiation* which is financed by *Naturvårdsverket, SMHI* and *Strålsäkerhetsmyndigheten*

[4]Temperature data from Norrköping for the year 2012 is given as hourly measurements. The data is provided by *Dataleveranser* at SMHI

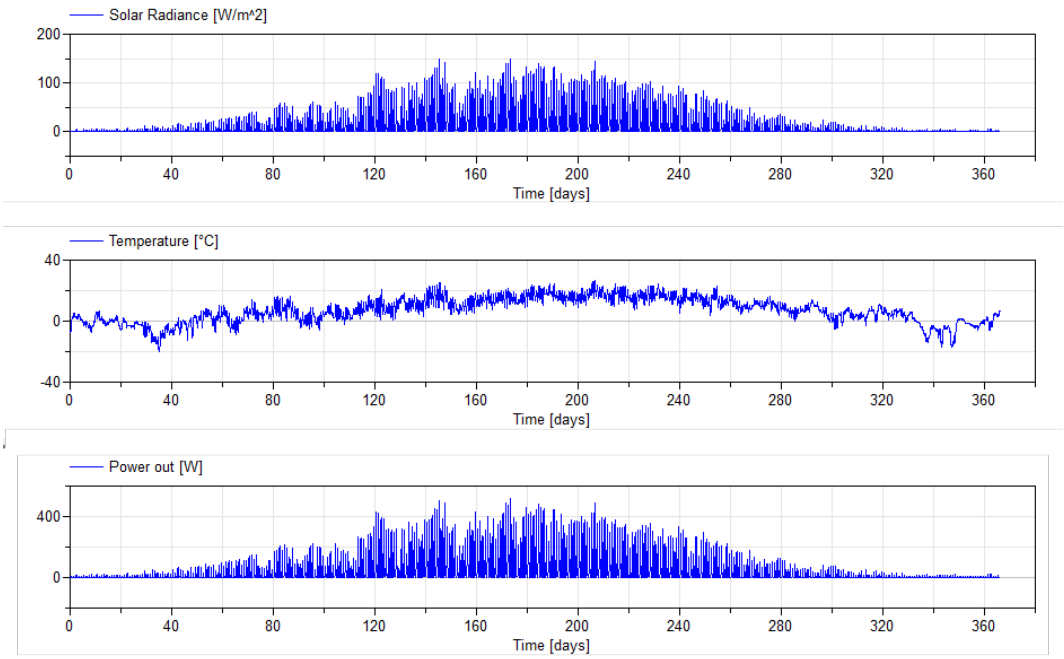


Figure 7.4: Diagram of the input and output data from the solar power model used. In the top sub plot the solar radiance is plotted, in the middle the temperature data is plotted and in the bottom the outputted power produced by the solar panel is plotted. The radiance and temperature data is given for Norrköping 2012 and the total amount of energy generated for 2012 is 381 kWh.

radiance and the resulting power generated can be seen in figure 7.4. The parameters for the solar panel in equation 4.7 are chosen as:

$$\begin{aligned} \eta_{PV} &= 14.4\ \% \\ n_{PV} &= 18 \\ S_{PV} &= 1.3\ \mathrm{m}^2 \end{aligned} \tag{7.2}$$

To simulate the consumption for a house the electrical devices from table 7.1 are used. In the last column of the table there is a time interval in which the device is likely to be turned on. This is modelled such that the device is turned on in a stochastic sense. Every time instances in the time interval are equally probable i.e. the time the loads is turned on is *uniformly distributed*. The generated consumer power profile can be seen in figure 7.5.

The Dymola implementation of the smart house for long term is similar to the one for short term but with much more loads. A figure of the implementation is seen in figure 7.6.

Type	**power [W]**	**operation time [min]**	**maximum delay [min]**	**time interval[hour]**
Dishwasher	1900	180	360	7-8
Washing machine	2100	180	600	19-20
Tumbler dryer	1900	120	600	19-21
Water heater	800	360	600	6-8
Microwave	1000	15	N/A	6-8
Oven	2100	60	N/A	18-20
Hair dryer	1200	10	N/A	6-8
Coffee maker	600	400	N/A	6-8
Toaster	1000	15	N/A	7.5-8.5
Fridge	400	∞	N/A	N/A
Freezer	300	∞	N/A	N/A
Computer	100	180	N/A	6-7 & 18-21
TV	80	180	N/A	18-20
Electric devices stand by	50	∞	N/A	N/A
Lights	50	180	N/A	N/A

Table 7.1: Table of all the loads the houses are equipped with. *Power* specifies the amount of power that the device consumes while turned on, *operation time* specifies for how long the device is turned on, *maximum delay* is the maximum time for which the device can be postponed, and *time interval* is the interval for which the devices are turned on, the exact time is *uniformly distributed* inside the interval. (The power consumption and operation time is inspired by "El nära och långt borta -hur kan hushållen agera på elmarknaden?" which is a book published by *Energimyndigheten*)

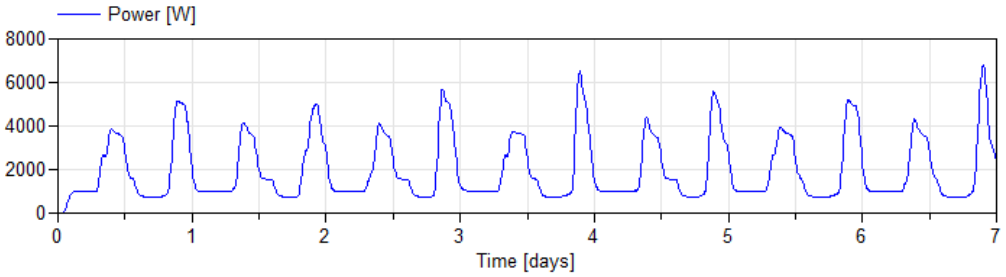


Figure 7.5: The generated consumer profile is shown. Notice the stochastic behaviour of the loads.

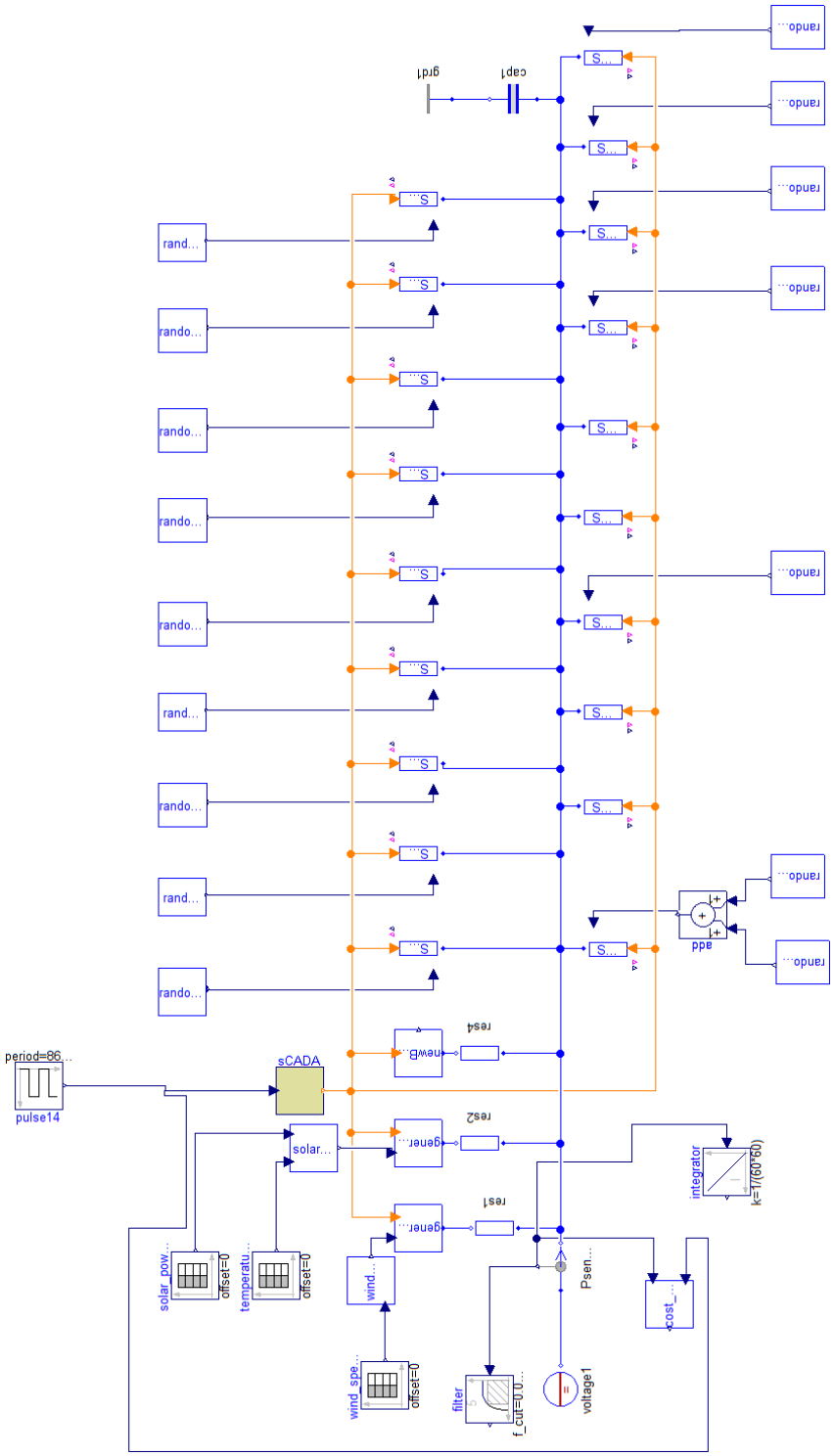


Figure 7.6: Dymola model for long term scenario.

Chapter 8

Results

In this chapter simulation results from the scenarios introduced in chapter 7 are presented. The system performance is evaluated during external power losses and the control strategy is explained. The long time aspects of using these control laws are also evaluated. The power performance and the economical cost are presented.

8.1 Short term scenario:

The most important task for the system controllers is to maintain an acceptable voltage level in the grids. The external grid voltages can be seen in figure 8.1. At $t = 80$ s the external high voltage grid fails and the system controller on the medium voltage grid is forced to operate in islanding mode. Doing so the system controller is forced to use its battery to do voltage control.

The plot is divided into three different zones, where zone 1 is where all grids operates normally, zone 2 is where the medium level grid operates in islanding mode and zone 3 when the houses operate in islanding mode. In zone 1 the voltage controller power up the system. When the correct voltage level is reached the houses connects to the grid and begin to turn on loads. The impact of this can be seen in zone 1. Entering zone the external high voltage grid fails and the medium voltage grid operates in islanding mode. This transmission can be seen in figure 8.1. Where the battery at the medium voltage grid start to control voltage. In zone 3 the battery at the medium voltage grid can not power the grid and the houses are therefore forced to disconnect from the grid and to operate in islanding mode.

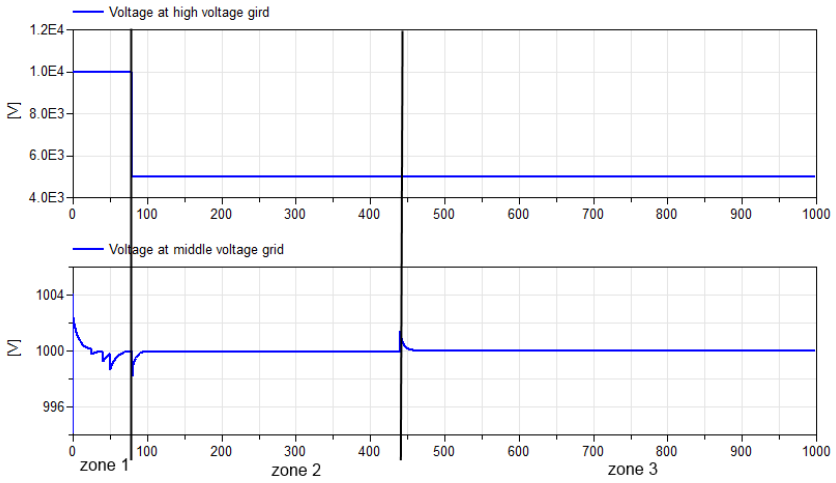


Figure 8.1: Simulation results from the short term scenario. In the top plot the voltage level for the external high level grid is visible. The voltage level for the medium voltage grid is plotted in the lower plot.

In figure 8.2 the voltage levels for the three houses are presented. This figure is divided into the same three zones. Notice that all houses manage to keep their voltage level close to the reference value of 230 V. Small voltage drops are noticed in zone 1 when the loads are turned on. Entering zone 2 house 1 maintains its voltage. House 2 and 3 experience a small voltage change since the system controller is forced to turn off the lowest priority load. Entering zone 3 all houses experience voltage drops. This is due to that the battery start to control voltage instead of the external connection. It takes a few seconds for the new controller to achieve good control of the voltage. House 1 and 2 experience larger voltage drops since these houses also needs to turn off loads.

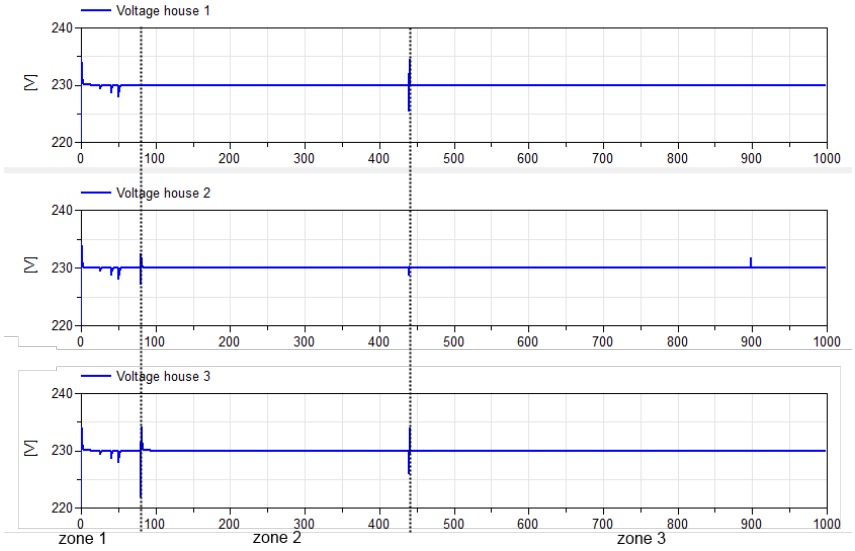


Figure 8.2: Simulation results from the short term scenario. In the top plot the voltage for house 1 is plotted, in the middle plot the voltage for house 2 is plotted and in the bottom house the voltage is plotted for the third house.

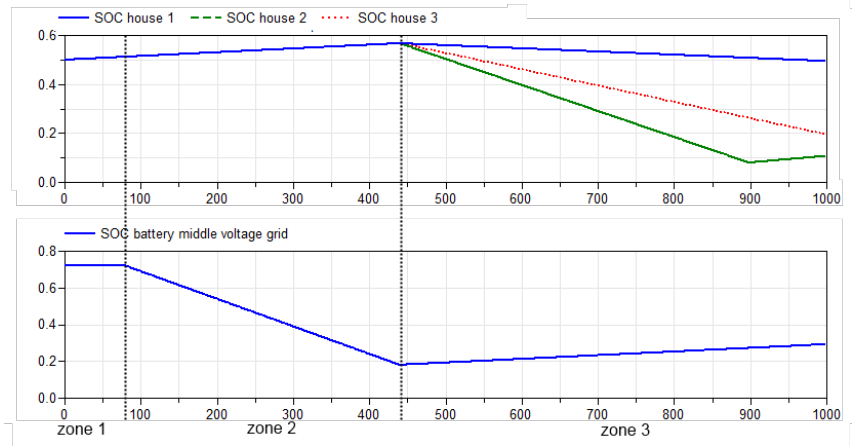


Figure 8.3: Simulation results from from the short term scenario. In the top subplot the voltage level for the battery in the different houses are plotted, all starting with an initial SOC of 0.5. In the bottom subplot the voltage level for the battery at medium voltage grid level is plotted, with the initial SOC of 0.7. Note that system controller on the medium voltage level considers the battery empty when it has reached SOC of 0.2.

During zone 1 and zone 2 the batteries at house level charges since there is power capacity to do so. The battery *SOC* can be seen in figure 8.3. In zone 3, when the houses operate in islanding mode, the battery is controlling voltage and the *SOC* is therefore decreasing for all houses. Except for house 2 after $t = 900$ s. The *SOC* for the battery in house 2 drops below 0.1 at $t = 900$ s and the system controller therefore tries to recharge the battery by using the

power from the micro generation and by disconnecting loads.

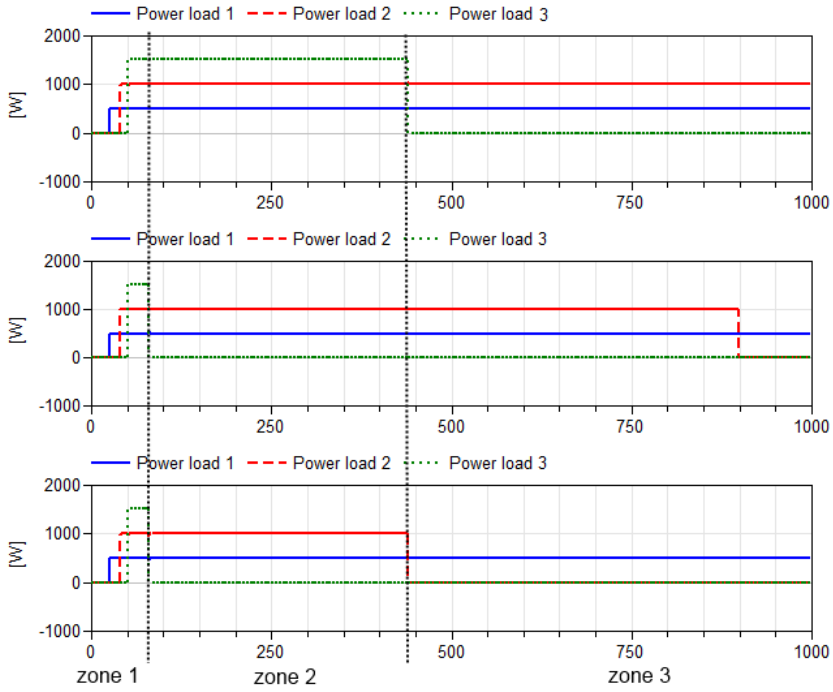


Figure 8.4: Simulation results from the short term scenario. In zone 1 the external high voltage grid operates normally and the loads turns on according to their priority in all the houses. The loads has priority according to their number, 1 is the highest and 3 the lowest. In zone 2 and 3 the loads behaves differently in the houses since the micro generation is not the same in the houses.

The *SOC* for the battery at medium voltage grid is also plotted in figure 8.3. Notice that the *SOC* for the battery drops to 0.2 at $t = 440$. This is what forces the houses to disconnect from the grid since the system controller considers the battery empty at SOC = 0.2. When the houses are disconnected the system controller at this level tries to recharge the battery by using the available generation at this level. This is why the *SOC* increases in zone 3.

In figure 8.4 the power of the loads in the houses are shown. When operating in zone 1, the loads turn on according to their priority and the system controller has no problem maintaining voltage. When entering zone 2, house 2 and 3 can not keep their lowest priory load (3) turned on , the system controller is forced to turn off the load to maintain the correct voltage level on the grid. When entering zone 3 the houses operate in islanding mode. House one is forced to turn off its lowest priority load, since the power generated from the micro generation and battery is not enough to keep all loads turned on. House 3 is forced to turn off its second load since the battery alone can not provide enough power for the two loads. House 2 is forced to turn off load two at $t = 890$ s since its battery discharges completely, this can be seen in figure 8.3. The battery can therefore no longer provide power to the system.

These results show that the controller program handles blackouts on the external high voltage grid well. With the help of an energy storage the system can maintain voltage and keep the highest priority loads turned on. The system is also capable to recharge the batteries to prepare the grid for further blackouts. The grid robustness is mainly determined by the size of the energy storage. In these simulations the storage has been chosen rather small to illustrate the different zones in a quite small time frame. If the energy storage were dimensioned larger the system controller would be able to keep the houses/loads connected longer.

8.2 Long term scenarios:

The main focus of these simulations will be to compare power peaks when using load delay and price differentiation with a system not using load delay. The economical aspects of this will also be evaluated by investigating the total electricity cost for one year running these control laws.

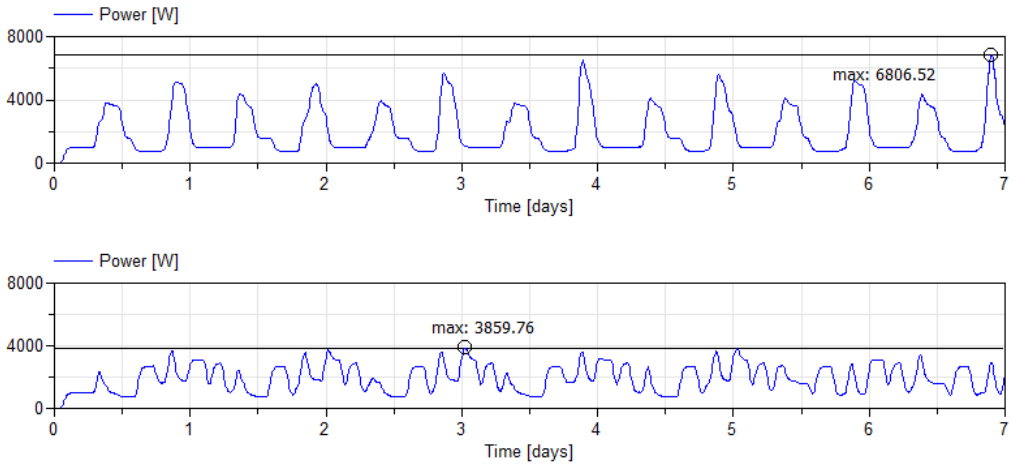


Figure 8.5: Simulation results comparing the power consumed for one week, the lower plot is from the house that uses price differentiation to delay some loads to run during times of low prices. The upper plot is from the house that does not use load delay.

In figure 8.5 the consumed power for a house is compared when using load delay with not using load delay. It is obvious from the figure that the system controller manages to reduce the power peaks with use of load delay. During the first week the consumed power peak for the system that is not using load delay is $P_{max} = 6806.52$ W. The system that is using load delay manage to reduce the peak to $P_{max} = 3859.76$ W.

The maximum power peaks during the whole simulation are compared in chart 8.1 for the different houses. Each house is simulated once using load delay and once without load delay. The different micro generation between the houses does not give any big influence on the power peaks. This could mainly be explained with the facts that using load delay big loads are delayed to run during night and maximum power is therefore reached during night-time. Micro generation would help to reduce power peaks in the grid but during night they tend to have quite small production and will not influence the power peaks noticeable. The systems using load delay manage to reduce the power peak with about 35%.

In chart 8.2 the total price for running all houses are presented. Each house has been simulated using load delays with three different price differentiations, 30 %, 40 % and 50 %. Furthermore this data has been compared with the total price when simulating the houses without load delay and using 30 % price differentiation.

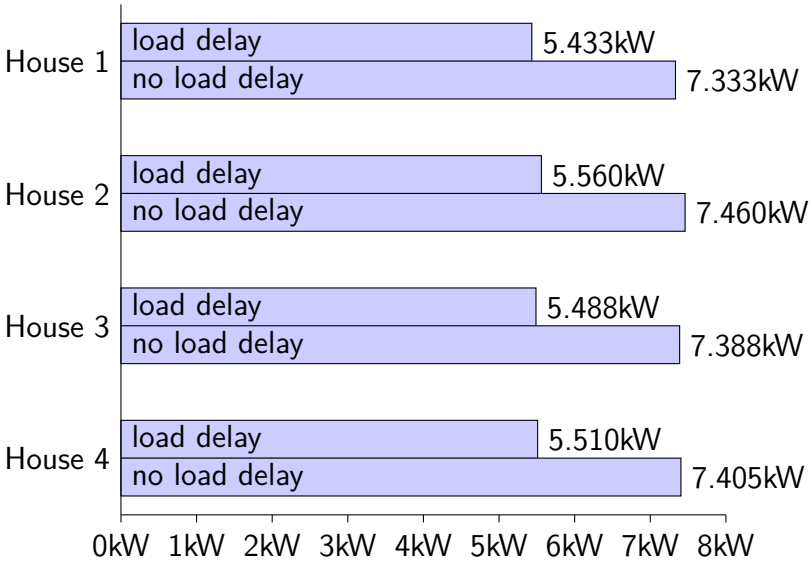


Chart 8.1: Simulation results from all the proposed houses. Power peaks are compared when using load delay with not using load delay.

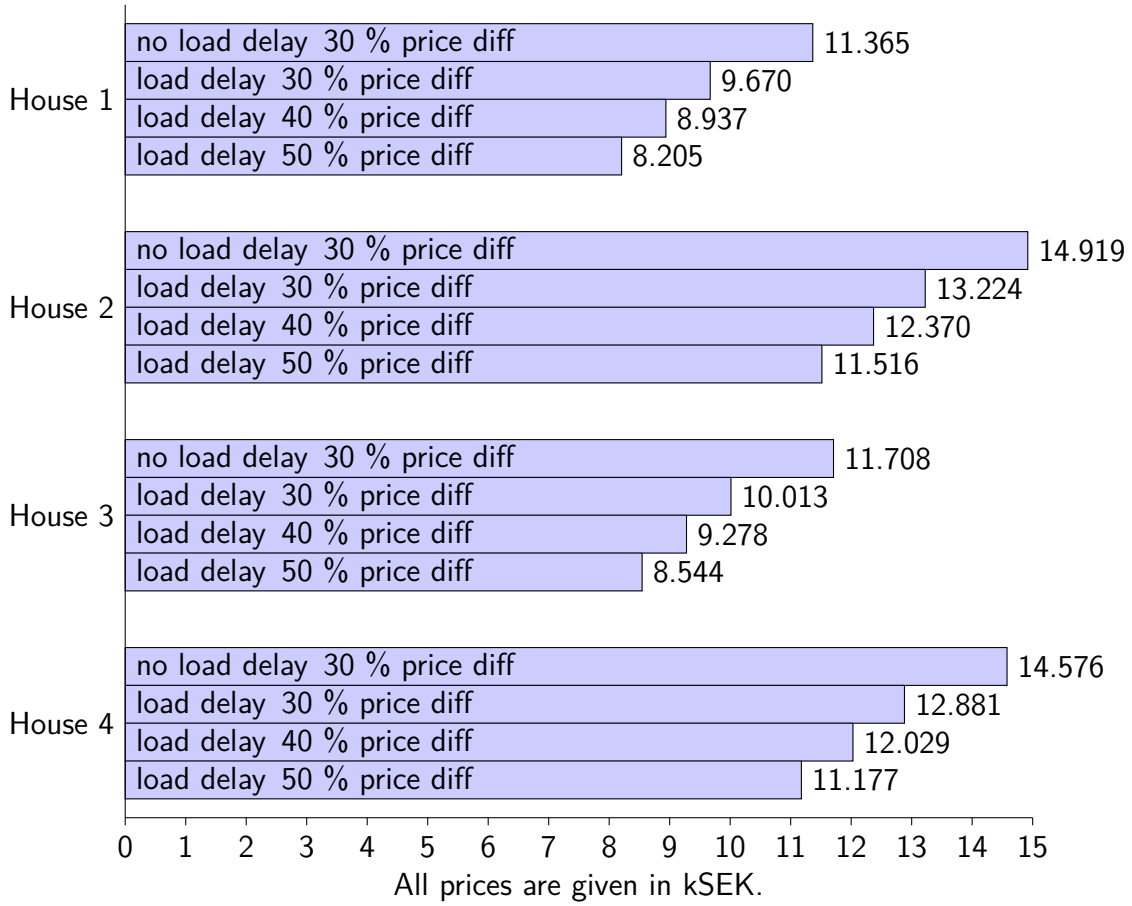


Chart 8.2: Simulation results from all the proposed houses. Total price for one year was compared using load delay with not using load delay for different price differentiation levels.

In figure 8.6 the *SOC* for the battery is plotted for the first week of simulation. The battery is charged during night, when the price is low. And discharges during day time, to simulate a car driving during day time. This power will therefore not go back into the internal grid and the battery will start almost empty every evening.

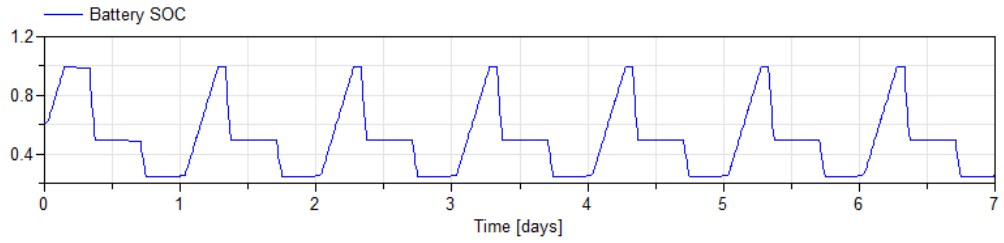


Figure 8.6: Simulation results for the battery from the long time scenario, the SOC is plotted.

These results shows that using these control laws the system controller manages to decrease the power peaks with about 35 %. Using 30 % price differentiation the yearly cost differs with 1.650 SEK when using load delay compared to not using load delay. The cost difference can of course be increased if the price differentiation is larger.

Chapter 9

Discussions and Conclusions

In this final chapter the project is evaluated and summarized. Some future work is suggested and the concluding results from the project are stated.

9.1 Evaluation of the Project

The purpose of this project was to develop generic models for smart houses using *Dymola/Modelica* and to design control laws using C/C++. Apart from houses a medium voltage grid was also modelled, which formed a smart cell together with the houses. The project goals are listen in section 1.4. Overall the project was considered a success. Models have been developed and can be configured in *Dymola* by dragging and dropping components to form an arbitrary smart house. These houses can be connected using external connections to a medium voltage grid together with generators and battery to form smart cells. The physical part of the modelling was easily done using *Dymola* and *EPL*. *EPL* components were used and configured to suit our modelling purposes. The cells are controlled using control laws developed in *C/C++*. Simulation was done which shows that the system controller manage to reduce power peaks by using price differentiation and load delay. Short term simulations also showed that the robustness of the grid is improved using the proposed control strategies and energy storage. The simulation also shows that the yearly cost was decreased for the end user when load delay was used. When using 30 % differentiation the yearly electricity cost could be decreased with 1.650 SEK for a house with a total consumption of 13 MWh per year. It is hard to tell whether this price incentive is big enough for end users to invest in these kinds of systems. If new houses would be prepared for installation the installation cost might not be very high and this system might repay. There are also robustness incentives for grid operators. By introducing the proposed control strategies and energy storages the grid would be more reliable and withstand short periods of blackouts.

Several studies have projected that the global oil reserve will run out somewhere between 2050 and 2075. This will drive up the electricity price since a large amount of the electricity used today are generated using fossil fuel, such as oil [24]. Given future higher electricity prices, the economical incentives for smart grids will increase in the future.

9.1.1 Implementation problems

The system controller was implemented using *C++* and interfaced with *Dymola* using pre-compiled external static libraries. All functions that are called need to explicitly be defined in *Dymola* with function name, input and output variables. This method results in lot of work if changes in function arguments would be done. If one input argument should be added to a function this will introduce changes in *Dymola's* function definition, the interface file, the C++ source code and further down the *C++* data structure. Therefore if one argument is added to a function, the code need to be changed in 8 places.

Another problem introduced by external *C++-program* approach was the timing for which blocks and system controllers were initialized. Since the resources need to know which system it belongs to, i.e the system id, the system controller needs to be initialized first. *Dymola* has no predefined causality

therefore this was problematic to do. This was solved by not initialize the system controllers and resources at the same sample.

Apart from the initialization problems debugging using an external C++-program was hard. The functionality of the *C++-program* could only be evaluated when the Dymola simulation was running. Since the program needs to be fed with signals from the Dymola simulation to work properly. Pre-compiled libraries were used and a debugger tool can therefore not be used to test control functionality. The only debugging available was explicitly checking in and out signals in Dymola and try to trace the error in the code using print outs into a console window while running the simulation.

9.2 Improvements

During the process of working with the thesis we realised some improvement and changes that could be done to further improve the simulations. The following sections provides some ideas for future work.

9.2.1 Modelling

The models that are used in this project are simplifications of the actual physical components. For example; loads are modelled as resistances and batteries as capacitances. It is essential that these simple models are exchanged for better and more accurate models of the physical components. Loads would have different model implementations depending on the components, a dishwasher should be modelled as an electric motor containing inductance, resistance and an EMF[1], a water heater should have heating and cooling effects of the water corresponding to usage and converters should model losses.

Model improvements are very important to enhance short term simulations and scenarios. It is less important for long term since fast dynamics does not effect costs as much as the total consumption. However, fast dynamics might increase power peaks for long term scenarios which of course is of interest.

9.2.2 Model Verification

Unfortunately no measurement data was available for model verification. It would be essential to improve the models by using measurement data from actual physical components to identify parameter values.

9.2.3 Boundary Values

Boundary values such as consumer inputs for turning on and off loads and configurations of priorities and operation time do not correspond to any actual user behaviour. The consumer ”on/off” input is modelled as a random behaviour between given time points. It would be desirable to have actual

[1]Electro Motive Force

data sets for consumer habits for a house to increase the accuracy of power consumption patterns in order to get more realistic values for power peaks and total costs.

9.2.4 Implementation

As mentioned in 9.1.1 there are a lot of problems concerning communication and control with the implementation method chosen. We believe that these problems can be circumvented by not using Dymola as simulation tool in this case. Modelica is very powerful in making the physical models but not for control with complex decision systems. We would thus suggest that only the physical components such as, loads, generators, batteries, grids and converters should be modelled using Modelica and the control of these components should be done in another program. Exporting models as *FMU's*[2] and importing them to *Simulink* is a proposed method since Simulink is suitable for dealing with signals.

Rather than having one program with all functionality for both *system controller* and *resources* (one .lib file where different functions are called for different blocks), as was the case for this thesis, a Simulink implementation of the control system could involve a *CAN-bus*[3] which resources and system controller are connected to. This would separate the programs running the controllers and resources in a greater extent than the current solution. There are already Simulink toolboxes for CAN-bus simulation. Writing the control algorithms in *MatLab*-functions rather than C++ might also be more intuitive for the inexperienced programmer and debugging will be easier than the current method.

9.2.5 Real-Time

One major benefit with using Simulink for control is the possibility for *automatic code generation*, i.e. generate C code which can be used on a micro controller to be used in real time. This is however also possible in Dymola if the control algorithm is written in Dymola and not as external libraries.

9.3 Conclusion

Using smart grids the power performance for the grid can be improved. It is shown that power peaks in the grid can be decreased in average with 35 % . With 30 % price differentiation the proposed control strategy reduces the yearly cost for an end user with 1.650 SEK. Whether this price reduction is

[2]Functional Mock-up Unit is an executable file, in this case, containing physical models and possibly its solver that can be exported to another simulation environment to be executed.

[3]Controller Area Network is a communication system often used in vehicles to simplify information exchange between the sensors and computers which, in modern cars, are a couple of hundred as of today.

enough to pay off installation costs for the end user is hard to tell. Since only a few small scale project that have been developed, cost for large scale implementations is hard to estimate.

Appendix A

General

A.1 Mid-Range controller

Mid-range control strategy was considered possible when controlling the generators atmedium voltage level, but did not end up in the finial control program. Mid-range control is a strategy involving two controllers used to control one measurement signal. This is useful if different actuators with different range and resolution are used. Suppose that the level in a water tank is controlled using two actuators. One has a fine resolution of 0.1 l/s but saturates at 2 l/s, the other has a resolution of 5 l/s and a maximum flow of 30 l/s. The idea behind mid-range control is to use the first actuator to react at small deviations from the set point and the second controller to react on large deviations. This is done be feeding the second controller, C_2, with the control signal from the first controller, C_1, and the desired control signal set point value, u_{sp}, for C_1, see figure A.1. [25, p. 378-380]

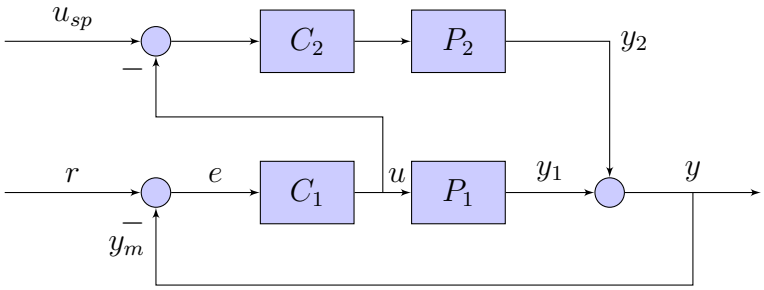


Figure A.1: Mid range controller

Appendix B

Modelica Source Code

B.1 Solar panel

Listing B.1: Modelica model for solarPanel

```
model solarPanel

  parameter Real conversion_efficient = 0.144;
  parameter Real numer_of_panels = 18;
  parameter Real array_area = 1.3;

  Modelica.Blocks.Interfaces.RealInput I_pv
    a;
  Modelica.Blocks.Interfaces.RealInput t_cr
    a;
  Modelica.Blocks.Interfaces.RealOutput P_pv
    a;

equation
  P_pv = conversion_efficient*numer_of_panels*array_area*
     I_pv*(1-0.005*(t_cr-25));
  a;
end solarPanel;
```

B.2 Wind Power Plant

Listing B.2: Modelica model for wind power plant

```
model windPowerPlant

  parameter Real rotor_diameter = 2;
  parameter Real eta = 1;
  parameter Real Cp = 0.3;
  parameter Real rau = 1.225;

```

```
8   Modelica.Blocks.Interfaces.RealInput wind_speed
9     a;
10  Modelica.Blocks.Interfaces.RealOutput generated_power
11    a;
12 equation
13  generated_power = if wind_speed >4 and wind_speed<12.5
14   then 1/2*rau*(rotor_diameter/2)^2*Modelica.Constants.pi*
       wind_speed^3*eta*Cp
15  else if wind_speed>12.5 and wind_speed<25 then 1127.48
      else 0;
16
17  a;
18 end windPowerPlant;
```

B.3 Generic Smart Block

Listing B.3: The Modelica code for generic smart block.

```
1  model genericSmartBlock
2
3  extends systemController.Blocks.MyDiscreteBlock;
4
5    import init_Block =
       systemController.ExternalFunctions.Block_init;
6    import set_Block =
       systemController.ExternalFunctions.Block_set;
7    import get_Block =
       systemController.ExternalFunctions.Block_get;
8
9    Modelica.Blocks.Interfaces.IntegerInput sysID "System␣
       identity"
10     a;
11
12 public
13   Modelica.Blocks.Interfaces.IntegerOutput breaker "Controls
       ␣the␣switch␣on/off"
14     a;
15
16 public
17   Modelica.Blocks.Interfaces.RealInput SOC
18     "Gives␣the␣state␣of␣charge␣for␣the␣battery␣if␣the␣
         resource␣is␣of␣type␣2␣else␣-1"
19     a;
20
21 public
22   Modelica.Blocks.Interfaces.RealInput resistance
23     "Gives␣the␣current␣resistance␣if␣the␣resource␣is␣load␣
         else␣-1"
```

```
24      a;
25  public
26    Modelica.Blocks.Interfaces.RealInput maxPower
27      "Gives the current maximum availiable power if the
          resource is generator else -1"
28      a;
29  public
30    Modelica.Blocks.Interfaces.RealInput voltageExternal
31      "Measures the voltage of the connected component"
32     a;
33  public
34    Modelica.Blocks.Interfaces.RealInput voltageGrid "Measures
        the grid voltage"
35     a;
36  public
37    Modelica.Blocks.Interfaces.RealInput powerOUT "Measures
        the outputten power"
38     a;
39  public
40    Modelica.Blocks.Interfaces.RealOutput controlCurrent
41      "Gives the curreent the block should output"
42     a;
43  public
44    Modelica.Blocks.Interfaces.RealInput consumerProfile
45      "Measures the grid voltage"
46     a;
47
48  // -- Parameters --
49  public
50  parameter Integer priority "Resource priority";
51  parameter Integer resourceType "Resource type";
52  parameter Real maxCurrent=63;
53
54  // -- Parameters for controller --
55  parameter Real K=-1 "Proportional gain";
56  parameter Real Ti=-1 "Integral gain";
57  parameter Real Tt=-1 "Antiwindup time constant";
58  parameter Real ymin=-1 "Minimum output value from controller
      ";
59  parameter Real ymax=-1 "Maximum ouput value from controller"
      ;
60  parameter Real priceLimit
61      "Component will not be turned on if price  > priceLimit,
          -1 default";
62  parameter Real maximumDelay=0 "Maximum delay for component, 
      0 default";
63  parameter Real operationTime=10000
64      "Operation time for component, 10000 is default";
65  constant Integer lowFlag = 0;
```

```
66
67  // -- variables --
68 Integer resourceID;
69 Real cv[:] = {resourceType, priority, maxCurrent,K,Ti,Tt,
       ymin,ymax, samplePeriod, lowFlag, priceLimit,
       maximumDelay, operationTime};
70 Real signalsOUT[signalsSizeOUT];
71 Real signalsIN[signalsSizeIN];
72
73 constant Integer signalsSizeOUT = 8;
74 constant Integer signalsSizeIN = 2;
75
76 algorithm
77     when sampleTrigger and secondTrigger then
78       resourceID := init_Block(sysID, cv, signalsSizeIN,
           signalsSizeOUT);
79     end when;
80
81     when sampleTrigger and  not firstTrigger and not
         secondTrigger then
82
83       // -- set singnals out
84       signalsOUT := {pre(voltageGrid), pre(powerOUT), pre(
           voltageExternal), pre(SOC), pre(maxPower), pre(
           resistance), -1, pre(consumerProfile)};
85       set_Block(sysID, resourceID, signalsOUT);
86
87       // -- set signals in
88       signalsIN      := get_Block(sysID, resourceID,
           signalsSizeIN);
89       breaker        := integer(signalsIN[1]);
90       controlCurrent := signalsIN[2];
91
92     end when;
93   a;
94 end genericSmartBlock;
```

B.4 External Connector Block

Listing B.4: The main Modelica code for external connector block.

```
1 model smartBlockExternal22april
2 extends systemController.Blocks.MyDiscreteBlock;
3
4   import init_Block =
       systemController.ExternalFunctions.Block_init;
5   import set_Block =
       systemController.ExternalFunctions.Block_set;
```

```
6   import get_Block =
        systemController.ExternalFunctions.Block_get;
7
8   Modelica.Blocks.Interfaces.IntegerInput sysIDHigh "System␣
        identity"
9     a;
10  public
11  Modelica.Blocks.Interfaces.IntegerOutput breaker "Controls
        ␣the␣switch␣on/off"
12    a;
13  public
14  Modelica.Blocks.Interfaces.RealInput SOC
15    "Gives␣the␣state␣of␣charge␣for␣the␣battery␣if␣the␣
          resource␣is␣of␣type␣2␣else␣-1"
16    a;
17  public
18  Modelica.Blocks.Interfaces.RealInput resistance
19    "Gives␣the␣current␣resistance␣if␣the␣resource␣is␣load␣
          else␣-1"
20    a;
21  public
22  Modelica.Blocks.Interfaces.RealInput maxPower
23    "Gives␣the␣current␣maximum␣availiable␣power␣if␣the␣
          resource␣is␣generator␣else␣-1"
24    a;
25  public
26  Modelica.Blocks.Interfaces.RealInput voltageLow
27    "Measures␣the␣voltage␣of␣the␣connected␣component"
28   a;
29  public
30  Modelica.Blocks.Interfaces.RealInput voltageHigh "Measures
        ␣the␣grid␣voltage"
31   a;
32  public
33  Modelica.Blocks.Interfaces.RealInput powerOUT "Measures␣
        the␣outputten␣power"
34   a;
35  public
36  Modelica.Blocks.Interfaces.RealOutput controlCurrent
37    "Gives␣the␣curreent␣the␣block␣should␣output"
38   a;
39  Modelica.Blocks.Interfaces.IntegerInput sysIDLow "System␣
        identity"
40    a;
41  public
42  parameter Integer priority "Resource␣priority";
43  parameter Integer resourceType "Resource␣type";
44  parameter Real maxCurrent=63;
45
```

```
46 // -- Parameters for controller --
47 parameter Real K=-1 "Proportional gain";
48 //annotation(dialog(group="Parameters",tab="Controller",
      enable = useController and not resourceType ==0));
49 parameter Real Ti=-1 "Integral gain";
50 //annotation(dialog(group="Parameters",tab="Controller",
      enable = useController and not resourceType ==0));
51 parameter Real Tt=-1 "Antiwindup time constant";
52 //annotation(dialog(group="Parameters",tab="Controller",
      enable = useController and not resourceType ==0));
53 parameter Real ymin=-1 "Minimum output value from controller
      ";
54 //annotation(dialog(group="Parameters",tab="Controller",
      enable = useController and not resourceType ==0));
55 parameter Real ymax=-1 "Maximum ouput value from controller"
      ;
56 //annotation(dialog(group="Parameters",tab="Controller",
      enable = useController and not resourceType ==0));
57
58  // -- variables --
59 Integer resourceIDHigh;
60 Integer resourceIDLow;
61 Real signalsINHigh[signalsSizeInHigh];
62 Real signalsOUTLow[signalsSizeOutLow];
63 Real signalsINLow[signalsSizeInLow];
64 Real signalsOUTHigh[6];
65
66 constant Integer signalsSizeInHigh = 3;
67 constant Integer signalsSizeOutHigh = 6;
68
69 constant Integer signalsSizeInLow = 2;
70 constant Integer signalsSizeOutLow = 7;
71
72 constant Integer highFlag = 1;
73 constant Integer lowFlag = 0;
74
75 Real cvHigh[:] = {resourceType, priority, maxCurrent,K,Ti,Tt
      ,ymin,ymax, samplePeriod, highFlag};
76 Real cvLow[:] = {resourceType, priority, maxCurrent,K,Ti,Tt,
      ymin,ymax, samplePeriod, lowFlag};
77
78 algorithm
79     when initial() then
80       // nothing
81     end when;
82
83     when sampleTrigger and secondTrigger then
84       resourceIDHigh := init_Block(sysIDHigh, cvHigh,
           signalsSizeInHigh, signalsSizeOutHigh);
```

```
85        resourceIDLow  := init_Block(sysIDLow, cvLow,
              signalsSizeInLow, signalsSizeOutLow);
86
87      end when;
88
89      when sampleTrigger and  not firstTrigger and not
            secondTrigger then
90      // first get the power blanace information form the high
            voltage grid
91      signalsINHigh       := get_Block(sysIDHigh,
            resourceIDHigh, signalsSizeInHigh);
92
93      // set this information to the low voltage side of the
            grid
94      signalsOUTLow    := {pre(voltageLow), pre(powerOUT), pre(
            voltageHigh), pre(SOC), min(pre(maxPower),
            signalsINHigh[3]),   pre(resistance), signalsINHigh
            [3]}; // min to take into account the power balance
            on the external grid
95      //signalsOUTHigh  := {pre(voltageLow), pre(powerOUT),
            pre(voltageHigh), pre(SOC), pre(maxPower),  pre(
            resistance)};
96      signalsOUTHigh  := {0, 0, 0, 0, pre(maxPower),  0};
97
98      //signalsOUTLow := {pre(voltageLow), pre(powerOUT), pre(
            voltageHigh), pre(SOC), pre(maxPower), pre(resistance
            ), 100};
99      set_Block(sysIDLow, resourceIDLow, signalsOUTLow);
100     set_Block(sysIDHigh, resourceIDHigh, signalsOUTHigh);
101
102     // get the signals from the low voltage external block!
103     signalsINLow       := get_Block(sysIDLow, resourceIDLow,
            2);
104     breaker            := integer(signalsINLow[1]);
105     controlCurrent     := signalsINLow[2];
106
107     end when;
108
109   annotation (Diagram(coordinateSystem(preserveAspectRatio=
        false, extent={{-100,
110             -100},{100,100}}), graphics), Icon(
                coordinateSystem(
111           preserveAspectRatio=false, extent
                ={{-100,-100},{100,100}}), graphics));
112 end smartBlockExternal22april;
```

Appendix C

C/C++ Source Code

C.1 Dymola interface

Listing C.1: SystemControlInter.h

```
1  #ifndef SYSTEMCONTROLINTER_H
2  #define SYSTEMCONTROLINTER_H
3
4  #ifdef __cplusplus
5  extern "C" {
6  #endif
7    // Functions acting on system controller from dymola
8    // used to initialize a system controller
9    extern int SystemControlInter_init(double* ↩
       configuration_vector,
10     int configuration_vector_size, int signals_size_scada)↩
         ;
11   // used to update the low voltage system controller
12   extern void SystemControlInter_update(int system_id, ↩
       double price);
13   // used to update the high voltage system controller
14   extern void SystemControlInter_updateHighVoltage(int ↩
       system_id,
15     double price);
16
17   // Functions acting on resources from dymola
18   // used to initializes block
19   extern int Block_init(int system_id, double* ↩
       configuration_vector,
20     int configuration_vector_size, int signals_in_size,
21     int signals_out_size);
22   // used to send information to dymola.
23   extern void Block_get(int system_id, int resource_id,
24     double* dymola_return_vector, int ↩
         dymola_return_vector_size);
```

```
25   // used to receive infromation from dymola
26   extern void Block_set(int system_id, int resource_id,
27     double* dymola_set_vector, int dymola_set_vector_size)
         ;
28 #ifdef __cplusplus
29 }
30 #endif
31 #endif
```

Listing C.2: SystemControlInter.cpp

```
1  #include "SystemControlInter.h"
2  #include "SCADA.h"
3  #include <vector>
4  #include <cstdio>
5
6  #include <iostream>
7  using namespace std;
8
9  static int id = 0;       // set default id to 0
10 static vector<SCADA*> systems; // vector with systems with
       dynamic size
11
12 extern int SystemControlInter_init(double* ↩
       configuration_vector,
13                  int configuration_vector_size,
14                  int signals_size_scada)
15 {
16   systems.push_back(new SCADA); //create a new SCADA-unit(↩
         system)
17   systems.at(id)->initSystem(configuration_vector, ↩
         configuration_vector_size,
18     signals_size_scada); //initiate SCADA-unit
19   id++; // increments the id
20   return id-1; // returns the current id
21 }
22
23 extern void SystemControlInter_update(int system_id, ↩
       double price)
24 {
25   systems.at(system_id)->updateSystem(price);
26 }
27
28 extern void SystemControlInter_updateHighVoltage(int ↩
       system_id, double price)
29 {
30   systems.at(system_id)->updateSystemHighVoltage(price);
31 }
32
```

```
33 extern int Block_init(int system_id, double* ↩
      configuration_vector,
34              int configuration_vector_size,
35              int signals_in_size,
36              int signals_out_size)
37 {
38   return systems.at(system_id)->addResource(↩
        configuration_vector,
39     configuration_vector_size,
40     signals_in_size, signals_out_size);
41 }
42
43 extern void Block_get(int system_id, int resource_id,
44              double* dymola_return_vector,
45              int dymola_return_vector_size)
46 {
47   systems.at(system_id)->getBlock(resource_id, ↩
        dymola_return_vector,
48     dymola_return_vector_size);
49 }
50
51 extern void Block_set(int system_id, int resource_id,
52              double* dymola_set_vector,
53              int dymola_set_vector_size)
54 {
55   systems.at(system_id)->setBlock(resource_id, ↩
        dymola_set_vector,
56     dymola_set_vector_size);
57 }
```

C.2 SCADA

Listing C.3: SCADA.h

```
1  #include "Block.h"
2  #include <vector>
3
4  class SCADA
5  {
6  public:
7    SCADA(void);
8    ~SCADA(void);
9
10   // functions used from Dymola
11   void initSystem(double* configuration_vector, int ↩
       configuration_vector_size,
12     int singals_size_scada);
```

```
13   void getBlock(int resource_id, double* ↩
        dymola_return_vector,
14     int dymola_return_vector_size);
15   void setBlock(int resource_id, double* dymola_set_vector↩
        ,
16     int dymola_set_vector_size);
17
18   int addResource(double* configuration_vector_in, int ↩
        configuration_vector_size,
19     int signals_in_size, int signals_out_size);
20   // functions used to update the systems
21   void updateSystemHighVoltage(double price);
22   void updateSystem(double price);
23   // alias
24   double *measure, controller_id;
25   bool *external_net;
26   bool flag;
27   double *ref;
28   // internal functions
29 private:
30   void SCADA::setController(int id);
31   void SCADA::turnOff(int load_id);
32   void SCADA::turnOn(int resource_id);
33   void SCADA::setController();
34   double SCADA::getMaximumAvailablePower();
35   double SCADA::getPresentConsumedPower();
36   bool SCADA::voltageAtSteadyState();
37   bool SCADA::checkVoltage();
38   bool SCADA::checkPrimaryController();
39   bool SCADA::checkExternalNet();
40   void SCADA::setUpControllers();
41   void SCADA::updateControllers();
42   void SCADA::updateBattery();
43   void SCADA::updateLoads(double maximum_available_power,
44                double present_generated_power,
45              double price);
46   bool SCADA::checkPower(double maximum_available_power,
47     double present_consumed_power);
48   int SCADA::findResource(int resource_type_in, int ↩
        high_low,
49     int on_off);
50   int SCADA::findResource(int resource_type_in, int ↩
        high_low,
51     int on_off, int consumer_profile, double price);
52   void SCADA::updateExternalConnections(double ↩
        available_power);
53   int SCADA::nbrOfResourcesOfType(int resource_type_in);
54   void SCADA::updateGenerator();
55   bool charge_battery_now;
```

```
56    double SCADA::getPowerExternalNet();
57    int SCADA::getNumberOfUnits(int resource_type, int ↩
          on_off);
58    void SCADA::updateBatteryHighVoltage(double price);
59    void SCADA::setControllerHigh();
60
61    // internal variables
62    std::vector<Block*> blocks;
63    // vector with all the resources
64    double* p_configuration_vector_scada;
65    // scada config vector
66    int current_resource_id;
67    // current resources id counter
68    int number_of_resources;
69    // total number of resources
70    double derivative_approximation_vector[2];
71    // used approx derivative
72    int present_sample;
73    // sample counter
74    double* p_voltage_ref;
75    // pointers to voltage ref in config vector
76    double* p_tolerance_voltage;
77    // pointers to tol voltage in config vector
78    double* p_tolerance_power;
79    // pointers to tol power in config vector
80    int primary_controller_id;
81    int external_net_id;
82 };
```

Listing C.4: SCADA.cpp

```
1  #include "SCADA.h"
2  #include <math.h>
3  #include <fstream>
4  #include <iostream>
5
6  #define LOAD 0
7  #define GENERATOR 1
8  #define BATTERY 2
9  #define EXTERNAL_NET 3
10
11 #define BATTERY_EMPTY 100
12 #define BATTERY_CHARGE_LEVEL 500
13 #define BATTERY_DISCHARGE_LEVEL 200
14
15
16 #define BATTERY_EMPTY_HIGH 100
17 #define BATTERY_CHARGE_LEVEL_HIGH 500
18 #define BATTERY_DISCHARGE_LEVEL_HIGH 200
```

```
19
20 #define EXTERNAL_NET_BAD 1000
21
22 #define HIGH 1
23 #define LOW 0
24
25 #define HIGHEST 1
26 #define LOWEST 0
27 #define DEFAULT -1
28 #define ON 1
29 #define OFF 0
30 #define CONSUMER_ON 1
31 #define CONSUMER_OFF 0
32 using namespace std;
33
34 SCADA::SCADA(void)
35 {
36   number_of_resources = 0;
37   controller_id = -1;
38
39   external_net = false;
40   flag = false;
41   external_net_id = -1;
42
43   derivative_approximation_vector[0] = 0;
44   derivative_approximation_vector[1] = 0;
45
46   present_sample = 0;
47 }
48
49 void SCADA::initSystem(double* configuration_vector,
50                        int configuration_vector_size,
51              int signals_size_scada)
52 {
53   p_configuration_vector_scada = (double*)malloc(sizeof(↩
       double)*configuration_vector_size);
54
55   int i;
56   for(i = 0; i < configuration_vector_size; i++)
57   {
58     *(p_configuration_vector_scada+i) = *(↩
         configuration_vector+i);
59   }
60
61   // configure the pointers
62   p_voltage_ref= p_configuration_vector_scada;
63   p_tolerance_voltage = p_configuration_vector_scada+1;
64   p_tolerance_power = p_configuration_vector_scada+2;
65   primary_controller_id = -1;
```

```
66
67    // charge flag
68    charge_battery_now = false;
69  }
70
71  void SCADA::getBlock(int resource_id, double* ↩
       dymola_return_vector,
72                       int dymola_return_vector_size)
73  {
74    return blocks.at(resource_id)->get(dymola_return_vector,
75                                       dymola_return_vector_size↩
                                          );
76  }
77
78  void SCADA::setBlock(int resource_id,
79                       double* dymola_set_vector,
80              int dymola_set_vector_size)
81  {
82    blocks.at(resource_id)->set(dymola_set_vector, ↩
         dymola_set_vector_size);
83  }
84
85  int SCADA::addResource(double* configuration_vector,
86                         int configuration_vector_size,
87                int signals_in_size,
88                int signals_out_size)
89  {
90      blocks.push_back(new Block());
91      blocks.at(current_resource_id)->initBlock(↩
           configuration_vector,
92                                                configuration_vector_size↩
                                                    ,
93                              signals_in_size,
94                              signals_out_size);
95      current_resource_id++;
96      number_of_resources++;
97      return current_resource_id-1;
98  }
99  // Below are functions called from the middle voltage ↩
       system controller ------
100 // Main loop used by the high voltage scada
101 void SCADA::updateSystemHighVoltage(double price)
102 {
103   double available_power = getMaximumAvailablePower() - ↩
         getPresentConsumedPower(); // calculates the power ↩
         available
104   setControllerHigh();
105   updateExternalConnections(available_power);
106   updateGenerator();
```

```
107   updateBatteryHighVoltage(price);
108
109   if(primary_controller_id != -1) // check that a ↩
          controller is initialized
110   {
111     // check if the battery is discharged, if that the ↩
          case update controller is not called.
112     if (*(blocks.at(primary_controller_id)->resource_type)↩
           == BATTERY && *(blocks.at(primary_controller_id)->↩
          state_of_charge) < BATTERY_EMPTY_HIGH)
113     {
114       *(blocks.at(primary_controller_id)->control_current)↩
            = 0;
115     }
116     else
117     {
118       blocks.at(primary_controller_id)->↩
            updateGeneratorPrimary(*p_voltage_ref);
119     }
120   }
121   present_sample++;
122 }
123
124 // used to find suitable controller
125 void SCADA::setControllerHigh()
126 {
127   if(number_of_resources > 1)
128   {
129     int external_controller_id;
130     int battery_id;
131     int iter;
132     // loops through all resources to find the externl ↩
          connector and
133     // battery id.
134     for(iter = 0; iter < number_of_resources; iter++)
135     {
136       if(*(blocks.at(iter)->resource_type) == EXTERNAL_NET↩
            &&
137         *(blocks.at(iter)->↩
            high_low_external_connection_flag) == LOW)
138       {
139         external_controller_id = iter;
140       }
141       else if(*(blocks.at(iter)->resource_type) == BATTERY↩
          )
142       {
143         battery_id = iter;
144       }
145     }
```

```
146     // external connextor is used as initial controller
147     if(primary_controller_id == -1)
148     {
149       primary_controller_id = external_controller_id;
150       setController(primary_controller_id);
151     }
152     // checks power from external connector
153     // if power is not enought the battery will be used
154     else if(*(blocks.at(primary_controller_id)->↩
            resource_type) == EXTERNAL_NET)
155     {
156       if(*blocks.at(primary_controller_id)->maximum_power ↩
             < 1000)
157       {
158         *(blocks.at(primary_controller_id)->↩
               control_current) = 0;
159         *(blocks.at(primary_controller_id)->on_off) = 0;
160         primary_controller_id = battery_id;
161         setController(primary_controller_id);
162       }
163     }
164     // checks power available on external connector when ↩
            the battery is used
165     else if(*(blocks.at(primary_controller_id)->↩
            resource_type) == BATTERY)
166     {
167       if(*blocks.at(external_controller_id)->maximum_power↩
             > 1000)
168       {
169         *(blocks.at(primary_controller_id)->↩
               control_current) = 0;
170         *(blocks.at(primary_controller_id)->on_off) = 0;
171         primary_controller_id = external_controller_id;
172         setController(primary_controller_id);
173       }
174     }
175   }
176 }
177
178 // updates the power to the houses
179 void SCADA::updateExternalConnections(double ↩
       available_power)
180 {
181   int iter;
182   int number_of_external_connections = ↩
         nbrOfResourcesOfType(EXTERNAL_NET);
183   if (number_of_resources == 1)
184   {
```

```
185       *(blocks.at(0)->↩
              power_available_higher_voltage_grid_set) =
186         *(blocks.at(0)->maximum_power);
187     }
188     else
189     {
190       double power_external_net = getPowerExternalNet();
191       for(iter = 0; iter < number_of_resources; iter++)
192       {
193
194         if(*(blocks.at(iter)->resource_type) == EXTERNAL_NET
195           && *(blocks.at(iter)->↩
                  high_low_external_connection_flag) == HIGH)
196         {
197           if(*(blocks.at(primary_controller_id)->↩
                  resource_type) == BATTERY
198             && *(blocks.at(primary_controller_id)->↩
                    state_of_charge) < 1.5*BATTERY_EMPTY_HIGH
199             || flag == true)
200           {
201             flag = true;
202             *(blocks.at(iter)->↩
                    power_available_higher_voltage_grid_set) = 0;
203           }
204           else if(checkVoltage())
205           {
206             *(blocks.at(iter)->↩
                    power_available_higher_voltage_grid_set) =
207               power_external_net / (↩
                    number_of_external_connections - 1);
208           }
209         }
210       }
211     }
212 }
213
214 // calculates the available power
215 double SCADA::getPowerExternalNet()
216 {
217   int iter;
218   double accumelated_maximum_power;
219   for(iter = 0; iter < number_of_resources; iter++)
220   {
221     if((*(blocks.at(iter)->resource_type) == EXTERNAL_NET ↩
          && *(blocks.at(iter)->↩
          high_low_external_connection_flag) == LOW) ||
222         *(blocks.at(iter)->resource_type) == GENERATOR ||
223         *(blocks.at(iter)->resource_type) == BATTERY && *(↩
            blocks.at(iter)->state_of_charge) > ↩
```

```
          BATTERY_EMPTY_HIGH)
224     {
225       accumelated_maximum_power += *(blocks.at(iter)->↩
            maximum_power);
226     }
227   }
228   return accumelated_maximum_power;
229 }
230
231 // calculates the number of resources of a given type
232 int SCADA::nbrOfResourcesOfType(int resource_type_in)
233 {
234   int number_of_resources_of_type = 0;
235   int iter;
236   for(iter = 0; iter < number_of_resources; iter++)
237   {
238     if(*(blocks.at(iter)->resource_type) == ↩
          resource_type_in)
239     {
240       number_of_resources_of_type++;
241     }
242   }
243   return number_of_resources_of_type;
244 }
245
246 // updates the battery instructions
247 void SCADA::updateBatteryHighVoltage(double price)
248 {
249   int battery_id = findResource(BATTERY,HIGHEST, DEFAULT);
250   int price_limit_charge = 240;
251   int price_limit_discharge = 320;
252   if(number_of_resources > 1)
253   {
254     if(battery_id != primary_controller_id && -1 != ↩
          primary_controller_id && battery_id != -1)
255     {
256
257       if(checkVoltage() && *(blocks.at(battery_id)->↩
            voltage_external) <
258         BATTERY_CHARGE_LEVEL_HIGH && price < ↩
              price_limit_charge )
259       {
260       *(blocks.at(battery_id)->control_current) =
261         -blocks.at(battery_id)->powerLimitation(0.1*(↩
              price_limit_charge - price), 0.001);
262       }
263       else if(checkVoltage() && *(blocks.at(battery_id)->↩
            voltage_external) >
```

```
264         BATTERY_DISCHARGE_LEVEL_HIGH && price > ↩
               price_limit_discharge)
265       {
266         *(blocks.at(battery_id)->control_current) =
267           blocks.at(battery_id)->powerLimitation(0.1*(↩
                 price - price_limit_discharge), 0.001);
268       }
269       else
270       {
271         *(blocks.at(battery_id)->control_current) = 0;
272       }
273     }
274   }
275 }
276 // end of middle voltage conotrol functions ↩
    ------------------------------------
277 // Below are functions used to control house level
278 // Main loop for the controlller on house level
279 void SCADA::updateSystem(double price)
280 {
281   setController();
282   updateControllers();
283   updateBattery();
284   updateGenerator();
285
286   if(present_sample%27 == 0){
287     updateLoads(0,0,price);
288   }
289   present_sample++;
290 }
291
292 // Finds a suitabele controller
293 void SCADA::setController()
294 {
295   // checks it there are resources connected to the ↩
       controller
296   if(number_of_resources != 0)
297   {
298     int external_id = findResource(EXTERNAL_NET, HIGHEST, ↩
         DEFAULT);
299     int primary_controller_id_battery = findResource(↩
         BATTERY, HIGHEST, DEFAULT);
300
301     if(primary_controller_id == -1) // initializes a ↩
         controller
302     {
303       setUpControllers();
304     }
```

```
305     else if (*(blocks.at(primary_controller_id)->↩
            resource_type) == EXTERNAL_NET &&
306       *(blocks.at(primary_controller_id)->↩
            power_available_higher_voltage_grid)
307       < EXTERNAL_NET_BAD)
308     {
309       *(blocks.at(primary_controller_id)->control_current)↩
            = 0;
310       setUpControllers();
311     }
312     else if (*(blocks.at(primary_controller_id)->↩
            resource_type) == BATTERY &&
313       *(blocks.at(external_id)->↩
            power_available_higher_voltage_grid)
314       > EXTERNAL_NET_BAD)
315     {
316       *(blocks.at(primary_controller_id)->control_current)↩
            = 0;
317       setUpControllers();
318     }
319   }
320 }
321
322 // Checks the condition on the external net
323 bool SCADA::checkExternalNet()
324 {
325   int external_net_id = findResource(EXTERNAL_NET, HIGHEST↩
        , DEFAULT);
326   if(*(blocks.at(primary_controller_id)->resource_type) !=↩
        EXTERNAL_NET &&
327      *(blocks.at(external_net_id)->voltage_external) > ↩
          EXTERNAL_NET_BAD)
328   {
329       return true;
330   }
331   else
332   {
333     return false;
334   }
335 }
336
337 // finds a suitable contorller
338 void SCADA::setUpControllers()
339 {
340   int primary_controller_id_external = findResource(↩
        EXTERNAL_NET, HIGHEST, DEFAULT);
341   int primary_controller_id_battery = findResource(BATTERY↩
        , HIGHEST, DEFAULT);
342
```

```
343   if(*(blocks.at(primary_controller_id_external)->↩
          power_available_higher_voltage_grid) >
344     EXTERNAL_NET_BAD )
345   {
346     primary_controller_id = primary_controller_id_external↩
          ;
347   }
348   else if(*(blocks.at(primary_controller_id_battery)->↩
          state_of_charge) >
349     BATTERY_EMPTY || present_sample == 0)
350   {
351     primary_controller_id = primary_controller_id_battery;
352   }
353   setController(primary_controller_id);
354 }
355
356
357 // checks the condition on the controler
358 bool SCADA::checkPrimaryController()
359 {
360   if(primary_controller_id == -1)
361   {
362     return false;
363   }
364   else if(*(blocks.at(primary_controller_id)->↩
          power_available_higher_voltage_grid) <
365     EXTERNAL_NET_BAD
366     && *(blocks.at(primary_controller_id)->resource_type) ↩
          == EXTERNAL_NET)
367   {
368     *(blocks.at(primary_controller_id)->control_current) =↩
          0;
369     *(blocks.at(primary_controller_id)->on_off) = 0;
370     return false;
371   }
372   else
373   {
374     return true;
375   }
376 }
377
378 // calls the PI-controller
379 void SCADA::updateControllers()
380 {
381     if(primary_controller_id != -1)
382     {
383       double control_current_primary =
384         blocks.at(primary_controller_id)->↩
            updateGeneratorPrimary(*p_voltage_ref);
```

```
385     }
386 }
387
388 // Instructions the battery in the house
389 void SCADA::updateBattery()
390 {
391   int battery_id = findResource(BATTERY,HIGHEST, DEFAULT);
392
393   if(battery_id != primary_controller_id)
394   {
395     if(*(blocks.at(battery_id)->voltage_external) <
396       BATTERY_CHARGE_LEVEL
397       && *(blocks.at(battery_id)->voltage_grid) >
398       *p_voltage_ref* (1 - *p_tolerance_voltage))
399     {
400       *(blocks.at(battery_id)->control_current) = -0.5;
401     }
402     else
403     {
404       *(blocks.at(battery_id)->control_current) = 0;
405     }
406   }
407 }
408
409 // Updates the generator output power
410 void SCADA::updateGenerator()
411 {
412   int iter;
413   for(iter = 0; iter < number_of_resources; iter++)
414   {
415     if(*(blocks.at(iter)->resource_type) == GENERATOR)
416     {
417       blocks.at(iter)->updateGenerator();
418     }
419   }
420 }
421
422 // Manges and updates the loads
423 void SCADA::updateLoads(double maximum_available_power1,
424               double present_consumed_power1,
425             double price)
426 {
427   // resets the delay_counter
428   int i = 0;
429   for(i = 0; i <number_of_resources; i++)
430   {
431     if(*blocks.at(i)->consumed_power_profile == ↩
        CONSUMER_ON
432       && blocks.at(i)->delay_counter == 0)
```

```
433       {
434         blocks.at(i)->consumer_flag = true;
435       }
436       else if (blocks.at(i)->delay_counter > *blocks.at(i)->↩
              maximum_delay)
437       {
438         blocks.at(i)->consumer_flag = false;
439         blocks.at(i)->delay_counter = 0;
440       }
441     }
442
443     // Turns off loads that have run through operation_time ↩
        and resets the counter
444     for(i = 0; i <number_of_resources; i++)
445     {
446       if(*blocks.at(i)->resource_type == LOAD &&
447           blocks.at(i)->operation_time_counter > *blocks.at(↩
              i)->operation_time)
448       {
449         *(blocks.at(i)->on_off) = 0;
450         blocks.at(i)->consumer_flag = false;
451
452         blocks.at(i)->operation_time_counter = 0;
453       }
454     }
455
456     // calculates the available and consumed power
457     double maximum_available_power = ↩
        getMaximumAvailablePower();
458     double present_consumed_power = getPresentConsumedPower↩
        ();
459
460     // checks voltage level, steady state adn if there is
461     // power available
462     if(voltageAtSteadyState() && checkVoltage() &&
463       checkPower(maximum_available_power, ↩
          present_consumed_power))
464     {
465       // finds nest load to turn on, if there are any
466       int load_id = findResource(LOAD, HIGHEST, OFF, 0, ↩
          price);
467       if(load_id != -1)
468       {
469         double power_from_load = *(p_voltage_ref) *
470           *(p_voltage_ref) / *(blocks.at(load_id)->↩
              resistance);
471         // turns it on if there is enough power
472         if(power_from_load + present_consumed_power <
473           maximum_available_power)
```

```
474         {
475           turnOn(load_id);
476           blocks.at(load_id)->delay_counter = 0;
477         }
478       }
479     }
480     // turns of lowest priority load if grid conditions ↩
          changes
481     else if(maximum_available_power < present_consumed_power↩
          )
482     {
483       int load_id_off = findResource(LOAD, LOWEST, ON);
484       if(load_id_off != -1)
485       {
486         turnOff(load_id_off);
487       }
488     }
489 }
490
491 // checks whether there are power available or not
492 bool SCADA::checkPower(double maximum_available_power,
493   double present_consumed_power)
494 {
495   return (maximum_available_power > (1+*p_tolerance_power)↩
          *
496     present_consumed_power);
497 }
498
499 // checks the voltage level on the grid
500 bool SCADA::checkVoltage()
501 {
502   if(*(blocks.at(primary_controller_id)->voltage_grid) <
503     (1 +*p_tolerance_voltage)* *p_voltage_ref &&
504      *(blocks.at(primary_controller_id)->voltage_grid) >
505      (1 - *p_tolerance_voltage)* *p_voltage_ref)
506   {
507     return true;
508   }
509   else
510   {
511     return false;
512   }
513 }
514
515
516
517 // Turns off resources
518 void SCADA::turnOff(int resource_id)
519 {
```

```
520   *(blocks.at(resource_id)->on_off) = 0;
521 }
522
523 // Turns on resources
524 void SCADA:: turnOn(int resource_id)
525 {
526   *(blocks.at(resource_id)->on_off) = 1;
527 }
528
529 // configures the controller
530 void SCADA::setController(int controller_id)
531 {
532   *(blocks.at(controller_id)->on_off) = 1;
533   (blocks.at(controller_id)->setController(true));
534
535 }
536
537 // calculates the maximum available power on the grid
538 double SCADA::getMaximumAvailablePower()
539 {
540   int battery_id = findResource(BATTERY,HIGHEST,DEFAULT);
541   if(battery_id != -1){
542     if(*(blocks.at(battery_id)->voltage_external) < ↩
          BATTERY_EMPTY)
543     {
544       charge_battery_now = true;
545     }
546     else if (*(blocks.at(battery_id)->voltage_external) > ↩
          200)
547     {
548       charge_battery_now = false;
549     }
550   }
551
552   int i;
553   double accumelated_maximum_power = 0;
554   for(i = 0; i<number_of_resources; i++)
555   {
556     if(*(blocks.at(i)->resource_type) == GENERATOR ||
557         *(blocks.at(i)->resource_type) == EXTERNAL_NET ||
558         (*(blocks.at(i)->resource_type) == BATTERY
559       && *(blocks.at(i)->voltage_external) >
560       BATTERY_EMPTY && primary_controller_id == i
561       && charge_battery_now == false) &&
562         *(blocks.at(i)->on_off) == 1)
563     {
564       accumelated_maximum_power += *(blocks.at(i)->↩
            maximum_power);
565     }
```

```
566   }
567   return accumelated_maximum_power;
568 }
569
570 // returns the current consumed power
571 double SCADA::getPresentConsumedPower()
572 {
573   int i;
574   double accumelated_generated_power = 0;
575   for(i = 0; i<number_of_resources; i++)
576   {
577     if((*(blocks.at(i)->resource_type) == LOAD ||
578         (*(blocks.at(i)->resource_type) == BATTERY && ↩
            primary_controller_id != i))
579        && *(blocks.at(i)->on_off) == 1)
580     {
581       if (*(blocks.at(i)->resource_type))
582       {
583         accumelated_generated_power -= *(blocks.at(i)->↩
            power_out);
584
585
586       }
587       else
588       {
589         accumelated_generated_power += *(blocks.at(i)->↩
            power_out);
590       }
591     }
592   }
593   return accumelated_generated_power;
594 }
595
596 // checks if the voltage level is in steady state
597 bool SCADA::voltageAtSteadyState()
598 {
599   if(primary_controller_id != -1)
600   {
601     derivative_approximation_vector[0] = *(blocks.at(↩
        primary_controller_id)->voltage_grid);
602     double derivative_approximation =
603          (derivative_approximation_vector[0]-↩
             derivative_approximation_vector[1])/
604          (*(blocks.at(primary_controller_id)->↩
             sample_period));
605
606
607     if(derivative_approximation<0.01 && ↩
        derivative_approximation> -0.01)
```

```
608     {
609       derivative_approximation_vector[1] = ↩
             derivative_approximation_vector[0];
610       return true;
611     }
612     derivative_approximation_vector[1] = ↩
          derivative_approximation_vector[0];
613   }
614   return false;
615 }
616
617 // finds resource of a given type
618 int SCADA::findResource(int resource_type_in, int high_low↩
    , int on_off)
619 {
620     int i;
621     int temporary_high_priority = 5;
622     int temporary_low_priority = -1;
623     int resource_id = -1;
624     for(i = 0; i < number_of_resources; i++)
625     {
626         if(*(blocks.at(i)->resource_type) == ↩
             resource_type_in &&
627             (*(blocks.at(i)->on_off) == on_off|| on_off == ↩
                DEFAULT))
628         {
629             if(high_low == HIGHEST) //
630             {
631                 if(*(blocks.at(i)->priority) < ↩
                    temporary_high_priority)
632                 {
633                     resource_id = i;
634                     temporary_high_priority= (int)*(blocks↩
                        .at(i)->priority);
635                 }
636             }
637             else
638             {
639                 if(*(blocks.at(i)->priority) > ↩
                    temporary_low_priority)
640                 {
641                     resource_id = i;
642                     temporary_low_priority= (int)*(blocks.↩
                        at(i)->priority);
643                 }
644             }
645
646         }
647     }
```

```
648     return resource_id;
649 }
650
651 // finds next load to turn on, considering price limits
652 int SCADA::findResource(int resource_type_in, int high_low↩
    ,
653   int on_off, int consumer_profile, double price)
654 {
655   int i;
656   int temporary_high_priority = 5;
657   int temporary_low_priority = -1;
658   int resource_id = -1;
659   for(i = 0; i < number_of_resources; i++)
660   {
661     if(*(blocks.at(i)->resource_type) == LOAD)
662     {
663       //cout << "flag: " << blocks.at(i)->consumer_flag <<↩
          endl;
664     }
665
666     if(*(blocks.at(i)->resource_type) == resource_type_in ↩
        &&
667     (*(blocks.at(i)->on_off) == on_off|| on_off == DEFAULT↩
        ) &&
668     (blocks.at(i)->consumer_flag))
669     {
670       if(high_low == HIGHEST)
671       {
672         if(*(blocks.at(i)->priority) < ↩
             temporary_high_priority)
673         {
674           if (*(blocks.at(i)->price_limit) > price ||
675             *(blocks.at(i)->price_limit) == -1 ||
676             (double)blocks.at(i)->delay_counter >
677             *blocks.at(i)->maximum_delay - 1)
678           {
679
680             resource_id = i;
681             temporary_high_priority= (int)*(blocks.at(i)->↩
                 priority);
682           }
683         }
684       }
685       else
686       {
687         if(*(blocks.at(i)->priority) > ↩
             temporary_low_priority)
688         {
689           resource_id = i;
```

```
690             temporary_low_priority= (int)*(blocks.at(i)->↩
                    priority);
691           }
692         }
693
694       }
695     }
696     return resource_id;
697
698 }
699
700 // determines the number of resources of a given type
701 int SCADA::getNumberOfUnits(int resource_type, int on_off)
702 {
703     int counter, iter;
704     for(iter = 0; iter < number_of_resources; iter++)
705     {
706       if(*(blocks.at(iter)->resource_type) == resource_type
707         && *(blocks.at(iter)->on_off) == on_off)
708       {
709         counter++;
710       }
711     }
712     return counter;
713 }
```

C.3 Block

Listing C.5: block.h

```
1  #include "Pi.h"
2  class Block
3  {
4  public:
5    Block(void);
6    void initBlock(double* configuration_vector,
7                   int configuration_vector_size,
8              int signals_in_size,
9              int signals_out_size);
10   void get(double* dymola_return_vector, int ↩
       dymola_return_vector_size);
11   void set(double* dymola_set_vector, int ↩
       dymola_set_vec_size);
12   void updateBlock(double voltage_ref);
13
14   void setController(bool set);
15   void Block::updateGenerator();
```

```
16    double voltageGrid(void);
17    double Block::powerLimitation(double control_current, ↩
          double epsilon);
18
19    double* signals_out;
20    double* signals_in;
21    double* configuration_vector;
22
23    int configuration_vector_size;
24    int signals_in_size;
25    int signals_out_size;
26
27    // Set up pointers
28    //configuration vector
29    double *resource_type, *priority, *maximum_current, *k, ↩
          *Ti, *Tt, *y_min;
30    double *y_max, *sample_period, *↩
          high_low_external_connection_flag;
31    double *price_limit, *maximum_delay, *operation_time;
32    // signals out
33    double *voltage_grid, *power_out, *voltage_external, *↩
          state_of_charge;
34    double *maximum_power, *resistance, *↩
          power_available_higher_voltage_grid;
35    double *consumed_power_profile;
36    //signals in
37    double *on_off, *control_current, *↩
          power_available_higher_voltage_grid_set;
38    double updateGeneratorPrimary(double voltage_ref);
39    double updateGeneratorSecondary(double control_current);
40
41    // internal variables
42    bool consumer_flag;
43    int delay_counter;
44    int operation_time_counter;
45
46  private:
47    // controller variables
48    Pi* pi;
49    bool controller_in_use;
50
51    void updateExternalConnector(double voltage_ref);
52    double controllerUpdatePrimary(double voltage_ref);
53    double controllerUpdateSecondary(double control_current)↩
          ;
54  };
```

Listing C.6: block.cpp

```
1  #include "Block.h"
2  #include <iostream>
3
4  #define LOAD 0
5  #define GENERATOR 1
6  #define BATTERY 2
7  #define EXTERNAL_NET 3
8  #define ON 1
9  #define OFF 0
10 using namespace std;
11
12 Block::Block(void)
13 {
14   controller_in_use = false;
15 }
16
17 void Block::initBlock(double* configuration_vector_in,
18                       int configuration_vector_size_in,
19             int signals_in_size_in,
20             int signals_out_size_in)
21 {
22   this->configuration_vector_size = ↩
       configuration_vector_size_in;
23   this->signals_in_size = signals_in_size_in;
24   this->signals_out_size = signals_out_size_in;
25
26   signals_out = (double*) malloc(sizeof(double)*↩
       signals_out_size_in);
27   signals_in = (double*) malloc(sizeof(double)*↩
       signals_in_size_in);
28   configuration_vector = (double*) malloc(sizeof(double)*↩
       configuration_vector_size_in);
29
30   int i;
31   for(i = 0; i < configuration_vector_size; i++)
32   {
33     *(configuration_vector+i) = *(configuration_vector_in+↩
         i);
34   }
35
36   for(i = 0; i < signals_in_size; i++)
37   {
38     *(signals_in + i) = 0;
39   }
40
41   for(i = 0; i < signals_out_size; i++)
42   {
43     *(signals_out + i) = 0;
44   }
```

```
45
46    // configure the pointers configuration vector
47    resource_type = (configuration_vector+0);
48    priority = (configuration_vector+1);
49    maximum_current = (configuration_vector+2);
50    k = (configuration_vector+3);
51    Ti = (configuration_vector+4);
52    Tt = (configuration_vector+5);
53    y_min = (configuration_vector+6);
54    y_max = (configuration_vector+7);
55    sample_period = (configuration_vector+8);
56    high_low_external_connection_flag = (↩
         configuration_vector+9);
57    price_limit = (configuration_vector+10);
58    maximum_delay = (configuration_vector+11);
59    operation_time = (configuration_vector+12);
60
61    // configure the pointers signals out
62    voltage_grid = (signals_out+0);
63    power_out = (signals_out+1);
64    voltage_external = (signals_out+2);
65    state_of_charge = (signals_out+3);
66    maximum_power = (signals_out+4);
67    resistance = (signals_out+5);
68    power_available_higher_voltage_grid = (signals_out+6);
69    consumed_power_profile = (signals_out+7);
70
71    // configure the pointers signals in
72    on_off = (signals_in+0);
73    control_current = (signals_in+1);
74    // ----------
75    power_available_higher_voltage_grid_set = (signals_in+2)↩
         ;
76
77    //i internal variables
78    consumer_flag = false;
79    delay_counter = 0; // initialize delay counter to 0
80    operation_time_counter = 0;
81  }
82
83  // stores the value of signals_in to the ↩
       dymola_return_vecor and
84  // increments the operation and delay time counters
85  void Block::get(double* dymola_return_vector, int ↩
       dymola_return_vector_size)
86  {
87    // update the delay and poeration time
88    if(*resource_type == LOAD)
89    {
```

```
90      if(*on_off == ON)
91      {
92        operation_time_counter++; // increment the operation↩
              time counter;
93      }
94      else if(consumer_flag == true)
95      {
96        delay_counter++;      // increment the operation ↩
              time counter
97      }
98    }
99    int i;
100   for(i = 0; i < dymola_return_vector_size; i++)
101   {
102     *(dymola_return_vector + i) = *(signals_in+i);
103   }
104 }
105
106 // stores the value of the dymola_set_vector into the ↩
      vector signals_out
107 void Block::set(double* dymola_set_vector, int ↩
      dymola_set_vector_size)
108 {
109   int i;
110   for(i = 0; i < dymola_set_vector_size; i++)
111   {
112     *(signals_out + i) = *(dymola_set_vector + i);
113   }
114 }
115
116 // returns the control current for the generators
117 double Block::updateGeneratorPrimary(double voltage_ref)
118 {
119     double epsilon = 10; // voltage limit for the ↩
          powerLimitation function
120     *control_current = pi->update(*voltage_grid, ↩
          voltage_ref);  //calculate control signal
121     *control_current = powerLimitation(*control_current, ↩
          epsilon);  // limit control signal
122     return *control_current;
123 }
124
125 // initializes a new controller
126 void Block::setController(bool set)
127 {
128   pi = new Pi();
129   pi->init(*k,*Ti,*Tt,*y_min,*y_max, *sample_period);
130   controller_in_use = set;
131   *on_off = 1;
```

```
132 }
133
134
135 // return the voltage of the grid
136 double Block::voltageGrid()
137 {
138   return *voltage_grid;
139 }
140
141 // Limits the control current to satisfy the maximum power↩
      condition
142 // epsilon is the voltage limit
143 double Block::powerLimitation(double control_current, ↩
    double epsilon)
144 {
145   if(*voltage_grid<epsilon && *voltage_grid>-epsilon )
146   {
147     return control_current;
148   }
149   else if (control_current > (*maximum_power)/(*↩
       voltage_grid))
150   {
151     return (*maximum_power)/(*voltage_grid);
152   }
153   else if (control_current < -(*maximum_power)/(*↩
       voltage_grid))
154   {
155     return -(*maximum_power)/(*voltage_grid);
156   }
157   return control_current;
158 }
159
160 // updates the generator by outputs the produced power to ↩
    the grid
161 void Block::updateGenerator()
162 {
163   if(*voltage_grid < 0.01 && *voltage_grid > -0.01 ){
164     *control_current = powerLimitation(*maximum_current↩
         ,0.0001);
165   }else
166   {
167     *control_current = *maximum_power/ *voltage_grid;
168     *control_current = powerLimitation(*control_current, ↩
         0.0001);
169   }
170 }
```

C.4 Pi controller

Listing C.7: pi.h

```
1  class Pi
2  {
3  public:
4    Pi(void);
5    void init(double k, double ti, double tt,
6            double y_min, double y_max, double sample_period↩
                );
7    double update(double voltage_measure, double voltage_ref↩
       );
8    ~Pi(void);
9
10 private:
11   double I; //integrated error
12   double k, Ti; //proporational gain and integrator time ↩
       constant
13   double Tt; //time constant for anti windup
14
15   double y_min, y_max; //max and min value for actuator
16   double sample_period; //sample period of controller
17 };
```

Listing C.8: pi.cpp

```
1  #include "Pi.h"
2  #include <iostream>
3  using namespace std;
4
5  // Initializing all control parameters to zero.
6  Pi::Pi(void)
7  {
8    this->I = 0;//integrated error
9    this->k = 0;//proporational gain
10   this->Ti = 0; //integrator time constant
11   this->Tt = 0;//time constant for anti windup
12
13   this->y_min = 0;//max value for actuator
14   this->y_max = 0;//min value for actuator
15   this->sample_period = 0;//sample period of controller
16 }
17
18 // Sets all the control parameters.
19 void Pi::init(double k, double Ti, double Tt, double y_min↩
     ,
20             double y_max, double sample_period)
```

```
21 {
22   this->k = k;
23   this->Ti = Ti;
24   this->Tt = Tt;
25   this->y_min = y_min;
26   this->y_max = y_max;
27   this->sample_period = sample_period;
28 }
29
30 // Called to return the control current.
31 double Pi::update(double voltage_measure, double ↩
       voltage_ref)
32 {
33   double y, temp, error;
34   error = voltage_ref-voltage_measure;
35   temp = k*(error+(sample_period/Ti)*I);
36
37   //anti windup
38   if(temp>y_max)
39   {
40     y = y_max;
41   }
42   else if(temp<y_min)
43   {
44     y = y_min;
45   }
46   else
47   {
48     y = temp;
49   }
50   I = I + error + (y-temp)/(k*Tt);
51   return y;
52 }
53
54 // Destructor not used.
55 Pi::~Pi(void)
56 {
57 }
```

Bibliography

[1] J. Cook, D. Nuccitelli, S. A. Green, M. Richardson, B. Winkler, R. Painting, R. Way, P. Jacobs, and A. Skuce, "Quantifying the consensus on anthropogenic global warming in the scientific literature," *ENVIRONMENTAL RESEARCH LETTERS*, 2013.

[2] U. D. of Energy, *The Smart Grid: An Introduction*. Office of Electricity Delivery and Energy Reliability, 2008.

[3] W. Zhang, H. Liang, Z. Bin, W. Li, and R. Guo, "Review of dc technology in future smart distribution grid," *Innovative Smart Grid Technologies - Asia (ISGT Asia), IEEE*, 2012.

[4] A. Yoza, K. Uchida, A. Yona, and T. Senjyu, "Optimal operation of controllable loads in dc smart house with ev," *IEEE , Renewable Energy Research and Applications International Conference*, pp. 1–6, 2012.

[5] OECD, ed., *OECD Factbook 2011-2012: Economic, Environmental and Social Statistics*. Oecd Factbook, Brookings Inst Press, 2012.

[6] IAEA, *Nuclear Power Reactors in the World 2012 Edition*. IAEA, 2012. ISBN 978-92-0-132310-1.

[7] "Iea clean coal centre's coalpower database."

[8] I. A. E. A. (IAEA), "Iaea pris (power reactor information system)."

[9] M. Alaküla, L. Gertmar, and O. Samuelsson, *Elenergiteknik*. Department of Industrial Electrical Engineering and Automation, Lund Univerity, 2011.

[10] H. Pidd, "India blackout leave 700 million without power," *The Guardian*, July 2012.

[11] "U.s. energy information administration - electric power monthly," 2013.

[12] K. Moslehi and R. Kumar, "Smart grid - a realiability perspective," *Smart Grid, IEEE Transactions*, 2010.

[13] ABB, "Gotland - fullskalig testplats för smarta elnät." http://www.abb.com/cawp/db0003db002698/be5d45ce73178831c125788500384649.aspx, May 2013.

[14] "Smart grid gotland." http://www.smartgridgotland.se/, May 2013.

[15] CCI, "Clinton climate initiative." http://www.clintonfoundation.org/main/our-work/ by-initiative/clinton-climate-initiative/programs/c40-cci-cities/climate-positive-development-program.html, May 2013.

[16] ABB, "Norra djurgårdsstaden." http://www.abb.com/cawp/db0003db002698/2059d29c34a9f600c12578fb003d24e7.aspx, May 2013.

[17] S. Coughlin, "Smart grid: A smart idea for america?," *IEEE, highlighted articles*.

[18] S. Bengtlars and E. Lidén, "Risk and vulnerability analyses for smart grids," Master's thesis, Uppsala Univeristy, 2012.

[19] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004. ISBN 0-471-471631.

[20] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Prentice Hall P T R, 2nd ed., March 1988. ISBN 0-13-110370-9.

[21] J. Svensson, *Active Distributed Power Systems*. Lund University, 2006. ISBN 91-88934-43-8.

[22] M. Multin, F. Allerding, and H. Schmeck, "Integration of electric vehicles in smart homes - an ict-based solution for v2g scenarios," *IEEE PES Innovative Smart Grid Technologies*, pp. 1–8, 2012.

[23] J. F. Manwell, J. G. MacGowan, and A. L. Rogers, *Wind energy explained - theory, design and application*. Wiley, second ed., 2009.

[24] T. colorado River Commission of Nevada, "World fossil fuel reserves and projected depletion," 2002.

[25] K. J. Astrom and T. Hagglund, *Advanced PID Control*. ISA- The Instrumentation, Systems, and Automation Society, 2006. ISBN 1-55617-942-1.