Dynamic Modelling with OpenModelica

Optimising Building Insulation

Bachelor’s Thesis

Bachelor of Engineering in Electrical and Automation Engineering

Spring 2022

John Steedman

Electrical and Automation Engineering | Abstract
Author John Steedman | Year 2022

Subject Dynamic Modelling with OpenModelica: Optimising Building Insulation

Supervisors Juhani Henttonen

This thesis shows how multi-domain modelling can be automated to allow the fast discovery of desired model configurations. For this purpose, finding the optimal building insulation of a simple building in various climate and construction scenarios was used here as an example.

A key outcome of the thesis was a demonstration of the provision of a robust, open source/libre tool chain that could be used to model and optimise complex systems in an automated manner. This led to a further outcome of optimised wall insulation for the modelled building.

The goal of reducing heat demand in a small Finnish cabin through the method of modelling combinations of insulation types and characteristics using OpenModelica and Python resulted in a marked reduction in required heat input whilst maintaining a comfortable indoor temperature.

Keywords Modelica, OpenModelica, Python, Building Insulation.
Pages 43 pages and appendices 11 pages

Electrical and Automation Engineering Tiivistelmä

Tekijä John Steedman Vuosi 2022

Työn nimi Dynamic Modelling with OpenModelica: Optimising Building Insulation

Ohjaaja Juhani Henttonen

Tämä opinnäytetyö osoittaa, kuinka monitoimialuemallinnus voidaan automatisoida niin, että halutut mallikonfiguraatiot löydetään nopeasti. Tätä varten käytetään esimerkkinä yksinkertaista rakennusta optimaalisen rakennuseristyksen löytämiseksi erilaisissa ilmasto- ja rakennusskenaarioissa.

Opinnäytetyön keskeinen tulos on demonstroida vankka, avoimen lähdekoodin/vapaan työkaluketjun tarjoaminen, jota voidaan käyttää monimutkaisten järjestelmien mallintamiseen ja optimointiin automatisoidusti. Tämä johtaa mallinnetun rakennuksen optimoidun seinäeristyksen lisätulokseen.

Tavoite pienentää pienen suomalaismökin lämmöntarvetta mallintamalla eristystyyppien ja -ominaisuuksien yhdistelmiä OpenModelica- ja Python-menetelmillä johti siihen, että tarvittava lämmöntuotto pieneni huomattavasti ja samalla säilytettiin miellyttävä sisälämpötila.

Avainsanat Modelica, OpenModelica, Python, Rakennuksen eristys.

Sivut 43 sivua ja liitteitä 11 sivua

Contents

1 Introduction ... 1

2 Models and Simulation ... 3

2.1 Causal and Acausal ... 4

2.2 OpenModelica Model Translation ... 6

2.3 Modelica ... 8

2.4 Introducing Modelling with Modelica ... 11

2.4.1 Types ... 12

2.4.2 Connectors ... 12

2.4.3 Partial Models ... 13

2.4.4 Models of the Circuit Components ... 14

2.4.5 Model of the Circuit ... 15

2.4.6 Simulate the Circuit Model ... 15

2.5 Further Modelling with Modelica ... 16

2.5.1 Heater Model ... 16

2.5.2 Heat Capacitor Model ... 18

2.5.3 Thermal Conductor Model ... 19

2.5.4 Simulation of a Water Heater ... 20

2.6 Automated Modelling Toolchain ... 21

2.6.1 Compiling and Simulating the Model ... 22

2.7 Modelling Building Systems ... 24

2.7.1 Walls ... 24

2.7.2 Windows ... 26

2.7.3 Building Systems Modelica Library ... 27

3 Optimising Building Insulation ... 28

3.1 Model ... 28

3.1.1 Ambient Climate ... 29

3.1.2 Building Structures ... 30

3.1.3 Building ... 31

3.1.4 Intra-Model Connections ... 31

3.2 Execution of Automated Simulations ... 32

3.2.1 Reduce Heat Demand 34
3.2.2 Change the Windows 36
4 Conclusions and Reflection 38
4.1 Lessons Learnt 38
4.2 Further Opportunities 39
References 41

Appendices

Appendix 1: Code Listing of ExampleCircuit Modelica Package
Appendix 2: Code Listing of Insulation Modelica Package
Appendix 3: Code Listing of Heater Python Code
Appendix 4: Code Listing of Insulation Python Code
Appendix 5: Sample Report of Multiple Simulation Scenarios
Appendix 6: Net Present Value Calculation for Heat Pump Investment

1 Introduction

"Modelling and simulation are indispensable when dealing with complex engineering systems." (Åström, Elmqvist, & Mattsson, 1998)

The aim of this thesis was to demonstrate the enormous potential for engineers of the Modelica modelling language, especially its open-source implementation, OpenModelica. Engineers use models of physical systems in order to improve the design, techniques, processes, training, and performance in a system. Easy availability of high-power computing now allows models to be created and simulated as mathematical models in a virtual environment. Physical models, or prototypes, are useful in cases such as automotive or building engineering but are often expensive and have limited experimental scope (Fritzson, 2015, p. 6).

In electrical engineering, modelling of electrical circuits using computers has been done for many decades, for example, the SPICE circuit emulator was introduced in the early 1970s (Nagel & Pederson, 1973). There have also been many domain specific modelling tools created for mechanical, chemical, and energy systems, for example (Åström, Elmqvist, & Mattsson, 1998). These tools are essentially standalone as modelling of systems across domain boundaries can be very complex or even impossible. Many general-purpose software packages for modelling and simulation are also available, one of the most well-known being MATLAB/Simulink.

A need arose for a cross-domain modelling environment that supported expressing model equations more naturally, with high level support for complex pre-prepared model components or objects. This so-called physical modelling first appeared commercially as Dymola in 1992 (Åström, Elmqvist, & Mattsson, 1998) and soon the need for a standardised modelling language was fulfilled by the advent of Modelica. The Modelica language is an open standard under the stewardship of the Modelica Association (Modelica Association, 2022). An open-source collection of tools and libraries is maintained by the Open Source Modelica Consortium (Open Source Modelica Consortium, 2020). Their modelling tool set, OpenModelica, is used throughout this document.

It is the author's contention that modelling and simulation in the majority of contexts is best done with Modelica, its standard library, and the OpenModelica software tool. Other closed-source, proprietary modelling tools are best used when the cost of one of their specialised libraries or toolboxes can be justified. For this document, the research question is: can OpenModelica be used to model, simulate, and dynamically analyse an engineering business area such as building insulation to provide insight and improvement opportunities? With availability and cost issues of both energy and building materials being very topical, I hope this thesis affirms that it can.

Figure 1. Simulating and dynamically analysing an engineering business area

Model: OpenModelica   Analyse/Optimise: Python   Visualise and Report

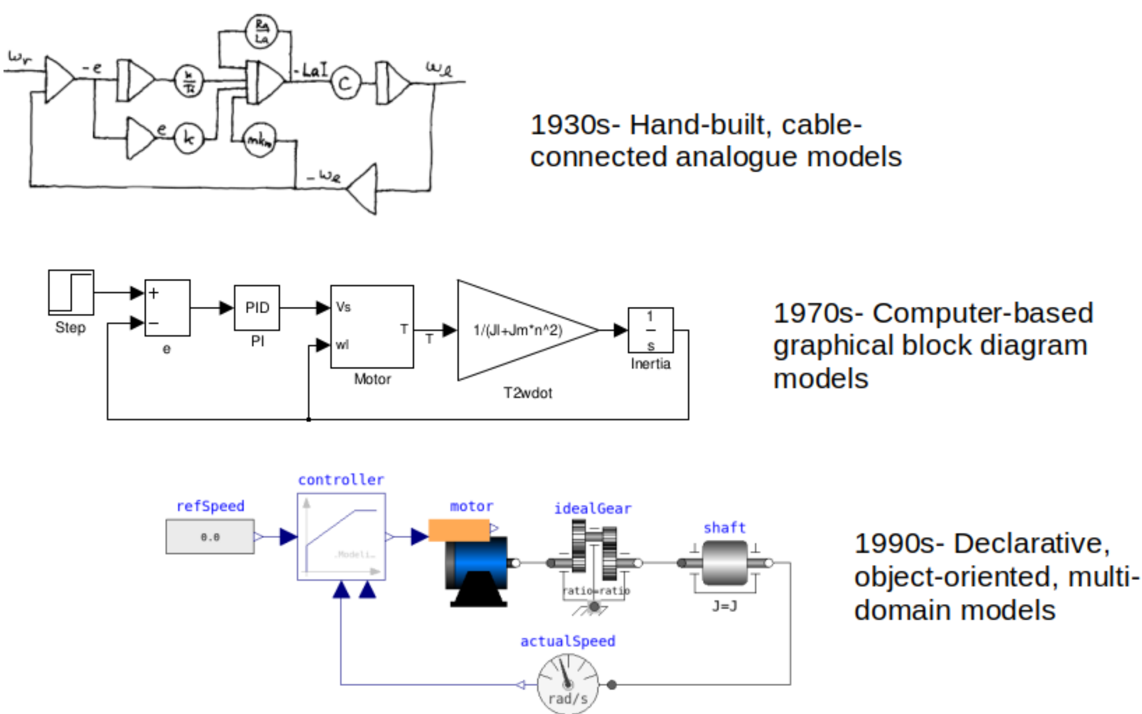
The techniques and tool chain defined in this document can be applied to any Modelica model (not just building systems). Therefore, model systems containing mechanical, electronic, electrical, thermal, hydraulic, electric power, control, or process-oriented subcomponents, for example can all benefit from automated simulation and optimisation. An overview of the simulation and analysis process is shown in *Figure 1*.

In the following chapters, first some pertinent simulation history and background are shared. Then the Modelica concept and language are explained using straightforward examples. A more complex model of building insulation is then introduced and simulated. Finally, the technique of dynamically simulating multiple scenarios of the same base model programmatically is explained and demonstrated.

2 Models and Simulation

The essential task of model simulators is to find numerical solutions to equations that represent the physical system being modelled in accordance with the basic rules, laws, and formulas of physics. Simulation techniques, styles and capabilities have evolved over the years. *Figure 2* shows a progression of modelling which starts with hardware analogues in the form of gears, cams, variables represented as voltage levels, cable interconnections, and so on. This is followed by a computer-based version of the analogue model expressed using graphical block diagram modelling software, in which the differential equations that correspond to the model must be represented using blocks for integration, addition, multiplication, and so on. This often requires that the underlying equations that describe the behaviour of the components of the physical model be reformulated manually to suit the modelling tool (Egel, 2009).

Figure 2. History of simulation, based on (Åström, Elmqvist, & Mattsson, 1998)



Finally, a Modelica version of the model is shown. This clearly shows the objects that participate in the model, the connections between them, all with clearly labelled parameters

and identifiers. As we will see in later chapters, models based on the open Modelica standard library can be inspected fully – that is they are open source, whereas in closed-source software such as Simulink, the models and toolboxes (model libraries) are proprietary with little or no underlying model detail available.

2.1 Causal and Acausal

In traditional modelling systems, inputs to a system affect the outputs, such that changes to inputs cause outputs to change, and that the causality direction is fixed. Graphical block diagrams, used in modelling tools such as Simulink, are causal models and require the model to be expressed as instructions that the computer can process step-by-step. The equation system being modelled may first have to be solved by extracting the derivatives before the model can be defined for simulation. As alluded to in the previous section, creating causal models takes extra translational effort. It also results in a rather rigid model that can execute well for a narrowly defined scenario but cannot be tweaked or altered easily or intuitively. The model diagram also bears little resemblance to the physical reality (Egel, 2009).

Many engineers would prefer to design and interact with models in a way that is as close to the physical system as possible (Egel, 2009). In most physical systems, what is an output and what is an input is not a characteristic of the actual system but is decided by the model user and what they consider to be the variables or parts of the systems that are interesting to study. Take for example the final model shown in *Figure 2*. The angular displacement of the shaft is affected by forces from the environment, but equally, forces between the environment and the shaft are affected by the angular displacement of the shaft (Fritzson, 2015, p. 5).

Acausal modelling is declarative in that the model comprises equations that represent equivalence rather than assignment statements, so, as in mathematics, a=b is equivalent to b=a. The equations of the model can be expressed as they are written in a physics or mathematics textbook. The declarative statements and equations are grouped as components of the model, or objects, which are connected together. The order in which the equations are expressed does not matter, only the solvability of the compile-time system of

equations is paramount. The data flow direction of the equations is not specified until compile-time when it is stated which variables of the simulated system are to be outputs and which are to be external inputs (Fritzson, 2015, p. 36). Acausal physical modelling objects are thus easily re-usable in different contexts and represent real objects quite well, for example, a motor, or a capacitor, and so on. Practical examples are described later in this document.

Analogue or block diagram models demand explicit state models as the blocks have a one-directional data flow from inputs to outputs and thus represent causal equations since there is a specified data-flow direction (Fritzson, 2015, p. 72). Modelling control systems with block diagrams can be useful, but for other modelling needs such as electrical circuits, they are not useful (Cellier & Kofman, Continuous System Simulation, 2006, p. 7).

Note that Modelica models are generally acausal, but if causality is desired, the prefixes ‘input’ or ‘output’ can be prefixed to variables to indicate data flow direction in a model. On the other hand, note too that a traditional causal modelling application such as MATLAB/Simulink also has its own acausal modelling environment called Simscape (Egel, 2009).

Having to design and input models into computers causally is somewhat artificial and arises from the historical need to match the model to the then capability of the modelling software and hardware. However, models and equations in nature are acausal, and this is the most expressive, re-usable, and straightforward way to express them. With modern hardware and software techniques, the modelling software now has the capability to parse acausal equations (for example as differential algebraic equations) and translate them automatically into simulation code (often as ordinary differential equations) thus avoiding this previously onerous and error-prone manual task. (Zupancic, Karba, Atanasijevic-Kunc, & Music, 2008)

2.2 OpenModelica Model Translation

For continuous-time problems, in which the physical properties being modelled vary continuously over time, the Modelica model is translated into a flat collection of equations of various types that the solver will solve during simulation:

1. Ordinary differential equations (ODE) of the form:

$$y' = f(t, y) \quad (1)$$
$$y(t_0) = y_0$$

(Petzold, 1982)

   Where *y* is a model variable, *f* is a function that acts on *y* at a given time *t*, and *y′* ('y prime') represents the transformation of *y*; in modelling, this is usually the next iteration of *y* after a time step, i.e., *y′* is *y* at $t + \Delta t$.

2. Differential algebraic equation (DAE) of the form:

$$f(t, y, y') = 0 \quad (2)$$
$$y(t_0) = y_0$$
$$y'(t_0) = y_0'$$

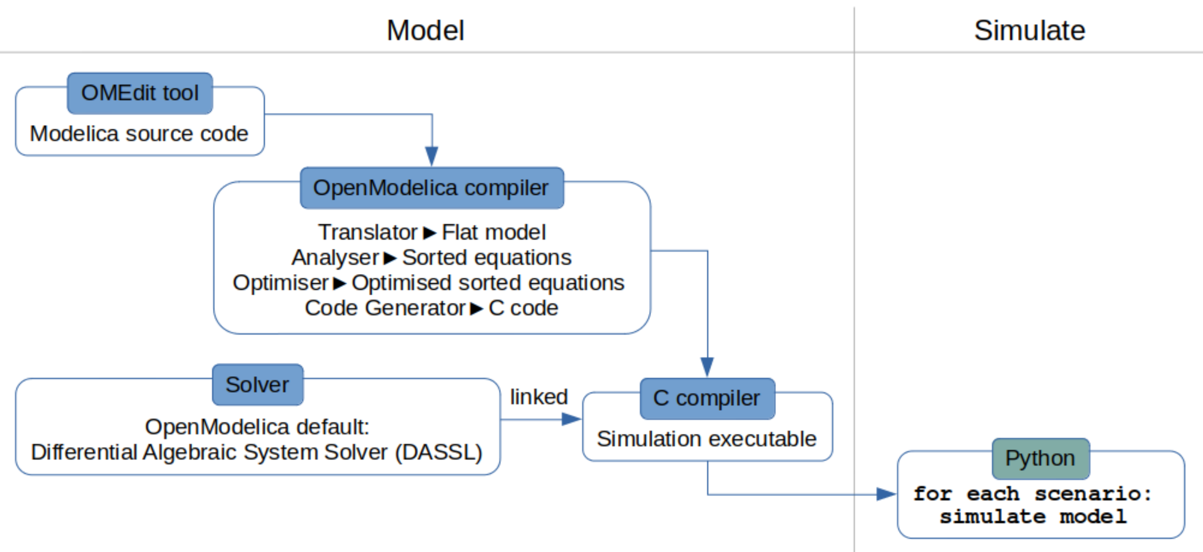
(Petzold, 1982)

3. Hybrid differential algebraic equations are also produced for mixed continuous-discrete problems.

The translation and execution of a model in the OpenModelica environment is illustrated in *Figure 3*. The acausal, object-oriented Modelica source code is first flattened into a series of function definitions, constants, variables, and equations. The data-flow dependencies of the equations are used as sorting criteria for order of execution, and then any equation optimisation methods such as algebraic simplification algorithms, or transformation of DAEs

to ODEs are applied. Now the equations are expressed as assignment statements and output as C code. A numeric equation solver is linked into the C file to solve the now smaller final system of equations. A C compiler then creates executable files that are used to simulate the model as needed, for example, in a loop through various scenarios of the model controlled by a Python script as investigated and demonstrated later in this document.

Figure 3. OpenModelica translation and execution steps



The solver defines the numerical integration method used during simulation on the equations generated by the OpenModelica compiler. Several solver choices are available for particular model types (OpenModelica, 2022); however, the default Differential Algebraic System Solver (DASSL) (Petzold, 1982) has the best stability and reliability for differential algebraic equation (DAE) systems, which are produced after compiling typical acausal models such as electrical or mechanical systems (Fritzson, 2015, p. 1002).

An example of how a physical component of a system can be modelled in Modelica is shown in *Figure 4*. The equation for current flow through a capacitor as typically found in a textbook (Fritzson, 2015, p. 746) is shown, along with the pictorial representation of a capacitor used in Modelica model diagrams. Also presented is the declarative textual version of the model which contains the actual equations and definitions used when simulating the response of the capacitor in the model over the period of the simulation.

Figure 4. Modelling current flow in a capacitor

Reality	Textbook	Modelica Library Diagram	Modelica Object

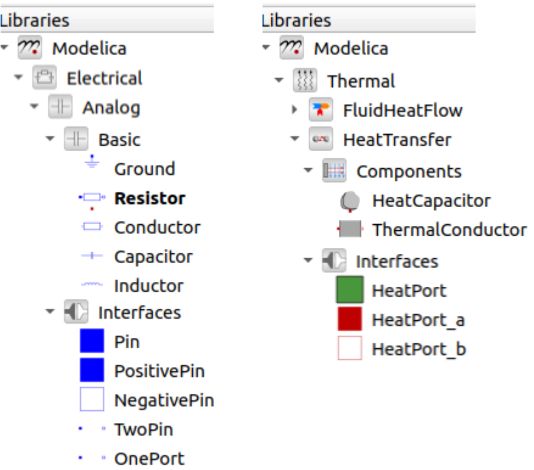
	$i(t)=C\frac{d}{dt}v(t)$	capacitor
		C=0.047 F
			class Modelica.Electrical.Analog.Basic.Capacitor
			"Ideal linear electrical capacitor"
			Real v(quantity = "ElectricPotential", unit = "V", start = 0.0) "Voltage drop of the two pins (= p.v - n.v)";
			Real p.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
			Real p.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
			Real n.v(quantity = "ElectricPotential", unit = "V") "Potential at the pin";
			Real n.i(quantity = "ElectricCurrent", unit = "A") "Current flowing into the pin";
			Real i(quantity = "ElectricCurrent", unit = "A") "Current flowing from pin p to pin n";
			parameter Real C(quantity = "Capacitance", unit = "F", min = 0.0, start = 1.0) "Capacitance";
			equation
			p.i = 0.0;
			n.i = 0.0;
			i = C * der(v);
			0.0 = p.i + n.i;
			i = p.i;
			v = p.v - n.v;
			end Modelica.Electrical.Analog.Basic.Capacitor;

Modelica is discussed in more depth, using practical examples, in the following sections.

2.3 Modelica

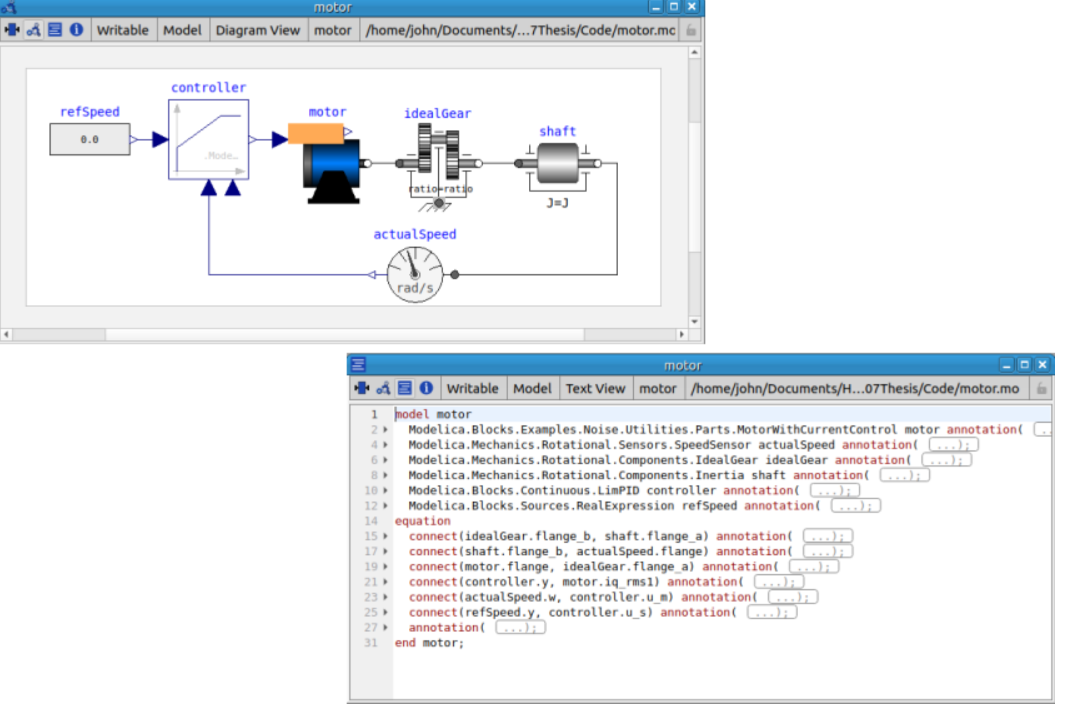
Modelica is object-oriented: mathematical models are declared in a structured manner defining any hierarchy or connections between model components, as well as common traits or functionality inherited by complex components from simpler ones. Example electrical and thermal component hierarchies are shown in *Figure 5*. In the electrical section of the library for example, the 'TwoPin' model inherits from the positive and negative pin models which in turn inherit basic characteristics from the 'Pin' model (this is further demonstrated later in sections 2.4.3 and 2.4.4).

Figure 5. Example electrical and thermal model component hierarchies



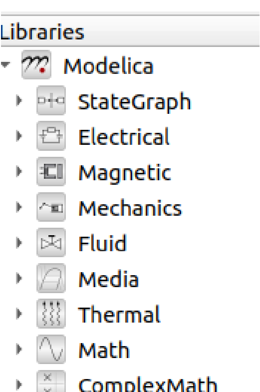
Modelica is high-level: models in Modelica can be defined graphically using familiar drag-and-drop methods. For example, the 'Resistor' model shown in *Figure 5* can be dragged from the libraries list and dropped onto the model canvas and then interconnected to other model components by clicking and dragging between connector pins. An alternative text-based view of the model can also be edited and is kept fully synchronised with the graphical view (see *Figure 6*).

Figure 6. Graphical and textual view of model in OpenModelica



Modelica is declarative: the dynamic properties of a model are declared as equations that define the system being modelled, i.e., there is no procedural algorithm defined as happens in imperative or procedural computer languages.

Figure 7. Domains in Modelica standard library

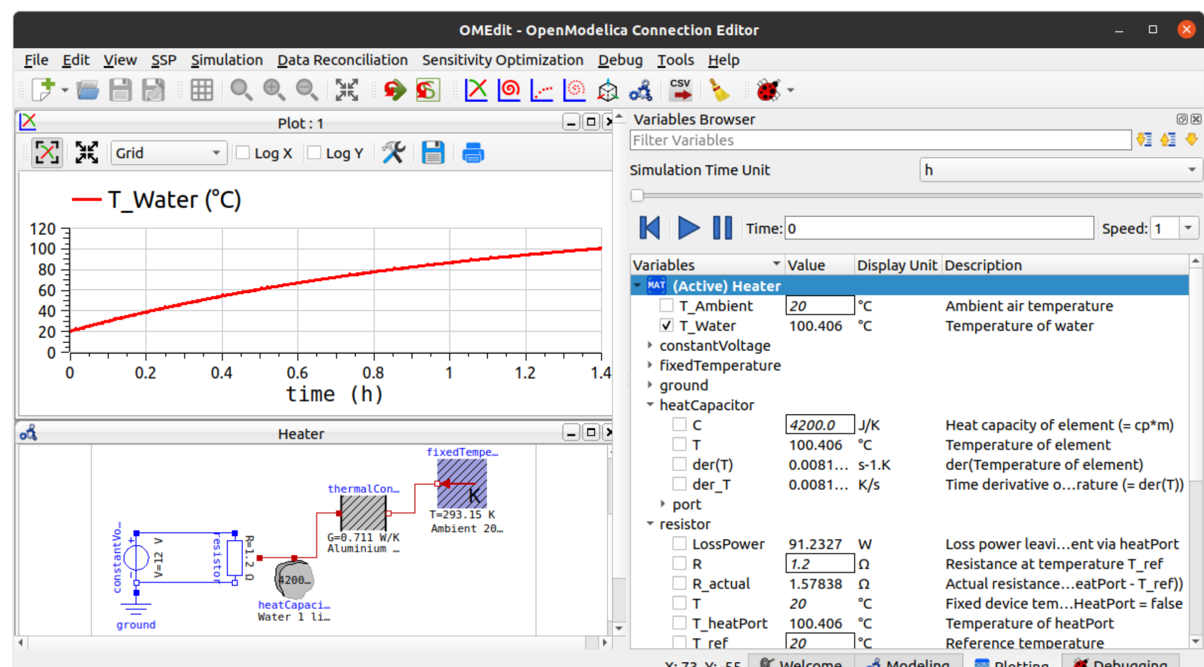


Modelica is multi-domain: *Figure 7* shows the main domains covered by the Modelica standard library. Model systems containing mechanical, electrical, electronic, hydraulic, thermal, control, electric power, or process-oriented subcomponents, for example can all benefit from automated simulation and optimisation. Further libraries are available, many of them open source, in areas such as building design, power system analysis, photovoltaics, vehicle dynamics, and so on (Modelica Association Libraries, 2022).

Modelica is the modelling language used in products such as Dymola (Dassault Systèmes, 2022), MapleSim (Maplesoft, 2022), and Wolfram System Modeler (Wolfram Research, 2022).

OpenModelica is an integrated interactive modelling, compilation, and simulation environment along with an open-source, industrial-strength implementation of the Modelica language. It is used to develop and test Modelica models using an application with a human-centric graphical user interface called OpenModelica Connection Editor ('OMEdit'). An example OMEdit session is shown in *Figure 8*.

Figure 8. Using OMEdit to simulate a water heater interactively

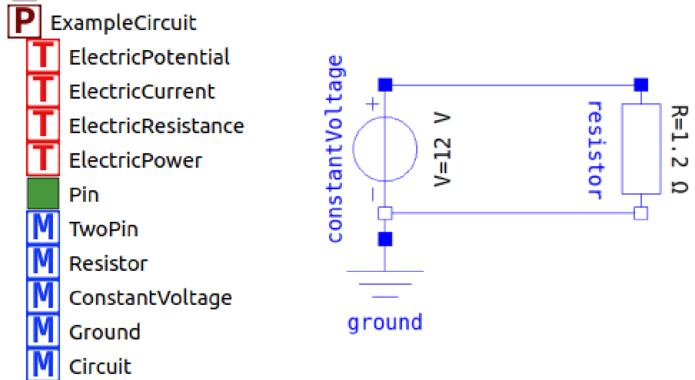


OpenModelica includes a compiler that takes a model expressed in the Modelica language and composes it into a target language such as C which can be further compiled to an executable program (Fritzson & et al., 2020, p. 246).

2.4 Introducing Modelling with Modelica

To illustrate the use of Modelica, a simple electrical circuit will be modelled using basic Modelica language features. This procedure is described in the following sections. A demonstration that begins in the electrical domain was selected to suit the context of electrical engineering; the example model is then extended into the thermal domain as an introduction to modelling heating and insulation of buildings. An overview of the initial circuit is shown in *Figure 9*.

Figure 9. Graphical representation of the simple circuit and its constituent parts



In more complex examples, the Modelica standard library (Modelica Association, 2020) would be used for many of the common parts of the model, for example ready-defined electrical types (Modelica.Units.SI.*) or electrical components (Modelica.Electrical.Analog.Basic.*). More on this in section 2.5.

2.4.1 Types

A model expressed in Modelica starts with the definition of the data types and units of the basic variables in the model. In the case of the simple electrical circuit, voltage, current, resistance and power were defined as follows:

```
type ElectricPotential = Real (unit="V");
type ElectricCurrent = Real (unit="A");
type ElectricResistance = Real (unit="Ohm");
type ElectricPower = Real (unit="W");
```

The data type 'Real' represents a floating-point number.

2.4.2 Connectors

A connector represents a point in the model where a physical connection occurs. This could be a wire, or heat or fluid flow, for example. In our simple circuit, we define a 'Pin' to allow interconnection between electrical components, as follows:

```
connector Pin "Pin of an electrical component"
  ElectricPotential v "Potential at the pin";
  flow ElectricCurrent i "Current flowing into the pin";
end Pin;
```

As can be seen above, there is an electrical potential (‘v’ or voltage) at the pin, as well as an electrical current (‘i’ or current) flowing through it (indicated by the Modelica ‘flow’ keyword). For flow variables, the Modelica compiler automatically generates equations to sum all connected elements of the given type to zero. In this case, the given type is the electric currents in all pins in the circuit, thus Kirchoff’s current law (Bird, 2014, p. 166) is handily enforced automatically.

This document describes Modelica-based models in the electrical and thermal domains. *Table 1* compares a selection of connector variables across some other engineering areas.

Table 1. A selection of connector variables, based on (Moraleda & Villalaba, 2018, p. 20)

Domain	**Across variable (potential)**	**Through variable (flow)**	**Energy Carrier**
Electrical	Voltage (V)	Current (A)	Charge (C)
Thermal	Temperature (K)	Heat flow rate (W)	Heat (J)
Mechanical translation	Position (m)	Force (N)	Linear momentum (kg·m/s)
Mechanical rotation	Angle (radian)	Torque (N·m)	Angular momentum (kg·m^2/s)

2.4.3 Partial Models

In Modelica, models form a hierarchy: for example, a circuit model may contain models of components that may in turn contain partial models that represent a part of the component. Our example circuit contains components with two pins (positive and negative), so it makes sense to have a common partial model to represent these two pins, as follows:

```
partial model TwoPin "Common parts of two pin electrical components"
  Pin p "Positive electrical pin";
```

```
  Pin n "Negative electrical pin";
  ElectricPotential v = p.v-n.v "Difference in electric potential between pins";
  ElectricCurrent i = p.i;
equation
  p.i + n.i = 0 "Conservation of charge";
end TwoPin;
```

Both pins inherit the properties of the earlier defined ‘Pin’ (See 2.4.2), so the positive pin, named ‘p’ has a voltage ‘v’ referred to as variable ‘p.v’ (using a dot between identifiers). The ‘TwoPin’ model also has its own value ‘v’ which refers to the potential difference (p.d.) between the two pins, in volts.

As well as parameters, models contain equations that define their behavior mathematically. This partial model simply affirms conservation of charge through components that have two pins.

2.4.4 Models of the Circuit Components

We now define the main components of the circuit as individual models. A resistor is a two-pin component, so its model extends the partial ‘TwoPin’ model defined earlier (see 2.4.3), as follows:

```
model Resistor "Ideal linear electrical resistor"
  parameter ElectricResistance R "Nominal resistance";
  extends TwoPin;
  ElectricPower LossPower "Loss power leaving component as heat";
equation
  v = p.i*R "Ohm's law";
  LossPower = v*p.i;
end Resistor;
```

The resistor model defines its voltage using the current at the positive pin multiplied by its nominal resistance. The power generated as heat (‘LossPower’) is also defined.

The remaining components are straightforward as they simply define the voltage of the circuit’s power supply (defaults to one volt) and ground (zero volts), as follows:

```
model ConstantVoltage "Source for constant voltage"
  parameter ElectricPotential V(start=1) "Value of constant voltage";
  extends TwoPin;
equation
```

```
  v = V;
end ConstantVoltage;

model Ground "Ground node"
  Pin p "Positive electrical pin";
equation
  p.v = 0;
end Ground
```

Note that the ground model only has one pin, so does not extend any other models.

2.4.5 Model of the Circuit

To finalise the model of the whole circuit, the start voltage and resistance are defined, and the components are interconnected. The whole circuit is defined as another model, but now the equation section defines the connections between the components, as follows:

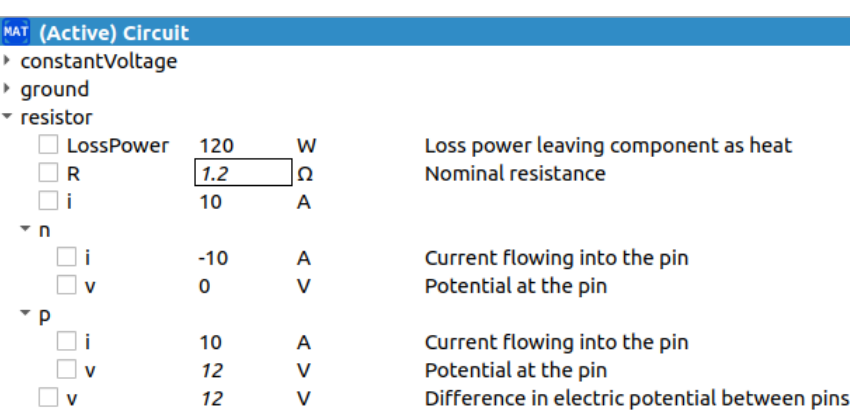
```
model Circuit "An example circuit"
  ConstantVoltage constantVoltage(V = 12);
  Resistor resistor(R = 1.2);
  Ground;
equation
    connect(constantVoltage.p, resistor.p);
    connect(constantVoltage.n, resistor.n);
    connect(constantVoltage.n, ground.p);
end Circuit;
```

All parts of the model (types, connectors, models) can be combined into a self-contained package as shown in appendix 1.

2.4.6 Simulate the Circuit Model

The circuit model as defined above was then simulated in the OpenModelica Connection Editor application. The results are shown in *Figure 10*. The power and current of the resistor have been calculated as expected but note that the pins also indicate voltage and direction of current.

Figure 10. Results of simulating the simple circuit model



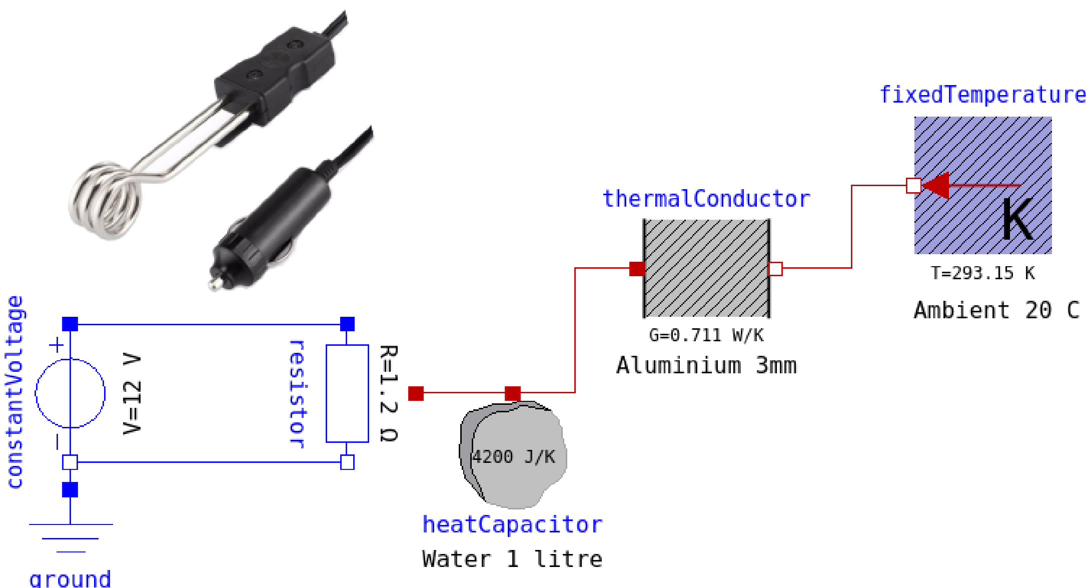
2.5 Further Modelling with Modelica

The simple circuit described in the previous section can be expanded upon to demonstrate the modelling of thermal energy flows in the same model system as electrical energy flows. This extended model can also be used to introduce heat capacitance and thermal conductivity which are the basis of the application of the Modelica building energy models utilised in later chapters.

2.5.1 Heater Model

As shown in *Figure 11*, the simple circuit can be viewed as a water heater that is used in a car or camping wagon to warm liquids using a 12v power source. The heater is placed into an aluminium container containing one litre of water (a ‘heatCapacitor’). The container wall is 3mm thick (a ‘thermalConductor’). The surrounding air temperature is 20°C (‘ambient’).

Figure 11. Graphical representation of the heater model and its constituent parts



To provide the heat source part of the heater, the 'Resistor' model can be extended to include a heat port, which is a Modelica connector:

```
connector HeatPort "Thermal port for 1-dim. heat transfer"
  Modelica.Units.SI.Temperature T "Port temperature";
  flow Modelica.Units.SI.HeatFlowRate Q_flow
    "Heat flow rate (positive if flowing from outside into the component)";
end HeatPort;
```

The 'HeatPort' of the resistor model then defines heat flow as negative 'LossPower' (note the negative sign):

```
final Q_flow=-LossPower
```

This models the heating effect of the electric current through the resistor flowing into the connected heat capacitor model component that represents one litre of water.

Many conductors increase in resistance as the temperature rises (Bird, 2014, p. 21), so an equation for the actual resistance value (calculated from the temperature coefficient of resistance) is added to the resistor model:

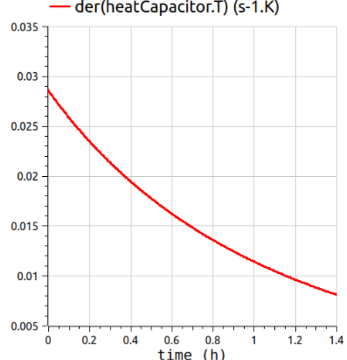
```
R_actual = R*(1 + alpha*(T_heatPort - T_ref));
```

Where 'R' is the nominal resistance, 'T_heatPort' is the temperature at the heat port, T_ref is 20 C, and alpha is 1 divided by 255 (which is approximately 0.004, the temperature coefficient of resistance of copper).

2.5.2 Heat Capacitor Model

In the model, one litre of water acts as a heat store or 'heatCapacitor'. The quantity of heat flowing into the heat capacitor is proportional to both the temperature change and the mass, as well as the specific heat of the water (~4200J/kg K). The temperature change or delta (Δ) varies over the simulation, as illustrated in *Figure 12*, so the Modelica 'der' or derivative function is used to calculate the time derivative of temperature. The textbook formula for the "heat required for temperature change ΔT of mass m" (Young & Freedman, 2012, p. 563) is:

Figure 12. Time derivative of temperature



$$Q = m\, c_p\, \Delta T \tag{3}$$

Where 'm' is mass and 'c_p' is specific heat. The equation relates thermal mass to thermal energy. *Figure 13* shows how this is rendered and encoded in Modelica.

Figure 13. Modelling heat flow in a heat capacitor

Reality	Textbook	Modelica Library Diagram	Modelica Object

For example, 1 kg water	$Q = mc_p \Delta T$	heatCapacitor
		4200 J/K
			class
			Modelica.Thermal.HeatTransfer.Components.HeatCapacitor
			"Lumped thermal element storing heat"
			parameter Real C(quantity = "HeatCapacity", unit = "J/K") "Heat capacity of element (= m*cp)";
			Real T(quantity = "ThermodynamicTemperature", unit = "K", displayUnit = "degC", min = 0.0, start = 293.15, nominal = 300.0) "Temperature of element";
			Real der_T(quantity = "TemperatureSlope", unit = "K/s", start = 0.0) "Time derivative of temperature (= der(T))";
			Real port.T(quantity = "ThermodynamicTemperature", unit = "K", displayUnit = "degC", min = 0.0, start = 288.15, nominal = 300.0) "Port temperature";
			Real port.Q_flow(quantity = "Power", unit = "W") "Heat flow rate (positive if flowing from outside into the component)";
			equation
			port.Q_flow = 0.0;
			T = port.T;
			der_T = der(T);
			C * der(T) = port.Q_flow;
			end
			Modelica.Thermal.HeatTransfer.Components.HeatCapacitor;

This is expressed in the equation section of the 'HeatCapacitor' model, where 'C' is a pre-calculated parameter representing mass (m) multiplied by specific heat (c_p):

```
model HeatCapacitor "Lumped thermal element storing heat"
  parameter Modelica.Units.SI.HeatCapacity C
    "Heat capacity of element (=m*cp)";
  Modelica.Units.SI.Temperature T(start=293.15, displayUnit="degC")
    "Temperature of element";
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a port;
equation
  T = port.T;
  C*der(T) = port.Q_flow;
end HeatCapacitor;
```

2.5.3 Thermal Conductor Model

In the model of the wall of the water container, G is the constant thermal conductance of a material. For example, aluminium has thermal conductivity of 237 $W \cdot m^{-1} \cdot K^{-1}$ (Lide, 2004, p. 12_219), so 3mm of aluminium has a value of G equal to 237 $W \cdot m^{-1} \cdot K^{-1}$ multiplied by 0.003m. The variable 'dT' is the difference in temperature between the two heat ports representing the inner surface and the outer surface of the container. The model declaration contains the equation for heat flow through the container wall, 'Q_flow':

```
model ThermalConductor
  "Lumped thermal element transporting heat without storing it"
  extends Modelica.Thermal.HeatTransfer.Interfaces.Element1D;
  parameter Modelica.Units.SI.ThermalConductance G
    "Constant thermal conductance of material";
equation
  Q_flow = G*dT;
end ThermalConductor;
```

2.5.4 Simulation of a Water Heater

The heat capacity of the one litre of water in the heat capacitor is defined as 4200 J/K and the thermal conductance of the aluminium container is defined as 0.711 W/K. The ambient temperature is set at 20C, an output variable is defined for the water temperature, and the model components are connected together as follows:

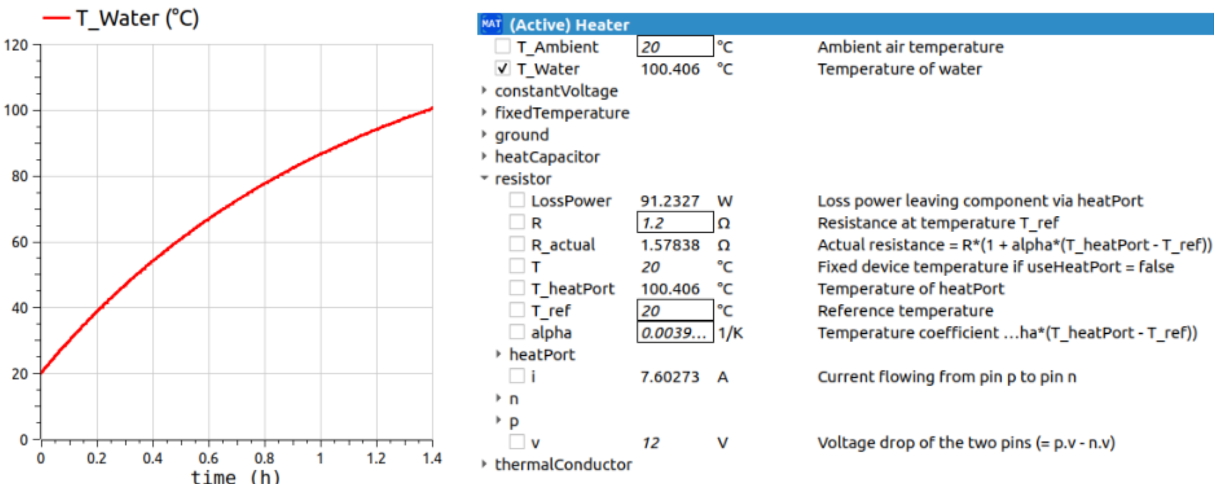
```
model Heater "Water heater"
  parameter Modelica.Units.SI.Temperature T_Ambient(displayUnit = "degC") = 293.15
            "Ambient air temperature";
  output Modelica.Units.SI.Temperature T_Water(displayUnit = "degC") =
            resistor.T_heatPort "Temperature of water";
  Modelica.Electrical.Analog.Basic.Ground ground;
  Modelica.Electrical.Analog.Sources.ConstantVoltage constantVoltage(V = 12);
  Modelica.Thermal.HeatTransfer.Components.HeatCapacitor heatCapacitor(C = 4200,
            T(fixed = true, start = T_Ambient));
  Modelica.Electrical.Analog.Basic.Resistor resistor(R = 1.2, T_ref = 293.15,
            alpha = 1 / 255, useHeatPort = true);
  Modelica.Thermal.HeatTransfer.Sources.FixedTemperature
            fixedTemperature(T(displayUnit = "K") = 293.15);
  Modelica.Thermal.HeatTransfer.Components.ThermalConductor
            thermalConductor(G = 0.711);
equation
  connect(constantVoltage.n, resistor.n);
  connect(constantVoltage.p, resistor.p);
  connect(constantVoltage.n, ground.p);
  connect(resistor.heatPort, heatCapacitor.port);
  connect(thermalConductor.port_a, heatCapacitor.port);
  connect(thermalConductor.port_b, fixedTemperature.port);
end Heater;
```

The above Modelica code describes the whole heater model. Compare the use of Modelica language in the model above with the simple circuit model listed in appendix 1. The heater model uses types, connectors, and partial models from the Modelica standard library rather than defining everything from scratch. Using the Modelica standard library makes model

descriptions much more compact and the use of standard, ready-made Modelica objects reduces errors.

The heater model is then simulated in the OpenModelica Connection Editor application. The simulation start time is set to 0 seconds, the stop time to 1.4 hours (5040s). The results are shown in *Figure 14*. It seems that a 12v water heater takes a very long time to boil a litre of water!

Figure 14. Results of simulating the heater model



Even though only the water temperature is plotted, many other variables are available in the simulation result. For smaller models, these can be viewed and plotted in the OMEdit interface, but as the model gets more complex and the number of variables becomes larger, automating the simulation, plotting and analysis steps becomes more desirable. The next section shows how this could be done with the heater model. Note that the heater model has only 32 variables, but the simple building model used later in this report has over 6000 variables.

2.6 Automated Modelling Toolchain

In the previous section, model design, creation, simulation and editing all took place in the OpenModelica Connection Editor application 'OMEdit'. These tasks can, however, be split into two stages: design and edit the model first in OMEdit as before but simulate the model

and analyse the results in a separate more automated environment. This allows the model to be tested, refined, and debugged in the first stage, often by a model domain area expert or small team. The second stage can then utilise the model in an automated or programmed environment that provides different sets of model inputs and interprets and presents the results of simulation in various ways. In this document, the programming language used in this automated stage is Python.

A Modelica model is stored in a plain text file, which maximises version control, compression, and file interchange possibilities. The model file can be loaded and simulated by other systems. In the following example, the heater model will be simulated from a Python 3 environment. A full listing of the Python program can be found in appendix 3; only the key commands are shown in the following sections.

2.6.1 Compiling and Simulating the Model

The model must be loaded and compiled once at the start of a session. Once compiled it can be simulated as many times as needed with different parameters sets, timings, and other settings. The Python code uses the 'OMPython' code library, which is part of OpenModelica. First, a connection to the Modelica system is started and the model 'Heater.mo' is compiled:

```
from OMPython import OMCSessionZMQ
from OMPython import ModelicaSystem
omc = OMCSessionZMQ()

# %% Compile model
model = ModelicaSystem("../Heater.mo", "Heater")
```

Next, the model is simulated with its default settings and start/stop times (which are stored in the model file):

```
model.simulate()
```

And then plotted:

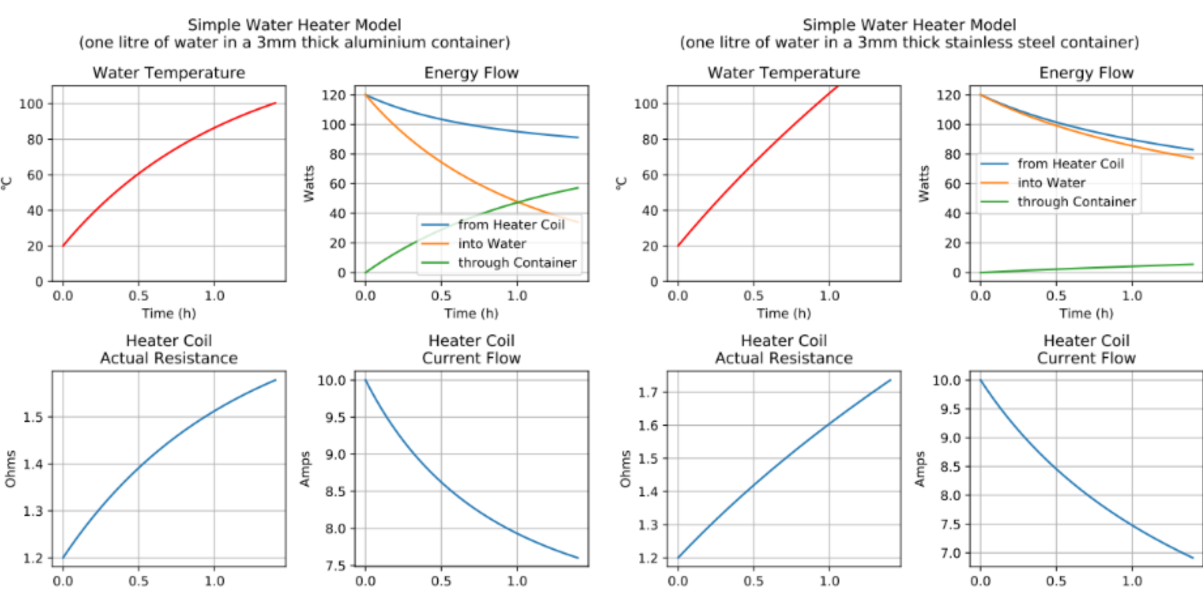
```
doPlot(scenario="(one litre of water in a 3mm thick aluminium container)")
```

A second simulation scenario can easily be setup and run by changing a model parameter:

```
# Stainless steel thermal conductivity = 16.3 W·m-1·K-1
model.setParameters([f"thermalConductor.G={16.3*0.003}"])
model.simulate()
doPlot(scenario="(one litre of water in a 3mm thick stainless steel
container)",
       pdfName="../figureHeater2.pdf")
```

The last two simulations result in the plots shown in *Figure 15*.

Figure 15. Comparing a couple of simulation scenarios



The final simulation changes another parameter and also shortens the timeframe of the simulation – now a 300ml cup of water can be boiled in around 15 minutes (see *Figure 16*):

```
# Specific heat capacity of water is around 4200 J/K
model.setParameters([f"heatCapacitor.C={4200*0.3}"])
# Set a shorter simulation period (15mins=900s)
model.setSimulationOptions(["stopTime=900"])
model.simulate()
doPlot(scenario="(300ml of water in a 3mm thick stainless steel
container)",
       pdfName="../figureHeater3.pdf", timeScale="min")
```

Figure 16. A faster simulation scenario

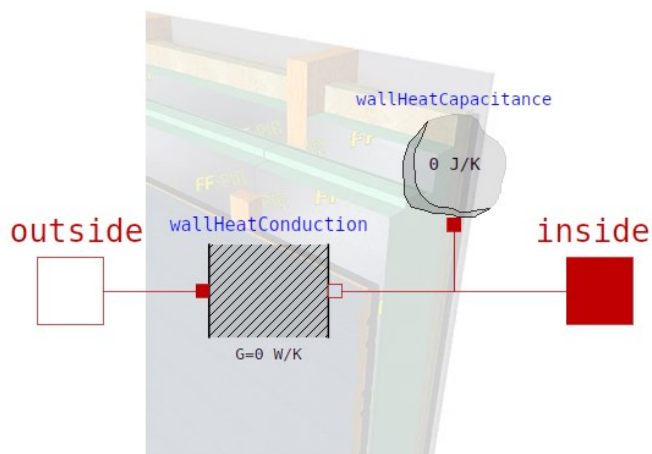
2.7 Modelling Building Systems

Models of the heating and cooling of buildings are used to estimate energy consumption, build energy-efficient buildings, and reduce costs. A key part of these models is the building envelope and its insulation properties. The building envelope consists of the roof, floor, walls, windows, and doors.

2.7.1 Walls

Figure 17 shows a simple Modelica model of a wall based on the heat capacitance and thermal conductance models already described in sections 2.5.2 and 2.5.3, respectively.

Figure 17. Simple Modelica model of heat flow in a wall



In this model the standard thermal variable types from the SI Units part of the Modelica standard library were used:

```
type SpecificHeatCapacity = Real (final quantity="SpecificHeatCapacity", final
unit="J/(kg.K)");
type ThermalConductivity = Real (final quantity="ThermalConductivity", final
unit="W/(m.K)");
type Density = Real (final quantity="Density", final unit="kg/m3", displayUnit="g/cm3",
min=0.0);
```

From these basic types, a general model for building insulation can be defined:

```
record MaterialGeneral
  parameter String name="";
end MaterialGeneral;

record MaterialThermalGeneral
  extends MaterialGeneral;
  parameter Modelica.Units.SI.ThermalConductivity lambda "Heat conductivity";
  parameter Modelica.Units.SI.SpecificHeatCapacity c "Specific heat capacity";
  parameter Modelica.Units.SI.Density rho "Density";
end MaterialThermalGeneral;
```

For the building in this model, the performance of Finnfoam PIR (polyisocyanurate) insulation board was studied. This requires that an instance of an insulation material called PIR insulation board be defined:

```
record PIRInsulation =  MaterialThermalGeneral(
  name="PIR (polyisocyanurate) insulation board",
  lambda=0.022,
```

```
  c=1500,
  rho=33)
```

The thermal conductivity, specific heat capacity and density values were sourced from the manufacturer's datasheet (FinnFoam_PIR, 2022). As will be seen in later sections, a wall is typically made up of multiple layers, each of which has its own thermal characteristics.

2.7.2 Windows

In a similar manner to walls, a basic 'TransparentConstruction' template can be defined containing the thermal and other properties of a window:

- Single, double, or triple glazed ('nPanes')
- The thickness of the glass (3mm=0.003m)
- Heat transfer performance for the glass part and the frame part of the whole window unit ('UValGla' and 'UValFra')
- The g-value, which is calculated as the total solar heat gain divided by the incident solar radiation, or the amount of heat from the sun that passes the window (Aguilar-Santana, Jarimi, Velasco-Carrasco, & Riffat, 2019). If the g-value is close to 1.0, the window transmits almost all of the solar radiation that hits its surface, so windows with high g-values transmit more heat energy from the sun into the building. A g-value closer to 0.0 means that the window transmits almost none of the solar radiation that hits its surface. Windows with low g-values help control unwanted solar heat gain and can save on cooling costs in certain locations or climates. The g-value is the European measure of solar energy transmittance of windows.

```
record TransparentConstruction
  parameter Integer nPanes(min=1)=1
    "Number of panes of the construction";
  parameter Modelica.Units.SI.Length thickness[nPanes]={0.003}
    "Thickness of each pane";
  parameter Modelica.Units.SI.CoefficientOfHeatTransfer UValGla
    "U-value of the glass construction";
  parameter Modelica.Units.SI.CoefficientOfHeatTransfer UValFra
    "U-value of the frame";
  parameter Real g(unit = "1")
    "g-value of the transparent construction";
  parameter Real b0(unit = "1")
    "Coefficient for radiation transmision curve";
```

To define a modern triple glazed window, an instance of the above-defined 'TransparentConstruction' was created using values from the window manufacturer (Piklas, 2019):

```
record WindowFIN =
  BuildingSystems.Buildings.Data.Constructions.TransparentConstruction(
  final nPanes = 3,
  thickness = {0.003, 0.003, 0.003},
  UValGla = 0.98,
  UValFra = 0.98,
  g = 0.55,
  b0 = 0.7)
  "Piklas MSEA triple glazed window with UValGla = 0.98 W/(m2.K) and g = 0.55";
```

2.7.3 Building Systems Modelica Library

For the walls and windows specified above, the structure and materials were defined from basic elements. Another approach is to use a Modelica library dedicated to the dynamic simulation of the behaviour of energy in rooms and buildings, such as the BuildingSystems library from Universität der Künste Berlin (UdK Berlin, 2022). This library is open source and includes the ability to read weather data files, has built-in definitions for many building materials and structures, as well as definitions for devices, machinery, pipes, valves, pumps, and so on used in indoor climate provision (heating, ventilation, and air conditioning (HVAC) equipment).

In the following section, components, and equations from the BuildingSystems library were used to define the basic model. Customised components and climate data were then used where required to simulate Finnish conditions.

3 Optimising Building Insulation

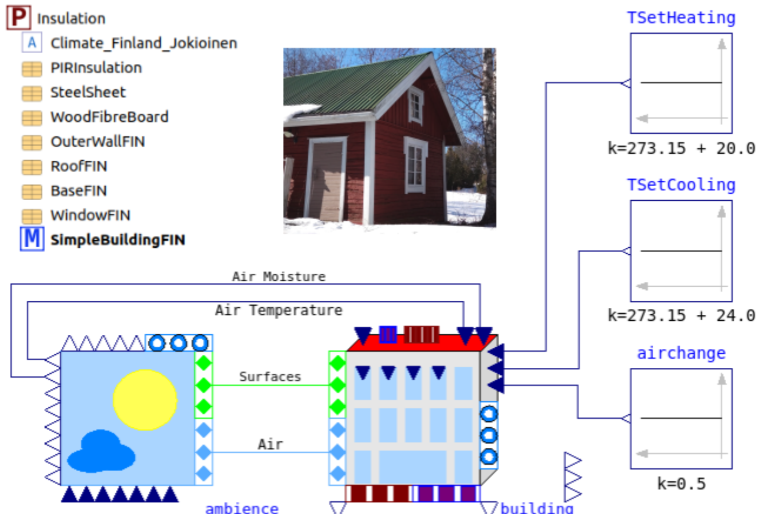
"The inside of a building can only be kept at a steady temperature above that outside by heating it at a rate which equals the rate at which it is losing energy. The loss occurs mainly by conduction through the walls, roof, floors, and windows." (Duncan & Kennett, 2021)

Earlier in this document, simple models in the electrical and thermal domains were deigned and simulated step-by-step. In this section of the document, a much more complex model will be specified and simulated. The topic of the model is a Finnish one-room cabin as may be found in some summer cottage locations or garden allotments.

3.1 Model

An overview of the model is shown in *Figure 18*. The model was part of a Modelica package named 'Insulation' that contained the climate data source as well as definitions for insulation materials and building structures that contain insulation. The model consisted of two main parts: the 'ambience' or local climate and the 'building' itself. Modelling of the interaction between these parts was through surface and air connections, with climate data also shared. Setpoints for the interior environment (heating temperature, cooling temperature and air changes per hour) were provided as inputs to the building model.

Figure 18. Overview of the simple building model



Pertinent parts of the model are described in the following paragraphs. Details of Modelica usage, the physics behind the model, and the attributes of PIR insulation can be found in section 2.7 of this document.

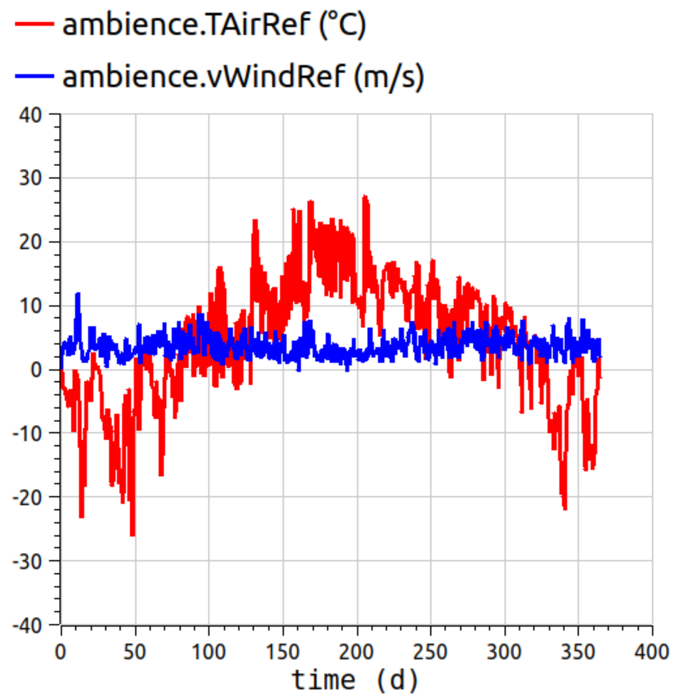
3.1.1 Ambient Climate

Climate data for the building's local environment was read in from a file on disk. The data in the file was sourced from the Finnish Meteorological Institute (Finnish Meteorological Institute, 2022). The location of Ilmala in Jokioinen near Forssa in southern Finland was chosen as it was the nearest weather station that recorded all weather data, including solar irradiance:

```
block Climate_Finland_Jokioinen
"Weather data for a typical location in South Finland (near Forssa)"
  extends BuildingSystems.Climate.WeatherData.BaseClasses.WeatherDataFileASCII(
  info="Source: Finnish Meteorological Institute (Ilmatieteenlaitos)",
  filNam=Modelica.Utilities.Files.loadResource(
                        "../Data/Finland_Jokioinen_Ilmala-v2021.txt"),
  final tabNam="tab1",
  final timeFac = 1.0/3600.0,
  final deltaTime = 1800.0,
  final columns={
    5,  // Direct solar radiation (W/m2)
    6,  // Diffuse radiation (W/m2)
    3,  // Air temperature (degC)
    8,  // Wind speed (m/s)
    9,  // Wind direction (deg)
    4,  // Relative humidity (%)
    7}, // Cloud amount (1/8)
  final scaleFac = {1.0,1.0,1.0,1.0,1.0,0.01,1.0},
  final latitudeDeg = 60.81397,
  final longitudeDeg = 23.49825,
  final longitudeDeg_0 = 1.0);
end Climate_Finland_Jokioinen;
```

The climate in this part of Finland varies from -25°C to 28°C in a typical weather year, as shown in *Figure 19*.

Figure 19. Overview of climate data used for the simulations



3.1.2 Building Structures

The wall was multi-layered, based on a Finnfoam building diagram (Finnfoam_RK, 2020). It had an inner plasterboard layer of 13mm, then PIR insulation board 100mm thick, and finally an outer wooden board layer of 28mm. The plasterboard and wood thermal properties were pre-defined in the BuildingSystems Modelica library, and the PIR layer was as defined earlier (see section 2.7.1):

```
record OuterWallFIN =
  BuildingSystems.Buildings.Data.Constructions.OpaqueThermalConstruction(
  final nLayers = 3,
  thickness = {0.013, 0.1, 0.028},
  material =
{BuildingSystems.HAM.Data.MaterialProperties.Thermal.Masea.Plasterboard(),
            PIRInsulation(),
            BuildingSystems.HAM.Data.MaterialProperties.Thermal.Wood()})
  "Outer wall construction (based on Finnfoam construction diagram)";
```

The thermal properties of the windows are as defined in section 2.7.2.

3.1.3 Building

Once the insulation materials and building structures were defined, they could be included in the definition of the building itself. The building dimensions indicate the size of the internal conditioned space (i.e., heated space). As this was a simple model, only one zone is present with no internal walls or ceilings. There were windows on all sides except the North wall. The windows were 1.2m square. The building model extended an existing base model 'Building1Zone1DBox' from the Building Systems Modelica library:

```
model SimpleFinnishBuilding
  "A simple Finnish building"
  BuildingSystems.Buildings.BuildingTemplates.Building1Zone1DBox building(
    width=3.7,
    length=5.8,
    height=2.25,
    InteriorWalls=false,
    InteriorCeilings=true,
    redeclare OuterWallFIN constructionWall1, // West
    redeclare OuterWallFIN constructionWall2, // North
    redeclare OuterWallFIN constructionWall3, // East
    redeclare OuterWallFIN constructionWall4, // South
    redeclare RoofFIN constructionCeiling,
    redeclare BaseFIN constructionBottom,
    redeclare WindowFIN constructionWindow1, widthWindow1=1.2, heightWindow1=1.2,
    redeclare WindowFIN constructionWindow2, widthWindow2=0.0, heightWindow2=0.0,
    redeclare WindowFIN constructionWindow3, widthWindow3=1.2, heightWindow3=1.2,
    redeclare WindowFIN constructionWindow4, widthWindow4=1.2, heightWindow4=1.2,
    nZones=1);
```

The roof/ceiling ('RoofFIN') consisted of an inner wood panel ceiling (12mm), mineral wool (150mm), Finnfoam PIR board (100mm), and metal sheet outer roof (0.6mm). The base/floor was suspended (i.e., not laid directly on the ground) and had a wooden floor (32mm), mineral wool insulation (100mm), and a windproof wood fibre board (25mm) layer. Further details of both of these and more can be found in the listing of the full 'Insulation' Modelica package in appendix 2.

3.1.4 Intra-Model Connections

There were three major model parameters that defined the desired indoor climate of the building: the room temperature setpoint when heat is needed, the room temperature setpoint when cooling is needed, and the number of room air changes per hour:

```
Modelica.Blocks.Sources.Constant TSetHeating(k = 273.15 + 20.0);
Modelica.Blocks.Sources.Constant TSetCooling(k = 273.15 + 24.0);
Modelica.Blocks.Sources.Constant airchange(k = 0.5);
```

In the ‘equation’ section of the the full ‘Insulation’ Modelica package listing in appendix 2, the connections between the model components were defined, such as the thermal interactions of the surfaces and the air between the ambient climate and the building. Climate data such as temperature and humidity were also shared between model components:

```
equation
connect(ambience.toSurfacePorts, building.toAmbienceSurfacesPorts);
connect(ambience.toAirPorts, building.toAmbienceAirPorts);
connect(ambience.TAirRef, building.TAirAmb);
connect(ambience.xAir, building.xAirAmb);
```

The model could also store various simulation (“experiment”) default parameters. In this case, the simulation lasts for one year (3.1536×10^7 seconds) with results recorded every hour (i.e., at intervals of 3600 seconds):

```
annotation (
   experiment(StartTime=1, StopTime = 3.1536e+07, Tolerance = 0.0001,
                  Interval = 3600),
   uses(Modelica(version="4.0.0"), BuildingSystems(version="2.0.0-beta")));
```

The model was designed for the latest version 4.0.0 of the Modelica language and standard library, and it used a beta version 2 of the BuildingSystems Modelica library.

3.2 Execution of Automated Simulations

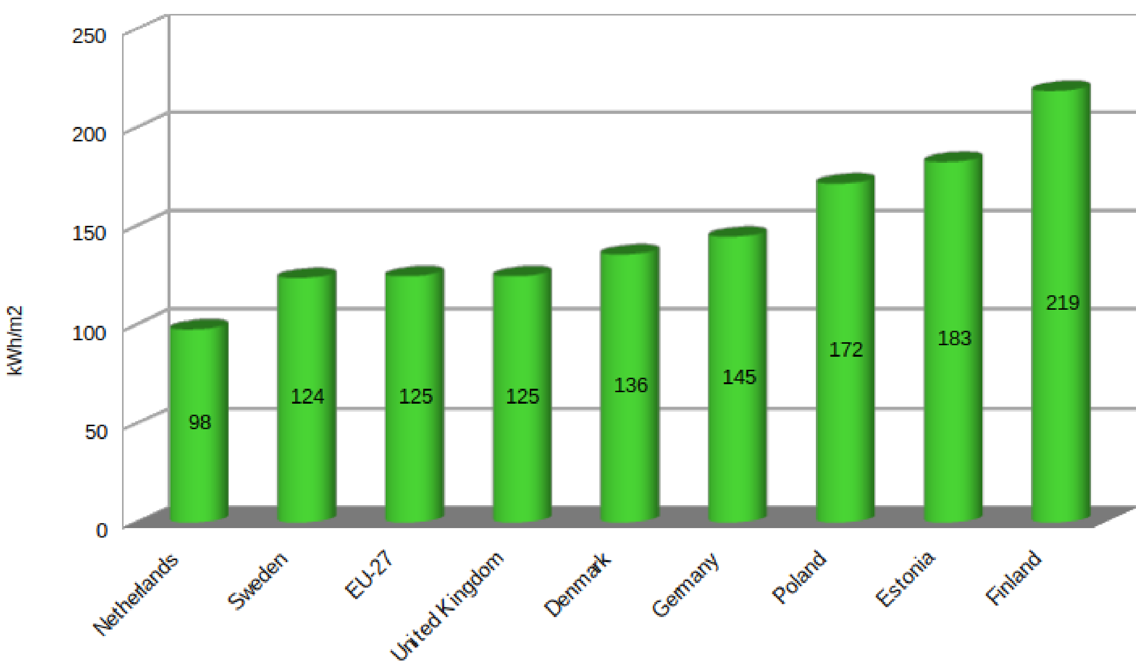
As discussed earlier in the document (especially in section 2.6), models can be simulated in a range of scenarios, programmatically. In the following sections, typical questions that house builders and homeowners ask about building insulation were grouped into collections of scenarios and then simulated and analysed. For example, how much energy can I save if I reduce the inside temperature, or make the insulation 20mm thicker?

The Python code used to automate the simulation executions is provided in appendix 4. When the code was executed, a list of scenarios was simulated in sequence. For each

scenario, the total watts for the whole year and the maximum watts required in any given hour was recorded. The building floor area data is extracted from the model and the energy use per square metre was then calculated. The results of each scenario were then graphed and reported as a PDF document.

As the results were a table of hourly results, heat demand for the whole year was calculated as the sum of hourly watts (W) divided by 1000 to give total kilowatt hours (kWh). The maximum watts result is useful for sizing heating devices, because a device with a heat production rating of just above the maximum needed would be the best one to install. Energy use per square metre is convenient for comparison with other regions and building types.

Figure 20. Household energy consumption for space heating, data from (EEA, 2015)

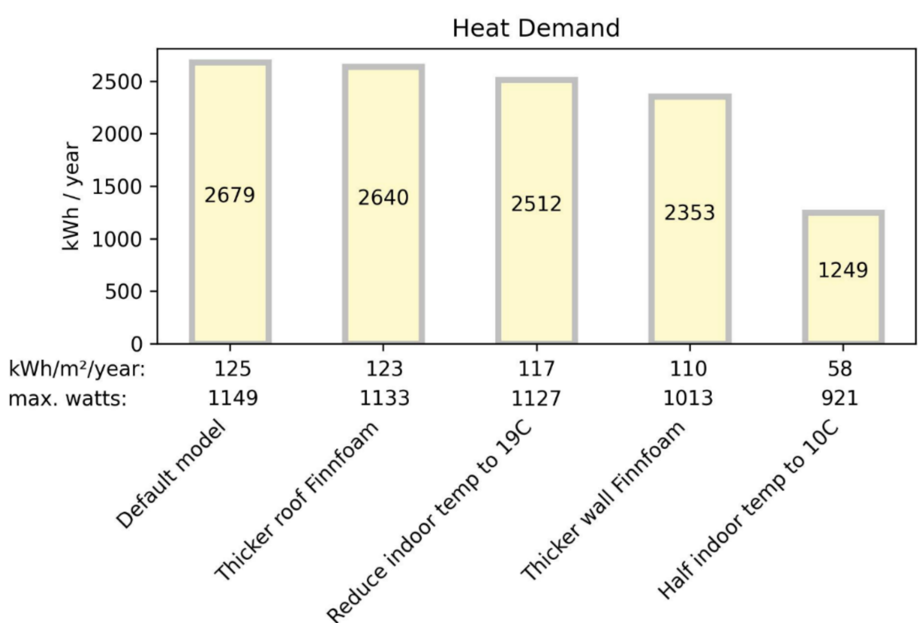


As demonstrated in *Figure 20*, there seems to be some scope for optimising heating through insulation improvements in Finland. In the following discussion, the price of electricity is set at 0.163 EUR/kWh (data from own power bill, Spring 2022).

3.2.1 Reduce Heat Demand

A common energy saving measure is to try and reduce heating demand. As energy prices rise, it is common to hear the suggestion “turn the thermostat down one degree and save money/save the planet!”. But how much is saved? Could there be a better way to save energy yet still maintain a comfortable temperature, such as by making the building insulation thicker? *Figure 21* shows the results of running these what-if scenarios through the dynamic modelling process. The code used to create the plot is listed in appendix 4, and the full report produced is shown in appendix 5.

Figure 21. Results of simulations that reduce heat demand



The default model (100mm Finnfoam on roof and walls, triple-glazed windows with u-value of 0.98 W/m^2.K) set the baseline and had a reasonable heating demand of 125 kWh/m^2/year compared to the Finnish average of 219 kWh/m^2/year (EEA, 2015). Assuming heating is by simple electrical resistance, if the indoor temperature is reduced by one degree, the saving per year would be 0.163 EUR x 167 kWh which equals 27.22 EUR. Halving the inside temperature to 10°C reduces energy usage by 54%, which may make sense if the house is not in use for longer periods of time.

The best reductions in heat demand without reducing the comfort level inside were by making the insulation a little bit thicker. Using insulation with a low u-value means that every extra milimetre of thickness has a measurable benefit; so, the wall can still have a reasonable total thickness, but have good insulation values. Adding 20mm of PIR board to the roof gave a small saving, but thickening the Finnfoam on the walls to 150mm gave the best reduction in heat demand whilst maintaining room temperature at 20°C.

As well as upgrading the insulation, an obvious energy saving improvement was to use a heater based on a heat pump. For example, a mid-priced (~1100 EUR + installation), recent model air source heat pump can have a seasonally adjusted coefficient of performance (SCOP) of 4.3, a nominal heating capacity of 3.4 kW, and a minimum heating capacity of 0.85 kW (Panasonic, 2022). This comfortably encompasses the maximum wattage required of 1149 W. Is this a worthwhile investment?

The net present value (NPV) method was used to evaluate the investment. This is used to calculate a rate of return ('internal rate of return' (IRR)) for the money invested in the system. The IRR when NPV is zero can be used to compare alternative investments: a higher IRR hints at a better investment.

Using these figures and the heat demand value from the simulation, an NPV calculation was performed over 10 years (likely lifetime of the heat pump). This gives a healthy IRR of 11.42% (see *Figure 22*) . If no loan is required, or if the loan rate can be kept below the IRR, then this investment is financially sound. Note that electricity price inflation is set at 4.5% which reflects recent steeper rises in the cost of electricity. A fully detailed calculation worksheet can be found in appendix 6.

Figure 22. Calculating the return on investment of a new heat pump

Energy Production System:	Air Heat Pump	10 years	Alternative Energy Source:	Grid Electricity
Energy Production Parameters			**Financial Parameters**	
Heat demand:	2679.00		Grid Electricity cost per kWh:	€0.16
SCOP:	4.30		Initial cost of Air Heat Pump system:	€2,099.00
Air Heat Pump saves (kWh):	2055.98	in year 1	Future value discount rate:	11.42%
Annual output degradation (linear):	0.60%		Grid Electricity price inflation rate:	4.50%

To sum up, the best energy saving strategy based on these simulation scenarios was to thicken the wall insulation and heat with a heat pump. Note that this may be seen as obvious without doing any analysis, but the simulations gave hard facts to help choose how much extra insulation to buy and to guide the choice of heat pump (over-sizing heating appliances wastes energy and costs more).

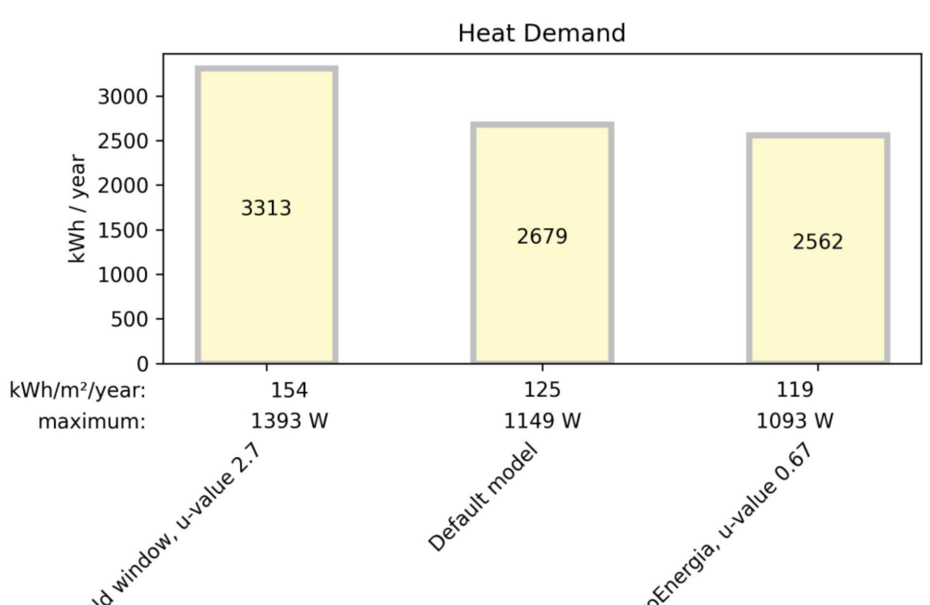
3.2.2 Change the Windows

Another way to improve the energy efficiency of a building is to upgrade the windows. A set of simulation scenarios was run to provide a comparison of:

- The model default window, which was the Piklas MSEA triple-glazed window with a U-value of 0.98 W/(m2.K) and a g-value of 0.55,
- Older double-glazed window with U-value of glass and frame of 2.7 W/(m2.K) and a g-value of 0.8, and
- The latest Piklas MEKA EkoEnergia triple-glazed window with a U-value of 0.67 W/(m2.K) and a g-value of 0.47. (Piklas, 2019)

The simulation results are presented in *Figure 23*.

Figure 23. Results of simulations with different window types



It is very clear that the older windows lose more heat to the environment than standard modern windows. As can be seen in *Figure 24*, renewing old windows for mid-range newer windows(the model default windows) gave a good internal rate of return over 25 years of more than 8%. As window lifetime should be easily more than 25 years, this is a good investment.

Figure 24. Calculating the return on investment of old windows to new MSEA windows

Energy Production System:	MSEA Windows	25 years
Energy Production Parameters		
Windows save (kWh):	634.00	
Annual output degradation (linear):	0.00%	in year 1

Alternative Energy Source:	Grid Electricity
Financial Parameters	
Electricity cost per kWh:	€0.16
Windows system:	€1,500.00
Future value discount rate:	8.23%
Electricity price inflation rate:	4.50%

What about updating to the very latest EkoEnergia windows from current standard windows (the model default windows)? The energy saving per year is only 117 kWh, so it takes far longer - more than 35 years - to start to see a return on investment, as shown in *Figure 25*.

Figure 25. Calculating the return on investment of MSEA windows to EkoEnergia windows

Energy Production System:	EkoEnergia Windows	35 years
Energy Production Parameters		
Windows save (kWh):	117.00	in year 1
Annual output degradation (linear):	0.00%	

Alternative Energy Source:	Grid Electricity
Financial Parameters	
Electricity cost per kWh:	€0.16
Windows system:	€1,710.00
Future value discount rate:	-0.43%
Electricity price inflation rate:	4.50%

The simulations show that upgrading old windows to newer energy efficient windows is worth doing but changing relatively new windows for the latest and greatest is not generally worth it.

4 Conclusions and Reflection

4.1 Lessons Learnt

The combination of Modelica models simulated automatically and programmatically is robust and usable in industry. The example building insulation model has around 6000 variables and 1000 complex equations, yet the simulation of five scenarios over a full year (8760 samples multiplied by 5) takes around three minutes on a mid-range laptop (see appendix 5). In the past general answers to building insulation performance queries have been available; now, very specific answers can be found by modelling the building and location much more precisely. Being able to run simulations automatically and programmatically allows the model to be used in customer interactions, in order to provide a PDF from the simulation run as the basis of a quote or recommendation, for example. More interestingly, the simulation and recommendations can be tailored precisely to the building being analysed, not generalized over a set of building types. With a set of Modelica objects defined for Finnish conditions, building materials, and construction techniques, it would be straightforward to offer a customised heating requirements checkup for any type of building.

A key objective in this thesis project was to raise awareness of the capabilities of Modelica as a way to model multi-physics engineering systems, especially in areas with cost constraints, but with a high impact; for example, lowering building heating costs saves owner-occupiers money and reduces reliance on problematic energy sources such as green-house gas producing power stations and nuclear power. It seems that OpenModelica is capable of this and much, much more. Being able to build models openly and logically from first principles (‘textbook equations’) and conversely, being able to drill-down in existing models to see the basic building blocks, is not only empowering for practicing engineers, but is especially delightful in engineering education.

4.2 Further Opportunities

For a more widespread use of the building insulation model, a friendlier way to add, edit and select scenarios could be added to the code. Adding a graphical user interface (GUI) above the Python code would allow the model to be used by those not expert in the details of the model, but who can utilise the simulated model results in their work. The GUI would allow model parameters to be changed and simulations made, but the actual model itself would not be altered.

Another key initiative of the Modelica Association is the Functional Mock-up Interface (Modelica Association FMI, 2022). This is an open-source standard that defines how models can be packaged up and shared with other tools. For example, the models defined in this document could be exported from OpenModelica (via the OMEdit application) to a functional mock-up unit (FMU) file and then utilised in other applications such as Dymola or more than 170 others (Modelica Association FMI tools, 2022). The FMU file is simply a zip compressed collection of configuration files, binaries, and C code that represent an executable version of the model. It allows others to use the model easily in their own modelling environments and is the basis of the co-simulation concept, whereby a model can be executed alongside other (perhaps real-time) processes in a manner that lets the model predict paths or potential faults for the process or even act as a full 'digital twin'. The models described in this thesis could have been loaded as FMUs and simulated using the Python library 'OMSimulator' (OpenModelica Simulator, 2022), a standalone simulation environment designed for execution of FMUs. However, the latest beta version of the BuildingSystems library required the latest version of the Modelica language, which the OpenModelica Simulator did not yet fully support. A future opportunity would be to utilise FMI with an updated version of the OpenModelica Simulator.

For automation engineers, companies that provide programmable logic controllers (PLCs), such as Beckhoff Automation, now provide support for FMI (Beckhoff Automation, 2022). In Beckhoff's TwinCAT 3 engineering environment with component TF1420, this allows FMU-based models to be compiled as runtime modules that can be parameterised and activated during the execution of PLC code. Building management systems, particularly control of

indoor climate, use a PLC-based controller. Further investigations could be undertaken to study the use of models of a building's HVAC response to changes in scenario such as occupancy level or changes in outdoor climate. For example, as part of the control loop of the PLC, the model could be used to 'goal seek' a particular outcome, such as a set indoor temperature, by altering parameters in multiple simulation runs to converge on the desired state.

References

Dassault Systèmes. (2022). *DYMOLA Systems Engineering*. Retrieved from Dassault Systèmes: https://www.3ds.com/products-services/catia/products/dymola/

Aguilar-Santana, J. L., Jarimi, H., Velasco-Carrasco, M., & Riffat, S. (2019, 12 5). Review on window-glazing technologies and future prospects. *International Journal of Low-Carbon Technologies, 15*(1), pp. 112-120. doi:10.1093/ijlct/ctz032

Åström, K. J., Elmqvist, H., & Mattsson, S. E. (1998). EVOLUTION OF CONTINUOUS-TIME MODELING AND SIMULATION. *The 12th European Simulation Multiconference, ESM'98.* Manchester: The European Multidisciplinary Society for Modelling and Simulation Technology. Retrieved from https://modelica.org/publications/papers/esm98his.pdf

Beckhoff Automation. (2022). *TF1420 | TwinCAT 3 Runtime for FMI*. Retrieved from Beckhoff New Automation Technology: https://www.beckhoff.com/en-en/products/automation/twincat/tfxxxx-twincat-3-functions/tf1xxx-tc3-system/tf1420.html

Bird, J. (2014). *Electrical Circuit Theory and Technology.* Abingdon: Routledge.

Cellier, F. E. (1991). *Continuous system modeling.* New York: Springer Science+Business Media.

Cellier, F. E., & Kofman, E. (2006). *Continuous System Simulation.* New York: Springer Science+Business Media.

Duncan, T., & Kennett, H. (2021). *Cambridge IGCSE Physics.* London: Hodder Education.

EEA. (2015, 01 06). *Household energy consumption for space heating per m2*. Retrieved from European Environment Agency: https://www.eea.europa.eu/data-and-maps/daviz/unit-consumption-of-space-heating

Egel, T. (2009). *Real Time Simulation Using Noncausal Physical Models.* Retrieved from Mathworks: https://se.mathworks.com/company/newsletters/articles/real-time-simulation-using-noncausal-physical-models.html

FinnFoam_PIR. (2022). *Technical specifications*. Retrieved from Finnfoam: https://www.finnfoam.com/products/ff-pir/properties

Finnfoam_RK. (2020, 12 17). *FF-PIR Rakennekortti - Seinärakene.* Retrieved from Finnfoam: https://www.finnfoam.fi/files/rakennekuvat/ff-pir/2021/FF-PIR_US04_puurunko_2021_fi.pdf

Finnish Meteorological Institute. (2022, 02 25). *The Finnish Meteorological Institute's open data*. Retrieved from Finnish Meteorological Institute: https://en.ilmatieteenlaitos.fi/open-data

Fritzson. (2015). *Principles of object oriented modeling and simulation with Modelica 3.3 : a cyber-physical approach.* Piscataway: IEEE Press.

Fritzson, P., & et al. (2020). The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control, 41*(4), 241–285. doi:10.4173/mic.2020.4.1

Lide, D. R. (2004). *CRC Handbook of Chemistry and Physics.* CRC Press.

Maplesoft. (2022). *MapleSim*. Retrieved from Maplesoft: https://www.maplesoft.com/products/maplesim/index.aspx

Modelica Association. (2020, 10 1). *Documentation of Modelica Standard Library - Version 4.0.0*. Retrieved from Modelica: https://doc.modelica.org/Modelica%204.0.0/Resources/helpDymola/Modelica.html

Modelica Association. (2022). *The Modelica Association*. Retrieved from Modelica Association: https://modelica.org/

Modelica Association FMI. (2022). *Functional Mock-up Interface*. Retrieved from FMI Standard: https://fmi-standard.org/

Modelica Association FMI tools. (2022). *Tools that support FMI*. Retrieved from FMI Standard: https://fmi-standard.org/tools/

Modelica Association Libraries. (2022). *Modelica Libraries - A collection of free and commercial libraries.* Retrieved from Modelica Association: https://modelica.org/libraries.html

Moraleda, A. U., & Villalaba, C. M. (2018). *Modeling and Simulation in Engineering Using Modelica.* Madrid: Universidad Nacional de Educación a Distancia.

Nagel, L. W., & Pederson, D. O. (1973). *SPICE (Simulation Program with Integrated Circuit Emphasis), Memorandum No. ERL-M382.* Berkeley: University of California.

Open Source Modelica Consortium. (2020). *Open Source Modelica Consortium*. Retrieved from OpenModelica: https://www.openmodelica.org/home/consortium

OpenModelica. (2022). *Solving Modelica Models, Integration Methods*. Retrieved from OpenModelica: https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/solving.html

OpenModelica Simulator. (2022). *OMSimulatorPython*. Retrieved from OpenModelica OMSimulator: https://www.openmodelica.org/doc/OpenModelicaUsersGuide/latest/omsimulator.html#omsimulatorpython

Panasonic. (2022). *CZ Inverter+ CZ25WKE*. Retrieved from Panasonic EU: https://www.aircon.panasonic.eu/FI_fi/model/cs-cz25wke-cu-cz25wke/

Petzold, L. R. (1982). Description of DASSL: a differential/algebraic system solver. *10. international mathematics and computers simulation congress on systems simulation and scientific computation.* Montreal. Retrieved from https://www.osti.gov/servlets/purl/5882821

Piklas. (2019, 4 26). *SUORITUSTASOILMOITUS (DECLARATION OF PERFORMANCE).* Retrieved from Piklas Ikkunat ja Ovet: https://www.piklas.fi/download_file/view/852/266

UdK Berlin. (2022). *BuildingSystems*. Retrieved from Universität der Künste Berlin: https://modelica-buildingsystems.de/index.html

Wolfram Research. (2022). *Wolfram System Modeler*. Retrieved from Wolfram Research: https://www.wolfram.com/system-modeler/

Young, H. D., & Freedman, R. A. (2012). *University Physics.* Addison-Wesley: San Francisco.

Zupancic, B., Karba, R., Atanasijevic-Kunc, M., & Music, J. (2008). *Continuous Systems Modelling Education – Causal or Acausal Approach?* Ljubljana: University of Ljubljana. doi:10.1109/ITI.2008.4588514

Appendix 1: Code Listing of ExampleCircuit Modelica Package

```
package ExampleCircuit
  type ElectricPotential = Real (unit="V");
  type ElectricCurrent = Real (unit="A");
  type ElectricResistance = Real (unit="Ohm");
  type ElectricPower = Real (unit="W");

  connector Pin "Pin of an electrical component"
    ElectricPotential v "Potential at the pin";
    flow ElectricCurrent i "Current flowing into the pin";
  end Pin;

  partial model TwoPin "Common parts of two pin electrical components"
    Pin p "Positive electrical pin";
    Pin n "Negative electrical pin";
    ElectricPotential v = p.v-n.v "Difference in electric potential between pins";
    ElectricCurrent i = p.i;
  equation
    p.i + n.i = 0 "Conservation of charge";
  end TwoPin;

  model Resistor "Ideal linear electrical resistor"
    parameter ElectricResistance R "Nominal resistance";
    extends TwoPin;
    ElectricPower LossPower "Loss power leaving component as heat";
  equation
    v = p.i*R "Ohm's law";
    LossPower = v*p.i;
  end Resistor;

  model ConstantVoltage "Source for constant voltage"
    parameter ElectricPotential V(start=1) "Value of constant voltage";
    extends TwoPin;
  equation
    v = V;
  end ConstantVoltage;

  model Ground "Ground node"
    Pin p "Positive electrical pin";
  equation
    p.v = 0;
  end Ground;

  model Circuit "An example circuit"
    ConstantVoltage constantVoltage(V = 12);
    Resistor resistor(R = 1.2);
    Ground;
  equation
    connect(constantVoltage.p, resistor.p);
    connect(constantVoltage.n, resistor.n);
    connect(constantVoltage.n, ground.p);
  end Circuit;
end ExampleCircuit;
```

Appendix 2: Code Listing of Insulation Modelica Package

```
package Insulation
  block Climate_Finland_Jokioinen
    "Weather data for a typical location in South Finland (near Forssa)"
    extends BuildingSystems.Climate.WeatherData.BaseClasses.WeatherDataFileASCII(
    info = "Source: Finnish Meteorological Institute (Ilmatieteenlaitos)",
    filNam = Modelica.Utilities.Files.loadResource("Finland_Jokioinen_Ilmala-v2021.txt"),
    final tabNam = "tab1",
    final timeFac = 1.0 / 3600.0,
    final deltaTime = 1800.0,
    final columns = {5, 6, 3, 8, 9, 4, 7},
    final scaleFac = {1.0, 1.0, 1.0, 1.0, 1.0, 0.01, 1.0},
    final latitudeDeg = 60.81397,
    final longitudeDeg = 23.49825,
    final longitudeDeg_0 = 1.0);
  end Climate_Finland_Jokioinen;

  record PIRInsulation =
    BuildingSystems.HAM.Data.MaterialProperties.BaseClasses.MaterialThermalGeneral(
    name = "PIR (polyisocyanurate) insulation board",
    lambda = 0.022,
    c = 1500,
    rho = 33)
    "PIR (polyisocyanurate) insulation board;

  record SteelSheet =
    BuildingSystems.HAM.Data.MaterialProperties.BaseClasses.MaterialThermalGeneral(
    name = "Steel sheet",
    lambda = 45.0,
    c = 466.0,
    rho = 7850.0)
    "Ruukki Classic C 475;

  record WoodFibreBoard =
    BuildingSystems.HAM.Data.MaterialProperties.BaseClasses.MaterialThermalGeneral(
    name = "Wood Fibre Board (Runkoleijona)",
    lambda = 0.049,
    c = 1200,
    rho = 250.0)
    "Runkoleijona tuulensuojaeriste";

  record OuterWallFIN =
    BuildingSystems.Buildings.Data.Constructions.OpaqueThermalConstruction(
    final nLayers = 3,
    thickness = {0.013, 0.1, 0.028},
    material = {BuildingSystems.HAM.Data.MaterialProperties.Thermal.Masea.Plasterboard(),
              PIRInsulation(),
              BuildingSystems.HAM.Data.MaterialProperties.Thermal.Wood()})
    "Outer wall construction (based on Finnfoam construction diagram)";

  record RoofFIN =
    BuildingSystems.Buildings.Data.Constructions.OpaqueThermalConstruction(
    final nLayers = 4,
    thickness = {0.012, 0.150, 0.1, 0.0006},
    material = {BuildingSystems.HAM.Data.MaterialProperties.Thermal.Wood(),
```

```
      BuildingSystems.HAM.Data.MaterialProperties.Thermal.DIN_V_4108.Mineralwolle040(),
      PIRInsulation(),
      SteelSheet()})
  "Roof construction";

record BaseFIN =
  BuildingSystems.Buildings.Data.Constructions.OpaqueThermalConstruction(
  final nLayers = 3,
  thickness = {0.032, 0.1, 0.025},
  material = {BuildingSystems.HAM.Data.MaterialProperties.Thermal.Wood(),
     BuildingSystems.HAM.Data.MaterialProperties.Thermal.DIN_V_4108.Mineralwolle040(),
     WoodFibreBoard()})
  "Base plate construction";

record WindowFIN =
  BuildingSystems.Buildings.Data.Constructions.TransparentConstruction(
  final nPanes = 3,
  thickness = {0.003, 0.003, 0.003},
  UValGla = 0.98,
  UValFra = 0.98,
  g = 0.55,
  b0 = 0.7)
  "Piklas MSEA triple glazed window with UValGla = 0.98 W/(m2.K) and g = 0.55;

model SimpleBuildingFIN
  "A simple Finnish building"
  // Wall or Window: 1 West, 2 North, 3 East, 4 South
  BuildingSystems.Buildings.BuildingTemplates.Building1Zone1DBox building(
  width = 3.7,
  length = 5.8,
  height = 2.25,
  InteriorWalls = false,
  InteriorCeilings = true,
  redeclare OuterWallFIN constructionWall1,
  redeclare OuterWallFIN constructionWall2,
  redeclare OuterWallFIN constructionWall3,
  redeclare OuterWallFIN constructionWall4,
  redeclare RoofFIN constructionCeiling,
  redeclare BaseFIN constructionBottom,
  redeclare WindowFIN constructionWindow1, widthWindow1 = 1.2, heightWindow1 = 1.2,
  redeclare WindowFIN constructionWindow2, widthWindow2 = 0.0, heightWindow2 = 0.0,
  redeclare WindowFIN constructionWindow3, widthWindow3 = 1.2, heightWindow3 = 1.2,
  redeclare WindowFIN constructionWindow4, widthWindow4 = 1.2, heightWindow4 = 1.2,
  nZones = 1);

  BuildingSystems.Buildings.Ambience ambience(redeclare block
       WeatherData = Climate_Finland_Jokioinen, nSurfaces = building.nSurfacesAmbience);

equation
  connect(ambience.toSurfacePorts, building.toAmbienceSurfacesPorts);
  connect(ambience.toAirPorts, building.toAmbienceAirPorts;
  connect(ambience.TAirRef, building.TAirAmb);
  connect(ambience.xAir, building.xAirAmb);
  connect(building.T_setHeating[1], TSetHeating.y);
  connect(building.T_setCooling[1], TSetCooling.y);
  connect(building.airchange[1], airchange.y);
```

```
  annotation(
    experiment(StartTime = 1, StopTime = 3.1536e+07, Tolerance = 0.0001,
                  Interval = 3600),
    uses(Modelica(version = "4.0.0"), BuildingSystems(version = "2.0.0-beta"));
  end SimpleBuildingFIN;
end Insulation;
```

Appendix 3: Code Listing of Heater Python Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import matplotlib.pyplot as plt

from OMPython import OMCSessionZMQ
from OMPython import ModelicaSystem


def doPlot(scenario="", pdfName="../figureHeater.pdf", timeScale="h"):
    # Define a function to plot the results of the simulation
    time, T_water, P_resistor, Q_water, Q_container, R_resistor, I_resistor = model.getSolutions(
        ["time", "T_Water", "resistor.LossPower",
         "heatCapacitor.port.Q_flow", "thermalConductor.Q_flow",
         "resistor.R_actual", "resistor.i"])  # return list of numpy arrays

    # Adjust x-axis timescale
    if (timeScale == "h"):
        # convert seconds to hours (divide by the number of seconds in an hour)
        time = time/(60*60)
    elif (timeScale == "min"):
        # convert seconds to minutes (divide by the number of seconds in a minute)
        time = time/(60)
    # Otherwise: seconds

    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(
        nrows=2, ncols=2, figsize=(6.4, 6.4), constrained_layout=True)
    fig.suptitle("Simple Water Heater Model\n" + scenario)

    # convert degrees Kelvin to degrees Centigrade
    T_water = T_water-273.15
    ax1.plot(time, T_water, color='red')
    ax1.set_xlabel(f'Time ({timeScale})')
    ax1.set_ylabel('\u2103')
    ax1.set_title("Water Temperature")
    ax1.grid(which='major', axis='both')
    ax1.set_ylim(0, 110)

    ax2.plot(time, P_resistor, label='from Heater Coil')
    ax2.plot(time, Q_water, label='into Water')
    ax2.plot(time, Q_container, label='through Container')
    ax2.set_xlabel(f'Time ({timeScale})')
    ax2.set_ylabel('Watts')
    ax2.set_title("Energy Flow")
    ax2.legend()
    ax2.grid(which='major', axis='both')

    ax3.plot(time, R_resistor)
    ax3.set_xlabel(f'Time ({timeScale})')
    ax3.set_ylabel('Ohms')
    ax3.set_title("Heater Coil\nActual Resistance")
    ax3.grid(which='major', axis='both')

    ax4.plot(time, I_resistor)
    ax4.set_xlabel(f'Time ({timeScale})')
    ax4.set_ylabel('Amps')
    ax4.set_title("Heater Coil\nCurrent Flow")
    ax4.grid(which='major', axis='both')

    # creates vector-based PDF
    fig.savefig(pdfName, dpi=600)


# %%Setup
os.chdir("./work/")

omc = OMCSessionZMQ()
print(omc.sendExpression("getVersion()"))

# %% Compile model
model = ModelicaSystem("../Heater.mo", "Heater")
```

```
# %% Simulate with default parameters
model.simulate()

# %% Deault scenario: aluminium container
# Aluminium thermal conductivity (k) = 237 W·m−1·K−1
doPlot(scenario="(one litre of water in a 3mm thick aluminium container)")

# %% Scenario: stainless steel container
# Stainless steel thermal conductivity = 16.3 W·m−1·K−1
model.setParameters([f"thermalConductor.G={16.3*0.003}"])
model.simulate()
doPlot(scenario="(one litre of water in a 3mm thick stainless steel container)",
       pdfName="../figureHeater2.pdf")

# %% Scenario: 300ml of water
# Stainless steel thermal conductivity = 16.3 W·m−1·K−1
# Specific heat capacity of water is around 4200 J/K
model.setParameters([f"heatCapacitor.C={4200*0.3}"])
# Set a shorter simulation period (15mins=900s)
model.setSimulationOptions(["stopTime=900"])
model.simulate()
doPlot(scenario="(300ml of water in a 3mm thick stainless steel container)",
       pdfName="../figureHeater3.pdf", timeScale="min")
```

Appendix 4: Code Listing of Insulation Python Code

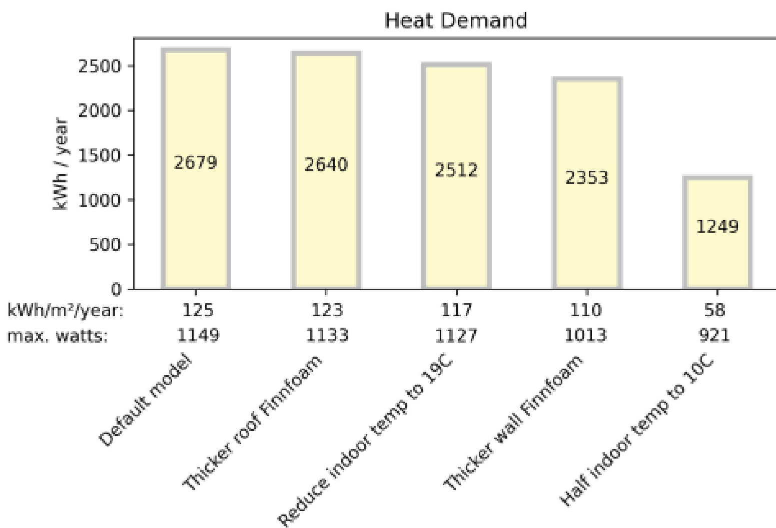
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# %%Setup
import os, time
import matplotlib.pyplot as plt

from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Image
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm

from OMPython import OMCSessionZMQ
from OMPython import ModelicaSystem

tik = time.perf_counter()

os.chdir("./work/")

omc = OMCSessionZMQ()
print(omc.sendExpression("getVersion()"))

# %% Compile model
# See https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/ompython.html#enhanced-ompython-
features
# Compiler (omc) flags reference at:
https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html
modelName = "Insulation.SimpleBuildingFIN"

model = ModelicaSystem(
    "../Insulation.mo",
    modelName,
)

# %% Run simulation scenarios
# Note: scenarios build upon each other, so settings made in earlier simulations
# stay the same for later simulations. Re-compile the model to start from defaults.
scenarios = [
    [
        "Default model",
        [],
    ],
    [
        "Reduce indoor temp to 19C",
        [
            f"TSetHeating.k={273.15 + 19.0}",
        ],
    ],
    [
        "Half indoor temp to 10C",
        [
            f"TSetHeating.k={273.15 + 10.0}",
        ],
    ],
    [
        "Thicker roof Finnfoam",
        [
            f"TSetHeating.k={273.15 + 20.0}",
            f"building.constructionCeiling.thickness[3]={0.12}",
        ],
    ],
    [
        "Thicker wall Finnfoam",
        [
            f"TSetHeating.k={273.15 + 20.0}",
            f"building.constructionWall1.thickness[2]={0.15}",
            f"building.constructionWall2.thickness[2]={0.15}",
            f"building.constructionWall3.thickness[2]={0.15}",
            f"building.constructionWall4.thickness[2]={0.15}",
            f"building.constructionCeiling.thickness[3]={0.1}",
        ],
    ],
```

```
]

model.setSimulationOptions(
    [
        "startTime=1",
        "stopTime=3.1536e+07",
        "stepSize=3600",
        "tolerance=1e-04",
        "solver=dassl",
    ]
)

results = []
for count, scenario in enumerate(scenarios):
    model.setParameters(scenario[1])
    model.simulate(simflags="-noEventEmit")
    Q_heat = model.getSolutions(["building.Q_flow_heating[1]"])
    results.append(
        (
            count,
            scenario[0],
            scenario[1],
            Q_heat.sum() / 1000,
            Q_heat.max(),
            Q_heat.size,
        ),
    )

tok = time.perf_counter()

# Sort results by kWh per year descending
results = sorted(results, key=lambda sortColumn: sortColumn[3], reverse=True)

# %% Collect data from the model for further calculations in the results
buildingLength, buildingWidth = list(
    map(float, model.getParameters(["building.length", "building.width"]))
)
heatedArea = buildingLength * buildingWidth

# %% Plot with matplotlib and numpy
fig, ax = plt.subplots(figsize=(6.4, 4.2), constrained_layout=True)

bars = ax.bar(
    list(range(0, len(results))),
    [result[3] for result in results],
    width=0.5,
    edgecolor="silver",
    color="lemonchiffon",
    linewidth=3,
)

ax.set_xticks(list(range(0, len(results))))
ax.set_xticklabels([result[1] for result in results], rotation=45, ha="right", y=-0.2)

ax.table(
    cellText=[
        [f"{result[3]/heatedArea:.0f}" for result in results],
        [f"{result[4]:.0f}" for result in results],
    ],
    loc="lower center",
    cellLoc="center",
    edges="",
    bbox=[0, -0.24, 1, 0.2],
    rowLabels=["kWh/m\xB2/year:", "max. watts:"],
)

ax.set_ylabel("kWh / year")
ax.set_title("Heat Demand")

ax.bar_label(bars, label_type="center", padding=3, fmt="%.0f")

# create JPEG of plot
cleanFilename = "".join([x if x.isalnum() else "_" for x in modelName])
fig.savefig(f"../figure{cleanFilename}.jpg", dpi=300)

# %% Create final results PDF report
styles = getSampleStyleSheet()
```

```
pdf = SimpleDocTemplate(
    f"../report{cleanFilename}.pdf",
    pagesize=A4,
    title=f"Heat Demand: {modelName}",
    author="",
    producer="",
)
flow = [Spacer(1, 0.1 * cm)]
flow.append(Paragraph(f"Model: {modelName}", styles["Title"]))
flow.append(Spacer(1, 0.8 * cm))
flow.append(Image(f"../figure{cleanFilename}.jpg", width=350, height=230))
flow.append(Spacer(1, 0.8 * cm))
flow.append(
    Paragraph(
        f"<i>Heated area of {heatedArea} m<sup>2</sup></i>",
        styles["Normal"],
    )
)
flow.append(Spacer(1, 0.2 * cm))

for result in results:
    flow.append(
        Paragraph(
            f"<b>{result[1]}:</b> {result[3]:.0f} kWh/yr,"
            + f" {result[4]:.0f} W max, "
            + f" {result[3]/heatedArea:.0f} kWh/m<sup>2</sup>/yr "
            + f"<i>({result[5]} samples)</i>",
            styles["Normal"],
        )
    )
    for parameter in result[2]:
        flow.append(Paragraph(f"{parameter}", styles["Normal"], bulletText=" "))
    flow.append(Spacer(1, 0.2 * cm))

flow.append(
    Paragraph(
        f"<i>Simulated {len(results)}"
        + f" scenario{'s' if len(results)>1 else ''} in"
        + f" {time.strftime('%Hh:%Mm:%Ss',time.gmtime(tok-tik))}</i>",
        styles["Normal"],
    )
)

pdf.build(flow)
```

Appendix 5: Sample Report of Multiple Simulation Scenarios

Model: Insulation.SimpleBuildingFIN



Heated area of 21.46 m^2

Default model: 2679 kWh/yr, 1149 W max, 125 kWh/m^2/yr *(8761 samples)*

Thicker roof Finnfoam: 2640 kWh/yr, 1133 W max, 123 kWh/m^2/yr *(8761 samples)*
TSetHeating.k=293.15
building.constructionCeiling.thickness[3]=0.12

Reduce indoor temp to 19C: 2512 kWh/yr, 1127 W max, 117 kWh/m^2/yr *(8761 samples)*
TSetHeating.k=292.15

Thicker wall Finnfoam: 2353 kWh/yr, 1013 W max, 110 kWh/m^2/yr *(8761 samples)*
TSetHeating.k=293.15
building.constructionWall1.thickness[2]=0.15
building.constructionWall2.thickness[2]=0.15
building.constructionWall3.thickness[2]=0.15
building.constructionWall4.thickness[2]=0.15
building.constructionCeiling.thickness[3]=0.1

Half indoor temp to 10C: 1249 kWh/yr, 921 W max, 58 kWh/m^2/yr *(8761 samples)*
TSetHeating.k=283.15

Simulated 5 scenarios in 00h:03m:02s

Appendix 6: Net Present Value Calculation for Heat Pump Investment

Item	Supplier/Product	Cost	Incl?	Price source
Air-water heat pump	Panasonic	1,099.00 €	y	https://www.netrauta.fi/ilmalampopumppu-panasonic-cz25wke
Installation & sundries		1,000.00 €	y	
		2,099.00 €		

Energy Production System:	Air Heat Pump	10 years	Alternative Energy Source:	Grid Electricity
Energy Production Parameters			**Financial Parameters**	
Heat demand:	2679.00		Grid Electricity cost per kWh:	€0.16
SCOP:	4.30		Initial cost of Air Heat Pump system:	€2,099.00
Air Heat Pump saves (kWh):	2055.98	in year 1	Future value discount rate:	11.42%
Annual output degradation (linear):	0.60%		Grid Electricity price inflation rate:	4.50%

Calculations

	Air Heat Pump	Grid Electricity		Other			
Year	**Generated kWh**	**Price/kWh**	**Total Cost**	**Income**	**Costs**	**Total**	**Present Value**
1	2,056	€0.16	€335.12			€335.12	€296.85
2	2,044	€0.17	€348.10			€348.10	€273.14
3	2,031	€0.18	€361.57			€361.57	€251.31
4	2,019	€0.19	€375.55			€375.55	€231.21
5	2,007	€0.19	€390.05			€390.05	€212.72
6	1,994	€0.20	€405.10			€405.10	€195.69
7	1,982	€0.21	€420.71			€420.71	€180.02
8	1,970	€0.22	€436.90			€436.90	€165.60
9	1,957	€0.23	€453.70			€453.70	€152.33
10	1,945	€0.24	€471.13			€471.13	€140.12
						Gross Present Value	€2,099.00
						Net Present Value	€0.00

Spreadsheet formulas:

	A	B	C	D	E	F	G	H
1		Energy Production System:	Air Heat Pump				Alternative Energy Source:	Grid Electricity
2		**Energy Production Parameters**					**Financial Parameters**	
3		Heat demand:	2679.00				=H1&" cost per kWh:"	€0.16
4		SCOP:	4.30				="Initial cost of "&C1&" system:"	=Initial_Cost.C4
5		="Year 1 "&C1&" saves (kWh):"	=C3-(C3/C4)				Future value discount rate:	11.42%
6		Annual output degradation (linear):	0.60%				=H1&" price inflation rate:"	4.50%
7								
8	**Calculations**							
9		**=C1**	**=H1**		**Other**			
10	**Year**	**Generated kWh**	**Price/kWh**	**Total Cost**	**Income**	**Costs**	**Total**	**Present Value**
11	1	=C5	=H3	=B11*C11			=SUM(D11:E11)-F11	=G11*(1-$H$5)^A11
12	2	=$C$5*(1-($C$6*A11))	=C11*(1+$H$6)	=B12*C12			=SUM(D12:E12)-F12	=G12*(1-$H$5)^A12
13	3	=$C$5*(1-($C$6*A12))	=C12*(1+$H$6)	=B13*C13			=SUM(D13:E13)-F13	=G13*(1-$H$5)^A13
14	4	=$C$5*(1-($C$6*A13))	=C13*(1+$H$6)	=B14*C14			=SUM(D14:E14)-F14	=G14*(1-$H$5)^A14
15	5	=$C$5*(1-($C$6*A14))	=C14*(1+$H$6)	=B15*C15			=SUM(D15:E15)-F15	=G15*(1-$H$5)^A15
16	6	=$C$5*(1-($C$6*A15))	=C15*(1+$H$6)	=B16*C16			=SUM(D16:E16)-F16	=G16*(1-$H$5)^A16
17	7	=$C$5*(1-($C$6*A16))	=C16*(1+$H$6)	=B17*C17			=SUM(D17:E17)-F17	=G17*(1-$H$5)^A17
18	8	=$C$5*(1-($C$6*A17))	=C17*(1+$H$6)	=B18*C18			=SUM(D18:E18)-F18	=G18*(1-$H$5)^A18
19	9	=$C$5*(1-($C$6*A18))	=C18*(1+$H$6)	=B19*C19			=SUM(D19:E19)-F19	=G19*(1-$H$5)^A19
20	10	=$C$5*(1-($C$6*A19))	=C19*(1+$H$6)	=B20*C20			=SUM(D20:E20)-F20	=G20*(1-$H$5)^A20
21								
22							**Gross Present Value**	=SUM(H11:H20)
23							**Net Present Value**	=H22-H4