ABSTRACT

Title: **SYSML EXECUTABLE MODEL OF AN ENERGY EFFICIENT HOUSE AND TRADE-OFF ANALYSIS**

Kersasp Aspi Cawasji
Master of Science, Systems Engineering, 2018

Thesis Directed By: Professor John S. Baras
Institute for Systems Research

With the growing complexity of energy efficient buildings, the methods of modeling and simulating such structures must account for monitoring several thousand design parameters across multiple diverse domains. As a result, modeling tools are now very specific to their respective domains and are growing more and more incongruous with each other. This calls for a way to integrate multiple modeling tools in the effort to create a single, large model capable to encapsulate data from multiple, different models.

Thus, in this thesis, different methods to perform an integration with Systems Modeling Language (SysML) and a simulation tool were identified, described and evaluated. Then, a new method was developed and discussed. Finally, the new method was demonstrated by developing a SysML executable model of a simple two-room house that utilizes solar power for space heating, with a heat pump used as a backup. Using the Functional Mock-up Interface (FMI) standard, the SysML model is integrated with a Modelica model, and a simulation is run in Simulink. Finally, a tradeoff analysis was performed for the purpose of design space exploration.

SYSML EXECUTABLE MODEL OF AN ENERGY EFFICIENT HOUSE AND TRADEOFF ANALYSIS

by

Kersasp Aspi Cawasji

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2018

Advisory Committee:
Professor John S. Baras, Advisor/ Chair
Associate Professor Mark A. Austin
Professor Raymond A. Adomaitis

Dedication

To my parents, Aspi and Jeroo. Thank you for the golden gift of a sound education.

Acknowledgements

I would like to sincerely express my gratitude to my thesis advisor, Dr. John Baras for his support and valuable guidance throughout the length of my time at the University of Maryland. With his vast magnitude of knowledge and expertise in several fields, he has inspired me to push the limits of my mind and shown me that there is always more to learn.

I would also like to thank Dr. Mark Austin and Dr. Raymond Adomaitis for taking the time out to be a part of my thesis committee and providing me with their valuable insight along the way. Dr. John MacCarthy also deserves a very special thanks for his mentorship over the two years that I have been here. Not only was he always available to aid in our studies but also played a large role in developing a strong work ethic for all his students.

In addition, I would also like to acknowledge my peers and colleagues Amar Vamsi Krishna, Kunal Mehta, Rishabh Agarwal, and Samvrit Srinivas, and my partner Mallika Deepak, who have always been a great source of support and encouragement throughout my journey at UMD. I am grateful for always having them to fall back on for advice, ideas and much need comic relief when stress levels were high.

Finally, I would like to sincerely thank my parents, Aspi and Jeroo, and my younger brother, Yashaan, for providing unending support, life-lessons, guidance and love all through my life, and more so during my formative college years. Despite them being so far away, I have always felt like a part of our close-knit family.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BAS	Building Automation System
BDD	Block Definition Diagram
BMS	Building Management System
DAE	Differential Algebraic Equations
FMI	Functional Mockup Interface
FMU	Functional Mock-up Unit
HVAC	Heating, Ventilation, Air Conditioning
IBD	Internal Block Diagram
INCOSE	International Council on Systems Engineering
MBSE	Model Based Systems Engineering
OMG	Object Management Group
SysML	Systems Modeling Language

Chapter 1

Overview

1.1 Introduction

With the growing complexity of energy efficient buildings, the methods of modeling and simulating such structures must account for monitoring several thousand design parameters across multiple diverse domains. As a result, modeling tools are now very specific to their respective domains and are growing more and more incongruous with each other. This calls for a way to integrate multiple modeling tools in the effort to create a single, large model capable of encapsulating data from multiple, different models.

Buildings are complex systems that have numerous interactions between different components, spanning several different domains. Thus, when it comes down to designing a new building for the purpose of construction, many challenges are faced in the building modeling stages. This is especially true now, in the era of living, breathing "smart" buildings. Such a structure involves a vast array of data captured from sensors like occupancy sensors, lighting sensors, thermostats, security sensors etc. In a smart building all of these sensors can either be a part of an individual Building Management System (BMS), a partially integrated BMS or fully integrated Building Automation System (BAS).

1.2 Problem Statement

When it comes to modeling such a large system, for the purpose of construction, all the tools to do that are not very congruent with each other. In this work, we will be approaching this problem from a Systems Engineering point of view with the intention of being able to run a simulation using a Systems Modeling Language (SysML) tool known as No Magic Cameo Systems Modeler. The definition and execution of engineering models, some of which may simply be represented as black boxes through the use of SysML constraint blocks is of great interest in terms of practicing Model Based Systems Engineering [1].

To do this, using the Functional Mock-up Interface (FMI) standard, the SysML model is integrated with a Modelica model, through an intermediate Simulink Model, in which the simulation runs. Finally, a tradeoff analysis is run through SysML, in Matlab, for the purpose of design space exploration to demonstrate that meaningful decisions can be carried out using this approach. In this case, the tradeoff is between the cost of the thermal insulation used in the construction of the house versus the heat-load needed by the heat pump to maintain a constant indoor air temperature.

1.3 Contribution of this Thesis

The major contribution of this thesis is that different methods to perform an integration with SysML and a simulation tool were identified, described and evaluated. Then, a new method was developed and discussed. Finally, the new method was demonstrated.

This thesis work provides step by step instructions of the implementation of an Agile Model Based Systems Engineering approach through the usage of a SysML/ Model/ Simulation integration. This means that SysML was used to drive the models and simulations used in performance analyses and tradeoff analyses that are performed during system development [2].

This was demonstrated by developing an executable SysML model of a two-room house that utilizes solar-thermal power for the purpose of space heating with a heat pump being used as a backup. Using the FMI 2.0 standard, a Modelica model of the house was integrated with the SysML model using Simulink as an intermediate interface, enabling users to perform a tradeoff analysis by varying design parameters through the SysML interface.

Chapter 2

Background

2.1 Need for MBSE in Today's World.

Model Based Systems Engineering (MBSE) is an engineering paradigm gaining traction towards inculcating a model-centric approach to engineering instead of the age-old document-centric approach. As defined in the International Council on Systems Engineering (INCOSE) Systems Engineering Vision 2020, Model Based Systems Engineering is referred to as the "formalized application of modeling to support system requirements, design analysis, verification, and validation activities beginning with the conceptual design phase and continuing throughout development and later life cycle phases" [3].

The MBSE approach encourages Systems Engineers to improve the precision and efficiency of their communication with other Systems Engineers as well as stakeholders through the usage of a common visual modeling language [4]. The most popular choice for this modeling language is the Object Management Group's System Modeling Language, commonly known as OMG SysML.

The need for MBSE is felt when clear communication is required between the system designers and various stakeholders across the Systems Development Life Cycle. MBSE is also able to capture and manage corporate intellectual property related to systems architectures, designs, and process [4]. Along with being able to provide a scalable structure, for problem solving, as well as being able to explore multiple

architectures with minimum risk, an MBSE approach also helps in catching errors early in the Development Life Cycle. Through all these functions, the MBSE approach is able to enhance system performance.

In this work, in the spirit of MBSE, an attempt was successfully made to integrate Cameo Systems Modeler, an OMG SysML environment with an FMU, an output of Dymola, a Modelica based modeling and simulation engine. More about Cameo and Dymola and their usage in this thesis will discussed in Chapter 3. The main reasoning for following such an approach was based on the reasoning that integrated models reduce inconsistencies, enable automation and support early and continual verification by analysis.

Chapter 3

Modeling Tools

In Section 2.1, the need for the Model-Based Systems Engineering approach was discussed. As models obviously play a key role in MBSE, it is important for all the different modeling tools to work together in harmony in order to approach the modeling of this energy efficient house from a Systems Engineering standpoint. In this regard, the three tools that were used to create the executable model were Cameo Systems Modeler, Dymola and Simulink.

3.1 Cameo Systems Modeler – SysML Environment

No Magic Cameo Systems Modeler is a commercial cross-platform collaborative Model-Based Systems Engineering (MBSE) environment, which provides smart, robust, and intuitive tools to define, track, and visualize all aspects of systems in the most standard-compliant SysML models and diagrams [5]. For the purpose of this thesis, version 18.5 sp3 of Cameo Systems Modeler was used in a Windows 10 environment.

The reason Cameo Systems Modeler was the chosen SysML environment is because it is one of the two most widely used SysML Environments in the industry. The other most commonly used tool is IBM Rhapsody. In addition to that, the Systems Engineering courses at UMD were taught using Cameo Systems Modeler and it was also readily available for research purposes.

Systems Modeling Language or SysML is the modeling language most widely used by Systems Engineers. The main idea is that SysML is a standardized medium for communication; the rules defined in it give the model's elements and relationships unambiguous meaning. The capability to construct and read well-formed models is at the heart of the MBSE approach [6]. It enables the visualization of the system's design in the form of the four pillars of Systems Engineering, namely, the system structure, behavior, parametric relationships, and requirements. SysML is based off UML however, it has multiple additions to it like Internal Block Diagrams, Parametric Diagrams, and Requirement Diagrams [6].

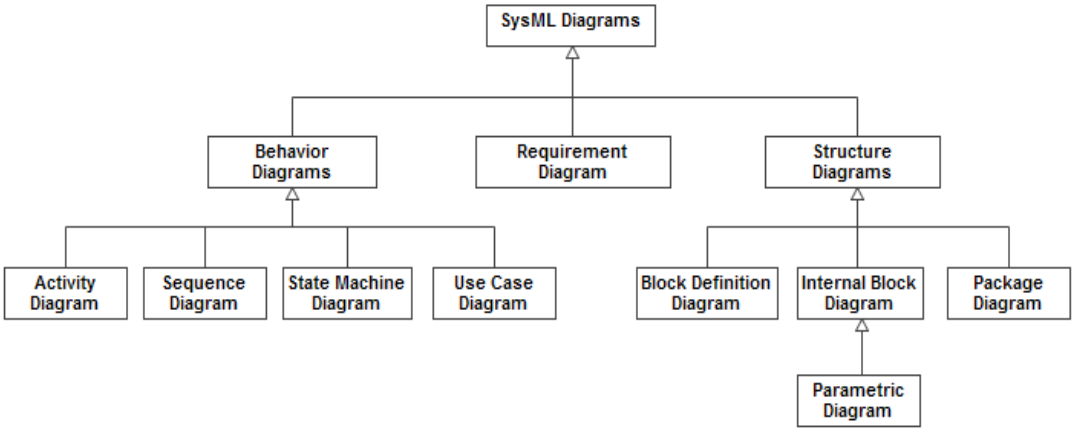

Figure 3.1: The SysML Diagram Taxonomy [7]

3.2 Dymola – Modelica Environment

Dymola is a commercial modeling and simulation tool that is based on the Modelica modeling language. It is capable of modeling integrated and complex systems from various domains like mechanical, electrical, control, and thermodynamics. It is known for its multi-engineering capabilities with compatible model libraries for many

different engineering fields. This allows for models of complete systems to be built that better represent the real world [8]. For the purpose of this thesis, Dymola was used as the modeling tool used to create the Functional Mock-up Unit (FMU) of the two-room house. FMUs will be discussed in the following section.

Modelica, the modeling language used in Dymola, is a non-proprietary, object-oriented, equation-based language to conveniently model complex physical systems [9]. The Modelica Standard library consists of over 1600 model components and 1350 functions over many domains. The Modelica version used in this thesis was 3.2.2. Since it is an open-source language, there are a large number of third party libraries that are built using Modelica.

In this thesis, the *BuildingSystems* Library v2.0.0 beta was extensively utilized for creating the model of the energy efficient house. This model will be extensively discussed in Section 5.3. The library can be found on the GitHub page at https://github.com/UdK-VPT/BuildingSystems. It was developed by a team in the Universität der Künste Berlin under the guidance of Dr. Christoph Nytsch-Geusen.

The Modelica open-source *BuildingSystems* library is developed for dynamic simulation of the energetic behavior of single rooms, buildings and whole districts [10]. Using this library, modeling a living space and its HVAC system became possible. The library also allowed for the use of renewable energy systems to be included in the model.

3.3 Functional Mockup Interface

A Functional Mock-up Unit (FMU) is a product of the Functional Mock-up Interface (FMI) standard. This standard is tool-independent that helps support model-exchange and co-simulation of dynamic models using .xml files and compiled C code [11]. The first version, FMI 1.0, was published in 2010, followed by FMI 2.0 in July 2014 [11]. For this thesis, the FMI 2.0 standard was utilized for the model exchange purpose. As mentioned in the standard documentation, the goal behind FMI for model exchange is that a modeling environment can generate C code of a dynamic system model that can be utilized by other modeling and simulation environments [12]. The model of interest is distributed in one zip file called FMU that contains several files like An XML file containing the definition of all exposed variables in the FMU and other static information; all needed model equations are provided with a small set of easy to use C functions; extraneous data is included in the FMU zip file, especially a model icon (bitmap file), documentation files, maps and tables needed by the FMU, and/or all object libraries or dynamic link libraries that are utilized [12].

3.4 Simulink/ Matlab

Simulink, like Dymola, is commercial graphical modeling and simulation environment developed by Mathworks Inc., in conjunction with Matlab. It offers a close integration with Matlab environment and can either drive Matlab or be scripted from it. Simulink is widely used in automatic control and digital signal processing for multidomain simulation and Model-Based Design [13]. Although Simulink is capable

of modeling multi-domain systems too, for large systems with several hundred components, it is very cumbersome to do so.

For the purpose of this thesis, Simulink wasn't used for the purpose of simulating the model of the energy efficient house. Rather, it was merely used as a shell or an interface that imported an FMU from the Dymola Model. This Simulink shell model was then imported in Cameo Systems Modeler to create the executable model making use of the preexisting Cameo Systems Modeler-Matlab Integration. The details of how exactly this was performed can be found in Section 5.4 and Section 5.5.

Chapter 4

Current Approaches to Creating an Executable Model

4.1 OpenModelica – SysML Integration Using OMG Specification

The first approach that was tried for creating the executable model was a Java based approach using an existing specification called the SysML – Modelica Transformation Specification Version 1.0 using No Magic Cameo Systems Modeler and OpenModelica, another Modelica based environment [14]. This method was introduced by the Object Management Group (OMG) in 2012 where the vision was to provide a bi-directional mapping between SysML and Modelica to leverage the benefits from both languages. By integrating SysML and Modelica, SysML's strength in descriptive modeling can be combined with Modelica's Differential Algebraic Equation (DAE) solving capability to support analyses and trade studies [14]. Using this approach, Cameo Systems Modeler users could use a plugin created by the Model Based Systems Engineering Center at Georgia Tech University to import and export Modelica models to SysML [15]. These plugins were named SysML4Modelica and Modelica4SysML.

This seemed to be the ideal method to convert a Modelica Model into a SysML representation of it. This would enable an Internal Block Diagram (IBD) of the system to be created. Next, having developed the structural, behavioral and parametric diagrams, of the system, they could be linked to the parameters and linkages and other data items from the imported Modelica model, the next step would be to execute this

SysML model and have it run a simulation within SysML itself. Using this functionality, a tradeoff analysis could then be performed.

So, to begin working on this method, the following steps were taken:

1. Install OpenModelicaCompiler 1.9.3.
2. Download and unzip the edu.gatech.mbsec.magicdraw.plugin.modelica2sysml.zip and edu.gatech.mbsec.magicdraw.plugin.sysml2modelica.zip files as folders.
3. Place the unzipped folders in the plugins folder of your MagicDraw installation directory (C:\Program Files\Cameo Systems Modeler\plugins).
4. Launch the MagicDraw application.
5. Download ModelicaImportTest.mdzip.
6. Download the associated SysML4Modelica profile, sysml4modelicaprofile.mdzip
7. Import a Modelica model into the MagicDraw SysML project.
    a. Go on Data (right click)->Modelica to SysML->Import Modelica. Select the Modelica .mo file.
    b. The Modelica file imported for this example was a small Spring Mass Damper as shown in Figure 4.1.

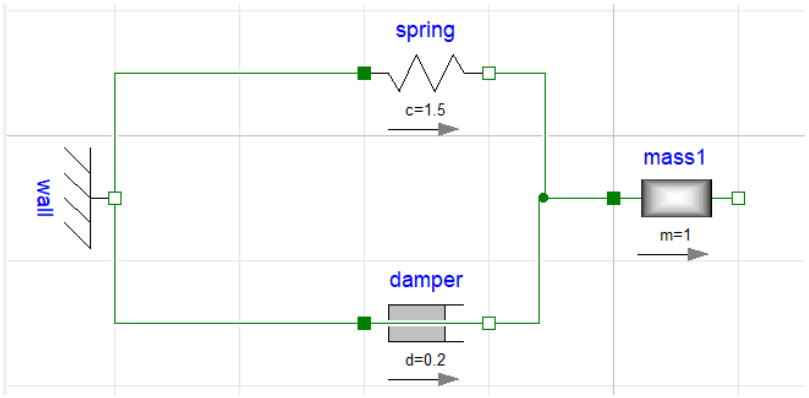

Figure 4.1: Test Modelica File of Spring Mass Damper System

At this stage, it should have been possible to automatically create the IBD. This could theoretically be done by selecting the Modelica class which contains connectors and then right click *New Diagram* and select a *SysML Internal Block Diagram*. Some manual refactoring would most likely still be necessary to make it look nice.

However, this last step was not possible, as when the Modelica file was imported, instead of separating itself into its hierarchical structure, only a single block, *sprdmp* was created as shown in Figure 4.2.

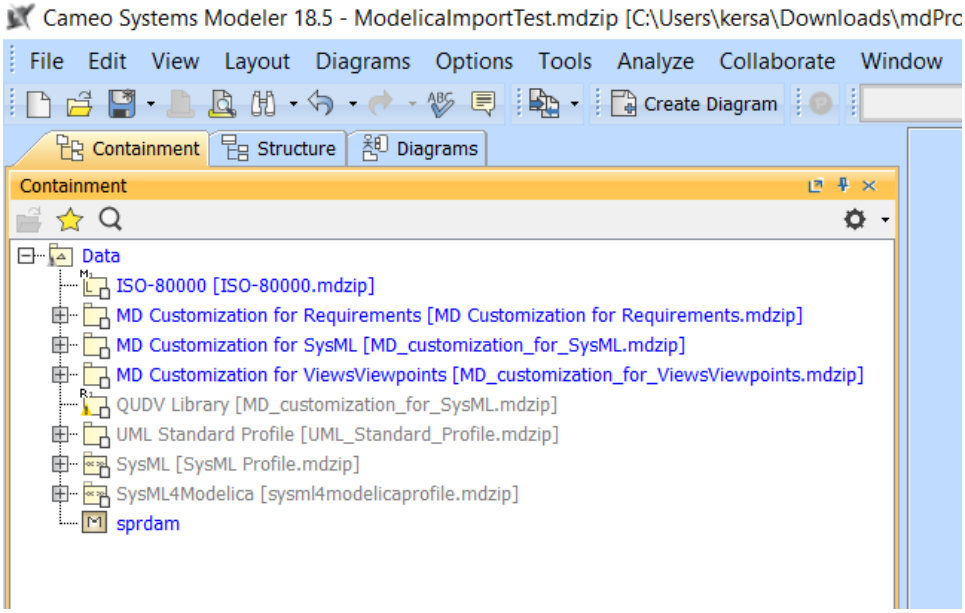

Figure 4.2: Containment Tree from ModelicaImportTest.mdzip

On opening the block and reading its specification, there was no indication of any other information from the Modelica model or any of its constituent blocks. There was also no mention of any ports, paths, values or properties listed in the specifications. In fact, there was no information at all and the specification was completely blank. In addition, there was no information about any of the equation, constraints or variables related to the Spring Mass Damper System. There was also an error pop-up that showed up when the Modelica Model was imported. This is shown in Figure 4.3. The unhandled errors were Java related as is visible from the figure. However, due to a lack of experience with Java, not much of an effort could be made in debugging the program and solving the problem.

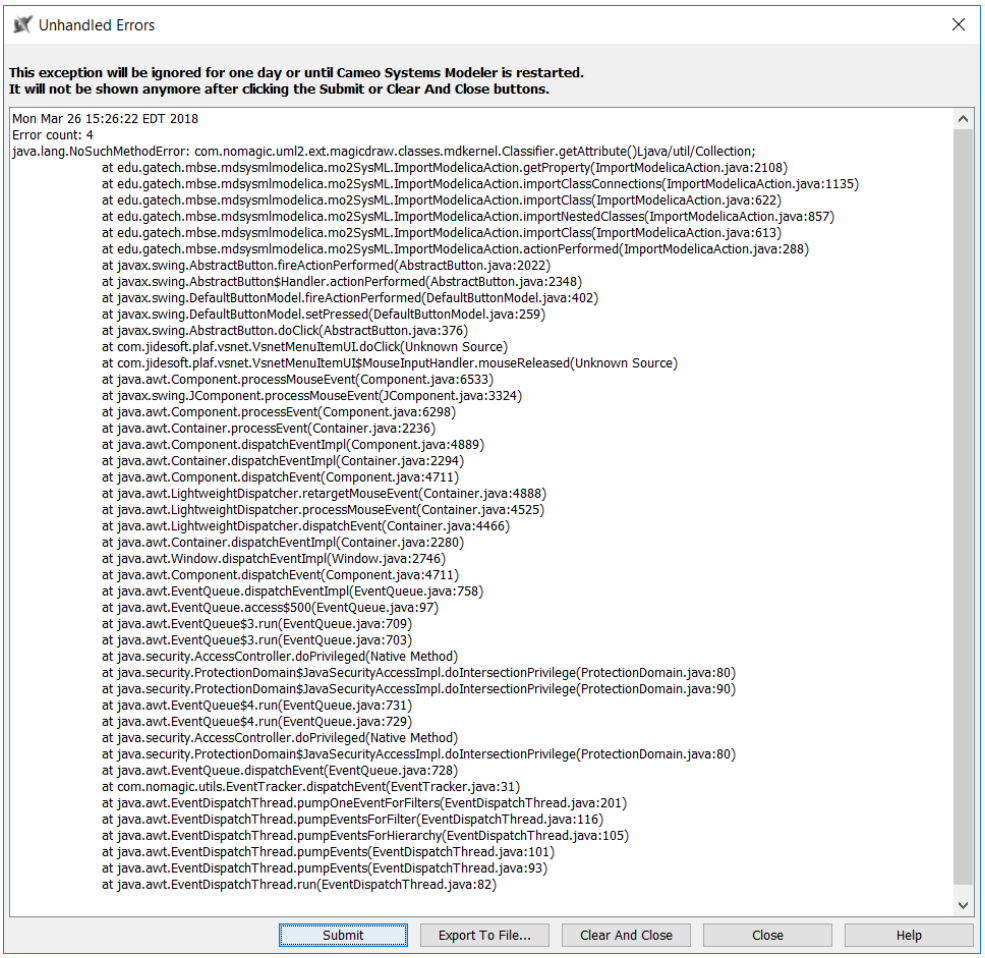

Figure 4.3: Error Window Displaying Unhandled Java Error

The reason for the failure of this method was speculated to be that the No Magic profile and the related plugins were developed sometime in 2012 with much older versions of Cameo Systems Modeler and OpenModelica. However, these examples were run using Cameo Systems Modeler v18.5 sp3. Since, previous versions of either of these software were not easily accessible with the requisite licensing, no further attempt was made to recreate this example and use it for the purpose of this thesis.

4.2 FMU Import into Cameo Systems Modeler Itself

The latest No Magic Cameo Systems Modeler beta version has crude support for importing FMUs using the older FMI 1.0 standard for Co-Simulation [16]. The Cameo Simulation Toolkit in Cameo Systems Modeler is capable of reading FMU files which are imported into the model in the form of FMU Blocks with the stereotype of ≪FMU≫. However, this functionality doesn't allow for FMUs to be imported into the system for the Purpose of Model-Exchange.

Also, for the purpose of the co-simulations, it was found that importing the FMU into Cameo didn't allow for the change in the input parameters that resided in the FMU block, making it practically useless for the purpose of tradeoff analyses. The restriction of the design space exploration ruled out this method for creating an executable SysML model. However, in the future, it is very likely that No Magic will update the Cameo Systems Modeler to be able to read FMU using the later FMI 2.0 standard for the purpose of model exchange as well as co-simulation. If this thesis work should be recreated two more years from now, following this method could be a much better and more robust solution.

Chapter 5

Executable Model Using SysML-Simulink-FMI Integration

5.1 Overview and High-Level Description of the Integration Procedure

In this section, the integration procedure will be defined.

Cameo Systems Modeler is used to create systems architecture of the two-room house in SysML. Dymola is used to create a multi-domain Modelica model of the same two-room energy efficient house. This Modelica model contains all the internal equations, constraints and relations that govern the two-room house. It is also capable of accepting user defined input values to the design parameters of the systems, performing the calculations, and producing the output values for the system metrics. This Modelica model is then exported as FMU as shown in Figure 5.1. The FMU is just a "skeletal structure" of the Modelica model and needs to be run from a different modeling tool to be able to accept input values and then output the corresponding metric values.

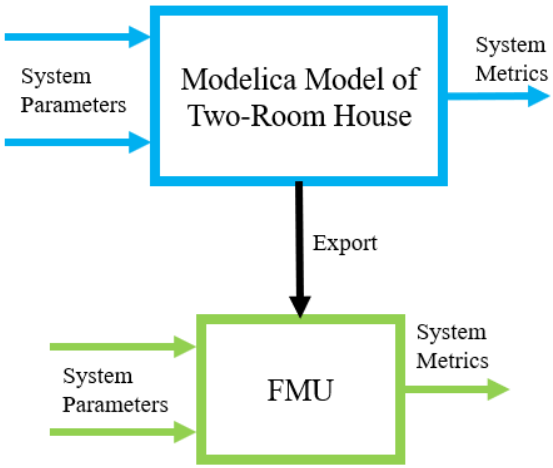

Figure 5.1: Export of Modelica Model into the FMU

Next, The FMU is imported into Simulink as shown in Figure 5.2. In this case, Simulink doesn't add anything new to the model, but merely acts as a shell or an interface to the FMU. Simulink was chosen as the interface since Cameo Systems Modeler and Matlab/Simulink have an existing integration that could be exploited for this usage.

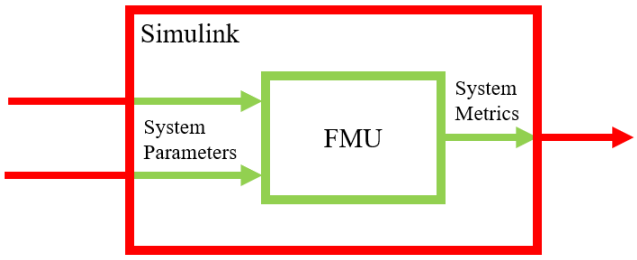

Figure 5.2: Simulink as in Interface between the FMU and the SysML Model

Now, through the SysML Model of the two-room house, this Simulink model is called. User-defined design parameter values that were inputted into the SysML model are now sent to Simulink. With these values, Simulink will run the FMU, calculate the values of the output metrics and send them back to SysML to be displayed back to the user. This is shown in Figure 5.3.

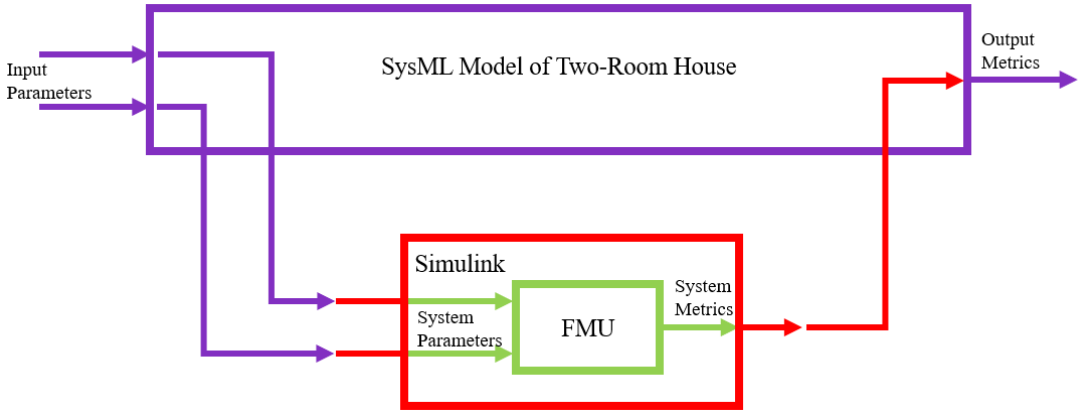

Figure 5.3: Interaction between the SysML Model and Simulink Model

5.2 SysML Model

This section describes the SysML model of the two-room energy efficient house. The first step to creating the executable model is to build the SysML architecture of the system. A few components of the architecture of a two-room house will be covered in the following sections.

5.2.1 Structure Diagrams

The first structure diagram created was the System Domain Block Definition Diagram (BDD) as is shown in Figure 5.4. This BDD articulates the structure of the system's domain, the system itself and its constituent elements. From the diagram it can be seen that the domain of the system comprises of the *System*, the *Users* and the *Environment*. The *System* itself is further comprised of the *Heating* system, the *Indoor Environment* and the *Building Structure* itself. Each of these subsystems are also broken down into their constituents. It is important to note that the *Building Structure* components are built using certain materials which are also shown as blocks in the Domain BDD. The operations of each component of the system as well as the values (variables) associated with each of the blocks should also be shown in a BDD. However, for the purpose of readability of the diagram, these details have been suppressed.

Next, the *Environment* is also broken down into its subsystems namely, the *Domestic Cold-Water Supply*, the *External Environment*, and the *Electricity Provider*. Finally, the *Users* of the system are also shown in their own block.

The hierarchical structure of the *Heat Pump System* is shown in Figure 5.5. This BDD also shows the operations and values associated with each of the constituent blocks of the *Heat Pump System*. Similarly, the *Solar Thermal System* and all its associated constituents are displayed in Figure 5.6 below. The hierarchical structure of the *Building Structure*, all its components and constituent building materials are shown in Figure 5.7

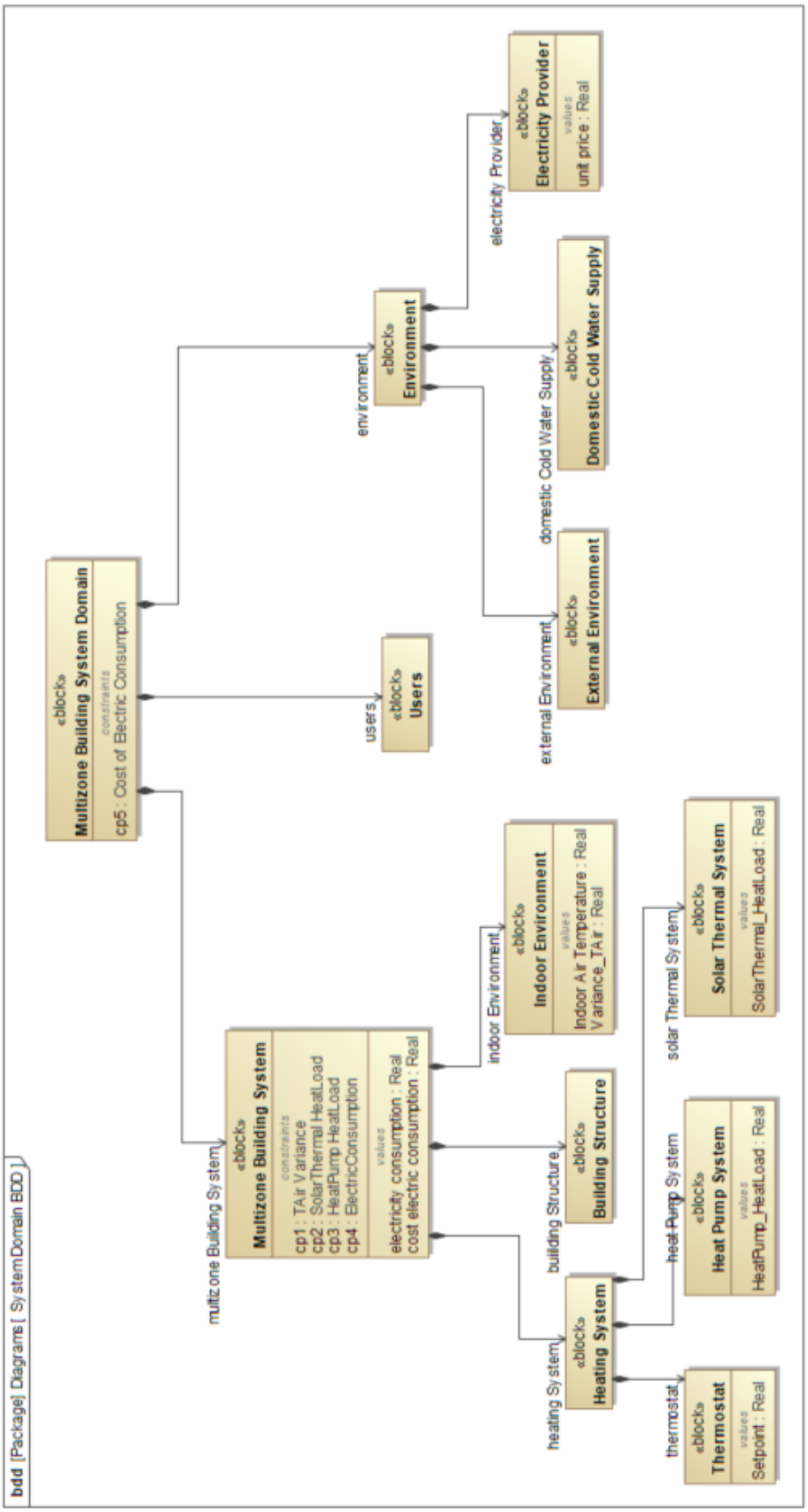

Figure 5.4: System Domain Block Definition Diagram

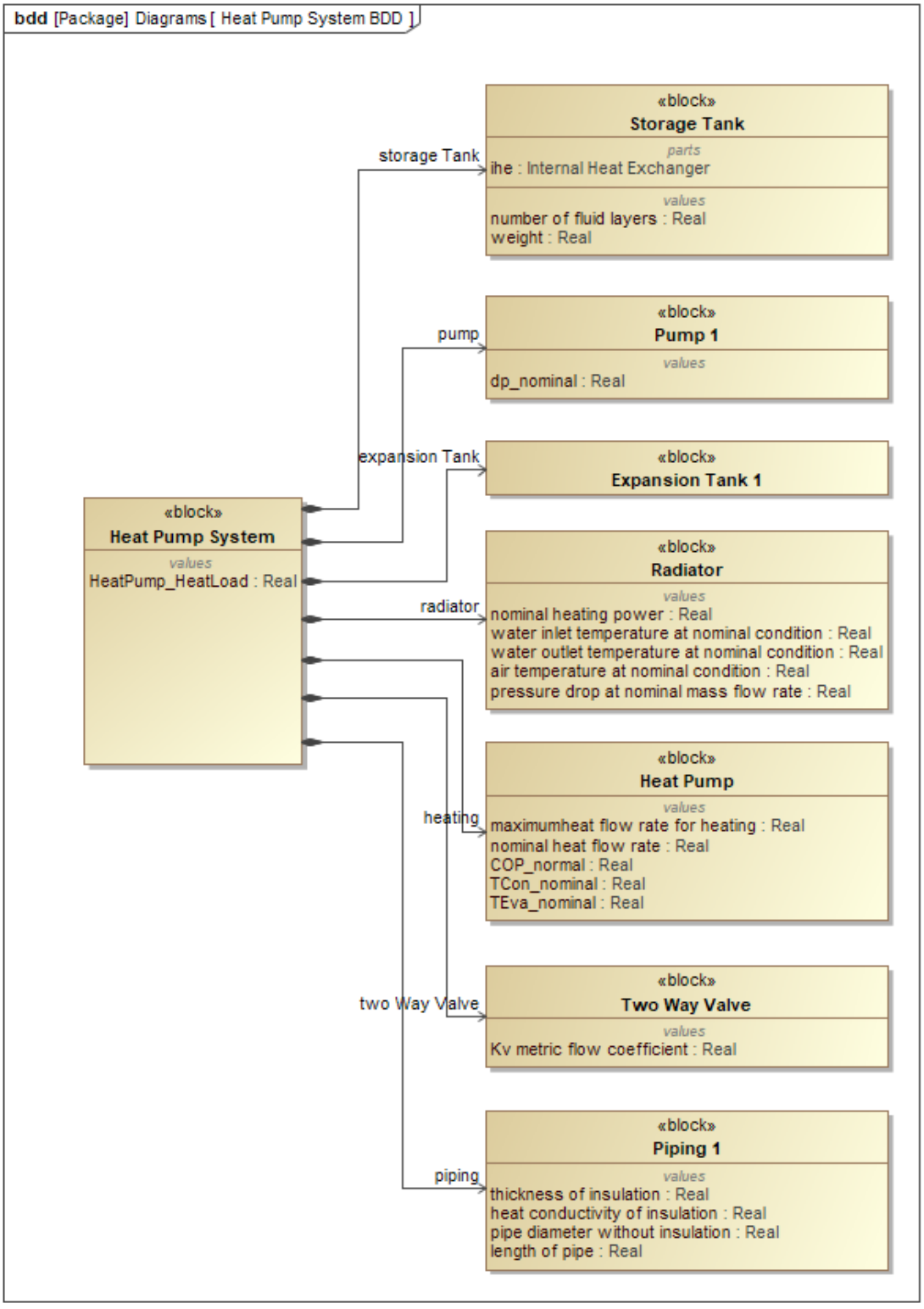

Figure 5.5: Heat Pump System Block Definition Diagram

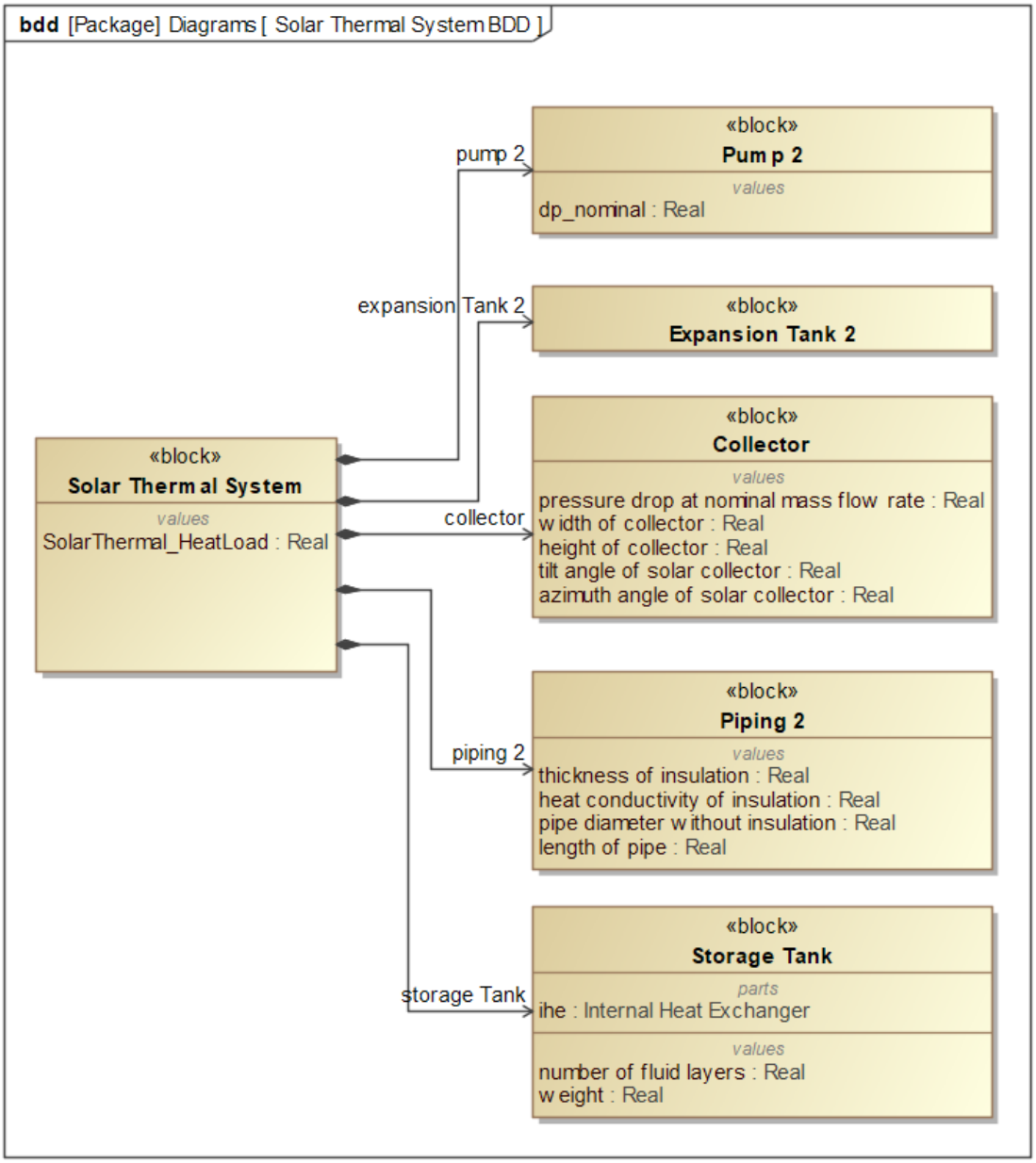

Figure 5.6: Solar Thermal System Block Definition Diagram

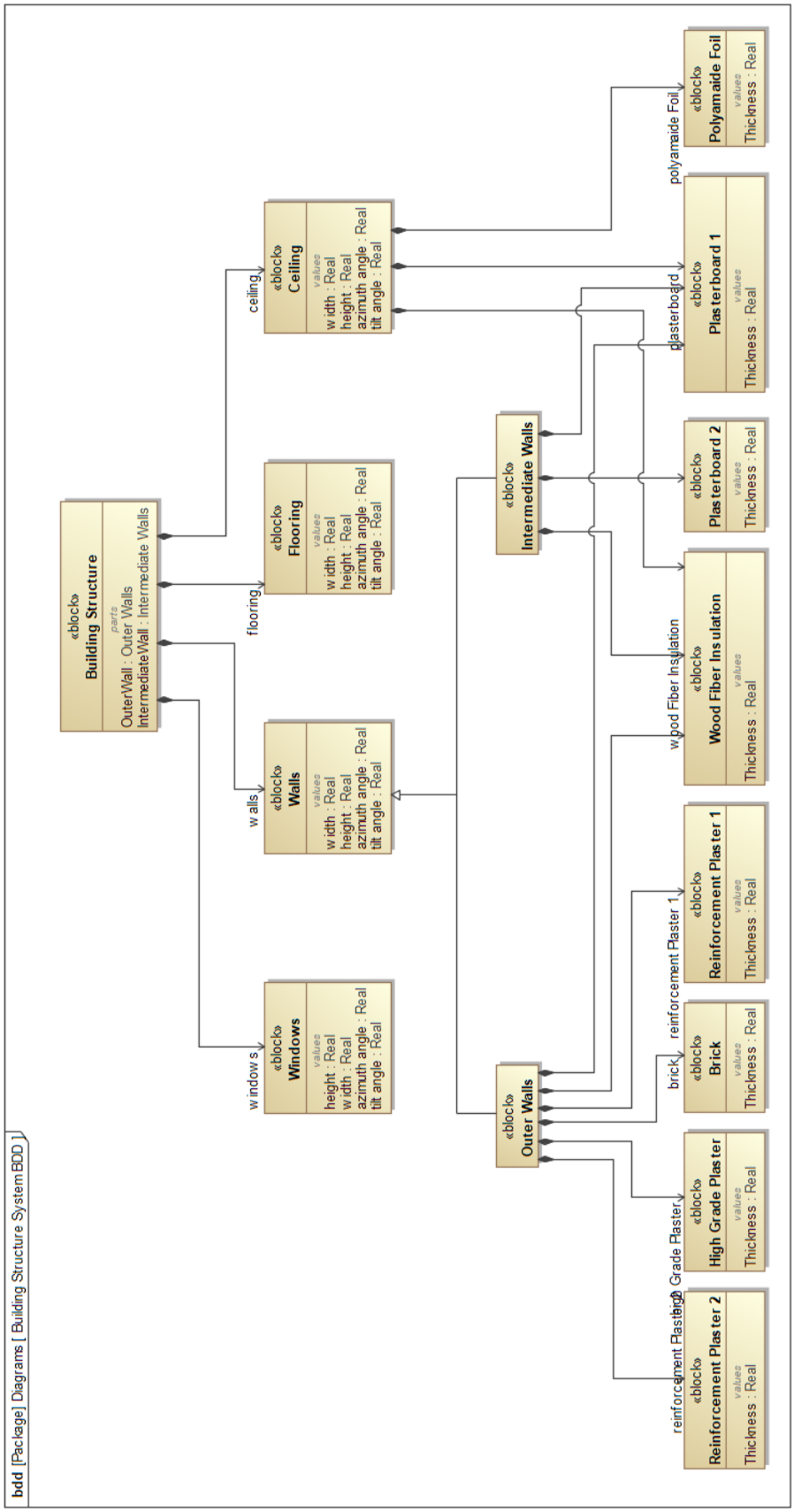

Figure 5.7: Building Structure System Block Definition Diagram

The next structure diagram, the Context Level Internal Block Diagram (IBD), shown in Figure 5.8, describes the internal structure of a single block, the *Multizone Building System Domain*. Since the *Multizone Building System Domain* comprises of the *Multizone Building System* itself and the *Environment*, this diagram also describes the connection between the system itself and the environment. The context level IBD also displays the interfaces and the various flows (energy and data) across connections among different parts and properties that form internal structure of the domain block. From the Context level IBD we see that the *Environment* and its constituent blocks supply electricity, ambient air temperature data, radiation and cold water to the system. The associated External Interface Diagram can be found in the Appendix in Figure A.1.

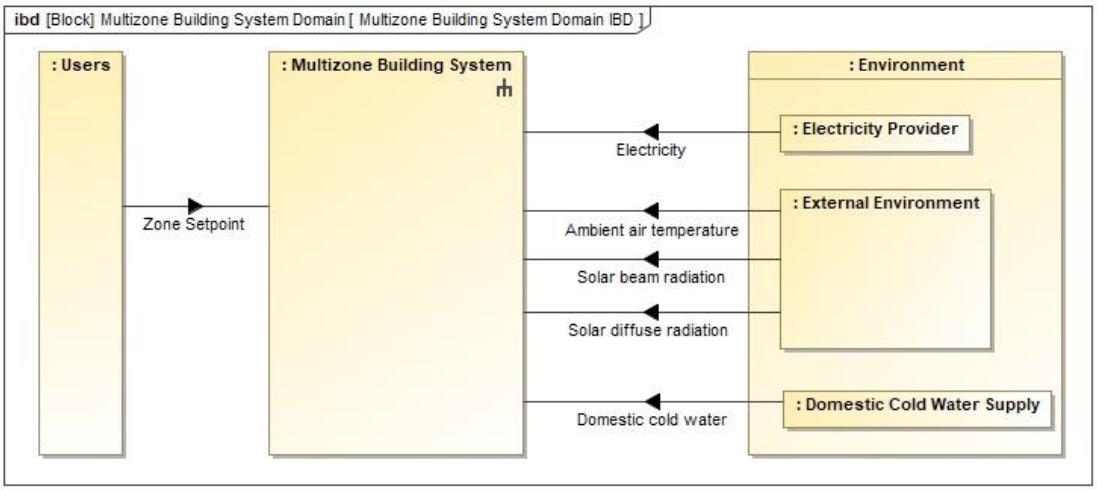

Figure 5.8: Context Level IBD

The System Level IBD shown in Figure 5.9 shows further detail of the interfaces and flows that are present within the system itself. The inputs into the system are solar irradiation, external air, electricity, ambient air temperature, and the user-set zone setpoint. The outputs of the system are exhaust air, the zone temperatures within the building, and the operative temperature within the building. As can be seen, all of the

inputs except the *Ambient Air Temperature* data feed into the *Heating/ Cooling System* which then, in turn, provides heat to the *Indoor Environment* as outputs and additionally, also outputs exhaust air. The *Indoor Environment* receives the *Ambient Air Temperature* data as well as heat from the *Heating/ Cooling System* and outputs the current *Zone Temperature* and *Zone Operative Temperature* to the system.

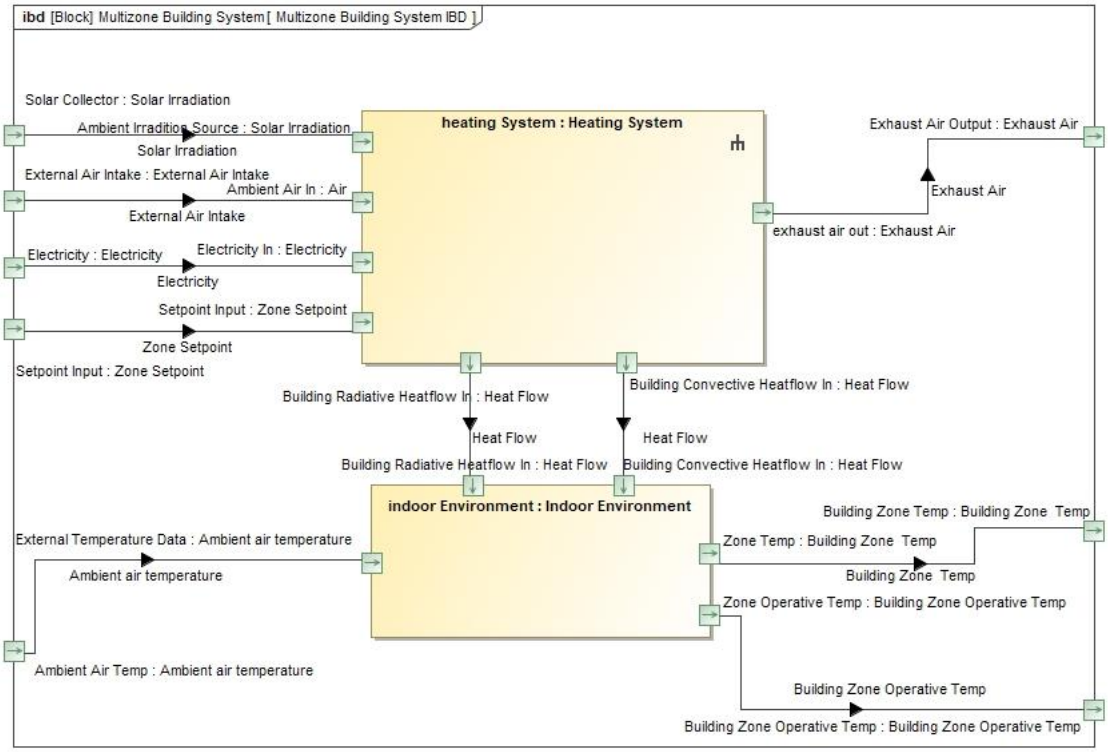

Figure 5.9: System Level IBD

In addition to the System Level IBD, a more detailed IBD of just the Heating Subsystem is shown in Figure 5.10. Since the process of heating a house and maintaining the specified *Zone Temperature* is a non-sequential and continuous process, this IBD serves better than an activity diagram would. This is because activity diagram is extremely useful to show a concrete chain of events that take place in a very specific order. However, for this system since most of the processes are taking place

concurrently, the system behavior is better described, showing in detail, the flow of information to and from each element contained within the system.

The IBD below in Figure 5.10 describes the Heating System in which the *Zone Temperature* is supplied to the *Thermostat*, which in turn on comparing the *Zone Temperature* and *Zone Setpoint*, provides a Boolean actuator signal to the *Two-Way Valve*. The *Two-Way Valve* in turn regulate the amount of fluid flow through to the *Radiator*. The *Radiator* is supplied with warm water from two thermal loops. One is the *Solar Thermal System* which is the main source of thermal energy to heat the house. The other loop is the backup *Heat Pump* loop that is only used to support the *Solar Thermal System* on days when there isn't enough solar thermal energy being provided to the house. The associated Internal Interface Diagram can be found in the Appendix in Figure A.2.

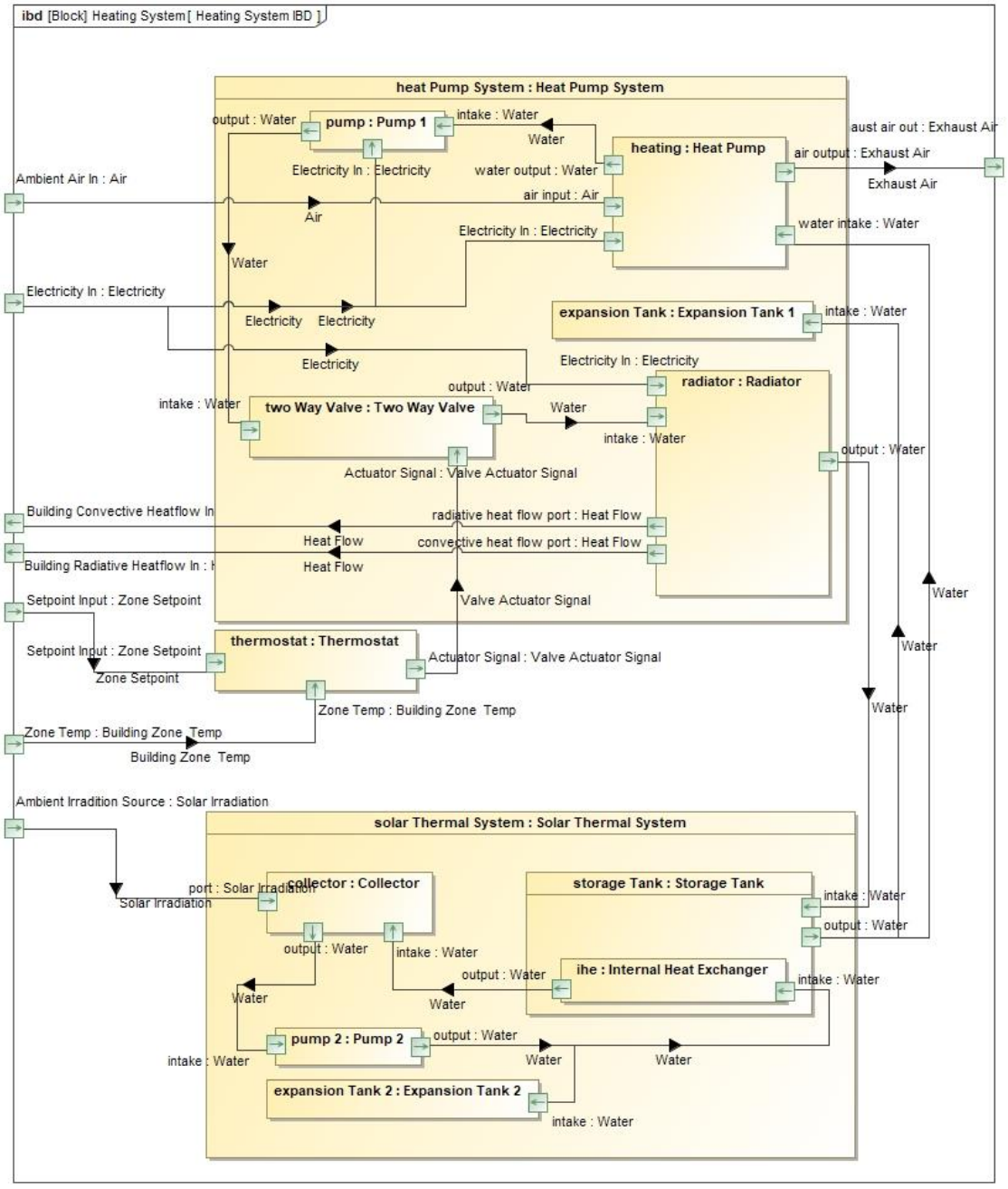

Figure 5.10: IBD of Heating System

5.2.2 Parametric Diagram

The diagram most important to creating the executable model is the SysML Parametric diagram. This diagram is used to express information about a system's

constraints [17]. The constraints are in the form of mathematical models that determine the set of valid values within the running system.

This Parametric Diagram allows the previously described Simulink model to be treated as a “black box” constraint within the Cameo Systems Modeler SysML Model. This is achieved by using a special plugin called ParaMagic, developed by InterCAX Inc.

To create this parametric diagram, first all the factors and metrics have to be identified. For this model of the two-room energy efficient house, the metrics are shown in Table 5.1

S No.	System Metric	Variable Name	Units
1.	Variance of Indoor Air Temperature	`Var_TAir`	K^2
2.	Heat Pump Heat-load Per Annum	`HeatPump_HeatLoad`	kWh
3.	Solar-thermal Heat-load Per Annum	`SolarThermal_HeatLoad`	kWh
4.	Cost of Electricity consumed Per Annum	`Cost_ElectricConsumption`	$
5.	Cost of Thermal Insulation	`Cost_Insulation`	$

Table 5.1: Table of System Metrics, Variable Names and Units

Although there are several different factors that affect the indoor air temperature, only few factors were selected as the design parameters. These are the parameters that can be realistically altered in order to improve the values of the metrics. Each design parameter, its associated SysML variable name and its nominal value can be found in

Table 5.2. The design parameters chosen were the thickness of the wood-fiber insulation layers used in the construction of the outer wall, intermediate wall, and the ceiling, the surface area of windows, and the indoor temperature setpoint.

S. No.	Design Parameter	Variable Name	Nominal Value
1.	Outer wall Wood fiber insulation thickness	`OW_WoodFibIns`	0.015 m
2.	Intermediate wall Wood fiber insulation thickness	`IW_WoodFibIns`	0.1 m
3.	Ceiling Wood fiber insulation thickness	`C_WoodFibIns`	0.255 m
4.	Window Area	`Window_height`	3.0 m^2
5.	Indoor Temperature Setpoint	`Temp_Setpoint`	20 °C

Table 5.2: Table of Design Parameters, Variable Names and Nominal Values

In addition to the design parameters, a number of design constants were also chosen. Setting certain values as design constants allowed for those parameters to be held constant for the purpose of the tradeoff analysis performed in this thesis work but at the same time it allows for those values to be changed in future works if required.

The design constants chosen were the thicknesses of the other materials that make up the outer-wall, the intermediate wall and the ceiling of the house. Each design constant, its associated SysML variable name and its nominal value can be found in Table 5.3.

S No.		Design Constant	Variable Name	Nominal Value (m)
1.	Outer wall materials	Plasterboard thickness	OW_Plasterboard	0.01
2.		Brick thickness	OW_Brick	0.24
3.		High grade plaster thickness	OW_HGplaster	0.02
4.		Reinforcement plaster thickness	OW_ReinfPlaster1	0.005
5.		Reinforcement plaster thickness	OW_ReinfPlaster2	0.005
6.	Intermediate wall materials	Plasterboard thickness	IW_Plasterboard1	0.015
7.		Plasterboard thickness	IW_Plasterboard2	0.015
8.	Ceiling Materials	Plasterboard thickness	C_Plasterboard	0.0125
9.		Polyamide foil thickness	C_PolyamideFoil	0.0005

Table 5.3 Table of Design Constants, Variable Names and Nominal Values

Having identified the factors, constants and metrics, a constraint block had to be first created in Cameo Systems Modeler [18]. This constraint was labeled *TAir Variance* and its specification window is shown in Figure 5.11. Next all the parameters as shown in the above table were inputted into the Constraint Block. The constraint inputted into the block was:

```
Variance_TAir=xfwExternal(matlab,scriptascii,
exec_script,
```

```
OW_Plasterboard,
OW_Brick,
OW_HGplaster,
OW_ReinfPlaster1,
OW_WoodFibIns,
OW_ReinfPlaster2,
IW_Plasterboard1,
IW_WoodFibIns,
IW_Plasterboard2,
C_Plasterboard,
C_PolyamideFoil,
C_WoodFibIns,
Window_width,
Window_height,
Temp_Setpoint)
```

The `xfwExternal` function is used to call an external solver into parametric diagrams. The variable on the left side of the equal-to sign is the metric being solved for. The first three arguments of the `xfwExternal` function describe the solver being called, here Matlab; the type of element called, here an ascii script; and the name of the function or script [18]. The remaining arguments of the function are the design constants and design parameters that will affect the metric. The Matlab script named `exec_script.m` being called by this function will be discussed in a later section. It can be found in Section A.2 of the Appendix Section.

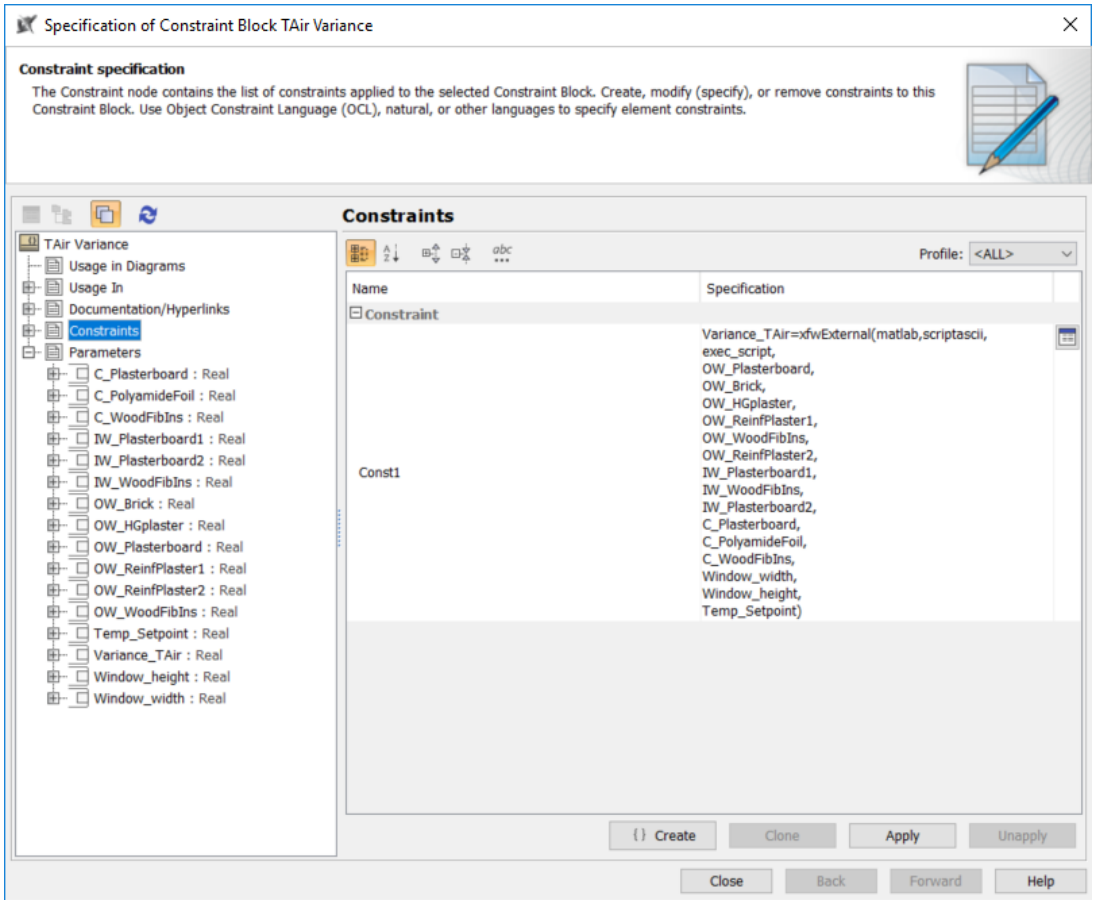

Figure 5.11: Constraint and Parameters in the Constraint1 Specification Window

Following this, the parameters in the Constraint Block were linked to the value properties of each block from the System Block Definition Diagram using the Parametric Equation Wizard. This is demonstrated in Figure 5.12. This was one of the most crucial steps in creating the Parametric Diagram.

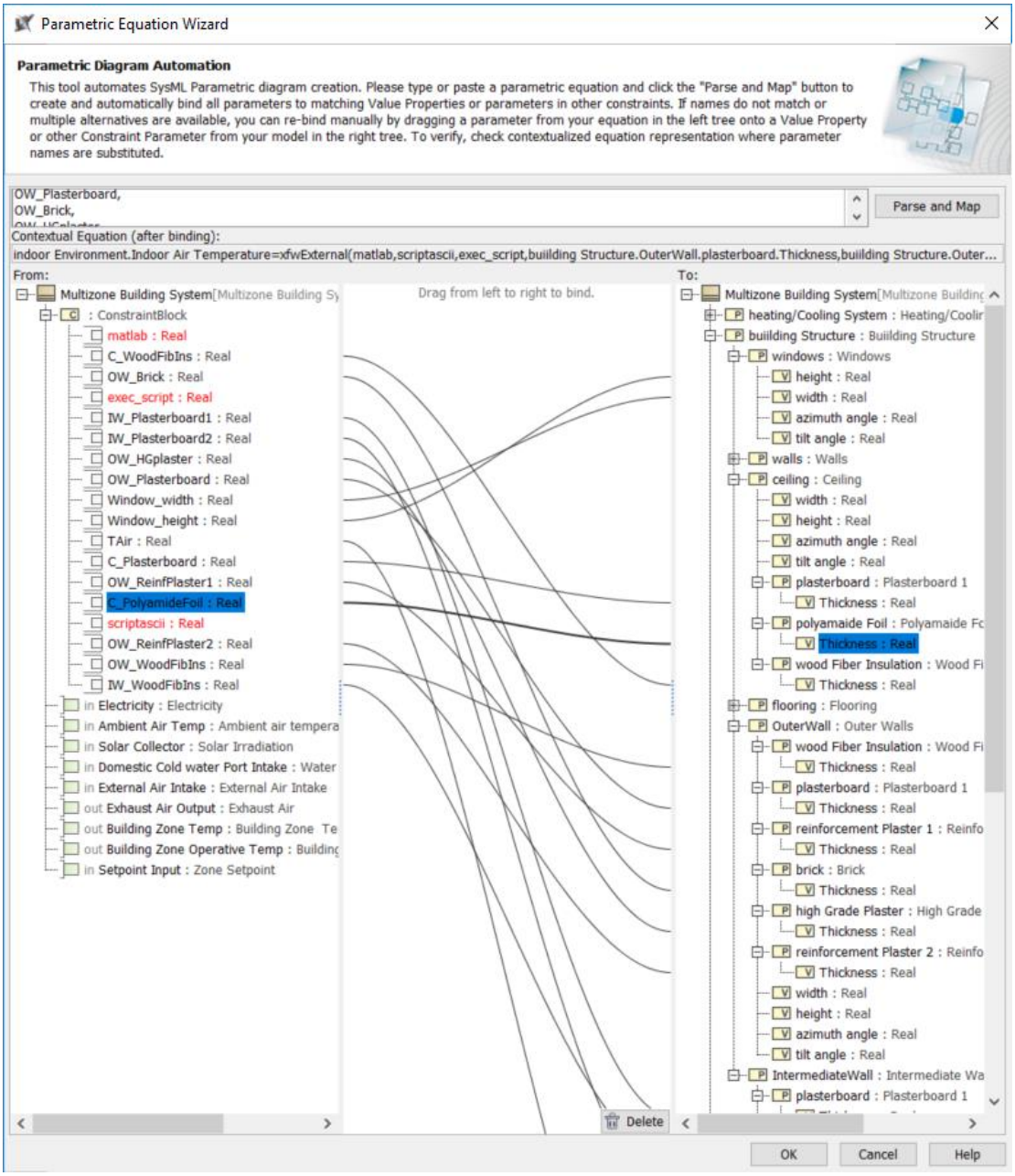

Figure 5.12: Linking Constraint Block parameter to Block values using the Parametric Equation Wizard.

The same process is carried out for the next three Constraint Blocks named *ElectricConsumption*, *HeatPump HeatLoad*, and *SolarThermal HeatLoad*. As the names suggest, each of these constraints compute the electricity consumed by the

system, the total yearly heat load provided by the heat pump, and the total yearly heat load provided by the solar thermal system. All these constraints were then added to the main Parametric Diagram under the main System Block. This diagram is shown in Figure 5.13. The reason each constraint had to be added into its own constraint block is a limitation of the ParaMagic plugin that was used to run the simulation of the Modelica model. Also, the reason the metric from the first constraint block, *Variance_TAir,* is fed into the remaining constraints is a work-around that had to be implemented for each of the constraints to be solved in the order of left to right as shown in the parametric diagram. It should also be noted that the first constraint block would run the main simulation and take the longest time to solve. Every subsequent constraint block calls a separate Matlab script that loads the data obtained from solving the first constraint, processes it as required and presents the corresponding metric values. The aforementioned Matlab scripts can be found in Section A.2 of the Appendix Section.

For readability purposes, Figure 5.13 has been cropped, showing the connection between the various system value properties and the constraint parameters in the constraint block. This cropped version is shown in Figure 5.14. The rest of the cropped pieces of the diagram can be found in Section A.1 of the Appendix Section.

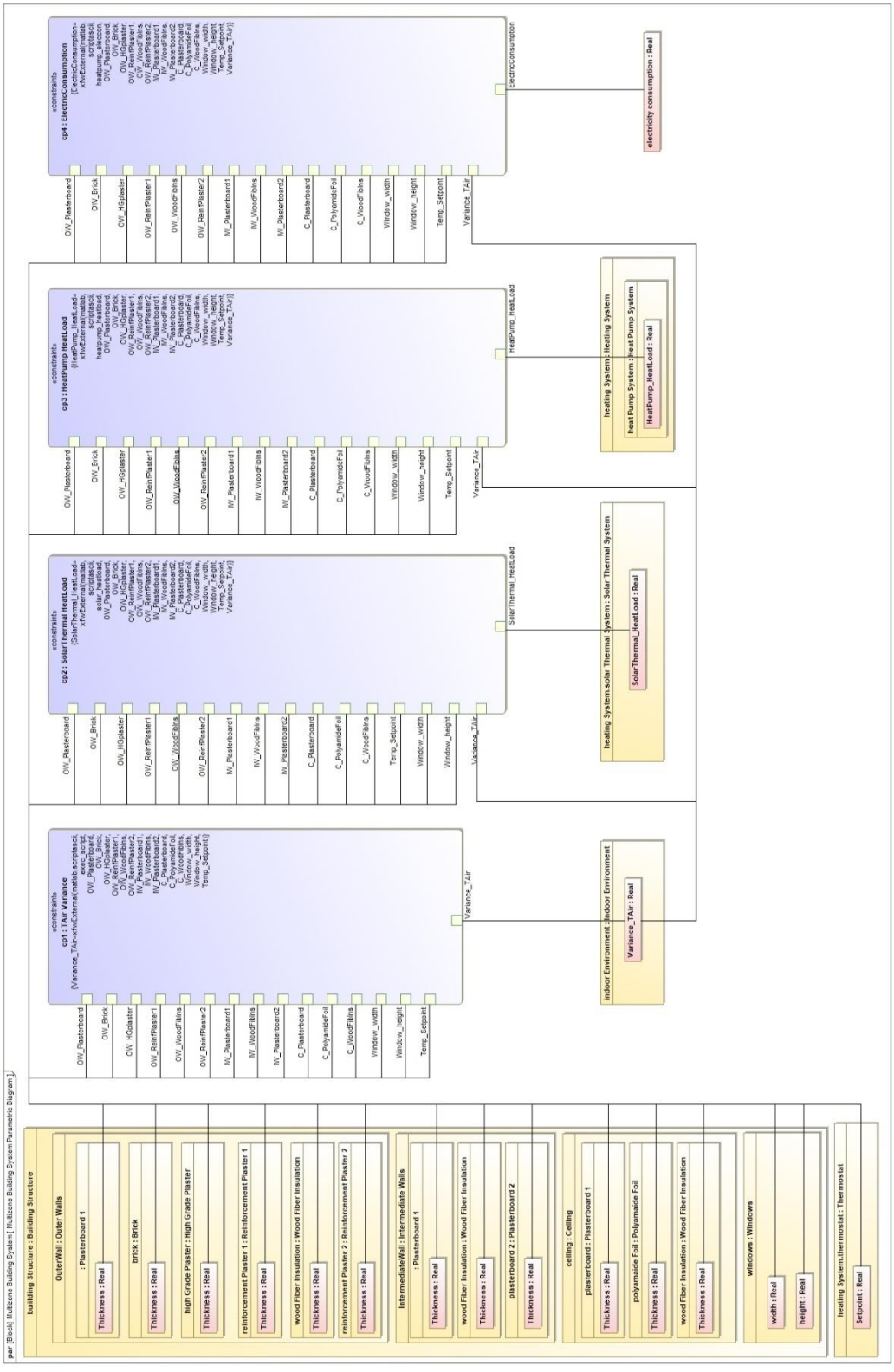

Figure 5.13: SysML Parametric Diagram showing the relationship between system values, constraints and metric values

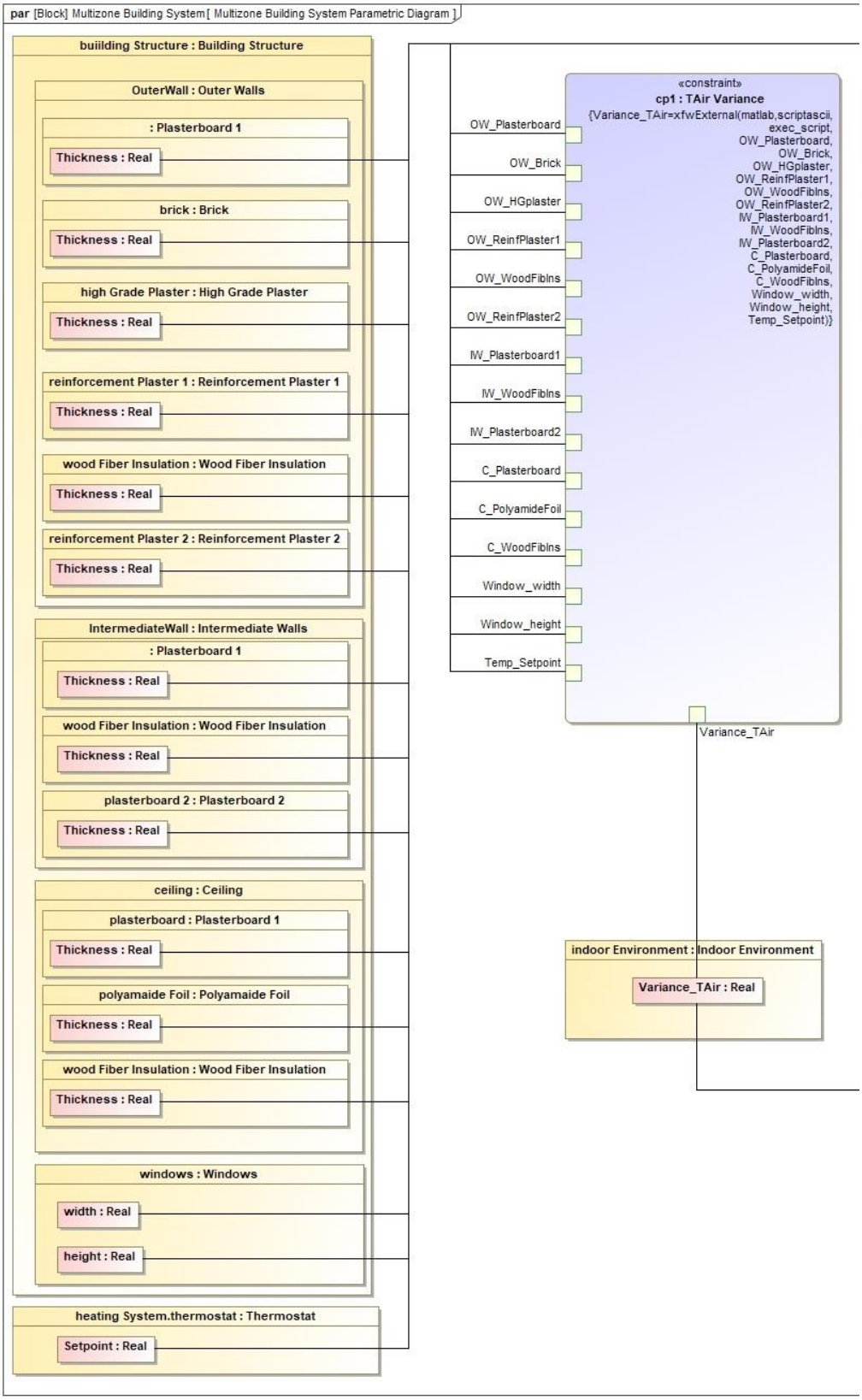

Figure 5.14: SysML Parametric Diagram - Cropped

In addition to this parametric diagram, another parametric diagram was created at the domain level to show the relationship between the cost of electricity consumed by the system, the amount consumed and the unit price for electricity. This is demonstrated in Figure 5.15. This concludes the description of the SysML Model of the system.

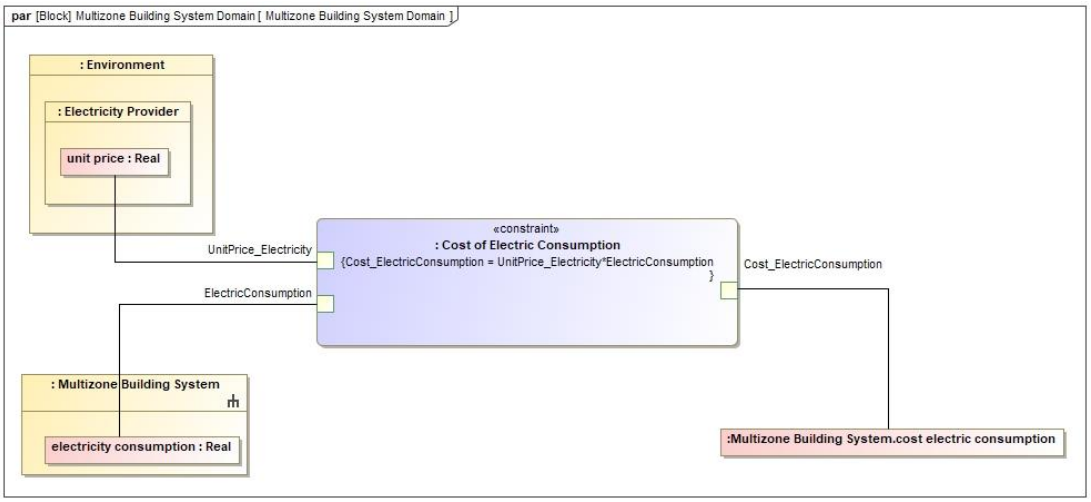

Figure 5.15: SysML Parametric Diagram Describing the Cost of Electricity Consumed by the System.

5.3 Dymola Model of Two-Room Energy Efficient House

5.3.1 Overview of Dymola Model

The Dymola model of the two-room energy efficient house is shown below in Figure 5.16. This Dymola Model contains all the equations, constraints and relations pertaining to calculating the values of the system metrics for a given set of input parameter values. This main model is named *SystemModel*.

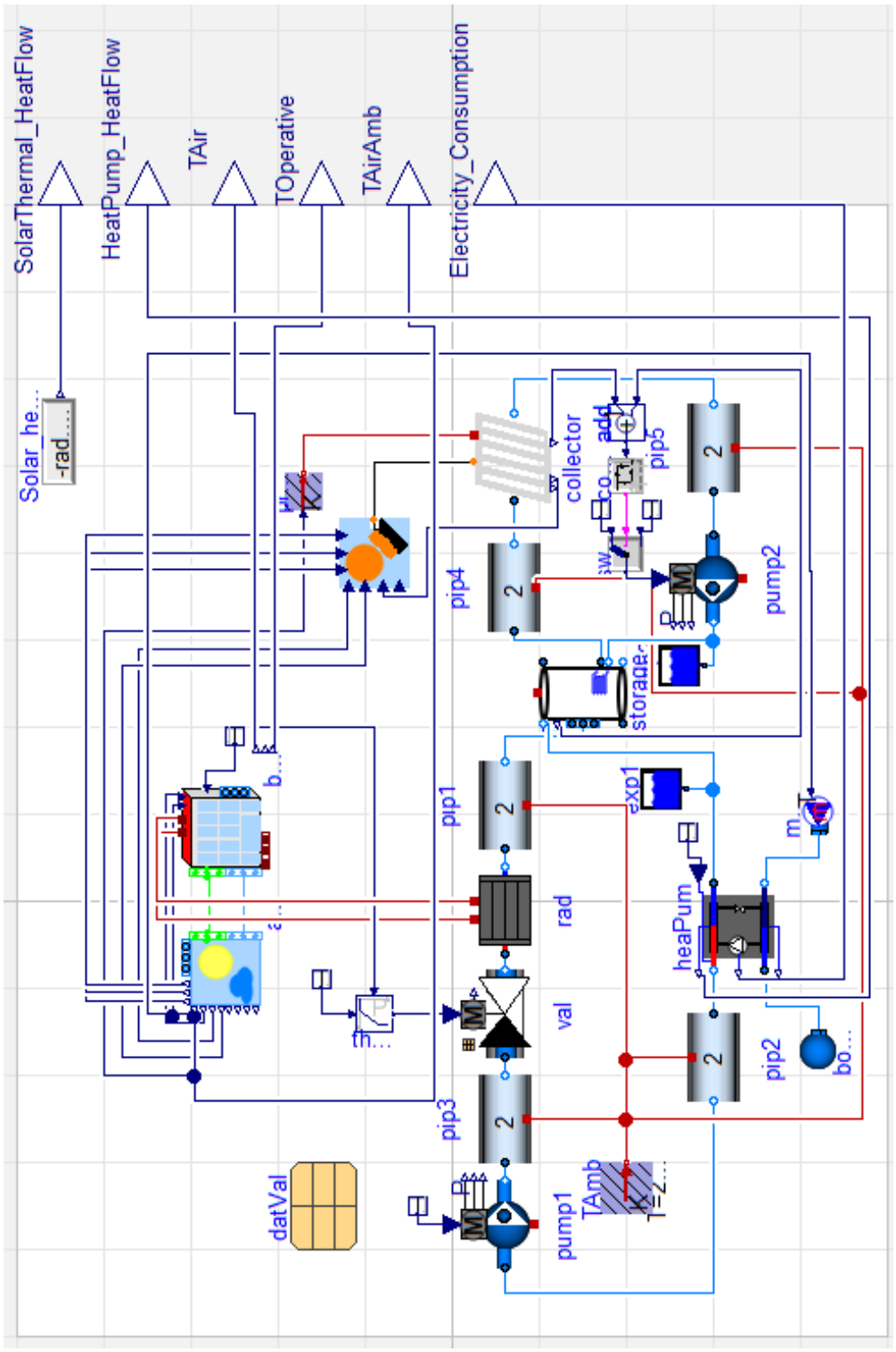

Figure 5.16: Dymola model - SystemModel

This model can be broadly categorized into 3 separate component clusters. These are the, *Building* model, the *Ambient* model and the *House-Heating* model. These components are highlighted in Figure 5.17. The *Building* block for this thesis was developed by the author using the *BuildingTemplate* template model class that can be found in the *BuildingSystems* Library at the path *BuildingSystems.Buildings. BaseClasses.BuildingTemplate*. The description of the process for creating the *Building* block can be found in Section 5.3.2. The *Ambient* block was instantiated by the author from *BuildingSystems.Buildings.Ambient* and configured as described in Section 5.3.3. Finally, the *Home-Heating* cluster of blocks was adapted from an existing example in the *BuildingSystems* Library from the path *BuildingSystems.Applications. HeatingSystems.SolarHeatingSystem*. This example can be found by downloading the *BuildingSystems* library from the GitHub page at https://github.com/UdK-VPT/BuildingSystems and loading it into the Dymola environment and then navigating to the aforementioned path. The details of the changes made to the implantation of this example and its usage in the Dymola model of the system can be found in Section 5.3.4.

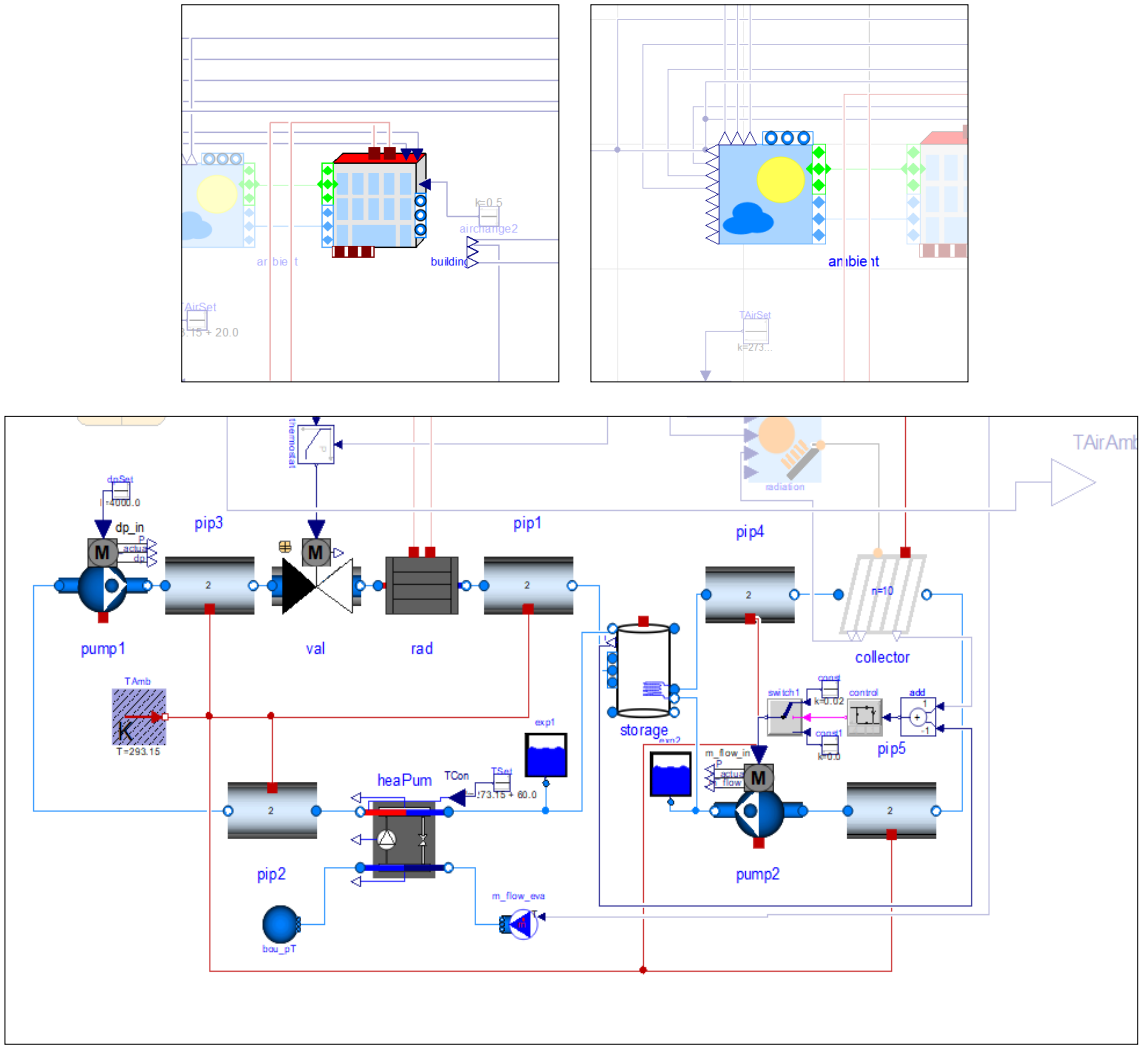

Figure 5.17: The *Building* block (top left), the *Ambient* block (top right), the *House-Heating* block cluster (bottom)

5.3.2 The *Building* Block

The *Building* block in the Dymola model was modeled as a simple cuboidal geometry separated into two symmetrical rooms with a length of 5 m, a depth of 5 m and a height of 3 m. Both these rooms are arranged side by side separated by an intermediate wall, thereby making the dimension of the entire house 10 m × 5 m × 3

m. with an internal air volume of 150 m^3. A 3D mockup of the building is shown in Figure 5.18 below for illustrative purposes.

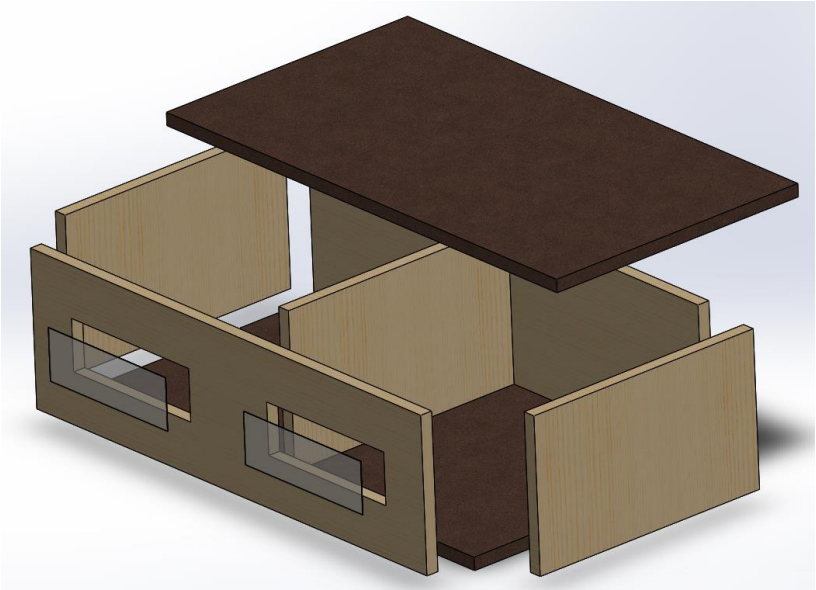

Figure 5.18 3D Structure of the Building

The *building* block contains two thermal zones blocks. One for each of the two rooms. These thermal zones were instantiated into the *building* model by dragging and dropping two component models from the class *BuildingSystems.Buildings. Zones.ZoneTemplateAirvolumeMixed*. Following this the construction elements like walls, ceiling and the floors were also created in the model by using the component from the class *BuildingSystems.Buildings.Constructions.Walls. WallThermal1DNodes.* The windows were added into the model from the class *BuildingSystems.Buildings. Constructions.Windows.Window*. The internal structure of the *building* block is shown in Figure 5.19.

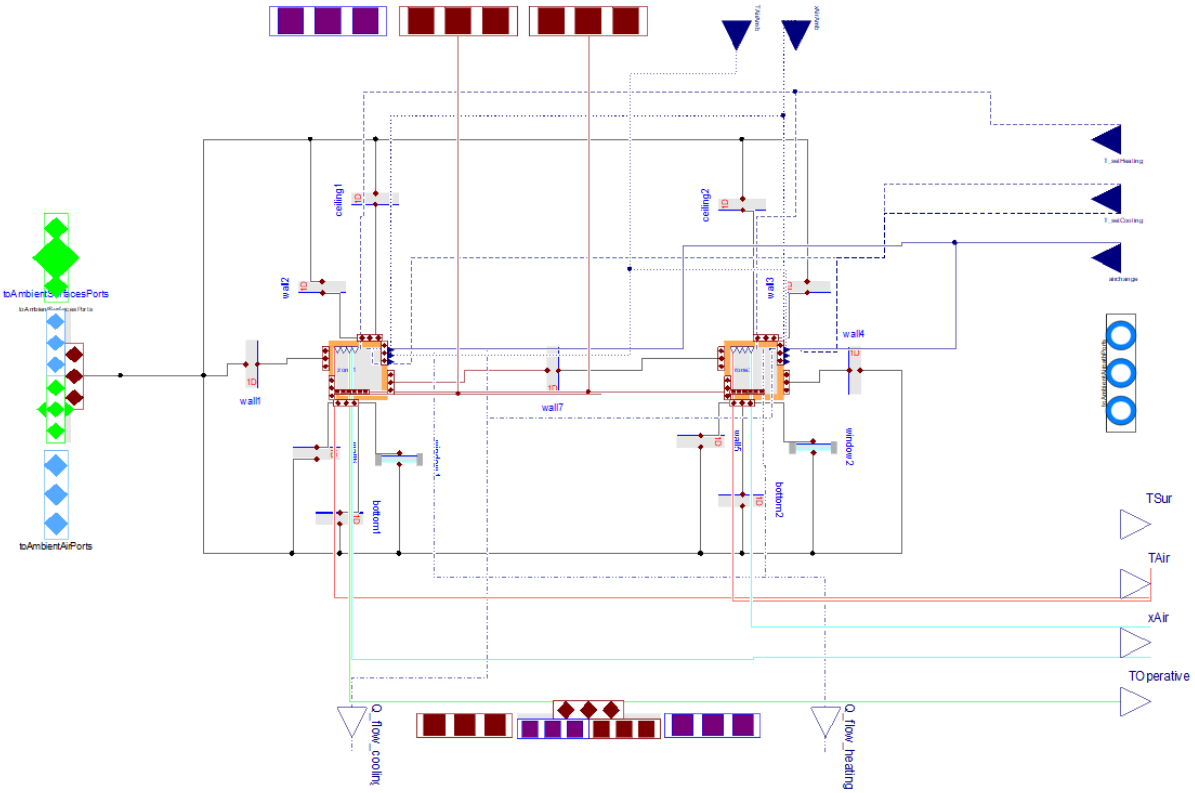

Figure 5.19: Internal Structure of the *Building* Block in Dymola

Each of these construction elements were configured by double clicking the element and defining the configuration parameters. These parameters are defined in Table 5.4. Each of the construction elements was also assigned a construction type. The construction type used for the outer-walls, intermediate walls, ceilings and floors for this house were *OuterWallSingle2014, IntermediateWallSingle2014, RoofSingle2014, BasePlateSingle2014.* Figure 5.20 shows the configuration data used for the construction element *wall1*.

Element	angleAzi	angleTil	Element	angleAzi	angleTil
wall1	90.0 °	90.0 °	**bottom1**	0.0 °	180.0 °
wall2	180.0 °	90.0 °	**bottom2**	0.0 °	180.0 °
wall3	180.0 °	90.0 °	**ceiling1**	0.0 °	0.0 °
wall4	-90.0 °	90.0 °	**ceiling2**	0.0 °	0.0 °
wall5	0.0 °	90.0 °	**window1**	0.0 °	90.0 °
wall6	0.0 °	90.0 °	**window2**	0.0 °	90.0 °
wall7	-90.0 °	90.0 °			

Table 5.4: Configuration data for Construction Elements

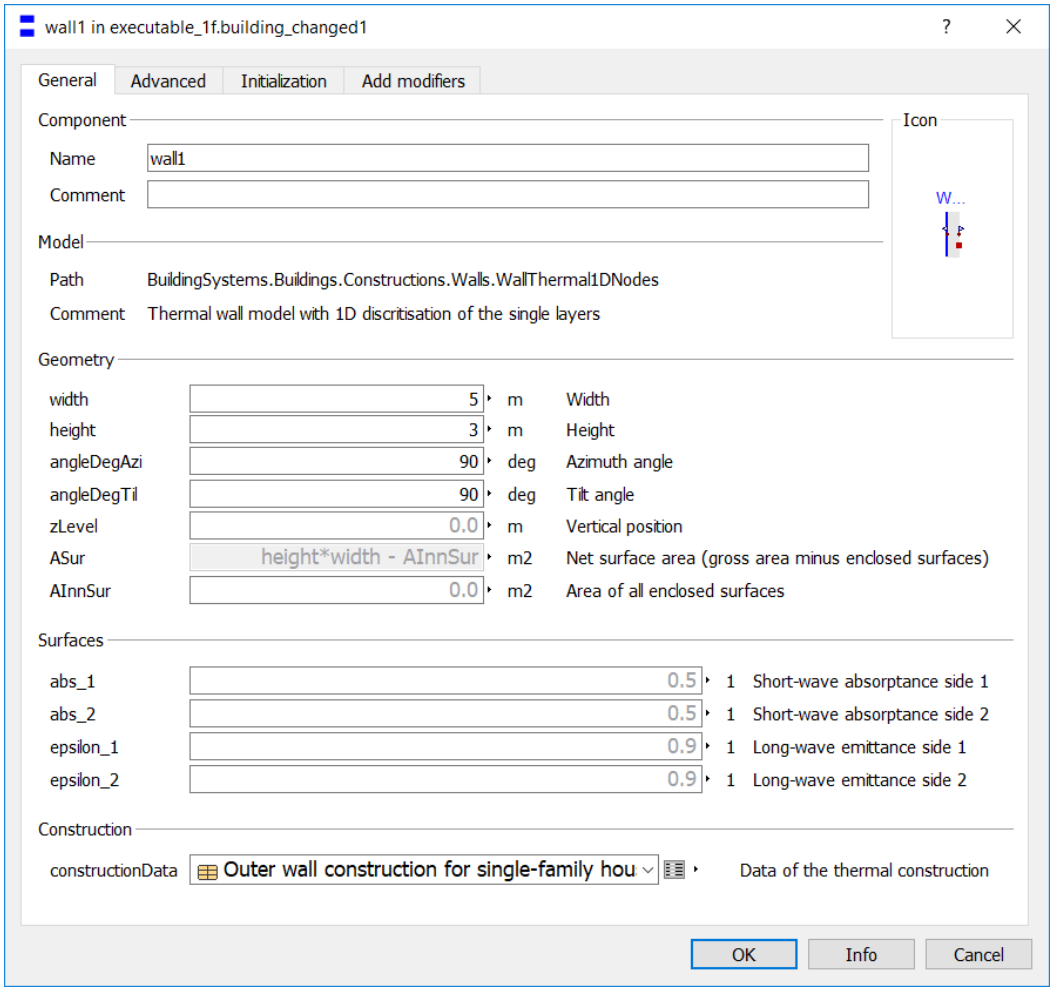

Figure 5.20: Configuration data for Construction Element *wall1*

Next, a *Temp_KOutput* output, extended from *BuildingSystems.Interfaces.Temp_KOutput*, was instantiated in the *building* model and was named TOperative. This can also be seen in the Figure 5.16. Then, the TOperative port from each of the two zones was connected to the TOperative output of the *building* model. Effectively, this created an output port called TOperative on the *building* block in the *SystemModel* model. Similarly, another *Temp_KOutput* output, extended from *BuildingSystems.Interfaces.Temp_KOutput*, was instantiated in the *SystemModel* model and was named TOperative. The TOperative port from the *building* model icon was then connected to the TOperative output in the *SystemModel* model.

Similarly, another *Temp_KOutput* output was also instantiated in the *SystemModel* model and named TAir. The TAir port from the *building* block icon in the *SystemModel* model was then connected to the TAir output in the *SystemModel* model.

These steps were necessary in order to define the outputs of the entire Dymola model as it was exported as an FMI and instantiated in Simulink. It also provided an easy way to plot outputs when running the Dymola simulation of the model as the newly created outputs are on the highest level and can be located easily.

5.3.3 The *Ambient* Block

Now that the *building* block is setup, the *ambient* block will be setup next. Double clicking on the ambient block, brings up its configuration window as shown in Figure 5.21. the parameter *nSurfaces* defines the number of building surfaces that are exposed to the ambient environment. The value for this parameter is derived from the parameter *building.nSurfacesAmbient*, which is also defines the number of building surfaces that are exposed to the ambient environment but from the *building* block's point of view.

Next, the weather file used in for the *weatherDataFile* parameter is *USA_SanFrancisco_weather.nc* which is titled *WeatherDataFile_USA_SanFrancisco.*

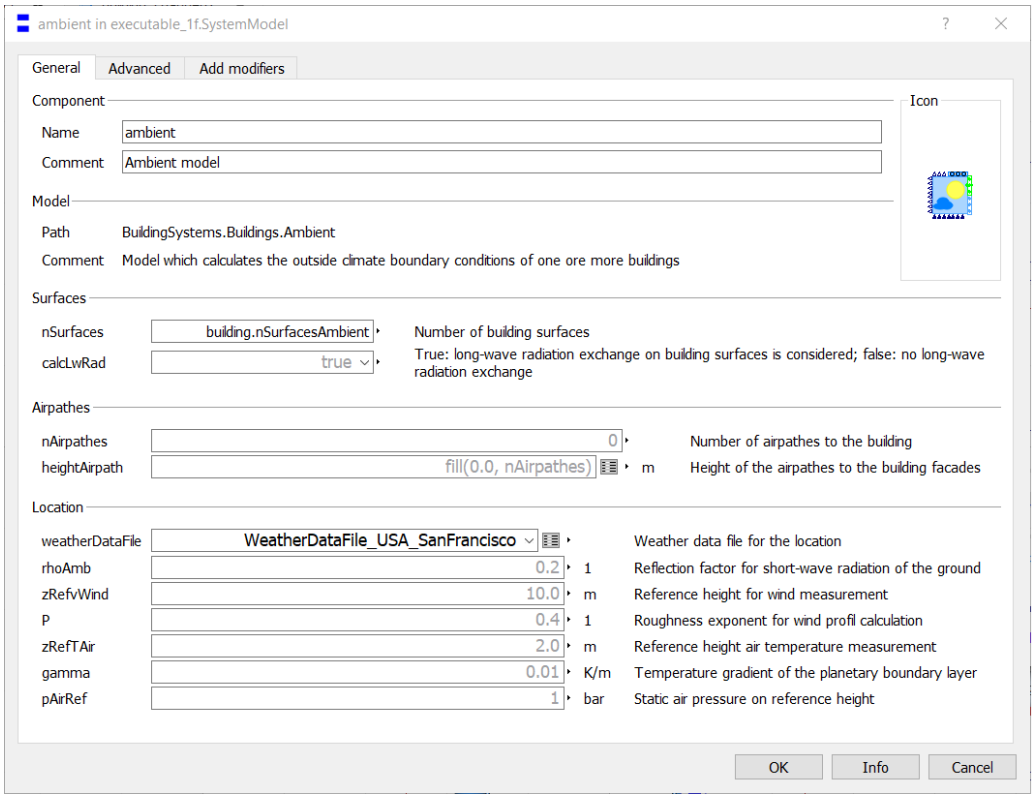

Figure 5.21: Configuration Data for Ambient Block

Next, a *Temp_KOutput* output, extended from *BuildingSystems.Interfaces. Temp_KOutput*, was instantiated in the *SystemModel* model and was named TAirAmb. This can also be seen in the Figure 5.16. Then, the TAirAmb port from the *Ambient* model icon was then connected to the *TAirAmb* output in the *SystemModel* model.

5.3.4 The House Heating Block Cluster

In the *SystemModel*, the rest of the blocks that support the heating of the model were adopted and modified from an example found in the BuildingSystems Library as extended from *BuildingSystems.Applications.HeatingSystems.SolarHeatingSystem*. This house-heating system simulates a solar thermal system which supplies a building with space heating. In the example a boiler is used to provide backup heating in case the solar system is unable to deliver the required amount of energy. However, in the model used for this thesis, the boiler was substituted by the air/water heat-pump. This was done in an effort to also use the heat-pump as an air cooler in the summer months. That way, the model would be able to simulate heating and cooling like most American houses across the country. However, using the same heat-pump in a reversible manner proved challenging due to the lack of documentation of the *HeatPump* block in the *BuildingSystems* Library documentation. Thus, it was decided that the heat-pump would be used only to provide heat in the winter months as a backup to the solar heating system. Implementing the *HeatPump* block instead of the boiler block in the example was not as straightforward as swapping one block out for the other. The HeatPump also required the addition of the new medium with which the heat exchange would take place. The additional heat-transfer medium was defined as *Air: Moist air model*. Two additional blocks had to be connected to the *HeatPump* block for it to function. First *Boundary_pT*, a block describing the boundary pressure and temperature of the air medium and second, *m_flow_eva* was also connected, specifying the flow source that produces a prescribed mass flow with temperature defined by the ambient condition as obtained from the *Ambient* block.

Although in the mentioned example, a *building* model and *ambient* model was already present, a new *building* and *ambient* model was developed as per the method described in Section 5.3.2 and Section 5.3.3 respectively, and those were used instead.

5.3.5 Simulation of the Dymola Model

Now that the model development was complete, the next step was to run the simulation of the Dymola model and once the model was verified to be working, the FMI would be generated next.

To simulate the system, first a compiler had to be selected in the Dymola Simulation Setup window in the Complier tab as shown in Figure 5.22. For this work, Visual Studio 2013/ Visual C++ Express Edition (12.0) was used as the compiler. The Test Compiler button verified that the compiler was running in 32 bit, as well as 64 bit mode.

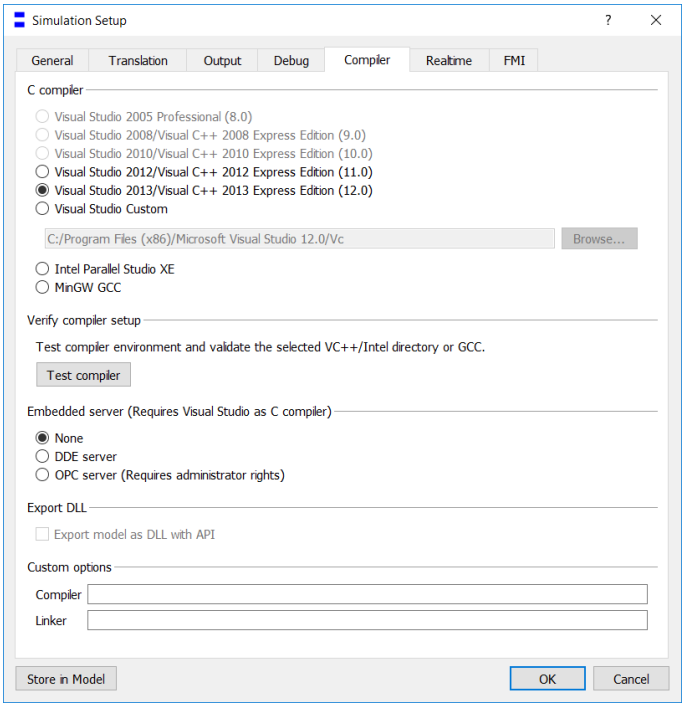

Figure 5.22: Compiler Setting in Simulation Setup Window

After the compiler was selected, in the General tab of the Simulation Setup Window, the start time and stop time of the simulation had to be defined. The start time was defined as 0 d and the stop time was defined as 365 d, to simulate a whole calendar year starting from 01 January. Although the simulation actually uses seconds as a timestep, it was more convenient to input the start and stop time in the unit of days. The other settings were left to their default values. This process is shown in Figure 5.23.

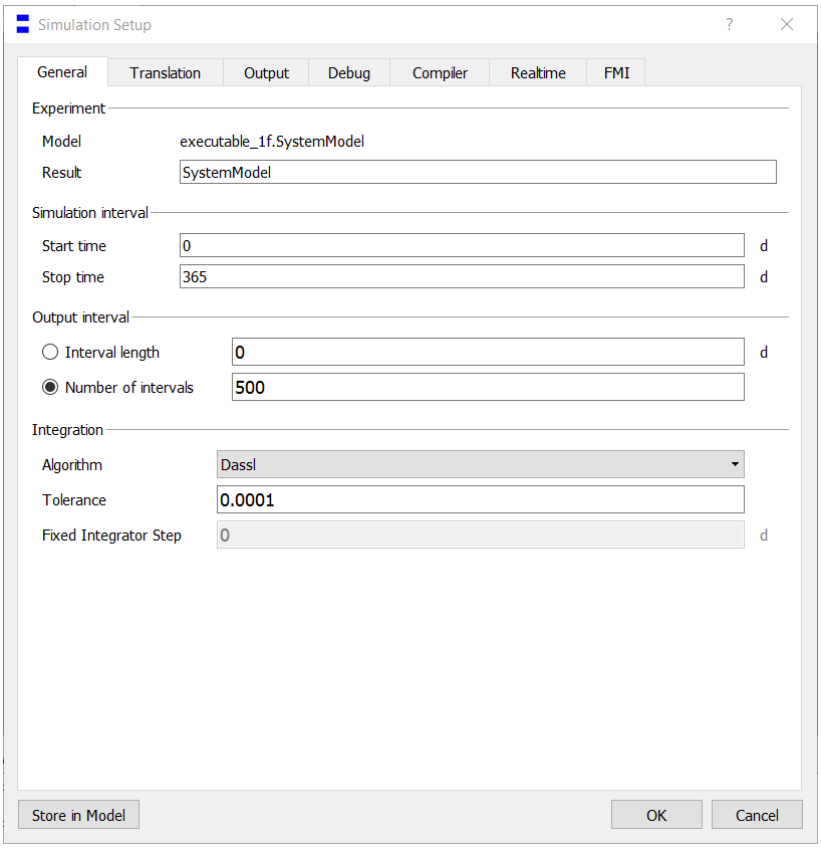

Figure 5.23: General Tab Settings in Dymola Simulation Setup Window

5.3.6 FMU Generation

At this stage, the Dymola model was ready, and the Functional Mock-up Unit (FMU) of the model was ready to be generated. To generate the FMU in Dymola, the Translate button in the Simulation Tab of Dymola was used as shown in Figure 5.24.

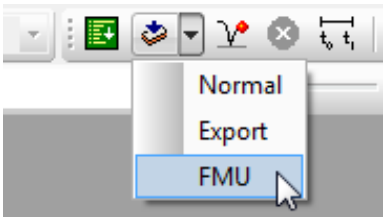

Figure 5.24: Translate Button in Simulation Tab of Dymola [19]

Following this, a dialog box appears to select the FMU Export settings as shown in Figure 5.25 and the following settings were selected. Since the FMU was going to be exported from Dymola and into Simulink, the model exchange type of FMU was selected. Clicking OK generated the FMU files in the chosen Dymola directory. This concludes the description of the Dymola Model for this thesis.

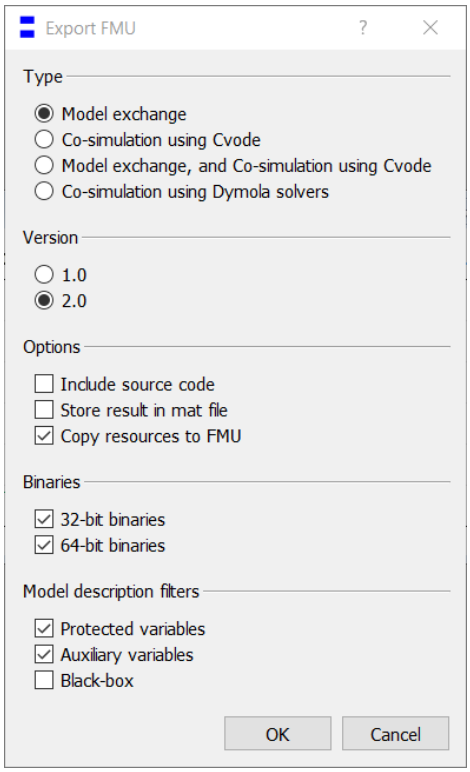

Figure 5.25: Export FMU Settings Window

5.4 Simulink as an Intermediate Model

5.4.1 Overview of Simulink Model

To integrate the Dymola model with Cameo Systems Modeler, Simulink was required to be used as an intermediate model. This is because the Dymola model was unable to be linked with SysML model either directly or through the use of the FMI standard. Since Matlab does have an existing integration with Cameo Systems Modeler, the idea was to somehow import the Dymola Model into Simulink which would then be programmatically accessible to Cameo through a Matlab script. This meant that the Simulink model would just be an empty “shell” containing the full Dymola model

without any alteration to it. The only value it adds to the model is the ability to link with SysML by exploiting the existing Cameo-Matlab integration.

5.4.2 Setup of FMI Kit in Simulink

For importing the Dymola model into Simulink, a special block called the FMI block was required. This special block could be found in the FMI Kit for Simulink that is made available by Dassault Systems with Dymola. The FMI Kit enables embedding FMUs into Simulink [20]. It also has full support for both export and import of both versions, 1.0 and 2.0 of FMIs. FMI Kit for Simulink is located in the `$Dymola_installation_folder/Mfiles/FMIKit_for_Simulink/` directory. To make the FMI Kit available in Matlab and Simulink the above directory and all its sub-folders had to be added to the Matlab path. In addition, the `ds_fmikit_setup.m` Matlab script had to be run every time an FMU would be imported into Simulink [20]. Thus, for the purpose of this thesis, this script was just added to the Matlab startup file, `startup.m` so that it would run every time Matlab started up.

5.4.3 Creating the Simulink Model

Now that the FMI Kit was set up for the Simulink, a new Simulink model was created and named `exec_model.slx`. To add an FMU block to this model, the following steps were taken.

1. Open the Simulink library browser (***View*** **>** ***Library Browser***) and drag the FMU block from the FMI Kit library into the model.

2. Double-click the FMU block, select **Load** and choose the FMU of the two-room, energy-efficient house, generated by Dymola
3. Click **OK**.

After following these steps, the model would look like Figure 5.26.

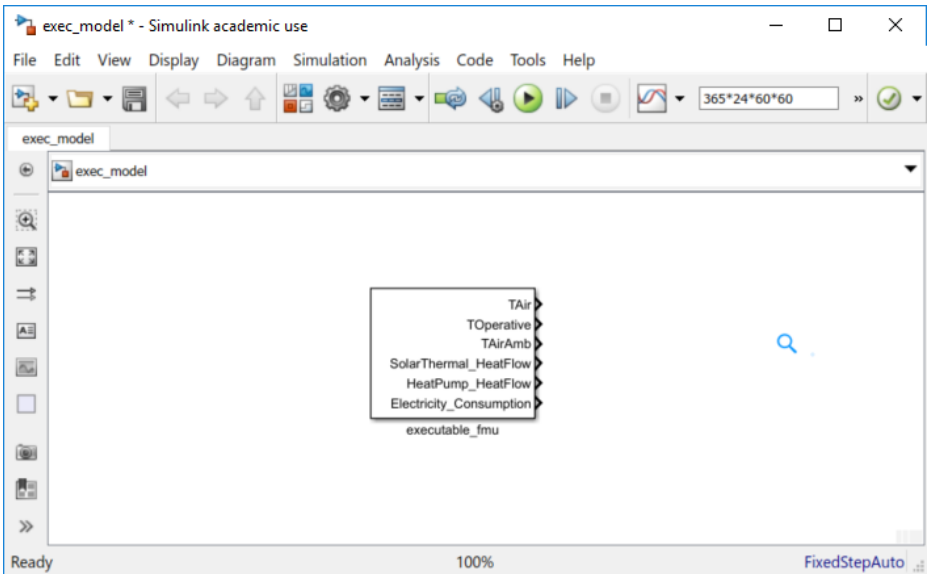

Figure 5.26: FMU Block imported into Simulink and FMU loaded into it.

Notice that the FMU block also has six outputs that correspond with the same six outputs from the Dymola model. In addition to these outputs, additional outputs can be created in the Simulink model using variables internal to the Dymola system by double clicking on the FMU block in the Simulink and navigating to the **Outputs** tab. This is demonstrated in the Figure 5.27.

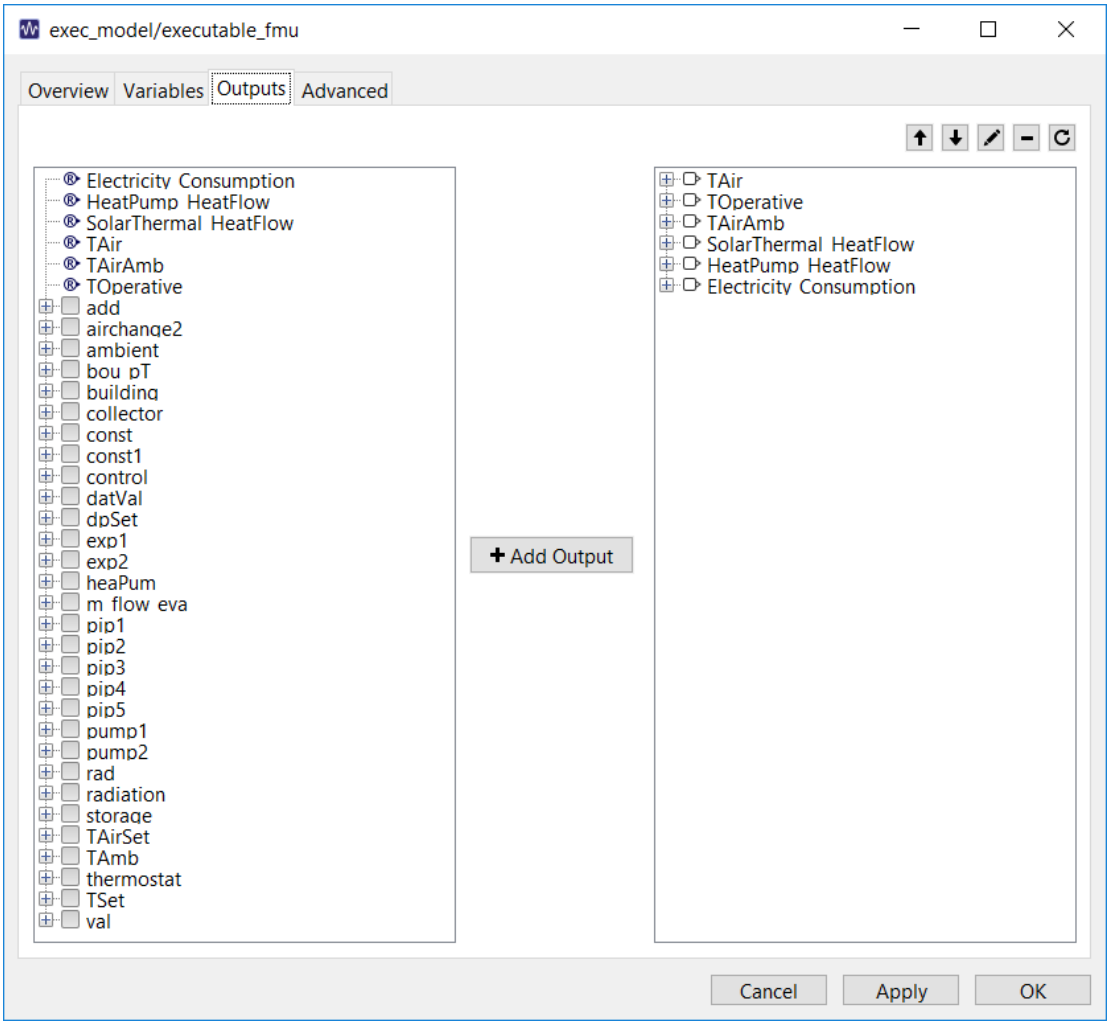

Figure 5.27: **Outputs** tab FMU block configuration window allows for additional outputs to be monitored.

Next, each of the output arrows in the model were connected to *Bus Creator* block from the Simulink Library browser. The *Bus Creator* block usually has only 2 inputs. So, allow the three outputs from the FMU block to the *Bus Connector*, double click on it and change the number of input signals to 3. Following, the output arrows were dragged from the FMU block and connected to the *Bus Connector* block. Next, a block called the *Scope1* was added to the model and connected to the *Bus Creator* block. This would allow the first three outputs from the *FMU Block* to be plotted on a single graph.

Similarly, another *Bus Creator* and two more *Scopes* were set up for the other outputs. Finally, six *Output Port* blocks were added to the model from the Simulink Library browser as shown in Figure 5.28. This concluded the building of the Simulink Model.

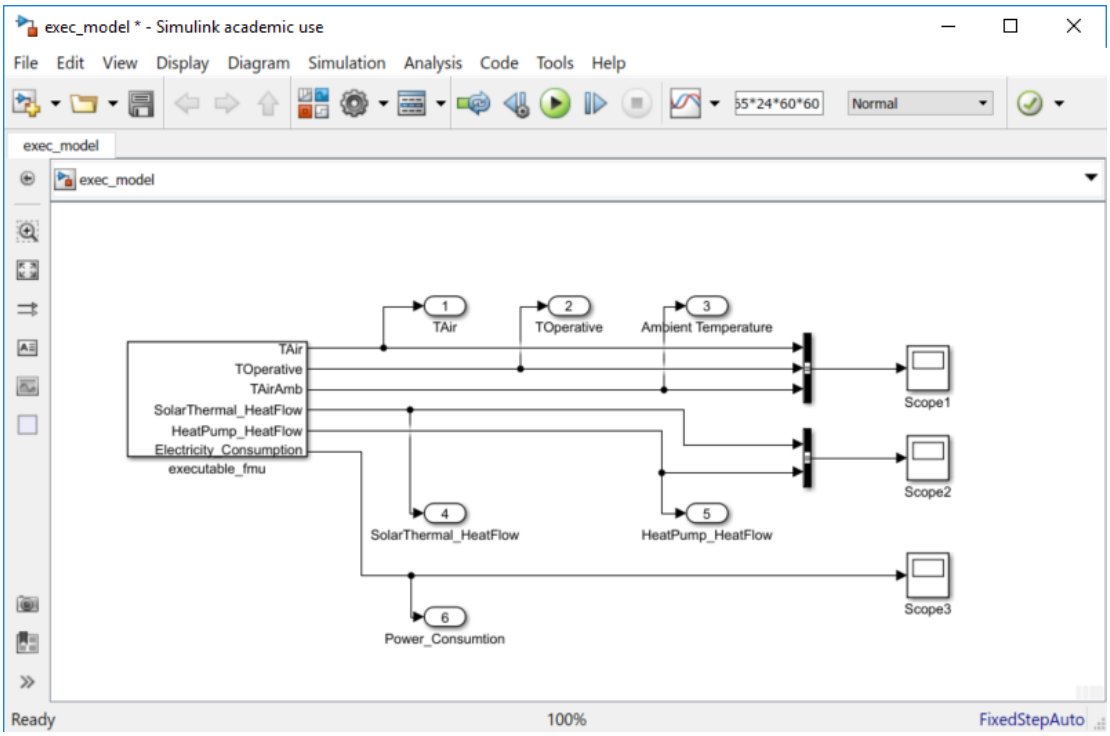

Figure 5.28: Final Simulink Model

5.4.4 Simulink Simulation Configuration

The settings for the Simulink simulation were configured by accessing the **Model Configuration Parameters** window in Simulink. In the **Commonly Used Parameters** tab the *start* and *stop time* of the simulation were specified as 0 s and 365*24*60*60 s respectively to simulate the Dymola model over a 1-year period. Next in the **Solver Options** tab, the *Fixed-step* type was selected and the *Solver* was set to *auto.* In the **additional options** section, the *fixed-step size* was set to 3.5 s. The reason for setting this obscure value as the fixed time step was because the intention was to choose a large

enough step size that would not slow down the simulation too much. The intended step-size was 30 s however, at this setting the Simulink simulation would always crash for an unknown reason. Thus, through hit and trial, 3.5 s was found to be the largest step size that Simulink would accept without crashing. In attempts to reduce the simulation time, the solver type was also changed to *Variable-step* however, this actually increased the simulation time by 2-3 times, so solver type was changed back to *fixed-step* with step size of 3.5 s. The Solver options used are shown in Figure 5.29.

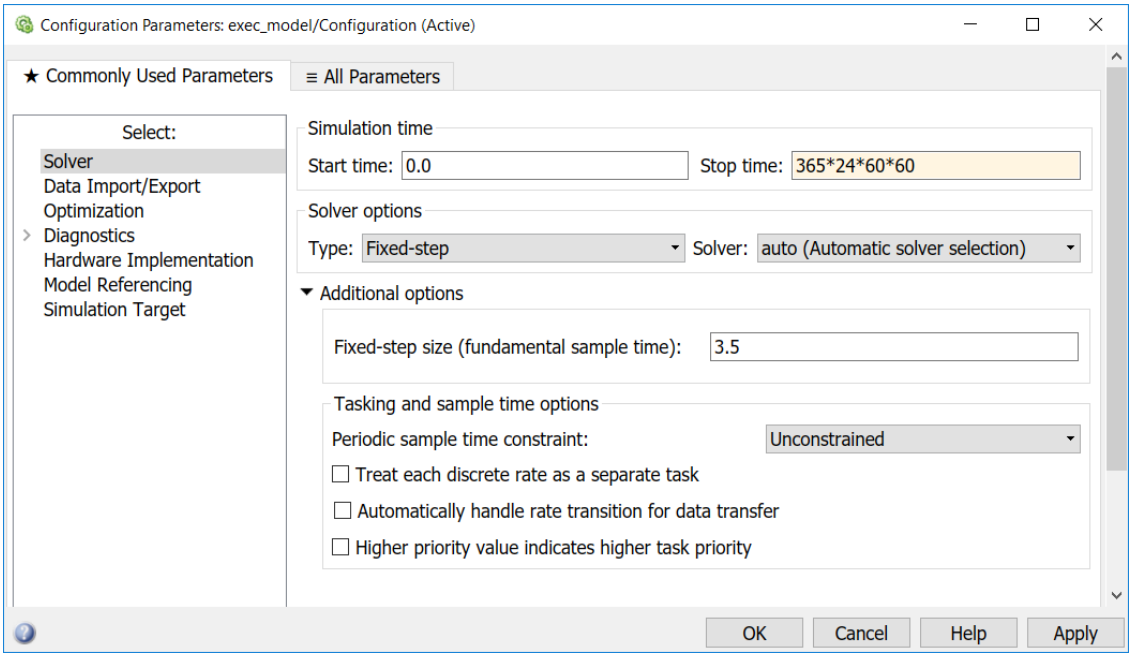

Figure 5.29: Simulink Solver Options

Next, the data import and export settings are configured in the **Data Import/Export** tab of the **Model Configuration Parameters** window. The format was changed to *Array* to enable the three outputs of the Simulink model to be captured as an array `yout` in the Matlab workspace. The data import and export settings are shown in Figure 5.30.

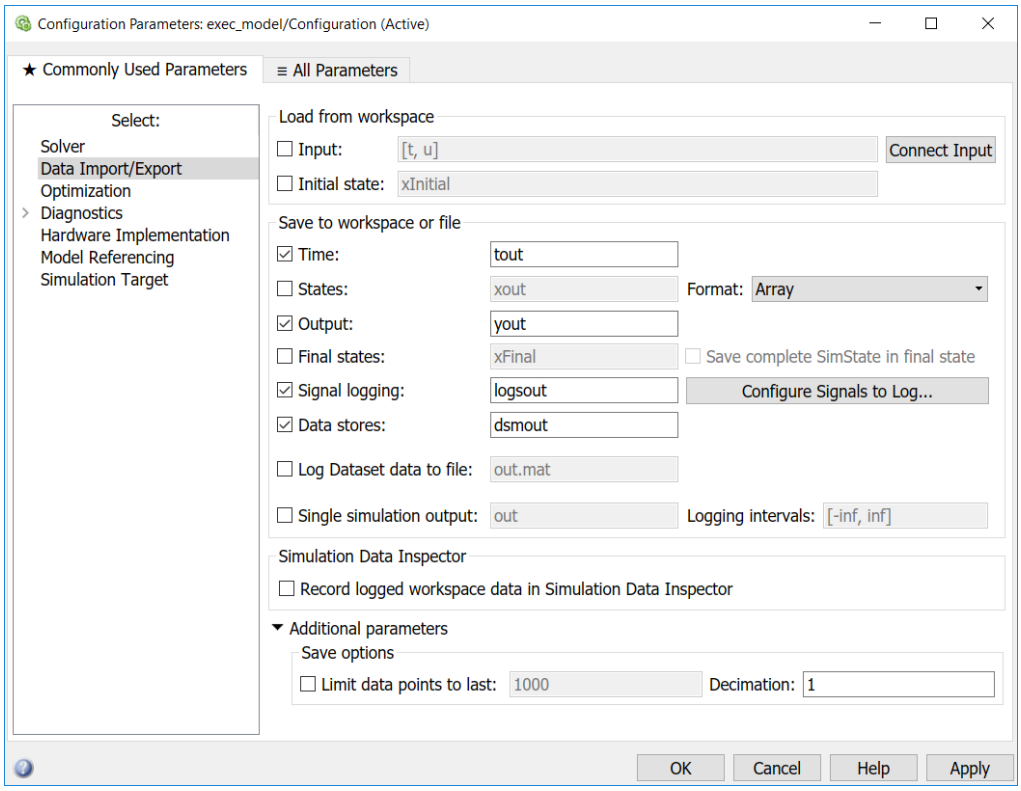

Figure 5.30: Data Import and Export Settings

After this, the Simulink simulation can be run just to verify that everything works. This concludes the description of the Simulink Model for this thesis. Just to reiterate, bereft of this Simulink "shell", the Dymola model cannot be directly integrated to the Cameo SysML model using any current method. It is only required as an intermediate model.

5.5 Integration Procedure

Having concluded building the SysML model of the system, the actual process to integrate all three models to create one fully executable model will be discussed in this section. First, the internal mechanics of the integration will be explained and the procedure to actually perform it will follow.

5.5.1 Integration Mechanics and Matlab Script

The way the integration works is that the user will input values for the design parameters in SysML in an “instance” of the model (Instances will be explained in detail below). These inputted values are then exported as a text file with the name *input.txt* by the SysML ParaMagic Plugin. The text file is then read by the Matlab script, assigning the inputted values to their corresponding variables and thereby inputting them into the Matlab workspace. This Matlab script then programmatically launches the Simulink model that was discussed earlier. Next, the script obtains the default parameters for the entire Simulink model from the FMU that resides in the Simulink model. Subsequently, these parameters are replaced with new parameters as specified in the input.txt file that is outputted by Cameo Systems Modeler. Following this step, using string manipulation methods, the old parameters string is edited to include the new parameters as mentioned by the input.txt file.

This is done by first, converting the old parameters string, which is saved as a `char` to a `string` for the purpose of string manipulation. Then the string is split into a `cell array` and the relevant cells are edited. The `cell array` is then converted back to a `string` and then back into `char` and fed back into the Simulink model to simulate.

Finally, the Simulink model is re-evaluated through the simulation and the metric is calculated over the period of the simulation time, in this case, one year. Lastly, the final value of the metric at the end of the simulation time is outputted from Matlab as a text file called *output.exe*. This file is then read by the ParaMagic Plugin in the SysML

model and the stored metric value is then displayed in SysML and updated into the SysML model.

To conduct a trade study, several instances are created in SysML, each with a different set of input parameters. Based on the resulting values of the metric, a suitable set of design parameters can be chosen.

5.5.2 Installing ParaMagic Plugin

To install the ParaMagic Plugin, the following steps were performed from the Cameo Systems Modeler Interface.

1. Install ParaMagic Plugin
    a. From the toolbar, Help > Resource/ Plugin Manager
    b. Expand the Plugins (Commercial) section
    c. Check the ParaMagic Box and Select Download/Install as shown in Figure 5.31
        i. With an evaluation license provided by the University of Maryland, this plugin will be accessible for 30 days.
2. Load ParaMagic profile
    a. From the toolbar, File > Use project > Use Local Project as shown in Figure 5.32
    b. In the “paths to used projects” section, select <install.root>\profiles
    c. Select the ParaMagic Profile.mdzip from the dropdown menu.
    d. Click finish

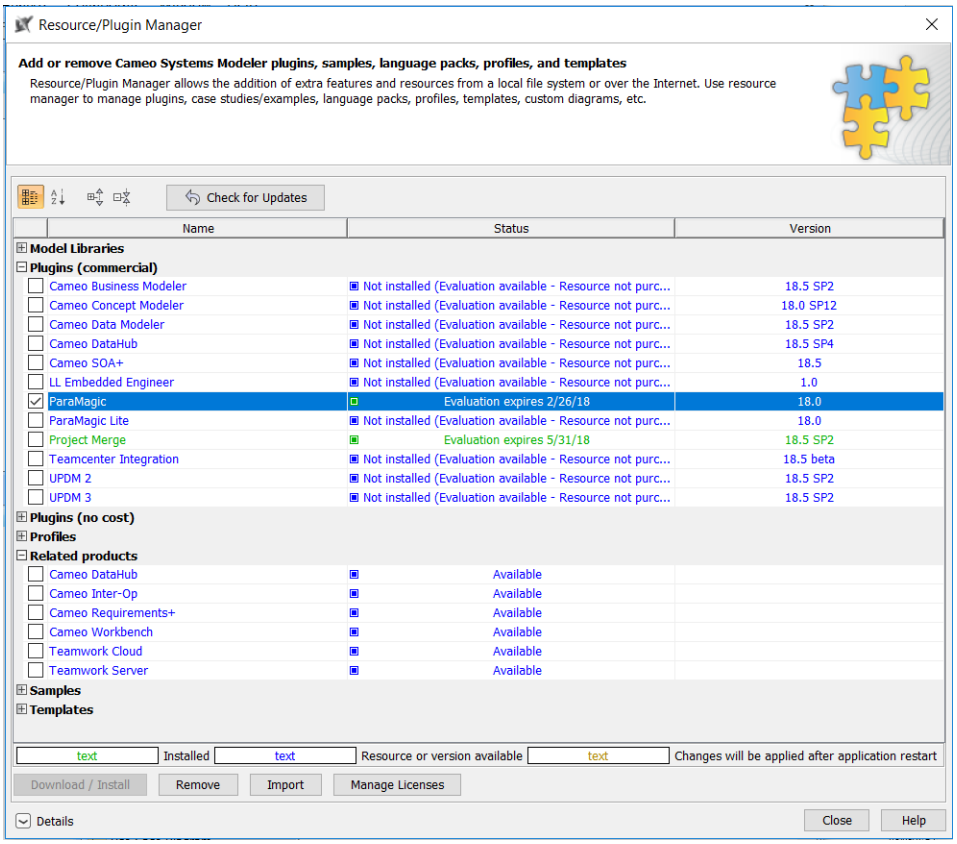

Figure 5.31: Cameo Systems Modeler Resource/ Plugin Manager

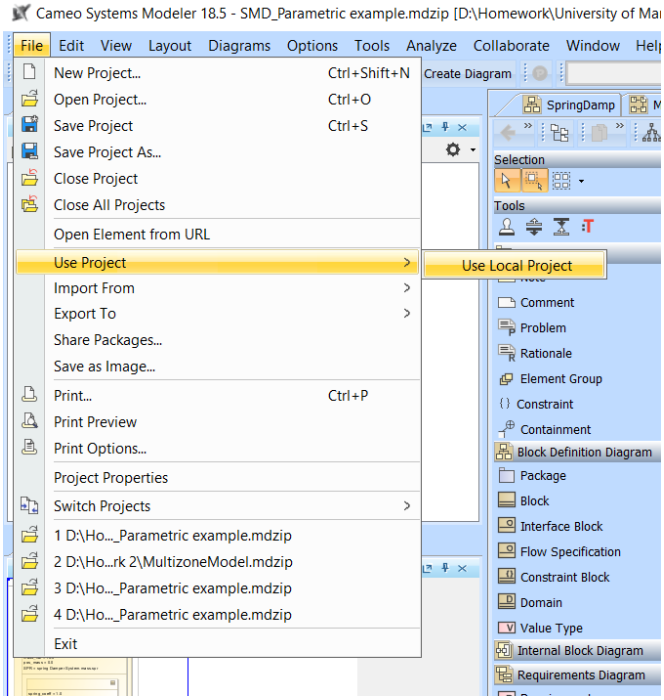

Figure 5.32: Loading ParaMagic Profile

5.5.3 Creating an Instance

Having created the SysML structure and Parametric Diagram, the next step was to create an instance of the system and then use the ParaMagic Plugin to solve the instance to solve for the metric. An instance is an example of the model with specific values assigned to the given parameters and which can be solved for the unknowns [21].

However, in order to create an instance, first the model had to be "validated". It is important to note that in this context, the validation of the model has nothing to the do with the Systems Engineering technical process of Verification and Validation. This validation simply means that the model was checked for consistency and screened for syntax errors and other minor errors. To validate the model, the root block, in this case, *Multizone Building System* has to be right clicked and then under the ParaMagic tab, the Validate button has to be clicked. This is demonstrated in Figure 5.33.

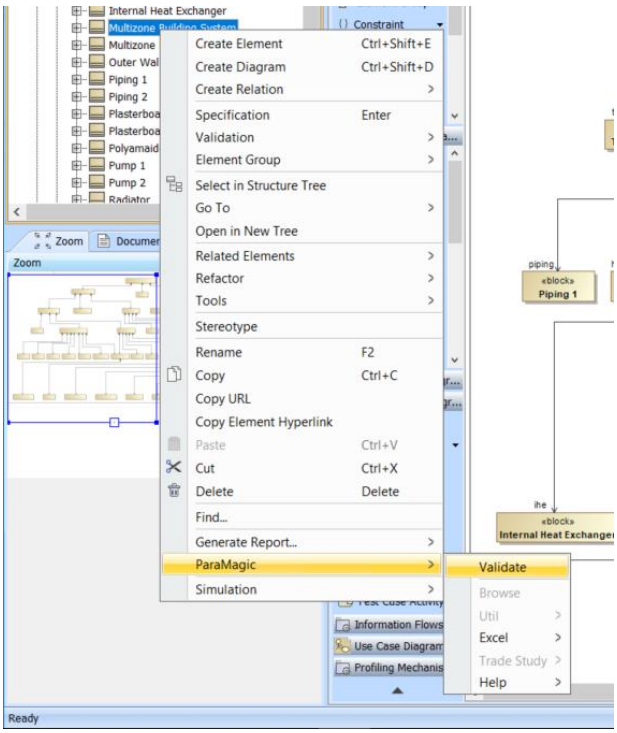

Figure 5.33 Validating System Model

This validation uncovered several minor errors in the model that needed to be resolved before proceeding. However, to aid in the localization of the errors, it is suggested that each block contained within the main root block be validated. This permits the user to resolve all lower level errors which will subsequently allow the main root block to be validated.

Next, an instance of the system was created using the Automatic Instantiation Wizard. This was performed by:

1. Right clicking on the root block, *Multizone Building System*
2. Navigate to **Tools > Create Instance**
3. Select all the part of the model that need to be instantiated as shown in Figure 5.34. In this case, all the value properties associated with the design parameters and their parent blocks need to be selected. In addition, the metric, *Indoor Air Temperature* and its parent block were also selected.

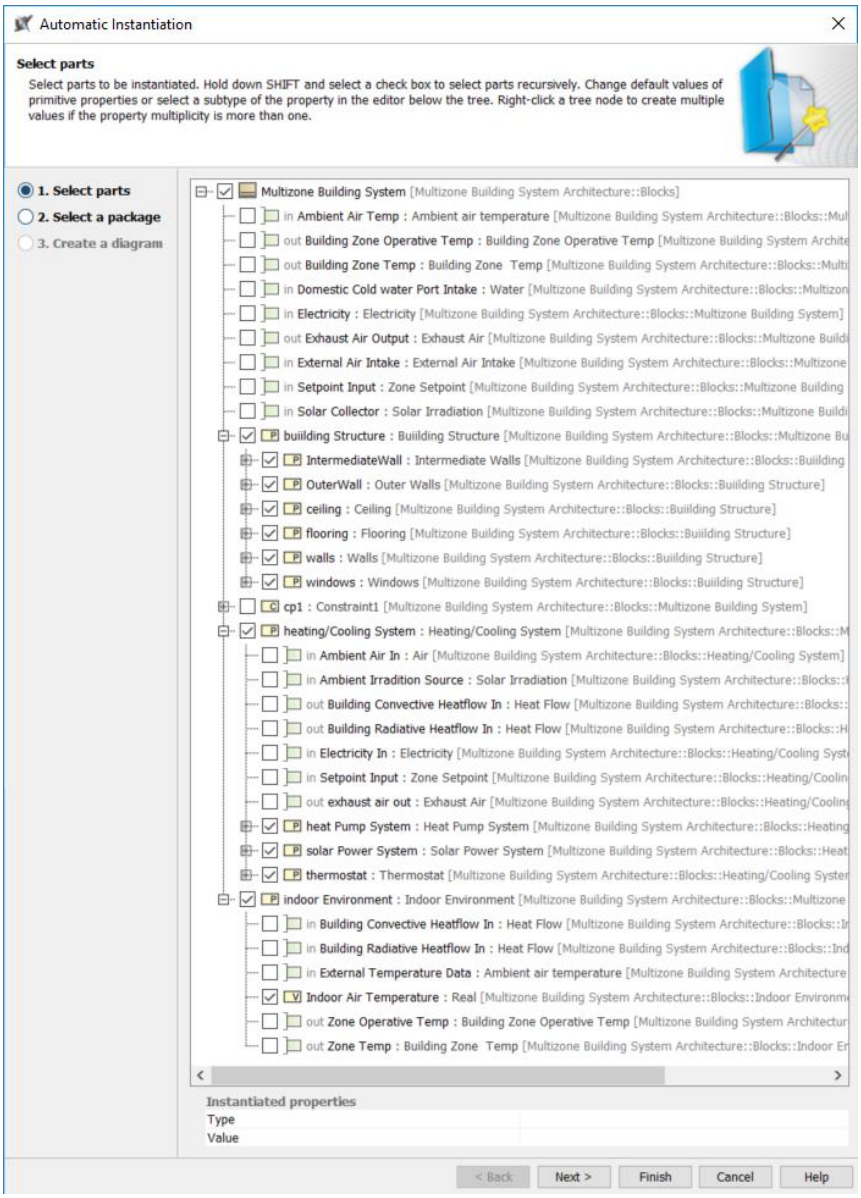

Figure 5.34: Selecting Parts in the Automatic Instantiation Wizard

4. Click **Next**

5. Create a Package under the root folder and name it *Instances*.

    a. Create a new package called *Instance01* under *Instances* (shown in Figure 5.35).

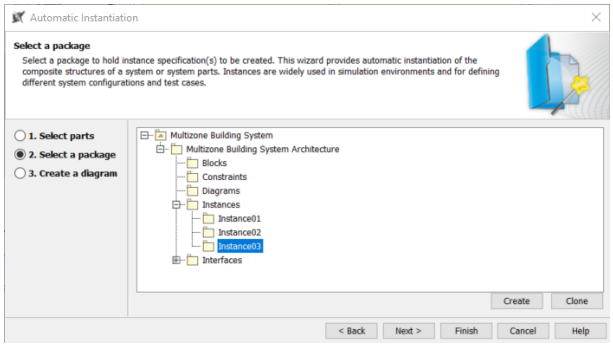

Figure 5.35: Creating Packages for the Instance

6. Click **Next**
7. Create the instance BDD under the package that was just created and name it Instance01 BDD (shown in Figure 5.36)

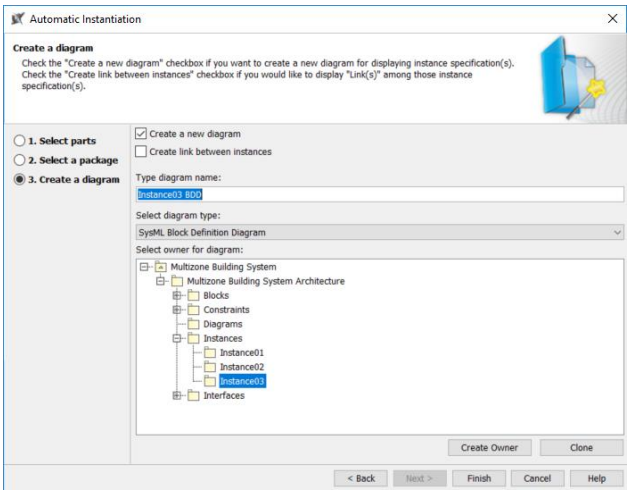

Figure 5.36: Creating the Instance BDD

8. Click **Finish**

Following these steps resulted in the creation of the instance as shown in Figure 5.37, Figure 5.38 Figure 5.39, and Figure 5.40. For the sake of readability, the actual Instance BDD has been cropped into the four aforementioned figures. In the figures, it can be seen that a large set of nested blocks are created. Each of these nested blocks

are an instance of that block from the system model. Most of the instances will also have a default value of 0 associated with its value type. Each of the design parameters and constants then had their value manually set to their nominal value as described in Table 5.2 and Table 5.3 respectively. Since the value of the other value properties are not known, they can be left as 0 since, they are not explicitly a part of the main constraint equations.

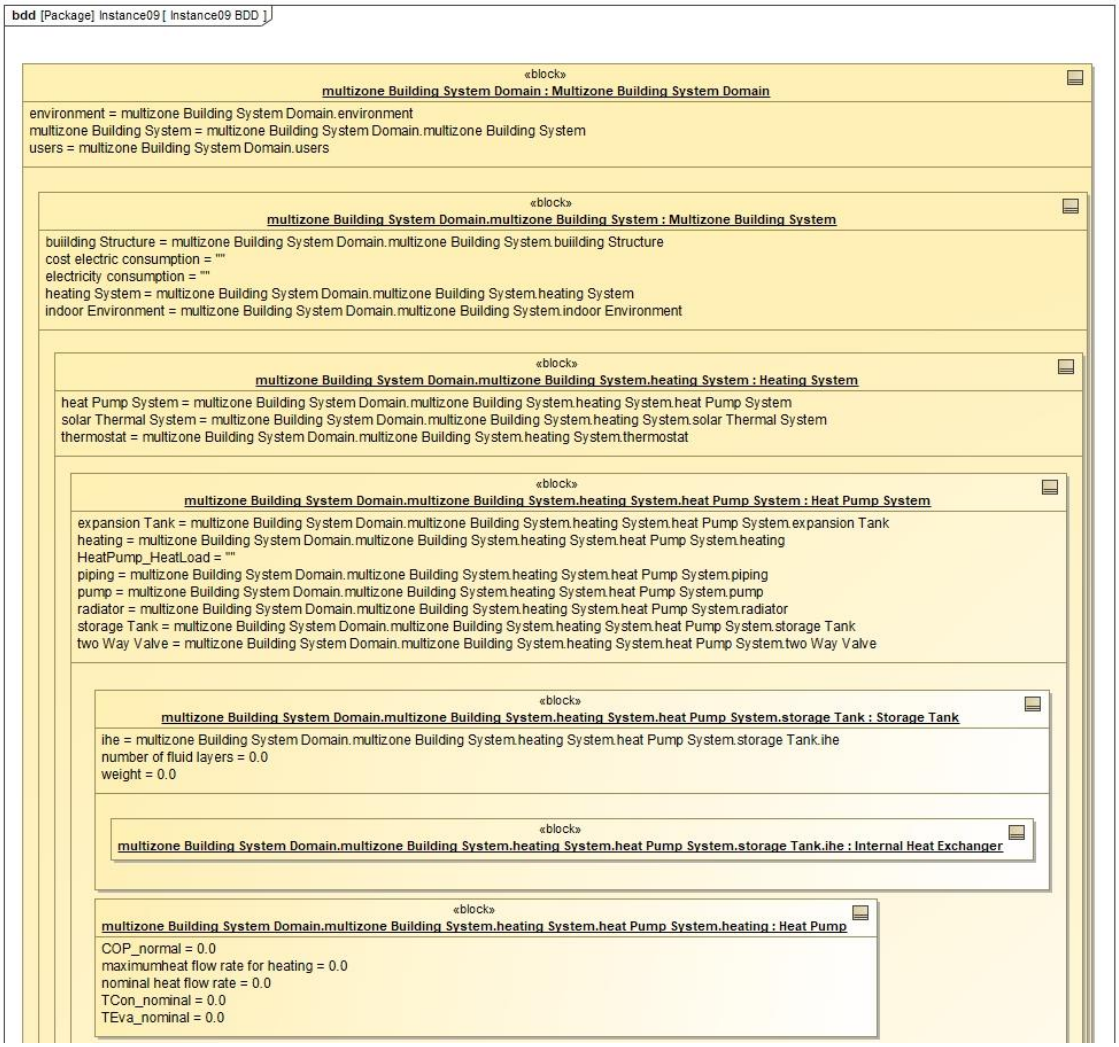

Figure 5.37: Instance of the System Model – 1

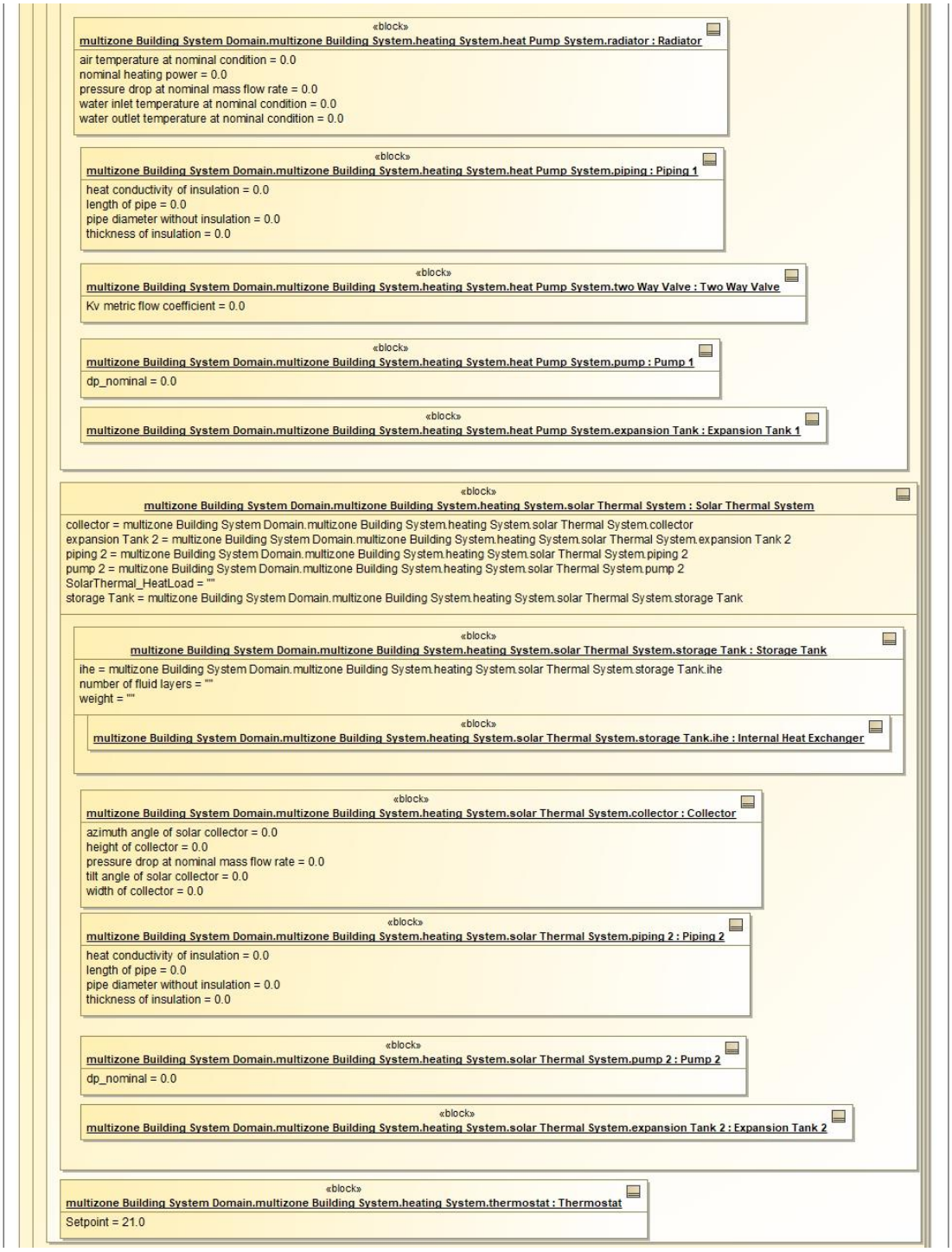

Figure 5.38: Instance of the System Model – 2

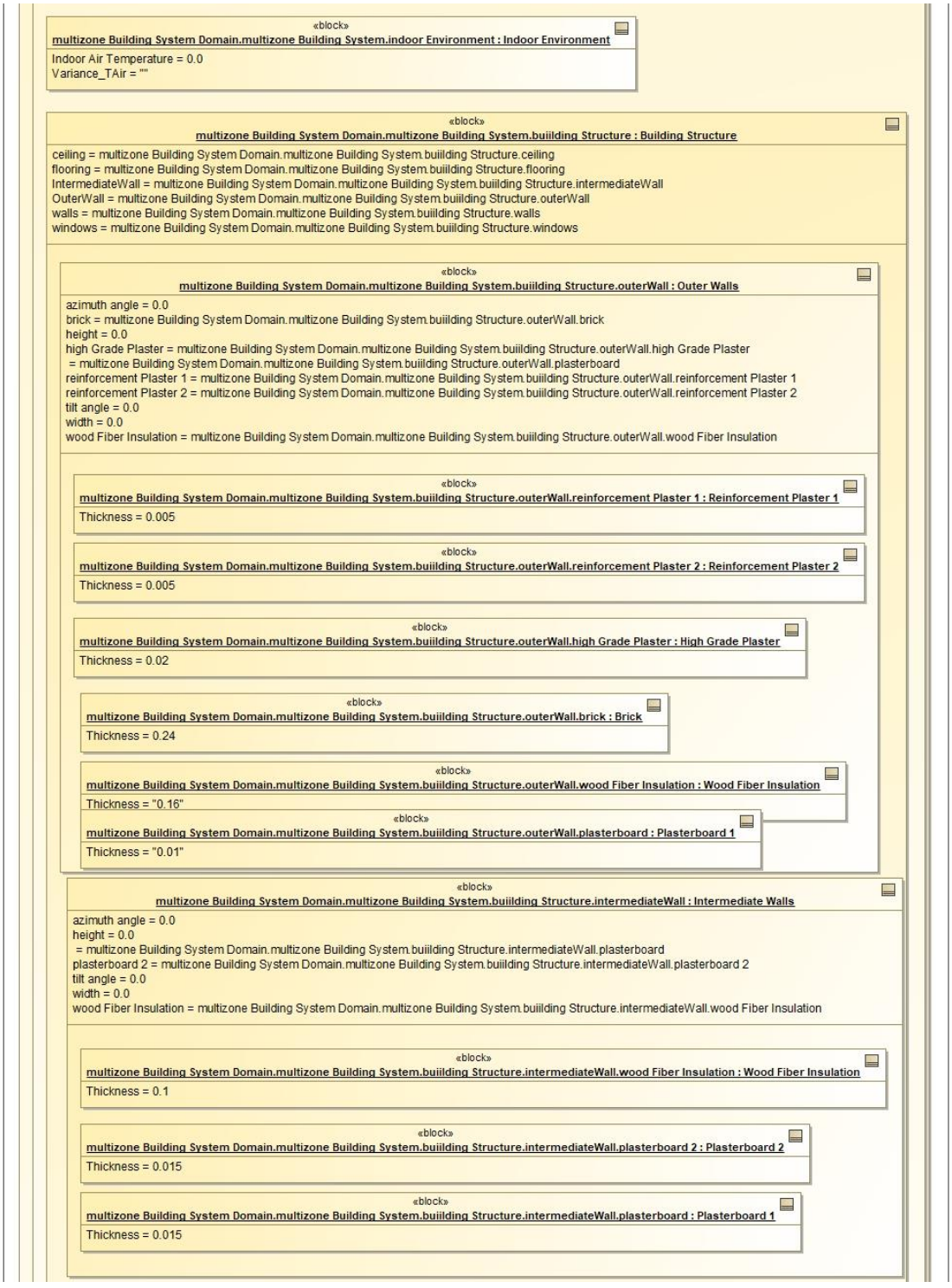

Figure 5.39: Instance of the System Model – 3

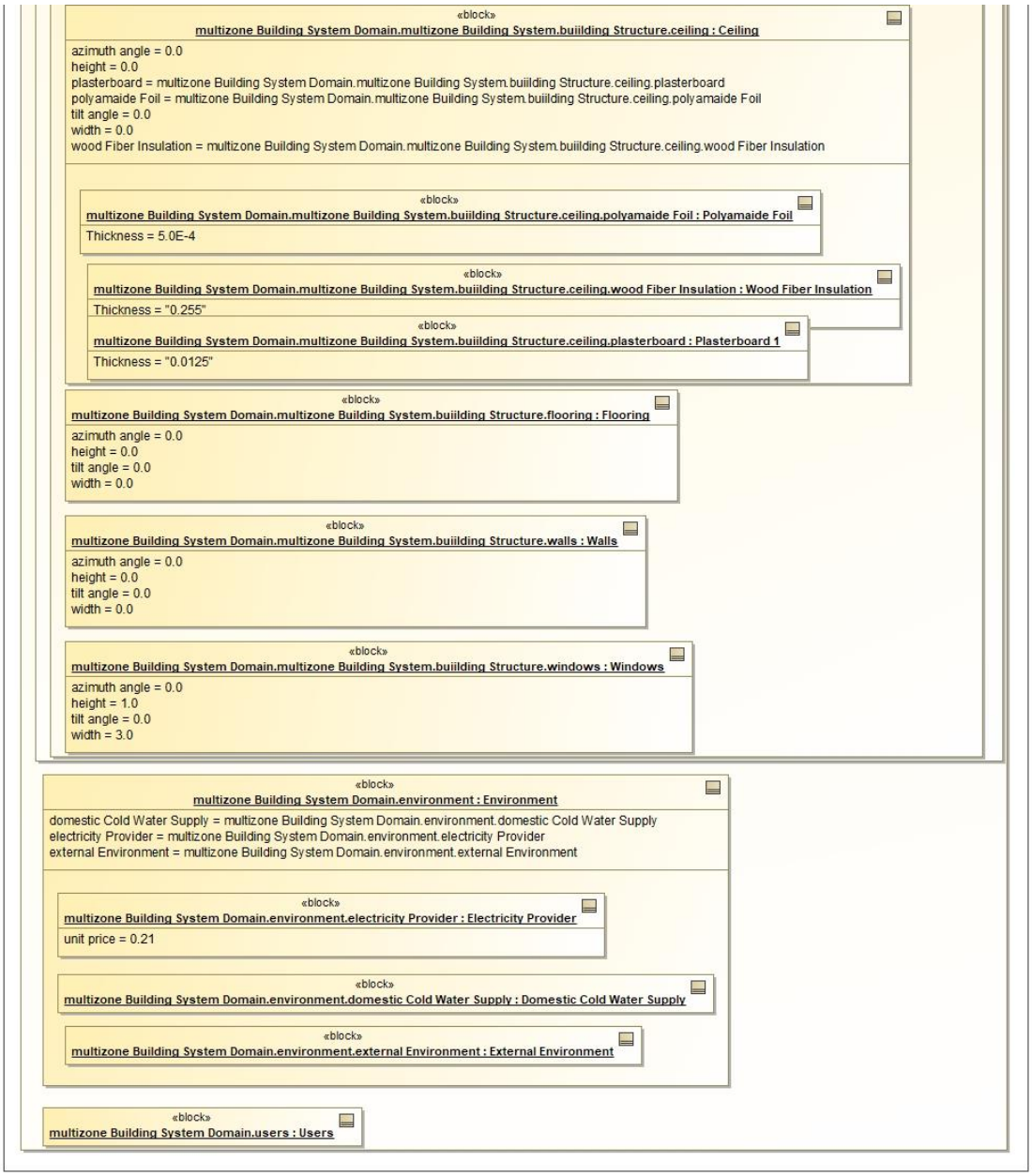

Figure 5.40: Instance of the System Model – 4

5.5.4 Solving the Instance

Finally, the instance was solved using the ParaMagic Plugin. To do this, the following steps were performed.

1. Right click on the *Instance* in the **containment tree**.
2. Navigate to the ParaMagic Button and click on Browse. This will launch the ParaMagic Browser.
3. Expand all the tabs to reveal the design parameters and their respective values. Notice that their causality should be set to *given.*
4. Set the causality of the metric, *Indoor Air Temperature* to *target*. The Browser should now look as shown in Figure 5.41.

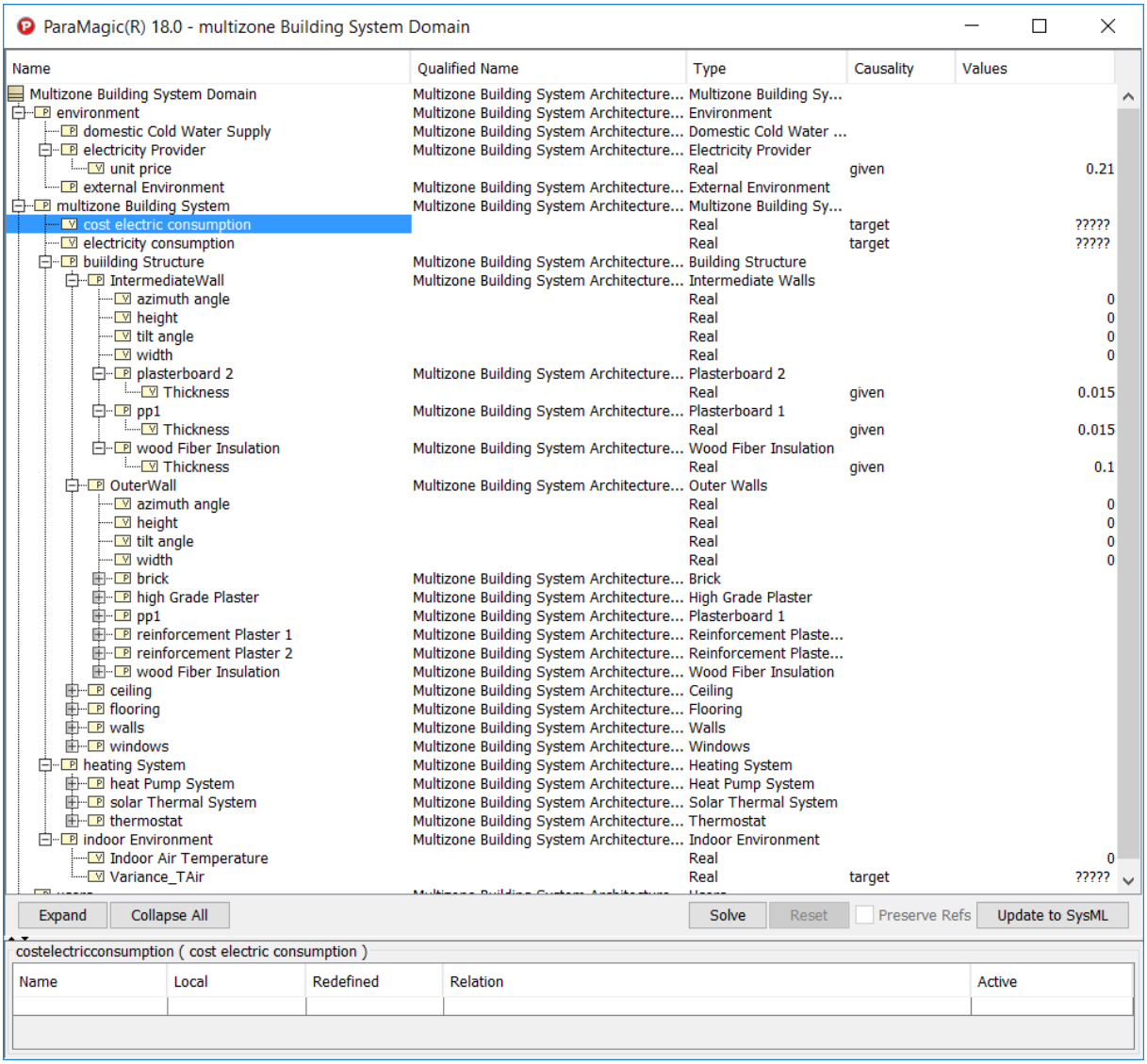

Figure 5.41: ParaMagic Browser

5. Click on Solve.

Following these instructions will cause Cameo to launch Matlab, and then launch the Simulink Model. The Simulink model will run and finally output 3 graphs as shown in Figure 5.42, Figure 5.43 and Figure 5.44. Finally, the ParaMagic Browser will update with the calculated value of the *target* parameter. This can then be updated in the SysML model clicking on the "Update to SysML" button. This concludes the Executable Model building process.

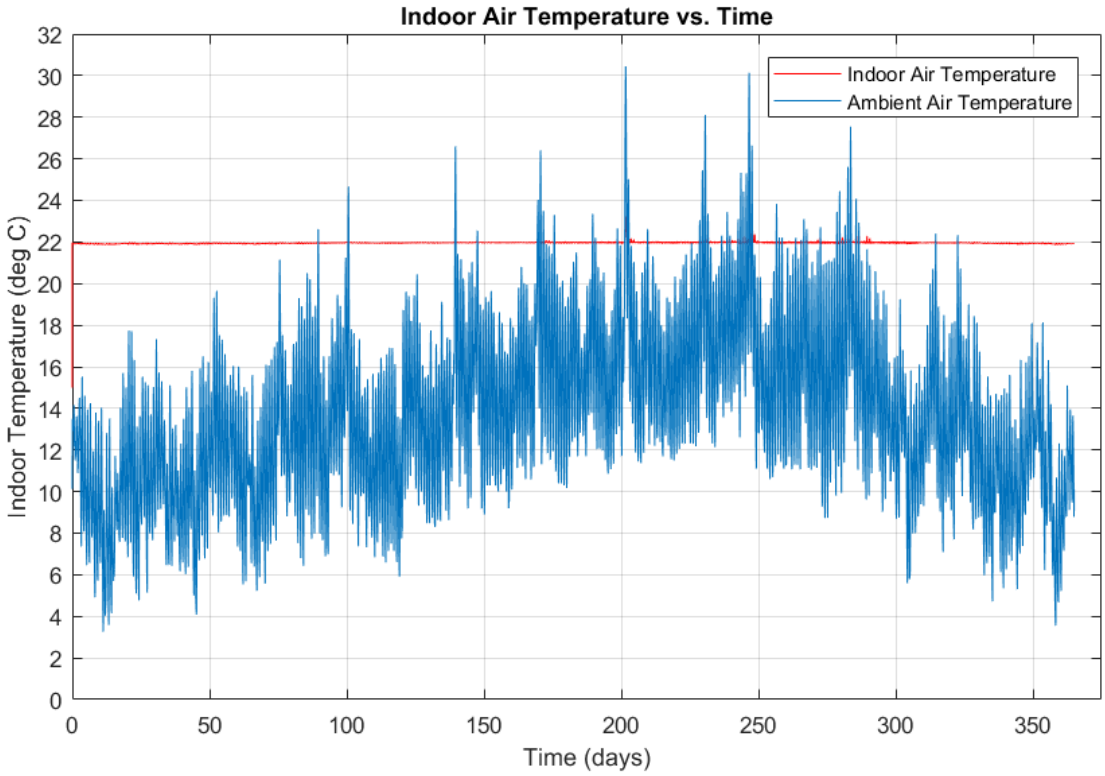

Figure 5.42: Plot of Indoor Air Temperature vs. Time

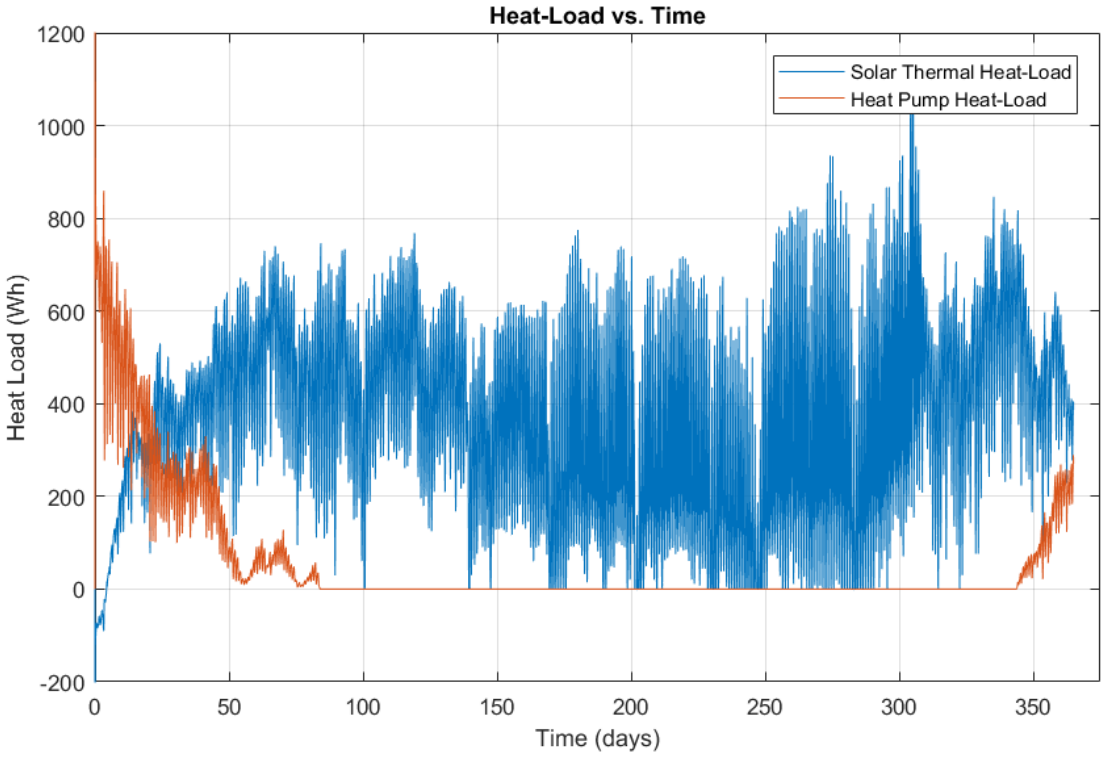

Figure 5.43: Plot of Solar Thermal and Heat Pump Heat Load vs. Time

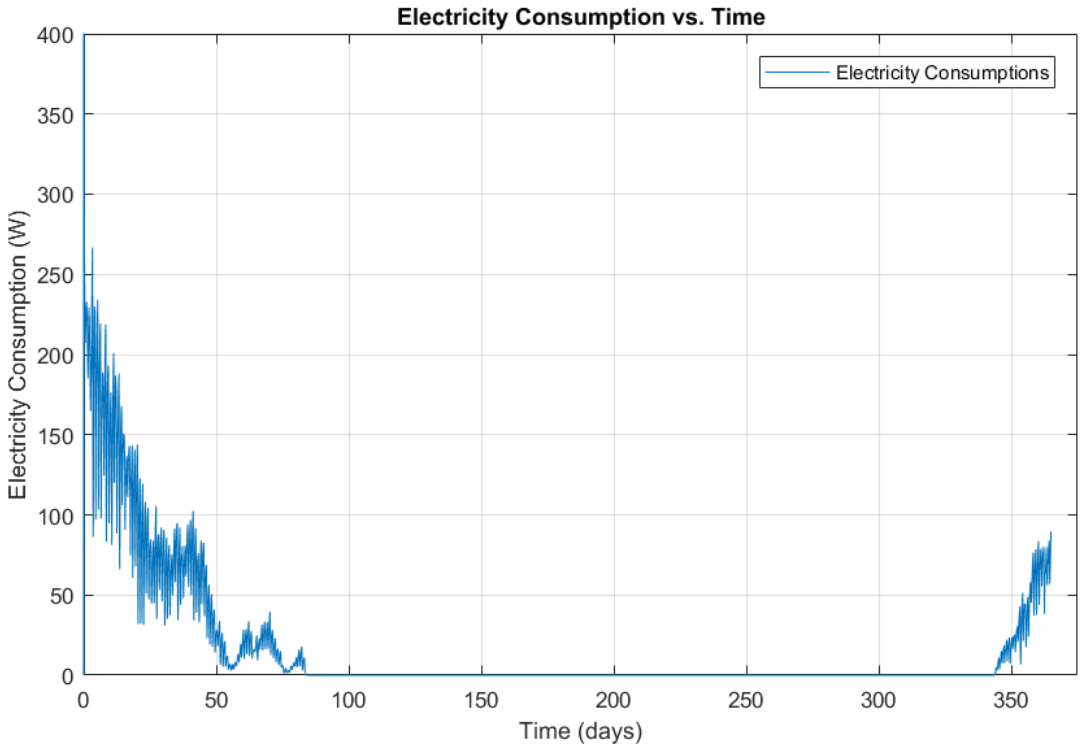

Figure 5.44: Plot of Electricity Consumption vs. Time

Chapter 6

Multi-Objective Trade-Off Analysis

6.1 Overview

In the previous section, creating a single *Instance* of the system model was discussed using a single set of design parameters. However, to be able to perform a meaningful trade-off analysis, several such instances need to be simulated and the results compared using multi-objective optimization techniques in order to satisfy the objective function.

Consol Optcad is one such multi-criteria optimization tool that uses a Feasible Sequential Quadratic Programming (FSQP) algorithm that would be best suited for a project like this [22]. The biggest advantage of this tool is its ability to change the parameters during the simulation in order to satisfy the constraints of the objective function, after having provided an initial parameter set. Another major benefit of such a tool is its ability to handle non-linear objective functions. Dimitrios Spyropoulos was able to utilize this tool in his thesis work to perform a multi-objective tradeoff analysis for an electric micro-grid system [23]. Despite making continued efforts to recreate Spyropoulos's work to integrate Consol Optcad with SysML for performing the trade-off analysis in this project, it couldn't be performed for a variety of reasons. Mainly because of a lack of comprehensive documentation of the integration procedure as well as a lack of time, resources and expertise to perform the integration from the beginning.

Yet, to demonstrate the power of the method, the trade-off analysis was performed in Matlab.

6.2 Pareto Frontier Analysis

For this system, since there are multiple conflicting metrics that define the quality of the system, a multi-objective trade-off analysis needs to be conducted in order to satisfy the objective function. One method of doing this analysis in by performing a Pareto Frontier Analysis. Taking various configurations of the design parameters of the system and performing a Pareto Frontier Analysis would provide a set of Pareto Optimal points that form the frontier and also demarcate the dominated region in the feasible solution set.

The first step to performing this analysis was to select discrete values of the previously defined design parameters and creating all possible configurations of them. Thus three levels of discrete values were chosen from the design parameters as shown in Table 6.1. The nominal values for all the parameters except window area correspond to the low values in the table. The nominal value for the Window Area was actually 3 m^2, thus corresponding with the high value as shown in the table.

	Outer wall Wood fiber insulation thickness	**Intermediate wall Wood fiber insulation thickness**	**Ceiling Wood fiber insulation thickness**	**Window Area**	**Indoor Temperature Setpoint**
Low	0.16 m	0.1 m	0.255 m	1 m^2	20 °C
Medium	0.24 m	0.2 m	0.3825 m	2 m^2	21 °C
High	0.32 m	0.3 m	0.510 m	3 m^2	22 °C

Table 6.1: Table of 5 Design Parameters and Discrete Value Levels

Having five metrics with three levels of values each would yield $3^5 = 243$ different configurations for which the associated metric values would have to be computed. Running a test simulation with one of these configuration sets showed that this would be very time prohibitive on the machine on which it was being run, taking into account its meager capability specifications. Thus, to cut down on the number of the configurations, it was decided that the Intermediate Wall Wood Fiber Insulation Thickness was not as important as the other design parameters since it was internal to the house and wouldn't affect the metrics to a large extent. Thus, this parameter was cut from the list, leaving us with a new configuration table of 4 design parameters with three discrete levels each. This table is shown in Table 6.2. This amounted to $3^4 = 81$ design configurations.

	Outer wall Wood fiber insulation thickness	Ceiling Wood fiber insulation thickness	Window Area	Indoor Temperature Setpoint
Low	0.16 m	0.255 m	1 m^2	20 °C
Medium	0.24 m	0.3825 m	2 m^2	21 °C
High	0.32 m	0.510 m	3 m^2	22 °C

Table 6.2: Table of 4 Design Parameters and Discrete Value Levels

6.2.1 Solving the Design Configurations

To solve the design configurations and find the value for each metric for every configuration, a simple automation tool in the Cameo Systems Modeler ParaMagic Plugin was used. This feature is called the ParaMagic Trade Study. ParaMagic uses the Excel Connection feature to set up the different initial parameter sets or "scenarios" as rows in a spreadsheet. Parameter sets are automatically read, the model is repeatedly executed, and output values written back to the spreadsheet [24].

6.2.2 Results of Pareto Frontier Analysis

Now that the each metric value was calculated for all the design configurations, a Pareto Frontier Analysis could be performed on the data. This was done by executing a fairly straightforward Pareto Frontier subroutine in Matlab. This Matlab script can be found in Section A.3.1 of the Appendix Section.

The objective function used in the Matlab script was supposed to minimize four of the metrics and maximize one. This is summarized in Table 6.3. The Variance of Indoor Air Temperature was minimized since it is desired for the indoor temperature to stay as close to the user-defined setpoint as possible, thereby making this a classic regulation

problem. Next, the Heat Pump Heat-load Per Annum was minimized to ensure that the heat pump isn't being used more than necessary. Since it was very electricity intensive to run and was designed to be a backup to the solar system, minimizing the amount of heating produced by the heat pump was a desirable trait. It was also desired that the heating provided by the solar thermal system be maximized, so the Solar-thermal Heat-load Per Annum metric was maximized. The Cost of Electricity consumed Per Annum metric was also minimized in order to reduce the operating cost of the system. Finally, the Cost of the Thermal Insulation used in the building construction was also minimized in order to keep the first costs of constructing the house low.

S No.	**System Metric**	**Units**	**Minimize or Maximize**
1.	Variance of Indoor Air Temperature	K^2	Minimize
2.	Heat Pump Heat-load Per Annum	kWh	Minimize
3.	Solar-thermal Heat-load Per Annum	kWh	Maximize
4.	Cost of Electricity consumed Per Annum	$	Minimize
5.	Cost of Thermal Insulation	$	Minimize

Table 6.3: Table of 5 System Metrics and Associated Optimization Actions

This script yielded that all 81 possible configurations were Pareto Optimal points. This was a much unexpected result and created cause for doubting the correctness of the analysis performed in the Matlab script. However, the script was verified to be correct and was retested with several other examples outputting correct results. This

meant that the analysis was indeed correct and all the possible configuration were Pareto Optimal Solutions.

The only reason that could lead to such a result was that there were very few data points for the number of metrics being analyzed. Since obtaining more data points was time-prohibitive, it was decided that since some metrics were dependent on others, their removal from the objective function should not be of much significance to the optimization problem. So the metric, Solar-Thermal Heat-load per annum was eliminated since maximizing it meant that the Heat Pump Heat-Load would be minimized. This is because of the total heat load required to keep the house warm is algebraic sum of the Solar-Thermal Heat-Load and the Heat Pump Heat Load. The other metric eliminated from the list was the Cost of Electricity consumed Per Annum. This is because, in the simplistic Dymola model of the house, the only appliance that consumed any electricity was the heat pump itself. Thus, minimizing the Heat Pump Heat-Load was essentially equivalent to minimizing the electricity consumption of the house.

Thus, the updated metrics that would be a part of the objective function in the Pareto Frontier Analysis are shown in Table 6.4

S No.	**System Metric**	**Units**	**Minimize or Maximize**
1.	Variance of Indoor Air Temperature	K^2	Minimize
2.	Heat Pump Heat-load Per Annum	kWh	Minimize
3.	Cost of Thermal Insulation	$	Minimize

Table 6.4: Table of 3 System Metrics and Associated Optimization Actions

Thus, solving the simulation for the aforementioned metrics yielded the following results as tabulated in Appendix Section A.3.2. The highlighted rows are the Pareto Points. The Pareto Analysis using these three metrics yielded 46 Pareto Optimal points as shown in Figure 6.1.

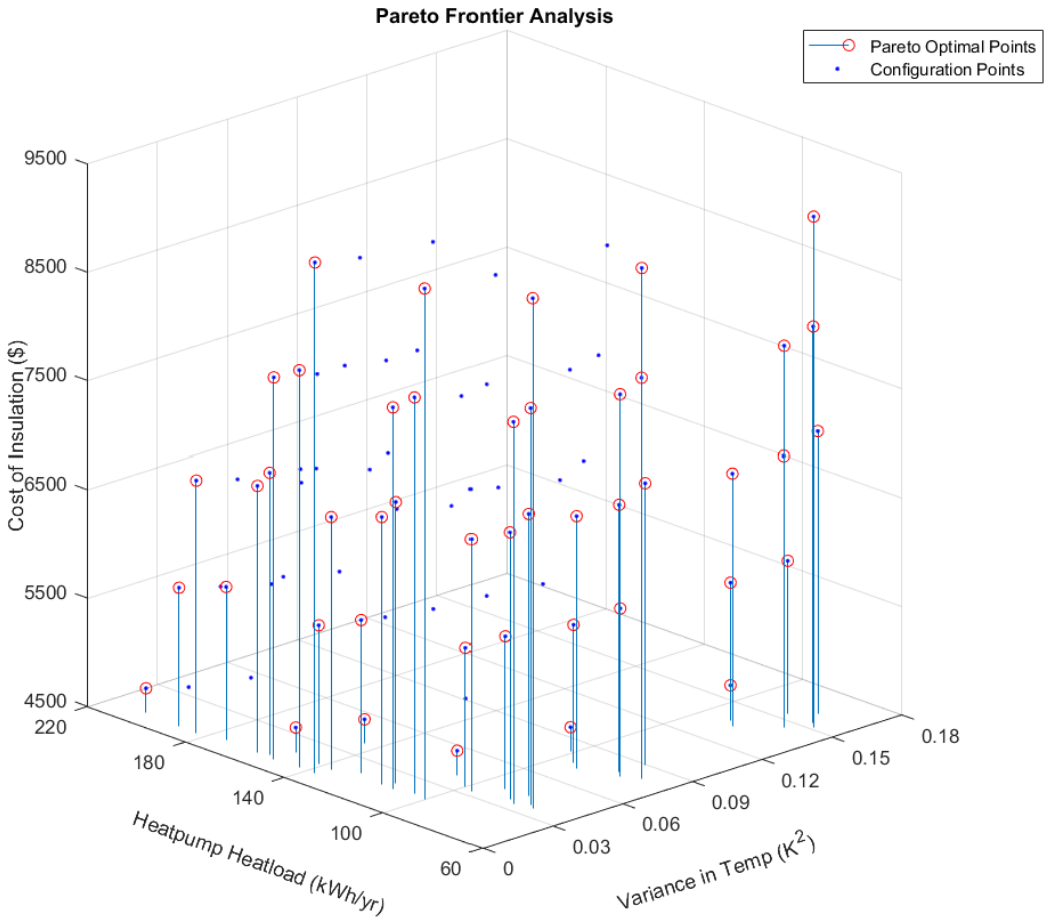

Figure 6.1: Pareto Optimal Solutions

If the above 3D plot is rotated as shown in Figure 6.2, a very clear Pareto Frontier Surface can be seen. Also, an interesting clustering of the various configurations is observed.

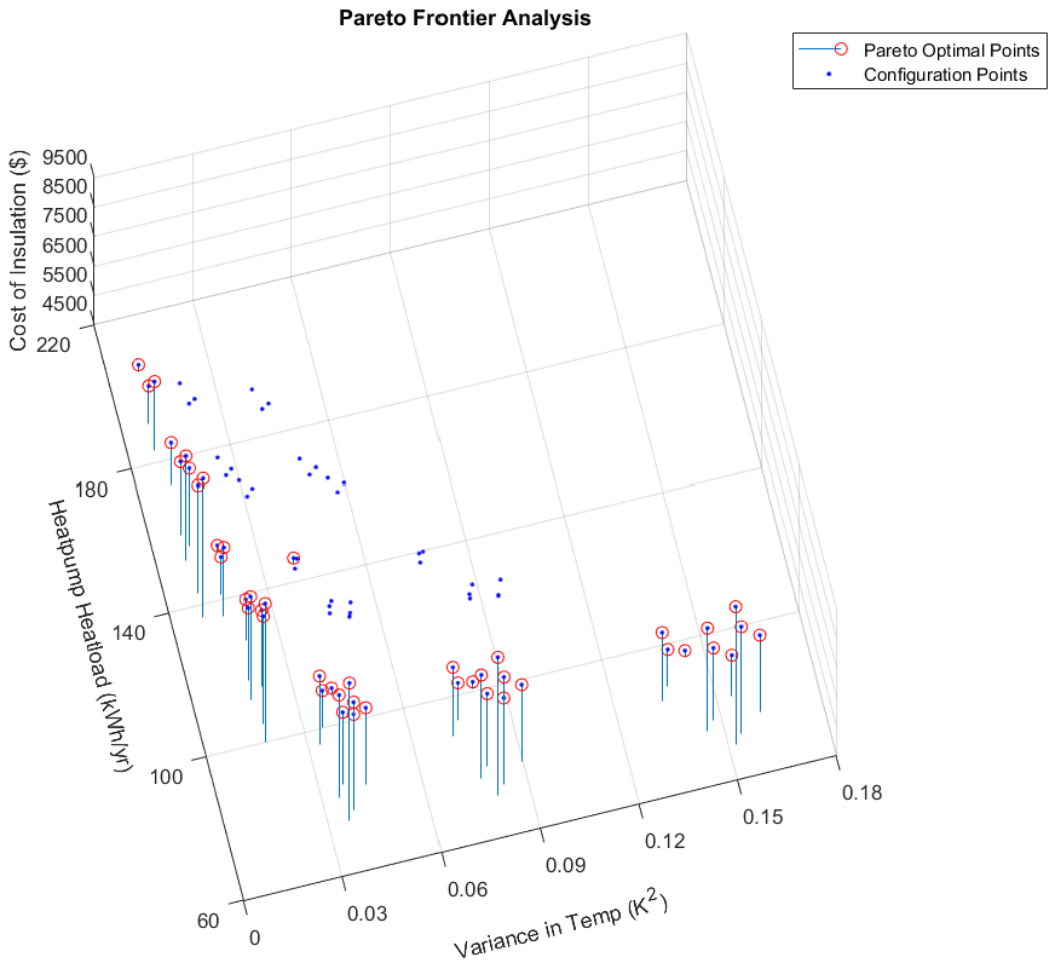

Figure 6.2: Pareto Optimal Solutions – Tilted View

However, out of 81 possible design configuration points, having 46 Pareto Optimal points isn't very helpful from a trade-off analysis point of view. This is because, if such a problem is presented to customer, he should have to choose from a set of 46 Pareto Points. Thus, some pruning of the Pareto Points was required to reduce the number of the choices to be presented. The first method of reducing the number of Pareto Optimals was simply by trimming the extreme values in all three dimensions. Since all three of the metrics have to be minimized, the numerically higher values of each metric could be deleted. Thus, any Pareto Point in the top 60% of the values of the Variance of the

Indoor Temperature, or the top 60% of the values of the Heat Pump Heat-Load, or the top 85% of the Cost of Insulation was eliminated. This reduced the Pareto Points from 46 to only 12 as shown in Figure 6.3.

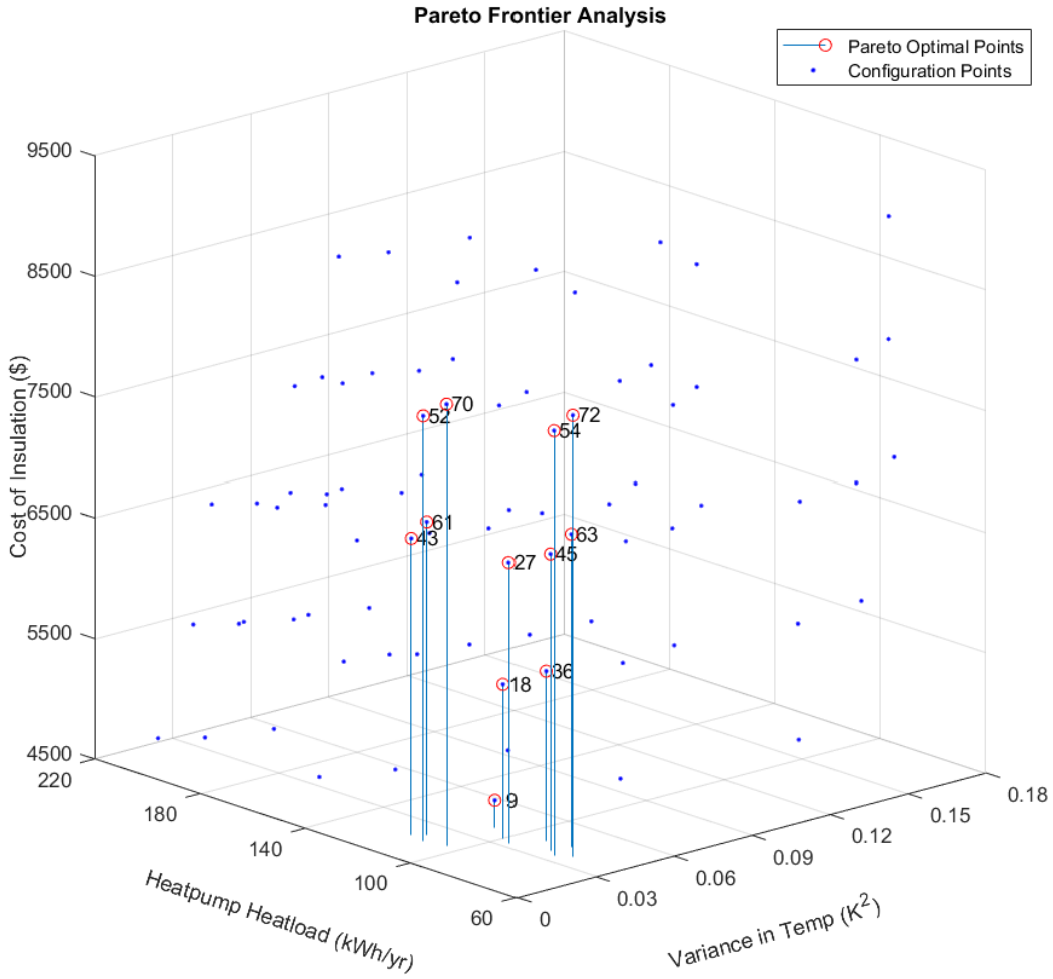

Figure 6.3: Pareto Optimal Solutions After First Elimination

The next level of thinning down the Pareto Points was by finding points very close to each other in clusters and picking only one from each cluster to represent the whole cluster as a single design configuration. To aid in this, each of the remaining Pareto Points was assigned an ID consistent with the design configurations that produced them from the table in Appendix Section A.3.2. From the 27-45-63 cluster, point # 45 was a

midpoint of amongst the three points in terms of all the three metrics. Thus point # 27 and # 63 were eliminated and # 45 was retained. From the 18-36 group, point # 18 was eliminated as it had a significantly higher value for the heat-load for roughly the same cost of insulation and variance of temperature. In a similar method, points # 72, #52, and #61 were also eliminated, leaving us with 6 Pareto Points as shown in Figure 6.4.

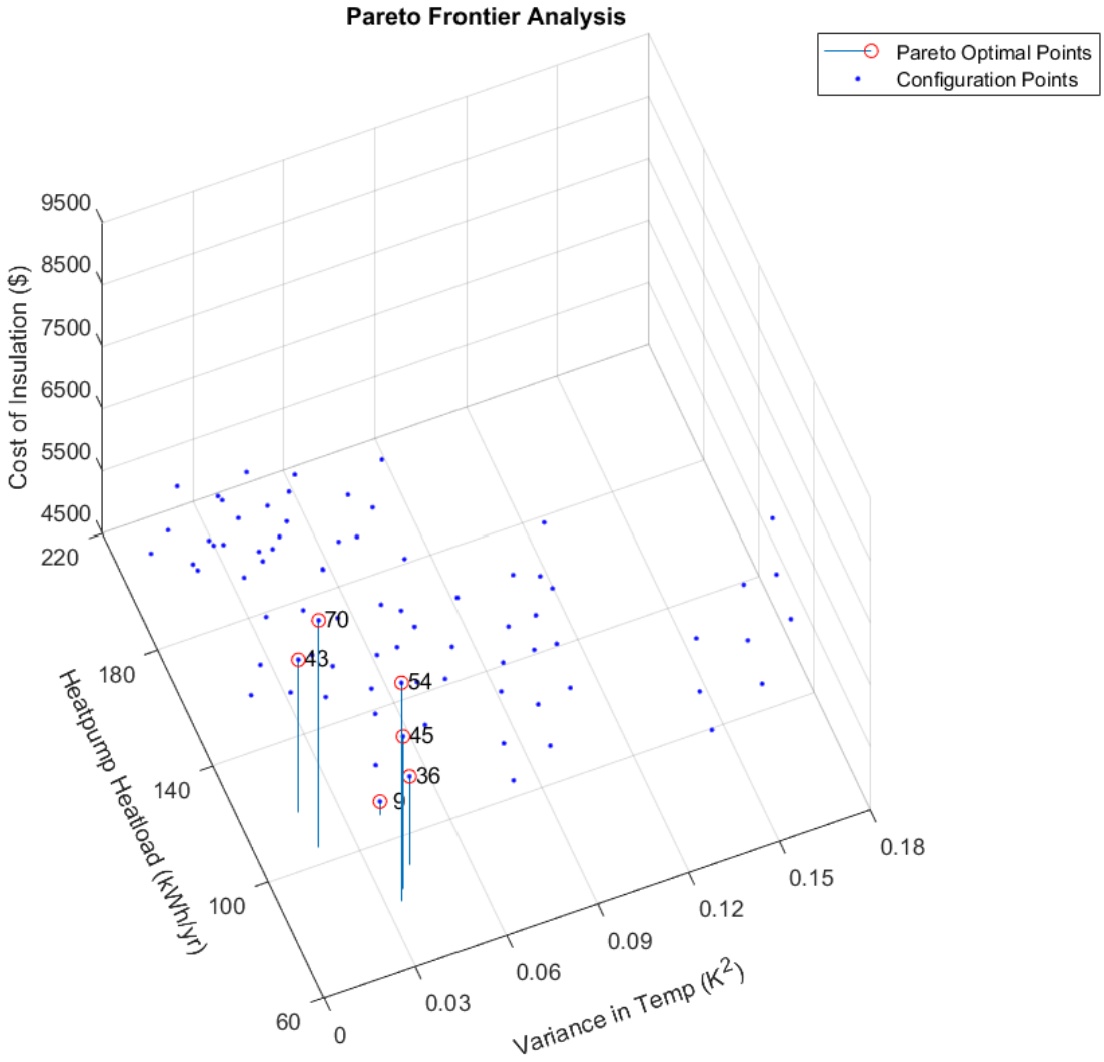

Figure 6.4: Pareto Optimal Solutions After Second Elimination

From these last six points, point # 70 and # 54 were the last to be eliminated as they provided a very small increment in the reduction of the heat-load and variance of indoor

temperature in comparison to point # 43 and # 45 respectively while costing close to $1200 more. Thus, having eliminated these 2 points, the final 4 points and their respective design configurations and metric values are shown in Table 6.5. Points # 9, # 36 and # 43 provide a good spread between the three metrics. Each of these points favors a particular metric and is a fair representation of the possible choices a customer would want to make based on the customer's preferences. Point # 9 could be chosen if the customer wants to spend the least amount of money on the thermal insulation of the house. Point # 36 could be chosen if the customers prioritizes the minimization of the heat-load, and point # 43 could be chosen if the customer wants to have the least variance in the indoor temperature, thereby prioritizing comfort. Lastly, point # 45 is a potential choice if the customer wants to minimize the heat-load and variance of indoor temperature, while willing to pay a significantly higher upfront cost for the extra thermal insulation. An interesting point to make note of is that for all four of these points, the Window Area is 1 m^2, smallest window area value from the discrete values chosen.

	Parameters				**Metrics**		
ID	**OW Insulation**	**C Insulation**	**Window Area**	**Temp Setpoint**	**Variance in TAir**	**Heat pump Heat load**	**Cost of Insulation**
9	0.16 m	0.255 m	1 m^2	20 °	0.041 K^2	108.18 kWh	$ 4727
36	0.24 m	0.255 m	1 m^2	20 °	0.042 K^2	90.76 kWh	$ 5915
43	0.24 m	0.3825 m	1 m^2	21 °	0.018 K^2	117.31 kWh	$ 6967
45	0.24 m	0.3825 m	1 m^2	20 °	0.037 K^2	83.91 kWh	$ 6967

Table 6.5: Table of Finalized Design Options and Associated Parameter and Metric Values

Chapter 7

Conclusions and Future Work

7.1 Summary of Thesis Work Performed

In conclusion, the major contribution of this thesis was that different methods to perform an integration with SysML and a simulation tool were identified, described and evaluated. Then, a new method was developed and discussed. Finally, the new method was demonstrated by developing an executable SysML model of a two-room house that utilizes solar-thermal power for the purpose of space heating with a heat pump being used as a backup. Using the FMI 2.0 standard, a Modelica model of the house was integrated with the SysML model using Simulink as an intermediate interface, enabling users to perform a tradeoff analysis by varying design parameters through the SysML interface.

Cameo Systems Modeler is used to create systems architecture of the two-room house in SysML. Dymola is used to create a multi-domain Modelica model of the same two-room energy efficient house. The Modelica model is then exported as FMU. The FMU is just a “skeletal structure” of the Modelica model.

Next, The FMU was imported into Simulink. Simulink didn’t add anything new to the model, but merely acts as a shell or an interface to the FMU. Simulink was chosen as this interface since Cameo Systems Modeler and Matlab/Simulink have an existing integration that could be exploited for this usage.

User-defined design parameter values that were inputted into the SysML model were sent to Simulink. With these values, Simulink ran the FMU, calculated the values of the output metrics sent them back to SysML to be displayed back to the user.

After the completion of the integration procedure of the SysML architecture with the FMU model of the two-room house, a multi-objective trade-off analysis was conducted. A Pareto Analysis was performed to identify the Pareto Frontier. Considering the initially discussed five metrics, all 81 design configurations turned out to be Pareto Points. Thus, the five metrics were cut down to three thereby yielding 46 Pareto points. Using various methods, these Pareto Points were trimmed down to 4 points that provided a good spread between the three metrics. Each of these points favored a particular metric that could be representative of a potential customer's preferences.

7.2 Evaluation of Integration Framework and Future Work

Although the integration of SysML with a modeling and simulation tool for the purpose reducing inconsistencies, enabling automation and supporting early and continual verification by analysis, was successful, the method of integration has a large scope for improvement. Since there was no way to simulate the FMU in Cameo Systems Modeler itself, the usage of Simulink as an intermediate model between SysML and the FMU caused the overall procedure to be fairly "clunky". This was very evident in the part where string manipulations to replace the default values for the design parameters, embedded in the FMU, with the new input parameters specified by the user in SysML, had to be performed in the Matlab script responsible for calling the

Simulink model. If no changes are made to the FMU for any part of this integration, the string manipulations would not pose any problems. However, even if a single change is made to the FMU through Dymola, the string manipulations would have to performed again to ensure that the right design parameter inputted by the user is mapped to the right variable residing in the FMU.

Next, the ParaMagic plugin that even made this integration possible in the first place also proves to be a bottleneck to its capabilities in some instances. The major drawback of this tool is observed when any constrain block in the SysML architecture calls a Matlab function or script to evaluate a system metric value. Although the Matlab script is capable to calculate multiple metric values in the same script, only a single metric value could be outputted back into the SysML model to be displayed to the user. Hence, for different system metrics to be computed by a single Matlab script, the same script had to be solved repeatedly, each outputting the value of a different metric. In this thesis, this issue was overcome by setting the first Matlab script as the main calculation module and setting the subsequent Matlab scripts to just pull the calculated values of different system metrics from the first Matlab script. However, it was still a major drawback of the ParaMagic tool and a cause for inefficiency in the integration procedure.

As for finding a better integration technique in the future, the FMU import Cameo Systems Modeler described in Section 4.2 of this thesis shows the most promise. This method is still in development and if successfully completed, will eliminate the need to use an intermediate modeling tool like SysML to execute FMUs. The Model-Exchange enabled FMU could be directly imported into the SysML architecture of any

system and could interact with the rest of the system architecture in a streamlined manner.

Another area worth exploring is the recreation of the OMG SysML-Modelica Transformation Specification mention in Section 4.1. For seemingly unknown reasons, this OMG standard is not compatible with the latest versions of Cameo Systems Modeler or Dymola. It would be of great value if the transformation was brought back to life, allowing for Modelica models to be transformed into SysML constructs. This method would in-fact eliminate the need for the FMU block altogether as the Modelica model would itself transform into SysML artifacts preserving all the data, equations, constraints and connections that are part of the Modelica model. However, this would only be most useful to those who are only working with SysML and Modelica as this method doesn't help with the integration of other modeling tools.

Appendices

A.1 Additional SysML Model Diagrams

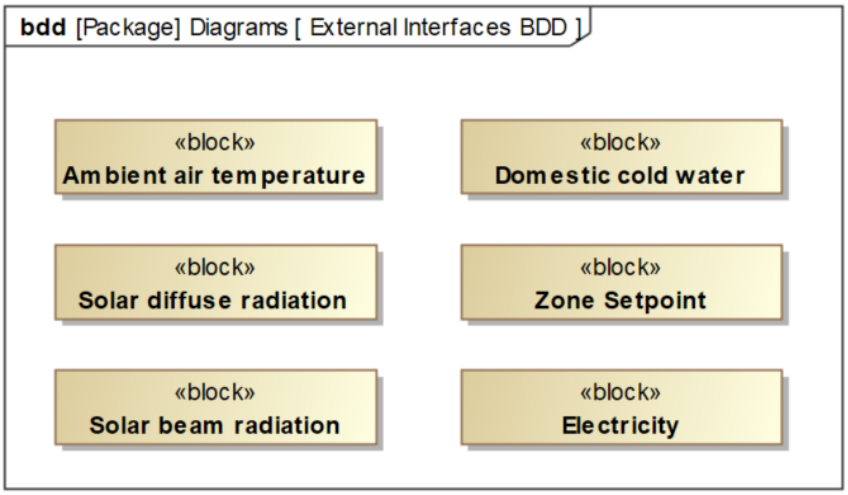

Figure A.1: External Interface BDD

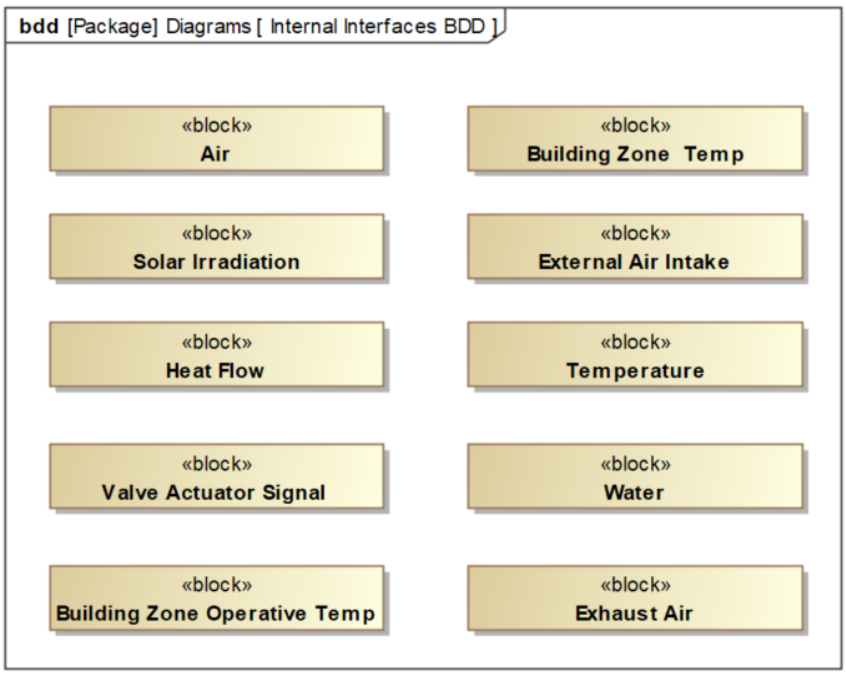

Figure A.2: Internal Interface BDD

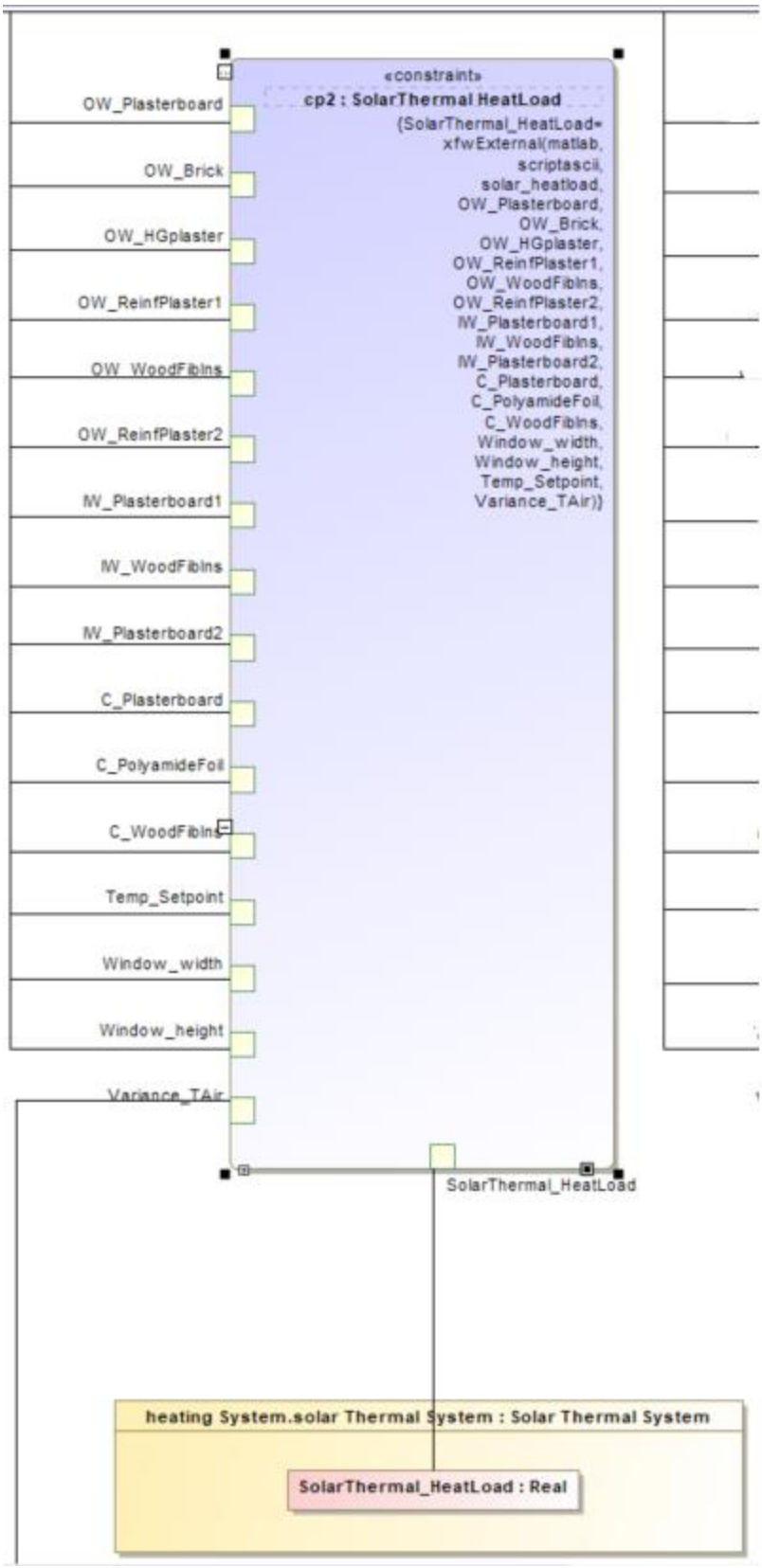

Figure A.3: Parametric Diagram – Constraint 2

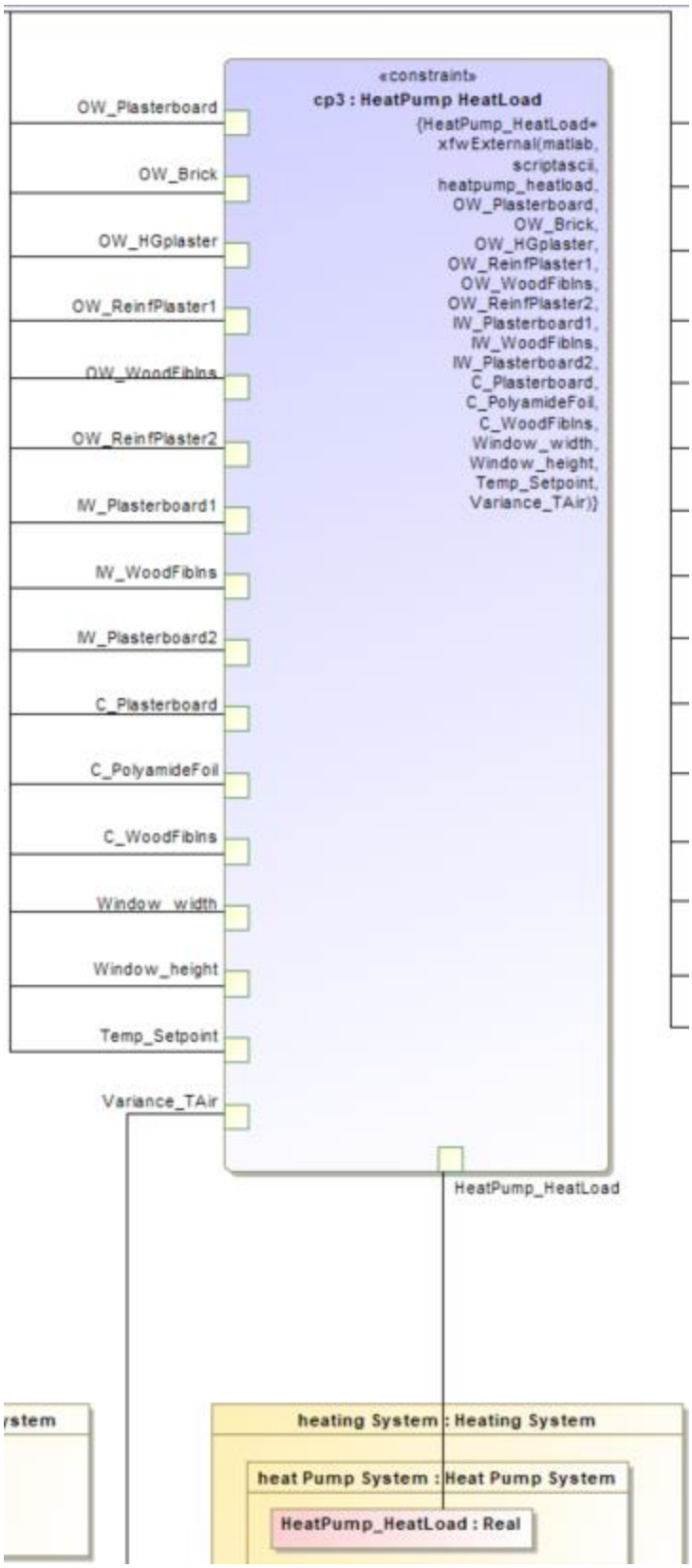

Figure A.4: Parametric Diagram – Constraint 3

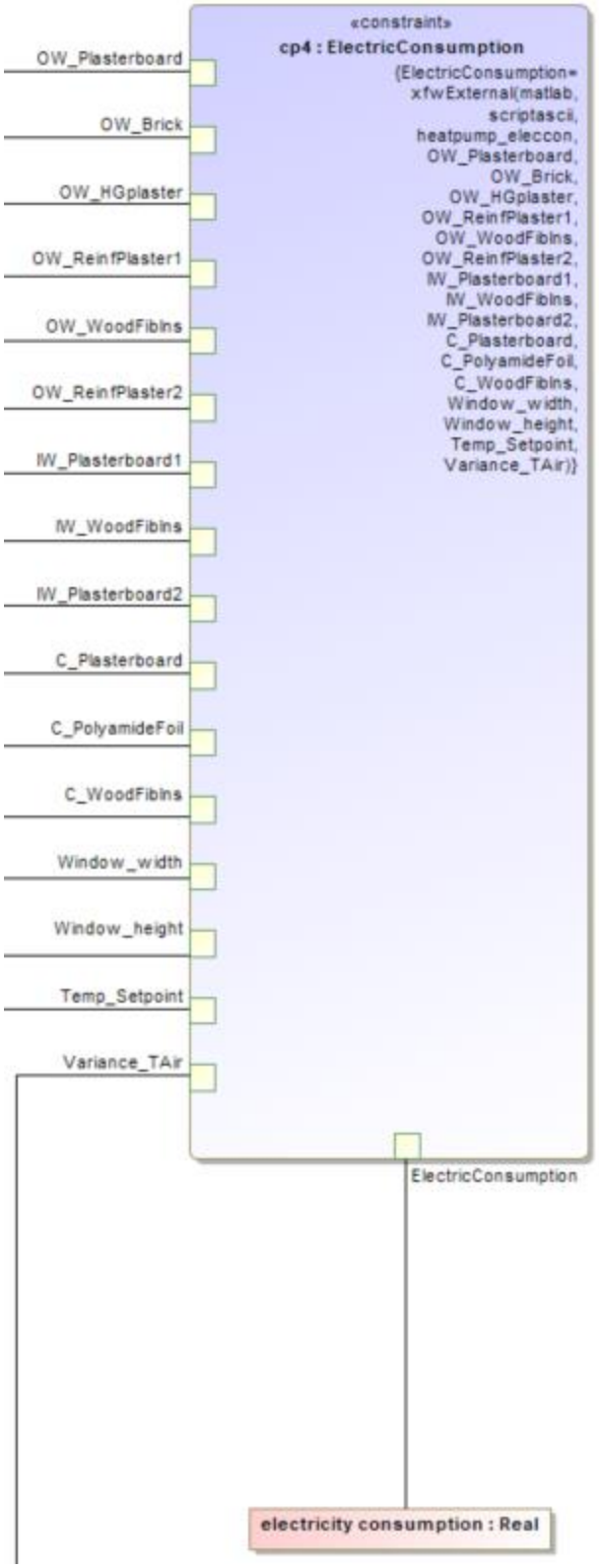

Figure A.5: Parametric Diagram – Constraint 3

A.2 Matlab Scripts used in the SysML Model of the Two-Room House

A.2.1 Constraint Block Matlab Script (`exec_script.m`)

```
%% Executable script
% This script is run through MagicDraw Cameo Systems Modeler. It
% essentially calls the simulink model for the Energy Efficient Home system
% which is being modelled in SysML in Cameo Systems Modeler.

clear
clc

%% Inputs into Simulink

% The following inputs are generated by Paramagic and stored in a text file
% called "input.txt" in the project directory. It contains the inputs to
% the constraints as mentioned in the constraint block in the SysML model
% in the order that is stated in the "xfwExternal" function.


% In this case is the inputs into the constraint equations are:
% pos = xfwExternal(matlab, scriptascii, exec_script, OW_Plasterboard,
% OW_Brick, OW_HGplaster, OW_ReinfPlaster1, OW_WoodFibIns,
% OW_ReinfPlaster2, IW_Plasterboard1, IW_WoodFibIns, OW_ReinfPlaster2,
% IW_Plasterboard1, IW_WoodFibIns, IW_Plasterboard2, C_Plasterboard,
% C_PolyamideFoil, C_WoodFibIns, Window_width, Window_height)


% OW_Plasterboard   = Thickness of Outer wall Plasterboard 1
% OW_Brick          = Thickness of Outer wall Brick
% OW_HGplaster      = Thickness of Outer wall High grade plaster
% OW_ReinfPlaster1  = Thickness of Outer wall Reinforced plaster
% OW_WoodFibIns     = Thickness of Outer wall Wood Fiber Insulation
% OW_ReinfPlaster2  = Thickness of Outer wall Plasterboard 2
%
% IW_Plasterboard1  = Thickness of Intermediate wall Plasterboard 1
% IW_WoodFibIns     = Thickness of Intermediate wall Wood Fiber Insulation
% IW_Plasterboard2  = Thickness of Intermediate wall Plasterboard 2
%
% C_Plasterboard    = Thickness of Ceiling Plasterboard
% C_PolyamideFoil   = Thickness of Ceiling Polyamide Foil
% C_WoodFibIns      = Thickness of Ceiling Wood FIber Insulation
%
% Window_width      = Width of window
% Window_height     = Height of Window
%
% Temp_Setpoint

% These values are stored in the Matlab workspace and will be called by the
% Simulink model which will also be launched subsequently in this script.

inSel= load('input.txt');

OW_Plasterboard   = inSel(1);
OW_Brick          = inSel(2);
OW_HGplaster      = inSel(3);
OW_ReinfPlaster1  = inSel(4);
OW_WoodFibIns     = inSel(5);
OW_ReinfPlaster2  = inSel(6);
```

```
IW_Plasterboard1  = inSel(7);
IW_WoodFibIns     = inSel(8);
IW_Plasterboard2  = inSel(9);

C_Plasterboard    = inSel(10);
C_PolyamideFoil   = inSel(11);
C_WoodFibIns      = inSel(12);

Window_width      = inSel(13);
Window_height     = inSel(14);

Temp_Setpoint     = inSel(15);


%% Open the Simulink Model
% The following commands launch the simulink model that you want to
% integrate as the "black box" constraint block in SysML Cameo.

mdl='exec_model';
open_system(mdl);
open_system([mdl '/Scope1']);
open_system([mdl '/Scope2']);
open_system([mdl '/Scope3']);

%% Obtain the default parameters
% This function obtains the default parameters for the entire model from
% the FMU that are now stored in the Simulink model. Subsequently, these
% parameters will be replaced with new parameters as specified in the
% input.txt file that is outputted by Cameo Systems Modeler.

old_param = get_param('exec_model/executable_fmu','parameters');

%% String Manipulation
% In this section, using string manipulation methods, the old paramter
% string will be edited to include the new parameters as mentioned by the
% input.txt file.
%
% First, we convert char to string for the purpose of string
% manipulation. Then the string is split into a cell array and the relevant
% cells are edited. The Cell array is then converted back to a string and
% then chars and fed back into the Simulink model to simulate.
old_param_str = string(old_param)

C = strsplit(old_param_str);


%%
% Outer Wall Construction Data Parameters - Reassignment
[C(1495), C(1527), C(1559), C(1591), C(1622), C(1653)] =
deal(OW_Plasterboard);
[C(1496), C(1528), C(1560), C(1592), C(1623), C(1654)] = deal(OW_Brick);
[C(1497), C(1529), C(1561), C(1593), C(1624), C(1655)] = deal(OW_HGplaster);
[C(1498), C(1530), C(1562), C(1594), C(1625), C(1656)] =
deal(OW_ReinfPlaster1);
[C(1499), C(1531), C(1563), C(1595), C(1626), C(1657)] =
deal(OW_WoodFibIns);
[C(1500), C(1532), C(1564), C(1596), C(1627), C(1658)] =
deal(OW_ReinfPlaster2);

% Intermediate Wall Construction Data Parameters - Reassignment
C(1469) = IW_Plasterboard1;
C(1470) = IW_WoodFibIns;
```

```
C(1470) = IW_Plasterboard2;

% Ceiling Construction Data Parameters - Reassignment
[C(1685), C(1711)] = deal(C_Plasterboard);
[C(1686), C(1712)] = deal(C_PolyamideFoil);
[C(1687), C(1713)] = deal(C_WoodFibIns);

% Window Parameters - Reassignment
[C(1774), C(1789)] = deal(Window_width);
[C(1775), C(1790)] = deal(Window_height);

% Temperature Setpoint (should be in Kelvin)
C(1894) = Temp_Setpoint + 273.15;


% % C(33) = strcat(num2str(k),']');

new_param_str = strjoin(C);

new_param_char = char(new_param_str)

set_param('exec_model/executable_fmu','parameters' ,new_param_char);

set_param(mdl,'SimulationCommand','Update')

%% Run simulation within Simulink

% This command runs the simulation from within simulink and then goes on
% the plot the results as shown in the Scope Block called "Scope".

evalc('sim(mdl)');

%% Output ascii text file for SysML to read.

% These commands output the values of the LHS part of the constraint
% equation in SysML into a text file called "output.txt". This text file is
% then read by the Paramagic plugin in Cameo, which subsequently takes the
% data and prints as the values that you have set as a "target" in the
% Paramagic browser.

% output = yout(size(yout,1),1);

TAir = yout(:,1);
logical_count = TAir > 294;
Var_TAir = var(TAir);

% SolarThermal_HeatFlow = yout(:,4);
% SolarThermal_HeatLoad = trapz(SolarThermal_HeatFlow)/(1000*60*60);

save('yout_datafile.mat','yout')
save('output.txt','Var_TAir','-ASCII');
bdclose
exit
```

A.2.2 Constraint Block Matlab Script (heatpump_eleccon.m)

```
clear
clc

load('yout_datafile')

HeatPump_ElectricConsumption_flow = yout(:,6);
HeatPump_ElectricConsumption =
trapz(HeatPump_ElectricConsumption_flow)/(1000*60*60);

save('output.txt','HeatPump_ElectricConsumption','-ASCII');
exit
```

A.2.3 Constraint Block Matlab Script (heatpump_heatload.m)

```
clear
clc

load('yout_datafile')

HeatPump_HeatFlow = yout(:,5);
HeatPump_HeatLoad = trapz(HeatPump_HeatFlow)/(1000*60*60);

save('output.txt','HeatPump_HeatLoad','-ASCII');
exit
```

A.2.4 Constraint Block Matlab Script (solar_heatload.m)

```
clear
clc

load('yout_datafile')

SolarThermal_HeatFlow = yout(:,4);
SolarThermal_HeatLoad = trapz(SolarThermal_HeatFlow)/(1000*60*60);

save('output.txt','SolarThermal_HeatLoad','-ASCII');
exit
```

A.3 Trade-Off Analysis

A.3.1 Matlab Script Used for Pareto Analysis (`Pareto_Analysis2.m`)

```
% This script reads the metrics in MS Excel generated by the Cameo Systems
% Modeler Trade-Study tool and performs a Pareto Analysis to find the
% Pareto points and eliminate the non-dominant solutions. since there are
% 5 metrics, the Pareto Points cannot be visualized in the form of a plot.
%
% The script will output the Configurations and the associated metric
% values of the Pareto Points.
%
% To run this script, the following files are required to be in the same
% folder:
% 1. paretofront.m
% 2. paretofront.c
% 3. paretofront.mexw32
% 4. paretofront.mexw64
% 5. Tradeoff Analysis.xlsx

clear
clc

Configs = xlsread('Tradeoff Analysis', 'Trade Study', 'A3:E83');
Metrics = xlsread('Tradeoff Analysis', 'Trade Study', 'H3:L83');

Var_TAir = Metrics (:,1);
SolarThermal_HeatLoad = Metrics (:,2);
HeatPump_HeatLoad = Metrics (:,3);
Cost_ElectricConsumption = Metrics (:,4);
Cost_Insulation = Metrics (:,5);

% Since the default for the paretofront function is to minimize, a -ve sign
% has to be added to SolarThermal_HeatLoad since we want to maximize that
% metric.

objective =[Var_TAir, HeatPump_HeatLoad, Cost_Insulation];

I = paretofront(objective);

% To find all the Non-Pareto points easily
NotI = logical(1-I);

figure
plot3(Var_TAir, HeatPump_HeatLoad, Cost_Insulation, 'b.',...
    Var_TAir(I), HeatPump_HeatLoad(I), Cost_Insulation(I), 'ro')

grid ON
axis([0, 0.18, 60,220, 4500, 9500])
xticks(linspace(0,0.18,7));
yticks(linspace(60,220,5));
zticks(linspace(4500,9500,6));
xlabel('Variance in Temp (K^2)')
ylabel('Heatpump Heatload (kWh/yr)')
zlabel('Cost of Insulation ($)')
title('Pareto Frontier Analysis')
legend('Configuration Points','Pareto Optimal Points')

% Axis Rotation Code snippet
```

```
h = rotate3d;
set(h, 'ActionPreCallback',
'set(gcf,''windowbuttonmotionfcn'',@align_axislabel)')
set(h, 'ActionPostCallback', 'set(gcf,''windowbuttonmotionfcn'','''')')
set(gcf, 'ResizeFcn', @align_axislabel)
align_axislabel([], gca)
axislabel_translation_slider;

%% Video Capture

% OptionZ.FrameRate=40;
% OptionZ.Duration=5.5;
% OptionZ.Periodic=true;
% CaptureFigVid([-20,20;-110,20;-190,20;-290,10;-380,20],
'WellMadeVid',OptionZ)

%%
% The first column of the ParetoConfigs is the ID number of the test case

ParetoConfigs = Configs(I,:);
ParetoMetrics = Metrics(I,:);

NonParetoConfigs = Configs(NotI,:);
NonParetoMetrics = Metrics(NotI,:);

% Number of Pareto Points
N = length(ParetoConfigs)

%% Additional Plotting techniques

% Met3 collects the thre metrics being plotted:
% Var_TAir, HeatPump_HeatLoad, Cost_Insulation

Met3 = [ParetoMetrics(:,1), ParetoMetrics(:,3), ParetoMetrics(:,5)];

xlin = linspace(min(Met3(:,1)),max(Met3(:,1)),30);
ylin = linspace(min(Met3(:,2)),max(Met3(:,2)),30);

[X,Y] = meshgrid(xlin,ylin);
f = scatteredInterpolant(Met3(:,1),Met3(:,2),Met3(:,3));
Z = f(X,Y);

figure
mesh(X,Y,Z) %interpolated
hold on
plot3(Met3(:,1),Met3(:,2),Met3(:,3),'ro',Var_TAir, HeatPump_HeatLoad,
Cost_Insulation, 'b.') %nonuniform
axis([0, 0.18, 60,220, 4500, 9500])
xlabel('Variance in Temp (K^2)')
ylabel('Heatpump Heatload (kWh/yr)')
zlabel('Cost of Insulation ($)')
title('Pareto Frontier Analysis')
legend('Configuration Points','Pareto Optimal Points')
hidden OFF

% Axis Rotation Code snippet
h = rotate3d;
set(h, 'ActionPreCallback',
'set(gcf,''windowbuttonmotionfcn'',@align_axislabel)')
set(h, 'ActionPostCallback', 'set(gcf,''windowbuttonmotionfcn'','''')')
set(gcf, 'ResizeFcn', @align_axislabel)
align_axislabel([], gca)
axislabel_translation_slider;
```

```
%%
figure
stem3(Met3(:,1),Met3(:,2),Met3(:,3),'MarkerEdgeColor','r')
hold on
plot3(Metrics(:,1), Metrics(:,3), Metrics(:,5),'b.')
hold off
axis([0, 0.18, 60,220, 4500, 9500])
xlabel('Variance in Temp (K^2)')
ylabel('Heatpump Heatload (kWh/yr)')
zlabel('Cost of Insulation ($)')
xticks(linspace(0,0.18,7));
yticks(linspace(60,220,5));
zticks(linspace(4500,9500,6));
title('Pareto Frontier Analysis')
legend('Pareto Optimal Points', 'Configuration Points')

grid on

% Axis Rotation Code snippet
h = rotate3d;
set(h, 'ActionPreCallback',
'set(gcf,''windowbuttonmotionfcn'',@align_axislabel)')
set(h, 'ActionPostCallback', 'set(gcf,''windowbuttonmotionfcn'','''')')
set(gcf, 'ResizeFcn', @align_axislabel)
align_axislabel([], gca)


%% First Pruning

for n = 1:length(Metrics)

    if Metrics(n,1) > ( min(Metrics(:,1)) + 0.4*(max(Metrics(:,1))-
min(Metrics(:,1))) )
        I(n) = 0;
    end

    if Metrics(n,3) > ( min(Metrics(:,3)) + 0.4*(max(Metrics(:,3))-
min(Metrics(:,3))) )
        I(n) = 0;
    end

    if Metrics(n,5) > ( min(Metrics(:,5)) + 0.8*(max(Metrics(:,5))-
min(Metrics(:,5))) )
        I(n) = 0;
    end
end
sum(I)

NotI = logical(1-I);

ParetoConfigs = Configs(I,:);
ParetoMetrics = Metrics(I,:);

NonParetoConfigs = Configs(NotI,:);
NonParetoMetrics = Metrics(NotI,:);

Met3 = [ParetoMetrics(:,1), ParetoMetrics(:,3), ParetoMetrics(:,5)];

figure
stem3(Met3(:,1),Met3(:,2),Met3(:,3),'MarkerEdgeColor','r')
hold on
plot3(Metrics(:,1), Metrics(:,3), Metrics(:,5),'b.')
```

```
hold off
axis([0, 0.18, 60,220, 4500, 9500])
xlabel('Variance in Temp (K^2)')
ylabel('Heatpump Heatload (kWh/yr)')
zlabel('Cost of Insulation ($)')
xticks(linspace(0,0.18,7));
yticks(linspace(60,220,5));
zticks(linspace(4500,9500,6));
title('Pareto Frontier Analysis')
legend('Pareto Optimal Points','Configuration Points')
grid on

% Axis Rotation Code snippet
h = rotate3d;
set(h, 'ActionPreCallback',
'set(gcf,''windowbuttonmotionfcn'',@align_axislabel)')
set(h, 'ActionPostCallback', 'set(gcf,''windowbuttonmotionfcn'','''')')
set(gcf, 'ResizeFcn', @align_axislabel)


ID = cellstr(num2str(Configs(I,1)));
dx = 0.002; dy = 0.1; dz = 0.1; % displacement so the text does not overlay
the data points
text(Met3(:,1) +dx ,Met3(:,2) + dy ,Met3(:,3) + dz,ID);


%% Second Pruning - Manual Pruning

[I(27), I(63), I(18), I(72), I(52), I(69), I(61), I(54), I(70)] = deal(0);

sum(I)

NotI = logical(1-I);

ParetoConfigs = Configs(I,:);
ParetoMetrics = Metrics(I,:);

NonParetoConfigs = Configs(NotI,:);
NonParetoMetrics = Metrics(NotI,:);

Met3 = [ParetoMetrics(:,1), ParetoMetrics(:,3), ParetoMetrics(:,5)];

figure
stem3(Met3(:,1),Met3(:,2),Met3(:,3),'MarkerEdgeColor','r')
hold on
plot3(Metrics(:,1), Metrics(:,3), Metrics(:,5),'b.')
hold off
axis([0, 0.18, 60,220, 4500, 9500])
xlabel('Variance in Temp (K^2)')
ylabel('Heatpump Heatload (kWh/yr)')
zlabel('Cost of Insulation ($)')
xticks(linspace(0,0.18,7));
yticks(linspace(60,220,5));
zticks(linspace(4500,9500,6));
title('Pareto Frontier Analysis')
legend('Pareto Optimal Points','Configuration Points')
grid on

% Axis Rotation Code snippet
h = rotate3d;
set(h, 'ActionPreCallback',
'set(gcf,''windowbuttonmotionfcn'',@align_axislabel)')
set(h, 'ActionPostCallback', 'set(gcf,''windowbuttonmotionfcn'','''')')
```

```
set(gcf, 'ResizeFcn', @align_axislabel)

ID = cellstr(num2str(Configs(I,1)));
dx = 0.002; dy = 0.1; dz = 0.1; % displacement so the text does not overlay
the data points
text(Met3(:,1) +dx ,Met3(:,2) + dy ,Met3(:,3) + dz,ID);
```

A.3.2 Table of Pareto Points.

	Parameters				Metrics		
ID	**OW Insulation**	**C Insulation**	**Window Area**	**Temp Setpoint**	**Variance in TAir**	**Heatpump Heat load**	**Cost of Insulation**
1	0.16	0.255	3	21	0.076	137.85	4727
2	0.16	0.255	3	22	0.040	191.25	4727
3	0.16	0.255	3	20	0.144	95.45	4727
4	0.16	0.255	2	21	0.039	144.67	4727
5	0.16	0.255	2	22	0.020	197.48	4727
6	0.16	0.255	2	20	0.081	100.86	4727
7	0.16	0.255	1	21	0.018	152.87	4727
8	0.16	0.255	1	22	0.009	204.97	4727
9	0.16	0.255	1	20	0.041	108.18	4727
10	0.16	0.3825	3	21	0.073	127.29	5779
11	0.16	0.3825	3	22	0.039	177.35	5779
12	0.16	0.3825	3	20	0.137	88.74	5779
13	0.16	0.3825	2	21	0.037	133.73	5779
14	0.16	0.3825	2	22	0.018	183.46	5779
15	0.16	0.3825	2	20	0.074	93.44	5779
16	0.16	0.3825	1	21	0.016	141.54	5779
17	0.16	0.3825	1	22	0.008	190.68	5779
18	0.16	0.3825	1	20	0.035	100.03	5779
19	0.16	0.51	3	21	0.072	121.89	6831
20	0.16	0.51	3	22	0.039	170.30	6831
21	0.16	0.51	3	20	0.134	85.45	6831
22	0.16	0.51	2	21	0.036	128.13	6831
23	0.16	0.51	2	22	0.018	176.24	6831
24	0.16	0.51	2	20	0.072	89.74	6831
25	0.16	0.51	1	21	0.016	135.74	6831
26	0.16	0.51	1	22	0.008	183.36	6831
27	0.16	0.51	1	20	0.033	95.92	6831
28	0.24	0.255	3	21	0.084	114.90	5915
29	0.24	0.255	3	22	0.046	160.93	5915
30	0.24	0.255	3	20	0.155	82.10	5915
31	0.24	0.255	2	21	0.043	120.61	5915
32	0.24	0.255	2	22	0.022	166.47	5915
33	0.24	0.255	2	20	0.086	85.50	5915
34	0.24	0.255	1	21	0.020	127.78	5915
35	0.24	0.255	1	22	0.010	173.36	5915
36	0.24	0.255	1	20	0.042	90.76	5915
37	0.24	0.3825	3	21	0.082	105.64	6967
38	0.24	0.3825	3	22	0.045	148.01	6967
39	0.24	0.3825	3	20	0.148	77.08	6967

40	0.24	0.3825	2	21	0.041	110.66	6967
41	0.24	0.3825	2	22	0.021	153.23	6967
42	0.24	0.3825	2	20	0.079	79.60	6967
43	0.24	0.3825	1	21	0.018	117.31	6967
44	0.24	0.3825	1	22	0.009	159.69	6967
45	0.24	0.3825	1	20	0.037	83.91	6967
46	0.24	0.51	3	21	0.081	101.06	8019
47	0.24	0.51	3	22	0.045	141.45	8019
48	0.24	0.51	3	20	0.145	74.56	8019
49	0.24	0.51	2	21	0.040	105.72	8019
50	0.24	0.51	2	22	0.021	146.47	8019
51	0.24	0.51	2	20	0.077	76.78	8019
52	0.24	0.51	1	21	0.017	112.02	8019
53	0.24	0.51	1	22	0.009	152.73	8019
54	0.24	0.51	1	20	0.035	80.56	8019
55	0.32	0.255	3	21	0.090	103.68	7103
56	0.32	0.255	3	22	0.050	145.00	7103
57	0.32	0.255	3	20	0.161	76.42	7103
58	0.32	0.255	2	21	0.046	108.42	7103
59	0.32	0.255	2	22	0.024	150.06	7103
60	0.32	0.255	2	20	0.090	78.70	7103
61	0.32	0.255	1	21	0.021	114.78	7103
62	0.32	0.255	1	22	0.011	156.38	7103
63	0.32	0.255	1	20	0.044	82.55	7103
64	0.32	0.3825	3	21	0.087	95.39	8155
65	0.32	0.3825	3	22	0.049	132.69	8155
66	0.32	0.3825	3	20	0.154	71.72	8155
67	0.32	0.3825	2	21	0.043	99.38	8155
68	0.32	0.3825	2	22	0.023	137.39	8155
69	0.32	0.3825	2	20	0.083	73.74	8155
70	0.32	0.3825	1	21	0.019	105.02	8155
71	0.32	0.3825	1	22	0.010	143.31	8155
72	0.32	0.3825	1	20	0.038	76.72	8155
73	0.32	0.51	3	21	0.087	91.40	9207
74	0.32	0.51	3	22	0.050	126.50	9207
75	0.32	0.51	3	20	0.152	69.25	9207
76	0.32	0.51	2	21	0.043	94.96	9207
77	0.32	0.51	2	22	0.023	130.96	9207
78	0.32	0.51	2	20	0.080	71.22	9207
79	0.32	0.51	1	21	0.018	100.13	9207
80	0.32	0.51	1	22	0.010	136.67	9207
81	0.32	0.51	1	20	0.036	73.91	9207

Bibliography

[1] S. Balestrini-Robinson, D. F. Freeman and D. C. Browne, "An Object-oriented and Executable SysML Framework for Rapid Model Development," *Procedia Computer Science,* vol. 44, p. 424, 2015.

[2] J. E. MacCarthy, *Approaches to Agile MBSE - 180326,* College Park, MD: University of Maryland, College Park, 2018.

[3] International Council on Systems Engineering (INCOSE), "Systems Engineering Vision 2020 (Version 2.03, TP-2004-004-02, September 2007)," INCOSE, 2007.

[4] "Model-Based Systems Engineering Overview," MBSE.Works, [Online]. Available: http://mbse.works/mbse-overview/. [Accessed 28 February 2018].

[5] No Magic Inc., "Cameo Systems Modeler," No Magic, [Online]. Available: https://www.nomagic.com/products/cameo-systems-modeler#intro. [Accessed 02 March 2018].

[6] L. Delligatti, "Chapter 2: Overview of the Systems Modeling Language," in *SysML Distilled: A Brief Guide to the Systems Modeling Language*, Crawfordsville, Indiana, Addison-Wesley, 2014.

[7] No Magic Inc., "Modeling SysML Diagrams," No Magic, [Online]. Available: https://docs.nomagic.com/display/SYSMLP182/Modeling+SysML+Diagrams. [Accessed 02 March 2018].

[8] Dassault Systemes, "Catia Systems Engineering - Dymola," Dassault Systemes, [Online]. Available: https://www.3ds.com/products-services/catia/products/dymola/key-advantages/. [Accessed 02 March 2018].

[9] M. Otter, "Modelica Overview," Modelica Association, 28 August 2013. [Online]. Available: https://www.modelica.org/education/educational-material/lecture-material/english/ModelicaOverview.pdf. [Accessed 03 March 2018].

[10] C. Nytsch-Geusen, "BuildingSystems," Universität der Künste Berlin, [Online]. Available: http://modelica-buildingsystems.de/index.html. [Accessed 03 March 2018].

[11] Modelica Association Project, "Functional Mock-up Interface," [Online]. Available: http://fmi-standard.org/. [Accessed 15 March 2018].

[12] Modelica Association Project, "Functional Mock-up Interface for Model Exchange and Co-Simulation," 25 July 2014. [Online]. Available: https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_Model Exchange_and_CoSimulation_v2.0.pdf. [Accessed 15 March 2018].

[13] C. D. Bodemann and F. D. Rose, "The Successful Development Process with Matlab Simulink in the Framework of ESA's ATV Project," [Online]. [Accessed 15 December 2017].

[14] Object Management Group, "About The Sysml-Modelica Transformation Specification Version 1.0," November 2012. [Online]. Available: http://www.omg.org/spec/SyM/. [Accessed 04 January 2018].

[15] C. Paredis and A. Reichwein, "Sysml-Modelica Integration," Model-Based Systems Engineering Center, Georgia Tech, [Online]. Available: http://www.mbsec.gatech.edu/research/projects/active/sysml-modelica-integration. [Accessed 29 November 2017].

[16] No Magic Inc., "Simulation of SysML models," No Magic Inc., [Online]. Available: https://docs.nomagic.com/display/CST190/Simulation+of+SysML+models. [Accessed 13 January 2018].

[17] L. Delligatti, "Chapter 9: Paramteric Diagrams," in *SysML Distilled: A Brief Guide to the Systems Modeling Language*, Crawfordsville, Indiana, Addison-Wesley, 2014.

[18] InterCAX LLC., "SysML Parametrics Tutorial - HomeHeating," in *ParaMagic® 18.0 - Tutorials*, Atlanta, Georgia, 2014, p. 65.

[19] Dassault Systemes, "6.10.2 Exporting FMUs from Dymola," in *Dymola Dynamic Modeling Laboratory User Manual - Volume 2*, 2016, pp. 309 - 310.

[20] Dassault Systemes, "6.10.5 FMU Export from Simulink/ FMU Import into Simulink: The FMI Kit for Simulink," in *Dymola Dynamic Modeling Laboratory User Manual - Volume 2*, 2016, pp. 339-343.

[21] InterCAX LLC., "SysML Parametrics Tutorial - Addition," in *ParaMagic® 18.0 - Tutorials*, Atlanta, Georgia, 2014, p. 16.

[22] D. R. Daily, "Trade-off Based Design and Implementation of Energy Efficiency Retrofits In Residential Homes," University of Maryland, College Park, MD, 2014.

[23] D. Spyropoulos, "Integration of SysML with Trade-off Analysis Tools," University of Maryland, College Park, MD, 2012.

[24] InterCAX LLC., "SysML Parametrics Tutorial - LittleEye Trade Study," in *ParaMagic® 18.0 - Tutorials*, Atlanta, Georgia, 2014, p. 76.

[25] I. Roth, "Smart Sensors Are Driving Smart Buildings," 16 December 2016. [Online]. Available: https://www.sensorsmag.com/components/smart-sensors-are-driving-smart-buildings.