

Design of a Temperature Regulation Loop System Using SysML

Sébastien Buet,
System Engineering Analyst



Abstract: This document demonstrates an example of a temperature regulation loop, developed with the SysML (System Modeling Language) plug-in of MagicDraw TM UML. It introduces and describes the different specific diagrams used in SysML (Requirement, Block Definition, Internal Block, and Parametric diagrams) as well as the specific SysML features (blocks, flow ports, value types, items flow, etc.). This document also provides guidelines for the creation of a SysML model and the utilization of the specific SysML features using MagicDraw UML 15.5 with the SysML 15.5 plug-in.

Keywords: SysML, UML, MagicDraw, block, regulation loop.

1. Introduction

This example models a regulation loop for controlling temperature in an ordinary room. This example was chosen due to the prevalence of regulation loops in almost every control system. It was also chosen because it is easy to understand and communicate, since almost everyone has an air-conditioning system or a heater at home and knows the basics of how it works.

First, we will describe what a regulation loop is and explain what it is supposed to do, as well as how it is supposed to work. Next, we will explain how we built this example and how we have used the different features of the SysML plug-in. Finally, we will discover the benefits and weaknesses of this current tool and how it could be improved for the modeling of such engineering issues.

2. System Description

2.1 What the system is modeling

This model was created to enhance the behavior of a regulator. The behavior of a regulator can affect several aspects upon the comfort of the user(s) and the energy consumption of the system. A maladjusted regulator or a regulator with inappropriate settings can never reach the goal of regulating something to a chosen value.

Regulation loop principle [1]:

The goal of the regulation is to maintain, in time, a constant value “H” with an action “A” where the environment changes.

The more basic regulation is the regulation by all or nothing.

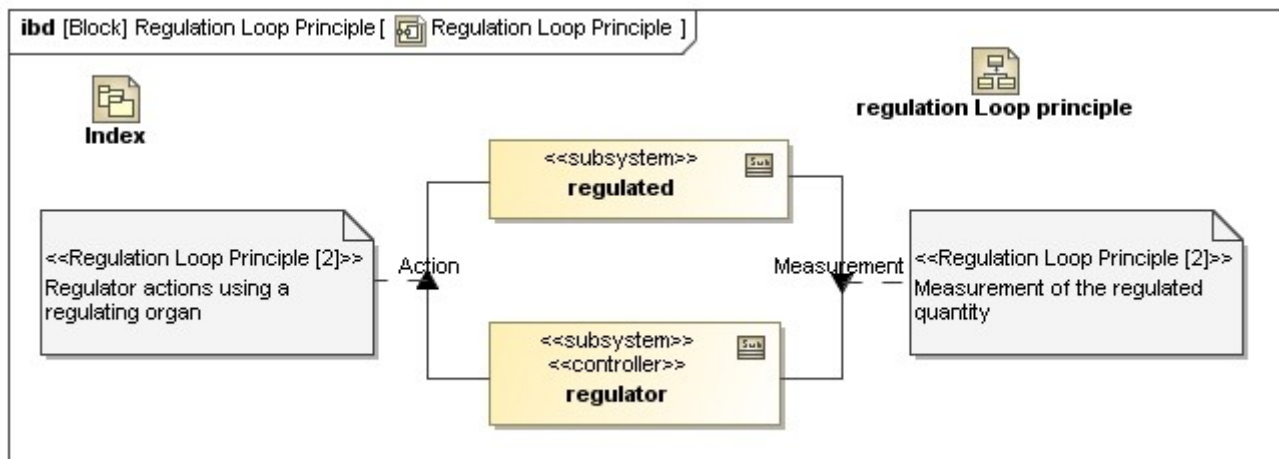


Figure 1: Regulation Loop principle [2]

In this model, we focus on the all or nothing regulator. This kind of regulator can only send two kinds of messages: 1 or 0 (All or Nothing, 0 or 100%). The action of the regulator is the same if the gap between the current temperature and the chosen is small or large, which is not favorable for a good dynamic behavior [3].

Summarized in Figure 2:

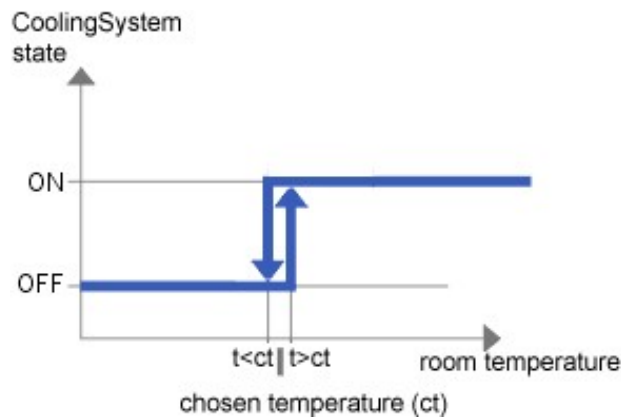


Figure 2: regulation scheme of an all or nothing regulator [2]

The system in this example is modeling a regulation loop to control the temperature of a room. For the modeling of this system, the SysML plug-in for MagicDraw was used. In SysML, this system is composed of different physical elements represented by SysML blocks and interconnected through ports/flow ports.

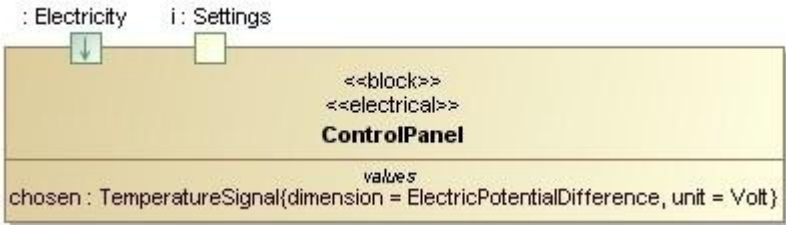

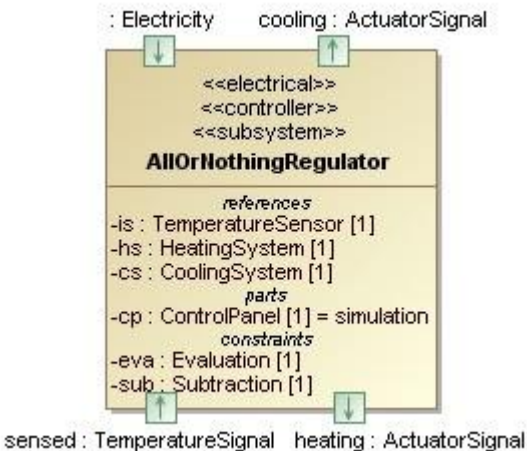
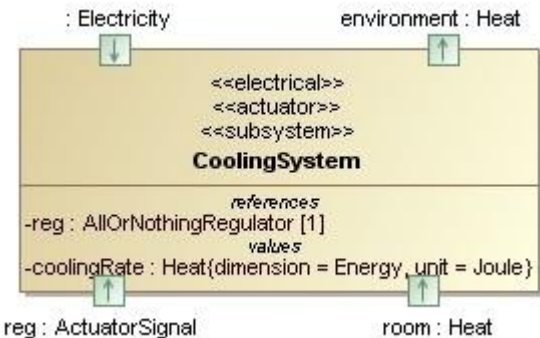
The OMG SysML™, Version 1.0 specification [4] describes the SysML blocks as follows:

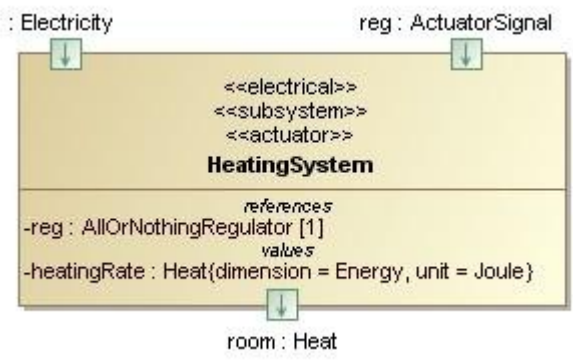
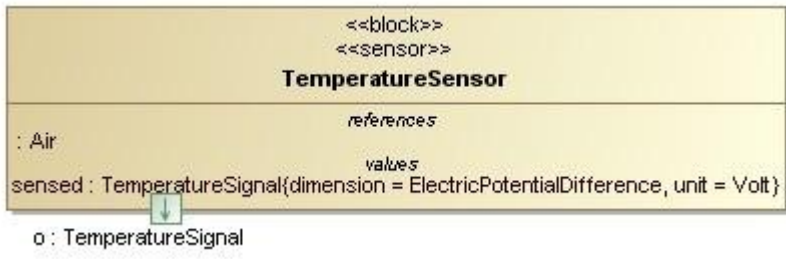
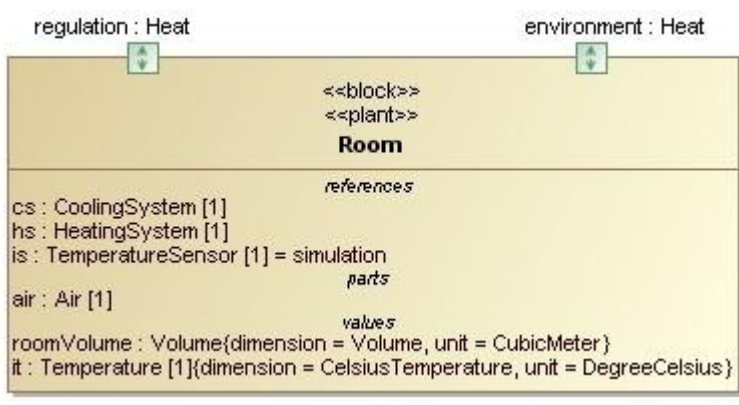
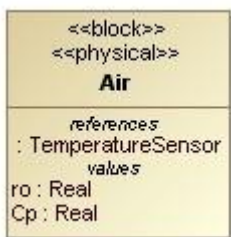

Blocks are modular units of system description. Each block defines a collection of features to describe a system or other element of interest. These may include both structural and behavioral features, such as properties and operations, to represent the state of the system and behavior that the system may exhibit.

Blocks provide a general-purpose capability to model systems as trees of modular components. The specific kinds of components, the kinds of connections between them, and the way these elements combine to define the total system can all be selected according to the goals of a particular system model. These include modeling either the logical or physical decomposition of a system, and the specification of software, hardware, or human elements. Parts in these systems may interact by many different means, such as software operations, discrete state transitions, flows of inputs and outputs, or continuous interactions.

For modeling the first level of detail within this temperature regulation loop system, the following elements were created:

Elements	Graphical representation
<i>TemperatureRegulationContext</i> System Context block	<p>The diagram shows a yellow rectangular block with a light brown border. Inside the block, at the top, is the text «system context». Below this, in bold, is the text TemperatureRegulationContext.</p>
<i>TemperatureRegulationSystem</i> System block	<p>The diagram shows a yellow rectangular block with a light brown border. Inside the block, at the top, is the text «system». Below this, in bold, is the text TemperatureRegulation. Underneath, the word 'parts' is followed by a list of components: -cs : CoolingSystem [1] = simulation, -hs : HeatingSystem [1] = simulation, -r : Room [1] = simulation, and -reg : AllOrNothingRegulator [1]. On the left side, there is an input port labeled ': Electricity' with a green arrow pointing into the block. On the top right side, there is an output port labeled 'cs : Heat' with a green arrow pointing out of the block. On the right side, there is a port labeled 'i : Settings' with a green arrow pointing into the block. On the bottom left side, there is an input port labeled 'gain : Heat' with a green arrow pointing into the block. On the bottom right side, there is an output port labeled 'loss : Heat' with a green arrow pointing out of the block.</p>

Elements	Graphical representation
<i>ControlPanel</i> block	 <pre> classDiagram class ControlPanel { <<block>> <<electrical>> values chosen : TemperatureSignal{dimension = ElectricPotentialDifference, unit = Volt} } </pre> <p>The diagram shows a yellow rectangular block labeled ControlPanel. It has two input ports at the top: a green square with a downward arrow labeled ": Electricity" and a yellow square labeled "i : Settings". The block contains the text "<<block>>", "<<electrical>>", and "values". Below this, it lists "chosen : TemperatureSignal{dimension = ElectricPotentialDifference, unit = Volt}".</p>
<i>Regulator</i> block	 <pre> classDiagram class Regulator { <<subsystem>> <<controller>> values -actions : Actions -measurement : Measurement } </pre> <p>The diagram shows a yellow rectangular block labeled Regulator. It has an input port on the left labeled ": Measurement" and an output port on the right labeled ": ActuatorSignal". The block contains the text "<<subsystem>>", "<<controller>>", and "values". Below this, it lists "-actions : Actions" and "-measurement : Measurement".</p>
<i>AllOrNothingRegulator</i> block	 <pre> classDiagram class AllOrNothingRegulator { <<electrical>> <<controller>> <<subsystem>> references -is : TemperatureSensor [1] -hs : HeatingSystem [1] -cs : CoolingSystem [1] parts -cp : ControlPanel [1] = simulation constraints -eva : Evaluation [1] -sub : Subtraction [1] } </pre> <p>The diagram shows a yellow rectangular block labeled AllOrNothingRegulator. It has four ports: ": Electricity" (input, top-left), "cooling : ActuatorSignal" (output, top-right), "sensed : TemperatureSignal" (input, bottom-left), and "heating : ActuatorSignal" (output, bottom-right). The block contains the text "<<electrical>>", "<<controller>>", and "<<subsystem>>". Below this, it lists "references" with "-is : TemperatureSensor [1]", "-hs : HeatingSystem [1]", and "-cs : CoolingSystem [1]"; "parts" with "-cp : ControlPanel [1] = simulation"; and "constraints" with "-eva : Evaluation [1]" and "-sub : Subtraction [1]".</p>
<i>CoolingSytem</i> block	 <pre> classDiagram class CoolingSystem { <<electrical>> <<actuator>> <<subsystem>> references -reg : AllOrNothingRegulator [1] values -coolingRate : Heat{dimension = Energy, unit = Joule} } </pre> <p>The diagram shows a yellow rectangular block labeled CoolingSystem. It has four ports: ": Electricity" (input, top-left), "environment : Heat" (output, top-right), "reg : ActuatorSignal" (input, bottom-left), and "room : Heat" (output, bottom-right). The block contains the text "<<electrical>>", "<<actuator>>", and "<<subsystem>>". Below this, it lists "references" with "-reg : AllOrNothingRegulator [1]" and "values" with "-coolingRate : Heat{dimension = Energy, unit = Joule}".</p>

Elements	Graphical representation
HeatingSystem block	 <pre> classDiagram class HeatingSystem { <<electrical>> <<subsystem>> <<actuator>> +Electricity : Electricity +reg : ActuatorSignal -reg : AllOrNothingRegulator [1] -heatingRate : Heat {dimension = Energy, unit = Joule} +room : Heat } </pre> <p>The diagram shows a yellow box representing the HeatingSystem block. It has two input ports at the top: Electricity and reg : ActuatorSignal. The box contains the following text: <<electrical>>, <<subsystem>>, <<actuator>>, HeatingSystem, references, -reg : AllOrNothingRegulator [1], values, -heatingRate : Heat{dimension = Energy, unit = Joule}, and room : Heat. There is an output port at the bottom labeled room : Heat.</p>
TemperatureSensor block	 <pre> classDiagram class TemperatureSensor { <<block>> <<sensor>> +Air : Air +sensed : TemperatureSignal {dimension = ElectricPotentialDifference, unit = Volt} +o : TemperatureSignal } </pre> <p>The diagram shows a yellow box representing the TemperatureSensor block. It contains the following text: <<block>>, <<sensor>>, TemperatureSensor, references, : Air, values, sensed : TemperatureSignal{dimension = ElectricPotentialDifference, unit = Volt}, and o : TemperatureSignal. There is an output port at the bottom labeled o : TemperatureSignal.</p>
Room block	 <pre> classDiagram class Room { <<block>> <<plant>> +regulation : Heat +environment : Heat -cs : CoolingSystem [1] -hs : HeatingSystem [1] -is : TemperatureSensor [1] = simulation -air : Air [1] -roomVolume : Volume {dimension = Volume, unit = CubicMeter} -it : Temperature [1] {dimension = CelsiusTemperature, unit = DegreeCelsius} } </pre> <p>The diagram shows a yellow box representing the Room block. It has two input ports at the top: regulation : Heat and environment : Heat. The box contains the following text: <<block>>, <<plant>>, Room, references, cs : CoolingSystem [1], hs : HeatingSystem [1], is : TemperatureSensor [1] = simulation, parts, air : Air [1], values, roomVolume : Volume{dimension = Volume, unit = CubicMeter}, and it : Temperature [1]{dimension = CelsiusTemperature, unit = DegreeCelsius}.</p>
Air block	 <pre> classDiagram class Air { <<block>> <<physical>> -tempSensor : TemperatureSensor -ro : Real -Cp : Real } </pre> <p>The diagram shows a yellow box representing the Air block. It contains the following text: <<block>>, <<physical>>, Air, references, : TemperatureSensor, values, ro : Real, and Cp : Real.</p>
Settings block	 <pre> classDiagram class Settings { <<block>> -tChosen : Temperature {dimension = CelsiusTemperature, unit = DegreeCelsius} +set(tChosen : Temperature) } </pre> <p>The diagram shows a yellow box representing the Settings block. It contains the following text: <<block>>, Settings, values, tChosen : Temperature{dimension = CelsiusTemperature, unit = DegreeCelsius}, and set(tChosen : Temperature).</p>



Elements	Graphical representation
<i>iSettings</i> interface	 <pre> classDiagram class iSettings { +set(tChosen : Temperature) } </pre>
<i>User</i> actor	 <pre> classDiagram actor User </pre>

Table 1: Elements constituting the system

Stereotypes used:

- <<electrical>>: categorize the electrical devices.
- <<actuator>>: categorize the actuator devices.
- <<sensor>>: categorize the sensor devices.
- <<plant>>: categorize the controlled devices.
- <<controller>>: categorize the control devices.
- <<physical>>: categorize the physical elements.

The block elements are represented in a Block Definition Diagram. In this case, the Block Definition Diagram is used to show the different elements of interest within the system.

2.2 What the system does

The user chooses the temperature he wants inside the room by using the *iSettings* interface connected to the *ControlPanel*. The *ControlPanel* will send this information to the *AllOrNothingRegulator*. The *AllOrNothingRegulator* will then receive from the *TemperatureSensor* the current room temperature. By comparing the current inside temperature and the temperature chosen by the user, the *AllOrNothingRegulator* will send a signal to the *CoolingSystem* and/or *HeatingSystem* in order to adjust the current inside temperature to the temperature desired by the user.

The following Internal Block Diagram shows how the different blocks, previously presented, are interconnected:

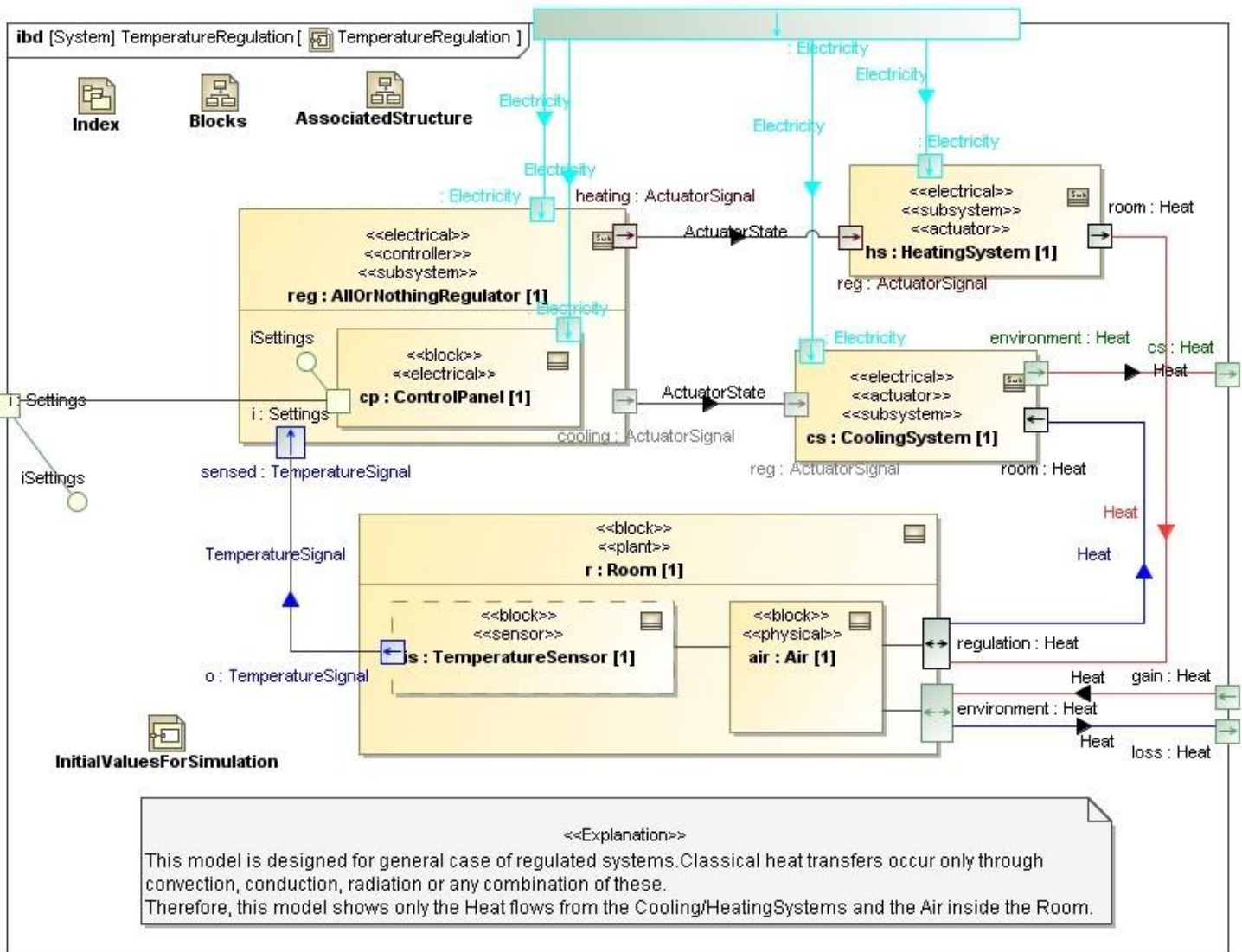


Figure 3: Internal Block Diagram of the TemperatureRegulation

This system does not accurately model thermal loss such as, the thermal loss through room partitions, the loss induced by wind, as well as thermal gain induced by sun radiation and occupation of the room (persons and electric equipment). In order to carry out the modeling of such gains and loss, we need to create some scenarios regarding the occupation of the room (number of people present in the room, the electric devices, schedule of occupation...), the wind velocity (direction, velocity, frequency), and the sun radiation (watts provided function of the sun inclination, direction, and season). These last parameters affect the efficiency of the loop regulation and should be taken into account for an accurate model. Such implementation was deemed not necessary for this model because the main objective of this example was to study the behavior of regulators.

System engineers can use an Internal Block Diagram (IBD) to specify the parts and connections of the system. On the other hand, this system can also be represented using a Block Definition Diagram (BDD), where associations between blocks are shown; this corresponds with a classic UML representation of the system.

In this system, the *ControlPanel* is part of the *AllOrNothingRegulator* and the *Air* is part of the *Room*. The *Room* is also composed of "roomVolume : Volume" and "it : InsideTemperature" properties (not shown on the next BDD). There are association relationships between the *TemperatureSensor*, the *HeatingSystem*, the *CoolingSystem*, and the *Room* and between the *TemperatureSensor*, the *CoolingSystem*, the *HeatingSystem*, and the *AllOrNothingRegulator*. The *Room*, the *CoolingSystem*, the *HeatingSystem*, and the *AllOrNothingRegulator* are parts of the *TemperatureRegulation*.

The associations between blocks are shown in Figure 4:

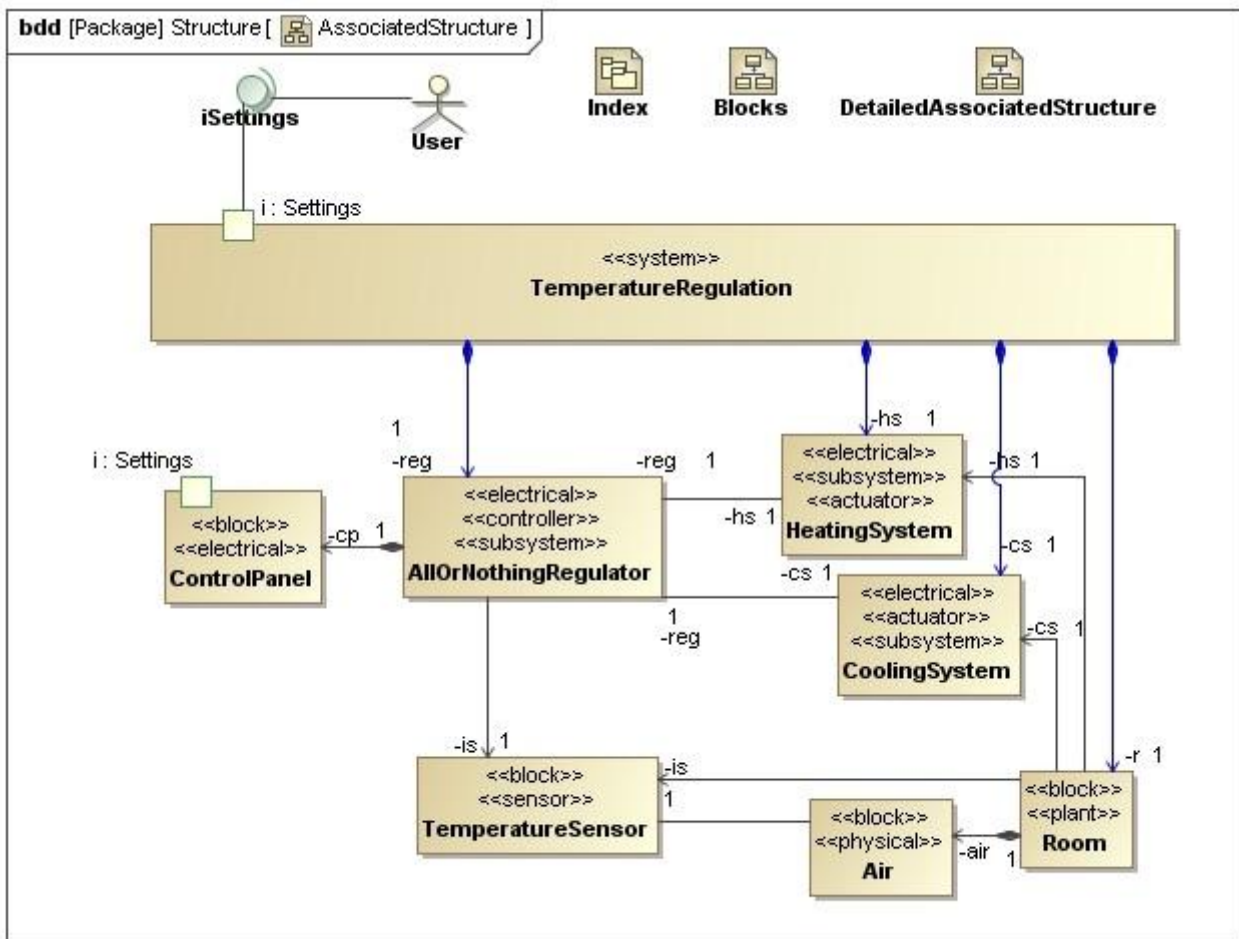


Figure 4: Block Definition Diagram of the TemperatureRegulationSystem

The OMG SysML™, Version 1.0 specification [4] describes the SysML block definition diagram as follows:

(...) The Block Definition Diagram in SysML defines features of blocks and relationships between blocks such as associations, generalizations, and dependencies. It captures the definition of blocks in terms of properties and operations, and relationships such as a system hierarchy or a system classification tree (...)

The next Block Definition Diagram introduces the different *ValueTypes* and *Enumeration* used in this system to type the value properties.

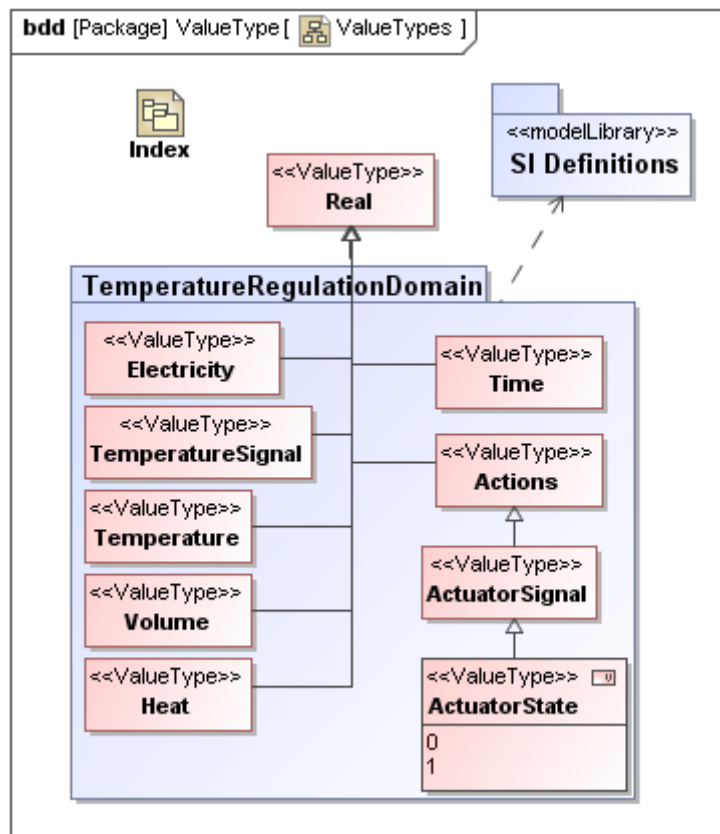


Figure 5: Block Definition Diagram of the ValueTypes and DataTypes

The OMG SysML™, Version 1.0 specification [4] describes the *ValueType* as follows:

A *ValueType* defines types of values that may be used to express information about a system (...). Value types may be used to type properties, operation parameters, or potentially other elements within SysML. SysML defines *ValueType* as a stereotype of UML *DataType* to establish a more neutral term for system values that may never be given a concrete data representation. (...) SysML *ValueType* adds an ability to carry a unit of measure or dimension associated with the value.

The *ValueType* element was introduced to provide a means to represent physical values that are expressed with specific units and dimension and to express mathematical concepts.

2.3 How the system works

After the *User* has set the chosen temperature of the room, the *AllOrNothingRegulator* will use the *TemperatureSignal* of the *TemperatureSensor* and the *TemperatureSignal* of the *ControlPanel* to decide whether the *HeatingSystem* or *CoolingSystem* will be turned ON in order to adjust the room temperature.

The *AllOrNothingRegulator* is a primitive regulator, yet it is one of the most used in standard electrical equipment.

In order to determine if the *CoolingSystem* or the *HeatingSystem* has to be turned ON, the *AllOrNothingRegulator* performs a few simple calculations:

- First, the *AllOrNothingRegulator* subtracts the sensed temperature from the chosen temperature.

- Second, according to the sign of the difference, the *AllOrNothingRegulator* will send:

- If the sign is positive: 1 to the *HeatingSystem* and 0 to *CoolingSystem*

- If the sign is negative: 0 to the *HeatingSystem* and 1 to the *CoolingSystem*

- Else: 0 to the *HeatingSystem* and 0 to the *CoolingSystem*

In order to carry out these calculations, ConstraintBlocks were created for the *AllOrNothingRegulator* as shown in the following table:

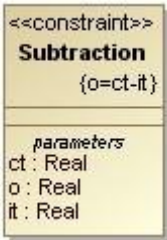
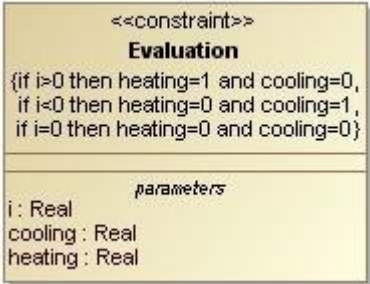
ConstraintBlock elements	Graphical representation
<i>Subtraction</i> ConstraintBlock	
<i>Evaluation</i> ConstraintBlock	

Table 2: ConstraintBlock elements constituting the AllOrNothingRegulator

Actually, the *AllOrNothingRegulator* is composed of one *Evaluation* ConstraintBlock and one *Subtraction* ConstraintBlock.

The different calculations performed by the *AllOrNothingRegulator* are represented in a Parametric Diagram. This diagram shows the connections between properties, ports, and ConstraintParameters.

The following Parametric Diagram shows the calculations performed by the *AllOrNothingRegulator*.

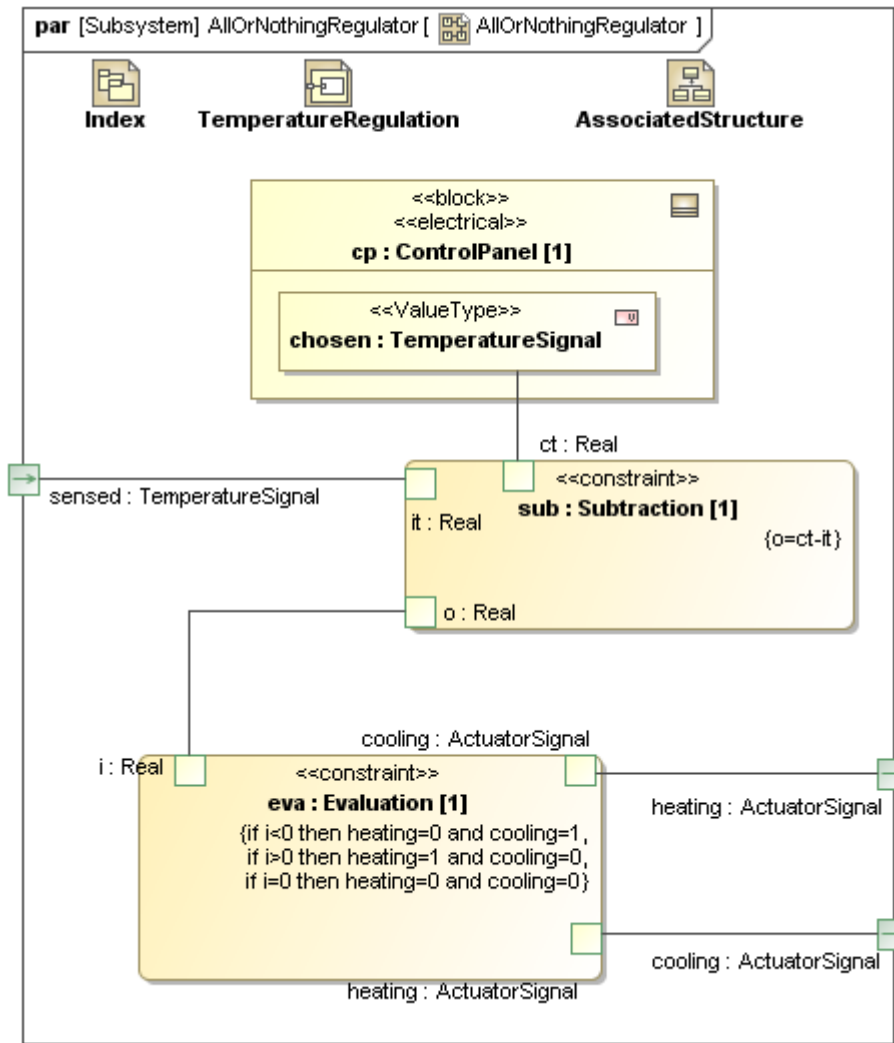


Figure 6: Parametric Diagram of the AllOrNothingRegulator

According to the *ActuatorSignal* sent by the *AllOrNothingRegulator*, the *CoolingSystem* and *HeatingSystem* will be turned ON (1) or OFF (0). Then the *HeatingSystem* will provide *Heat* to the *Room* and increase the room temperature; the *CoolingSystem* will remove *Heat* from the *Room* and then decrease the room temperature.

The regulation loop “starts again” after a time lapse, which is a function of the regulator type.

This loop ends when the room temperature reaches the chosen temperature, or when the user stops the regulation of the room temperature. In the real world, there are thermal losses (through the room partitions for instance) and gains (occupants of the room, sun radiation, etc.). Therefore, this regulation loop can only stop when the user stops it. The loss and gain can be represented using the *CoolingSystem* and *HeatingSystem*. The *CoolingSystem* can model the thermal loss of the room during the winter season (when the outside temperature is below the inside temperature (and the chosen temperature). The *HeatingSystem* can model the thermal gain during the summer season (when the outside temperature is above the inside temperature (and the chosen temperature). This can be done by modifying the *CoolingProfile* and the *HeatingProfile* and by setting the position of the system to ON during a simulation, for example. In these cases, the system is then transformed in a regular *CoolingSystem* or *HeatingSystem*; one system plays its role and the other system plays the thermal loss or gain respectively.

The *Heat* transfers of the model are not detailed, only the *Heat* flows were described. This solution was chosen in order to:

- first, be the less specific about the *CoolingSystem* and *HeatingSystem* used in the model (they could be convective system, conductive system or radiation systems);
- second, not describe the nature of loss and gain of the *Room*.

Classical transfer of thermal energy occurs only through [conduction](#), [convection](#), [radiation](#) or any combination of these. Heat transfer is of particular interest to [engineers](#), who attempt to understand and control the flow of heat through the use of [thermal insulation](#), [heat exchangers](#), and other devices [5]. However, in order to study the behavior of regulators, it was not deemed necessary to describe in detail the different *Heat* transfers that occur in this system.

3. How to build a SysML model

This example was built using MagicDraw 15.5 and the SysML 15.5 plug-in. This section explains how to use the specific features of this plug-in and how to make your model easily navigable. First, we will present the requirement diagram and the requirement blocks starting point of the project. Then we will focus on the block definition diagram and the block feature as well as focus on the internal block diagram and the flow ports features. Finally, once all the specific SysML diagrams are detailed, we will give guidelines to carry out an easily navigable model.

3.1 Requirements Diagram

This specific SysML diagram is used to detail the different requirements that the system has to comply with. The requirement diagram for this system is composed of a main requirement and other requirements connected to the main via SysML dependency relationships (DeriveReq, Satisfy, Copy, Verify) and/or UML dependency relationships (Trace, Refine, NestedClassifier). It is also possible to specify the type of the requirement; indeed, SysML proposes 8 different non-normative requirements (business, functional, extended, performance, interface, physical, usability requirement and design constraint). Moreover, it is possible to show in this diagram which features of the system will verify, satisfy, or refine a requirement. It is an efficient way to track the requirement compliance of a system.

For this example, only one requirement diagram has been created. The different requirements of this system are shown in figure 7:

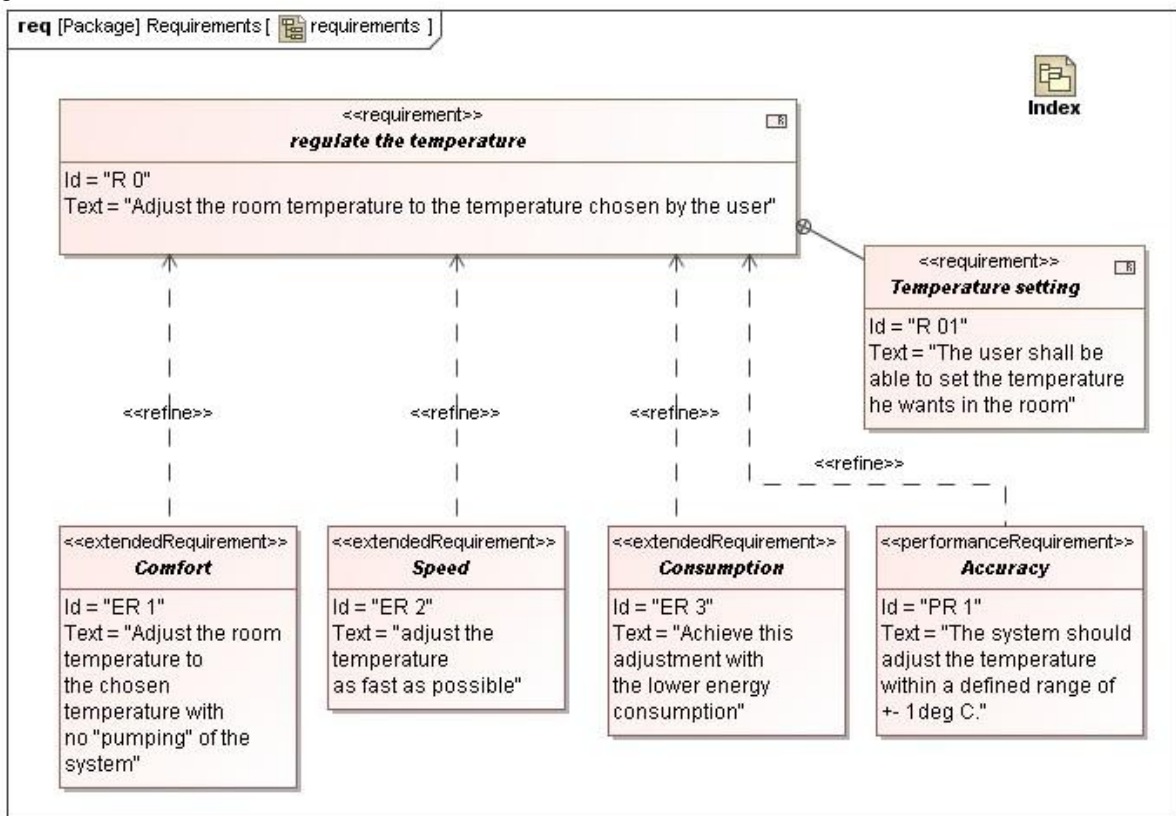


Figure 7: Requirement diagram

Some of the requirements have hyperlinks to a BDD in order to give more explanations of the requirement by using pictures, for instance.

3.2 Block definition diagram and Internal block definition diagram

The block definition diagram is an extension of the class diagram from UML. This diagram is used in a similar way to the class diagram in UML. It is used to describe the system and to show the different associations between the components comprising the system. Each component is represented by a block as a Holon (a part or a whole depending upon the perspective that is considered from), and each component could be composed of other components depending on the description level of the system. In order to be more specific, different kinds of non-

normative blocks are proposed: block, Domain block, External block, System block, Subsystem block, System Context block, Flow Specification block, and Constraint block.

First, a Block Definition Diagram presenting the system in its context is created. This diagram represents the level “0” of the system, as shown in figure 8:

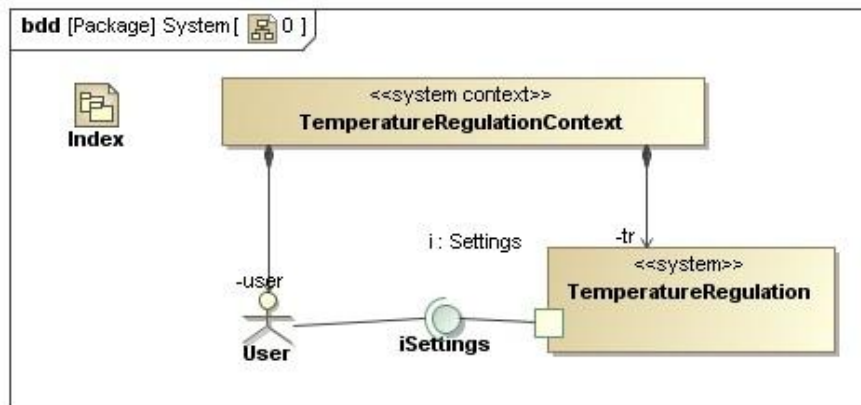


Figure 8: Block Definition Diagram of the System Context

In order to represent the system in this diagram, a *system context* block has been used (a block with the stereotype “system context” applied).

Then, another Block Definition Diagram has been created in order to represent the different elements constituting the system and their different ports.

The ports of the blocks could be shown on BDDs but should not be interconnected; those connections will be shown later on IBDs. This diagram shows what can flow through the different FlowPorts of the system.

Figure 9 shows these different elements:

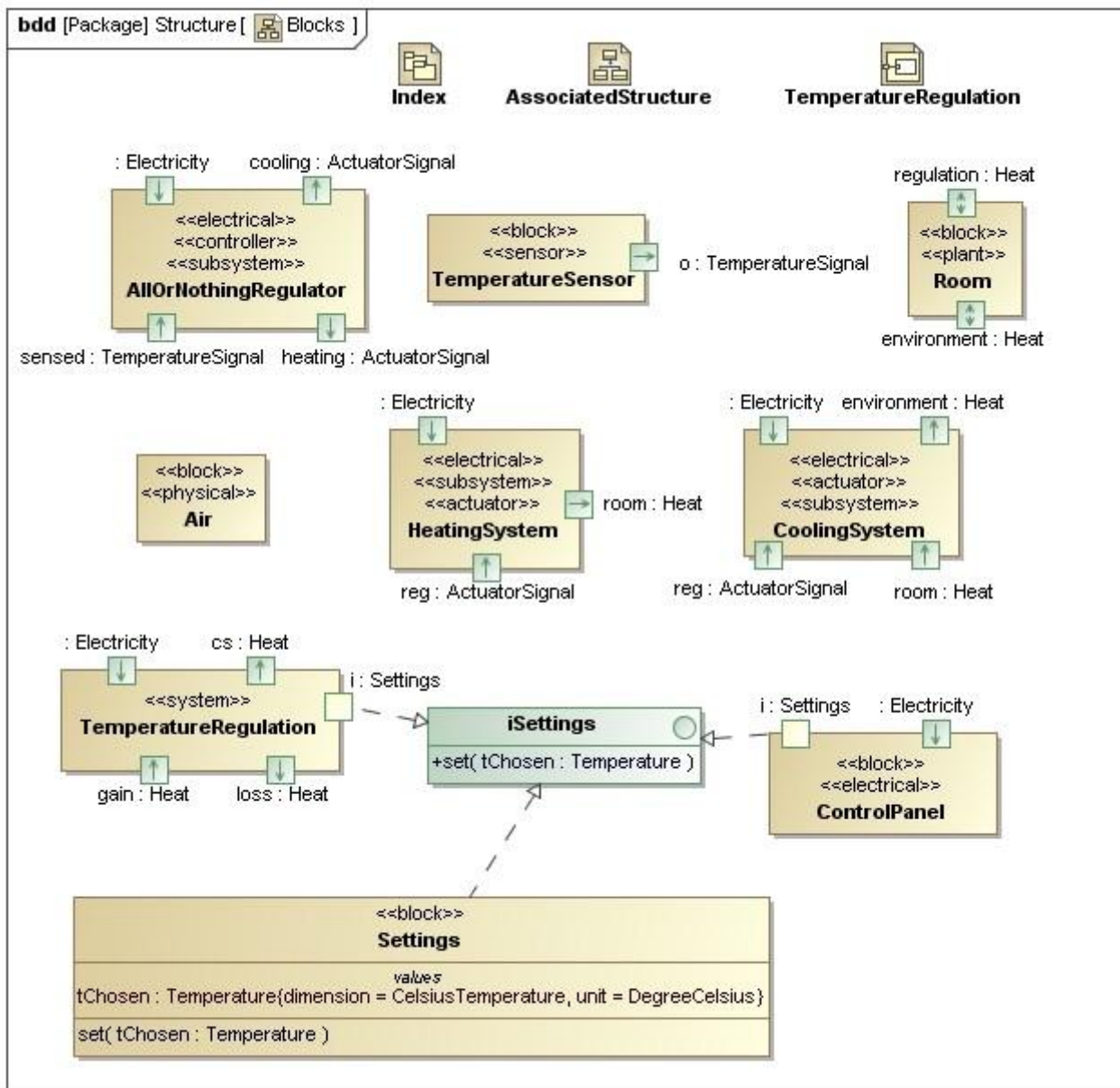


Figure 9: Block Definition Diagram of the elements constituting the system

Then, an internal block diagram was created in the *TemperatureRegulation* as presented in Figure 3, which shows how the different blocks constituting the system are interconnected. This diagram shows what really flows through the different Flow Ports of the system.

After interconnecting the different elements of the system on an IBD, we can generate a BDD, in which the different associations between blocks are represented. The IBD generated is the same as the one presented in Figure 4.

The other levels of detail for the elements of interest of this system were constructed following the same steps previously detailed.

The Parametric Diagram is used in a similar way to the Internal Block Diagram but is used to establish constraints between properties and constraint parameters using binding connectors.

3.3 Navigability inside the model

In this paragraph, we will detail how the navigability inside this model has been managed. The navigability was intended to be easy and intuitive. Every diagram is composed of different hyperlinks to different diagrams of the model. For example, every single diagram contains a hyperlink to the "Index" of the model (the package diagram groups the different packages constituting the system).

The diagrams also contain a hyperlink to the upper level, as well as the lower level of detail within the model (whenever it makes sense or whenever it is of interest). This is an efficient and intuitive way to navigate between the different diagrams.

The next figure shows how the navigability of the model, from the Index, has been managed:

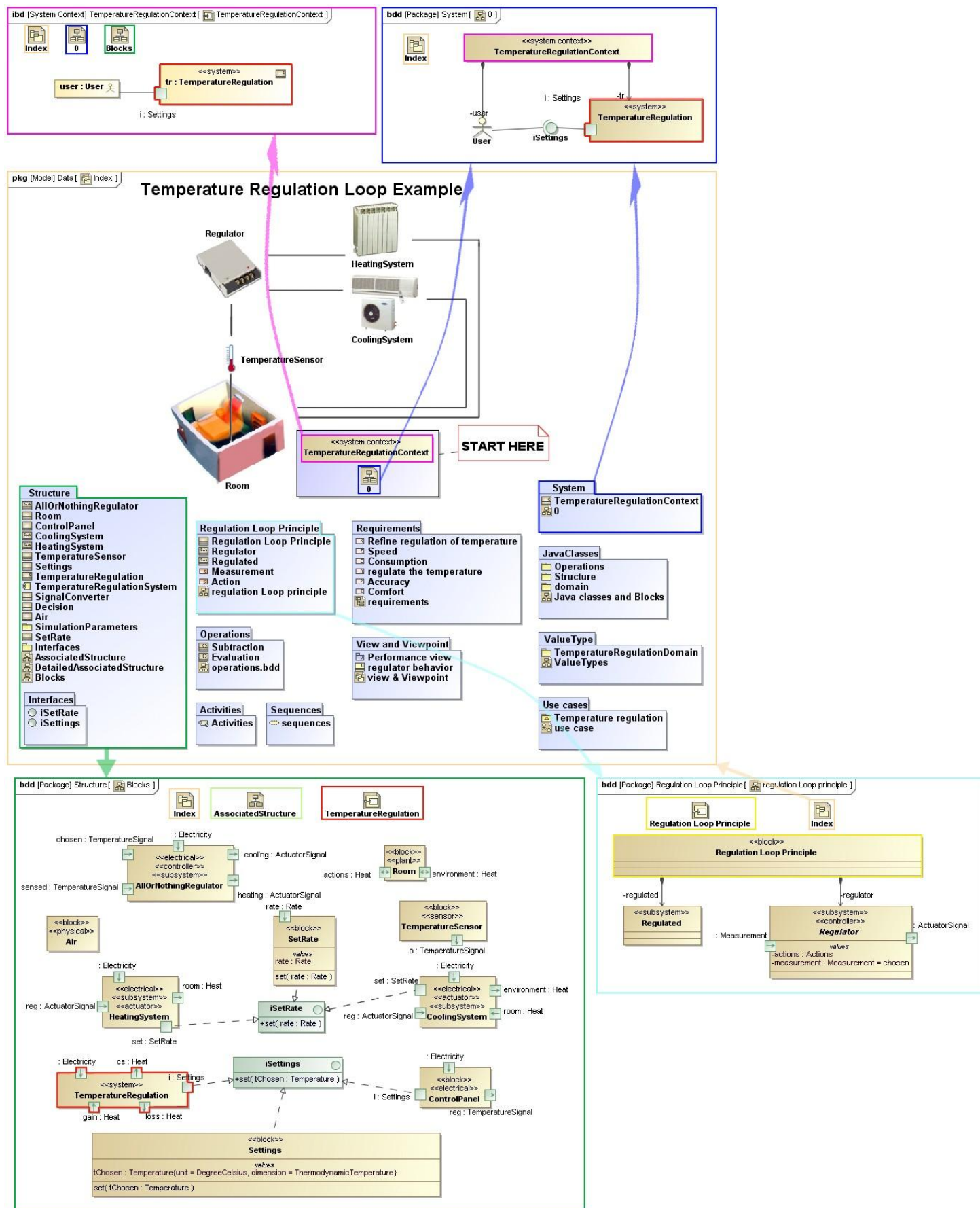


Figure 10: Navigability inside the model from the "Index" package diagram

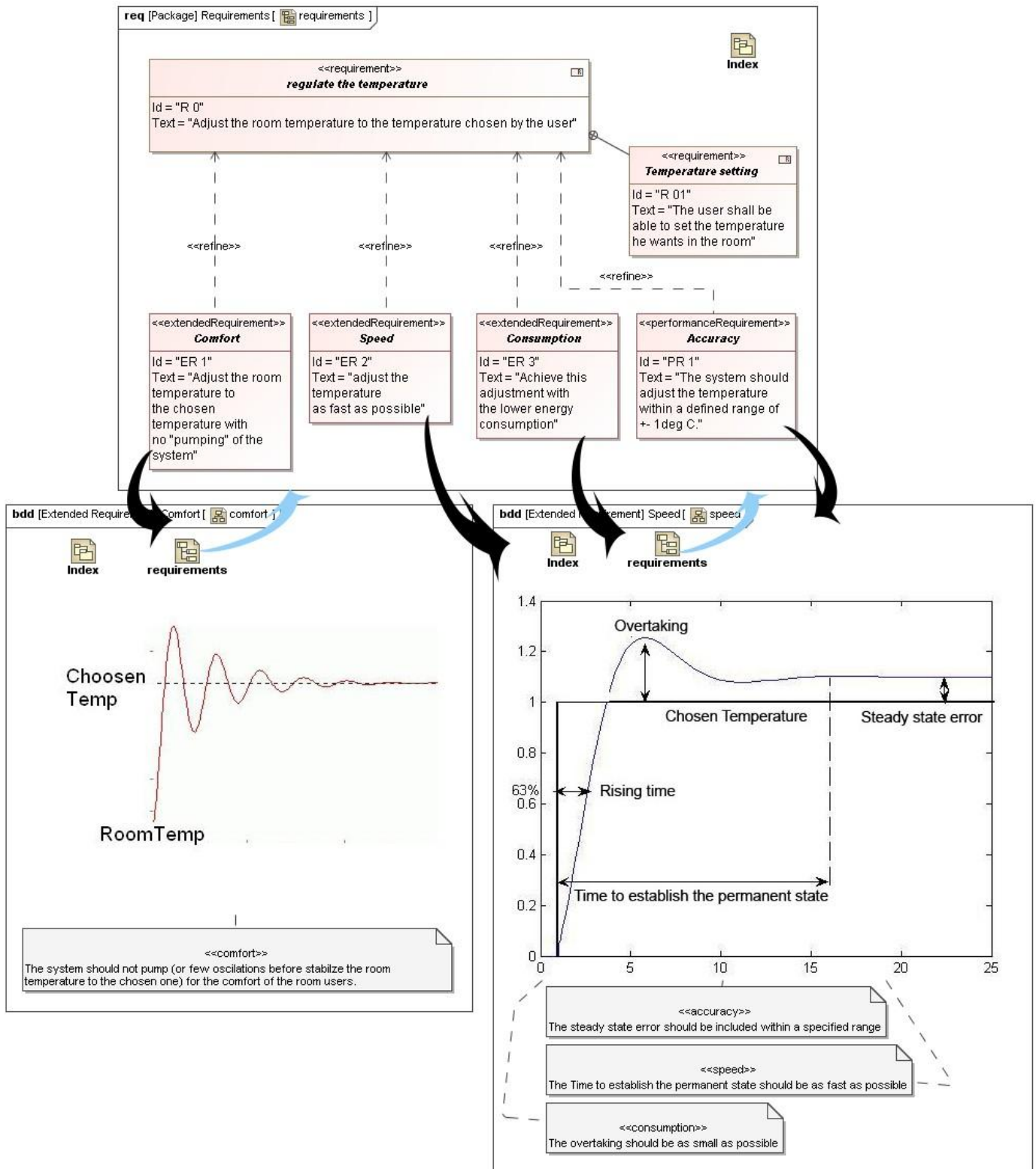


Figure 12: Navigability inside the Requirements

Whenever applicable, this principle has been followed: a system is composed of blocks nested inside other blocks. Then the hyperlinks allow us to navigate among them as well as open a block inside a block inside a block, and so on.

4. Simulation

Once the system is built, it is necessary to run it in order to do results analysis. The work done to create the system with MagicDraw UML 15.5 and the SysML 15.5 plug-in, allows an easier and faster creation of programming code. Some codes are needed in order to verify how the system behaves: this is done by running simulation. Using the MagicDraw UML code generation tool, the different blocks of this system have been created as class stubs in Java. The different constraints expressed in the ConstraintBlocks, as well as all blocks and valueTypes necessary for the simulation, have been implemented as Java classes. The internals of these class stubs have been hand-coded. The next block definition diagram presents the SysML blocks and the Java classes that simulate them:

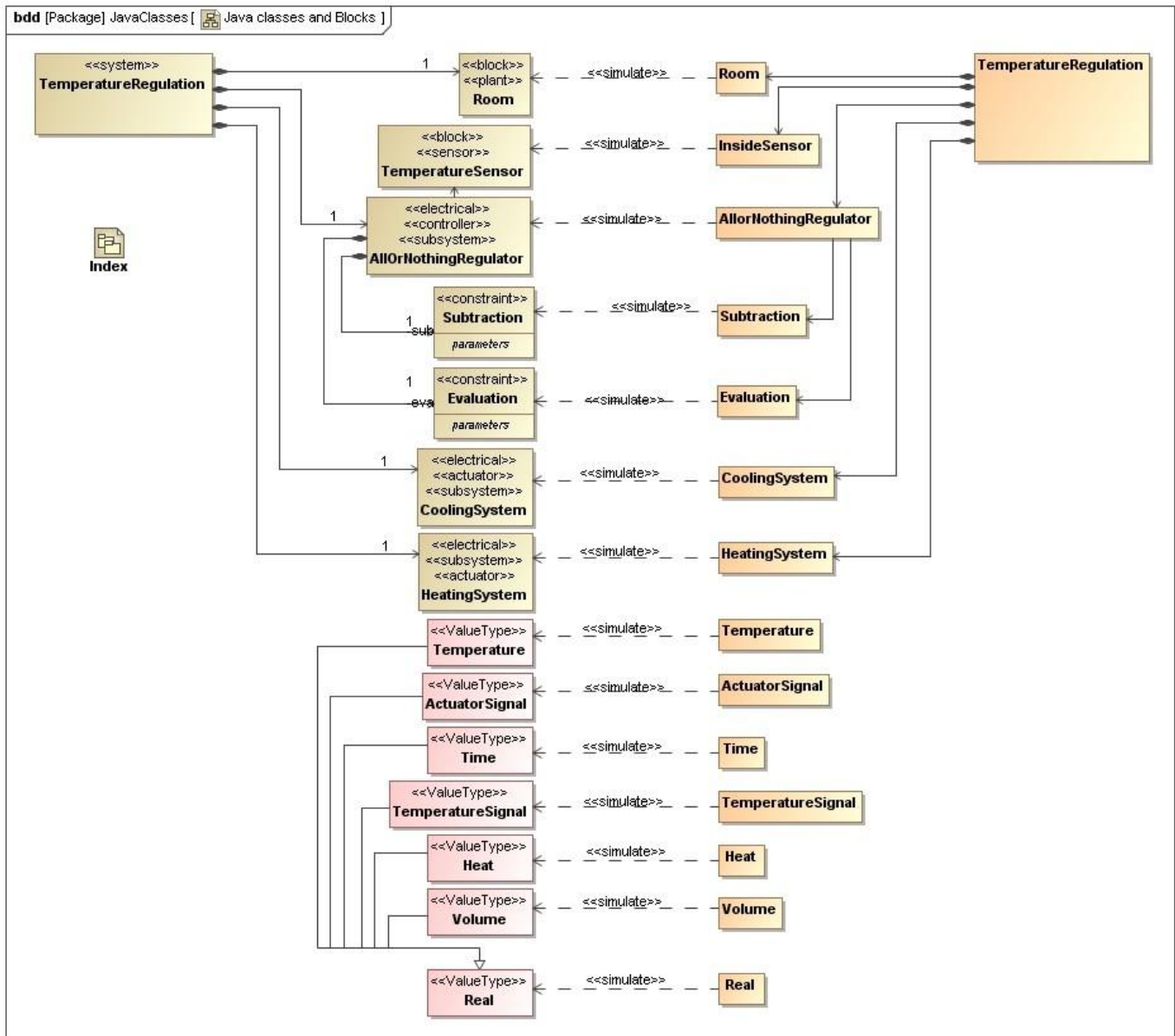


Figure 13: Block Definition Diagram of the Java classes simulating the SysML blocks

The initial room temperature and the temperature chosen by the user are set in the *AllOrNothingRegulator* in order to start the simulation. The initial room temperature, the chosen temperature, and the room volume are defined by the user. Once the *AllOrNothingRegulator* is initialized, it calculates its equations and returns to the *HeatingSystem* and *CoolingSystem* an ON/OFF signal in order to adjust the current room temperature to the chosen one. The *HeatingSystem* and *CoolingSystem* adds or removes *Heat* respectively in order to make the current room temperature converge to the chosen one. The *Heat* added or removed is a function of the *rate* set in the *CoolingSystem* or *HeatingSystem* respectively.

The *AllOrNothingRegulator* updates its output signals to the *HeatingSystem* and *CoolingSystem* according to a lapse time defined by the user (in the real world the user cannot define this lapse time, but for a behavior analysis of regulators it makes sense to see the effect of this lapse time on the regulator behavior). This system combines a continuous system as the *HeatingSystem* and *CoolingSystem* that provides continuous heat flow and a discrete system as the *AllOrNothingRegulator*.

Figure 14 presents results obtained using these Java classes created for the simulation:

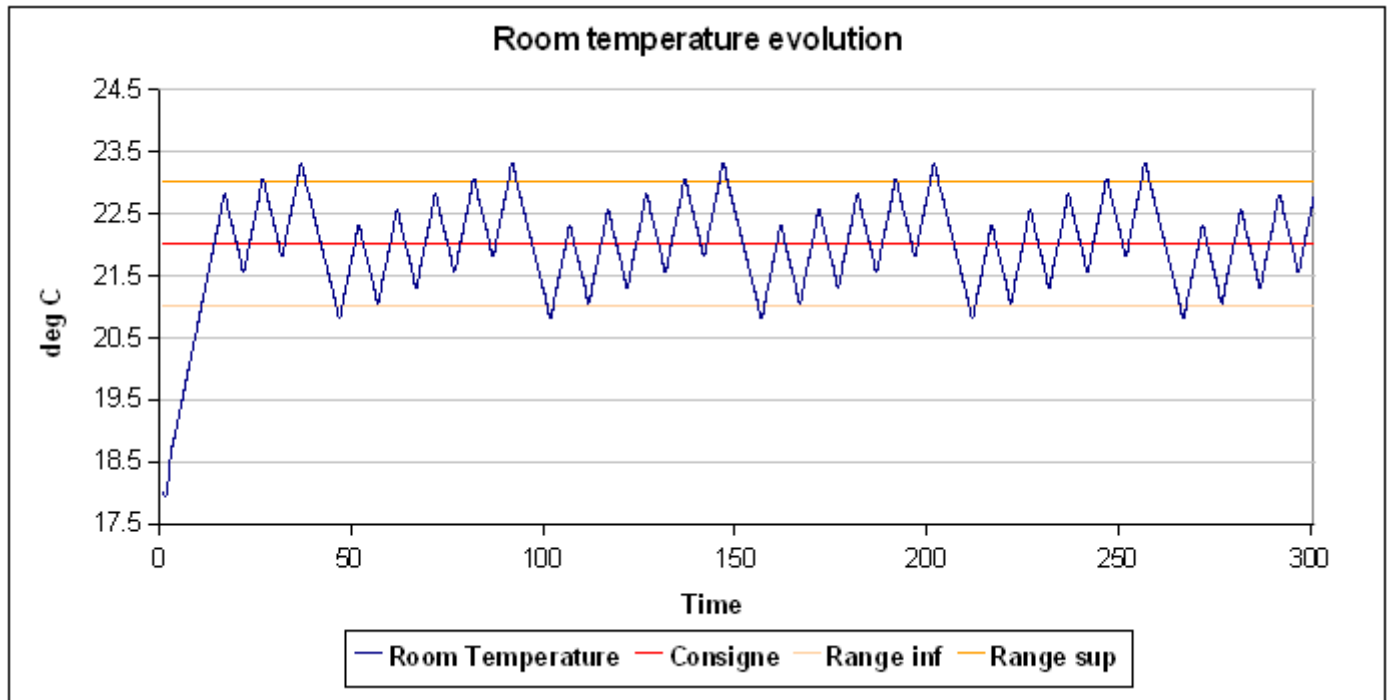


Figure 14: Simulation results using the java classes

The “Room temperature” curve results as an update of the *AllOrNothingRegulator* every 5 step times, the “Consigne” curve represents the temperature chosen by the user and the “Range inf” and “Range sup” curves represent the area in which the room temperature should be included (*Accuracy* requirement).

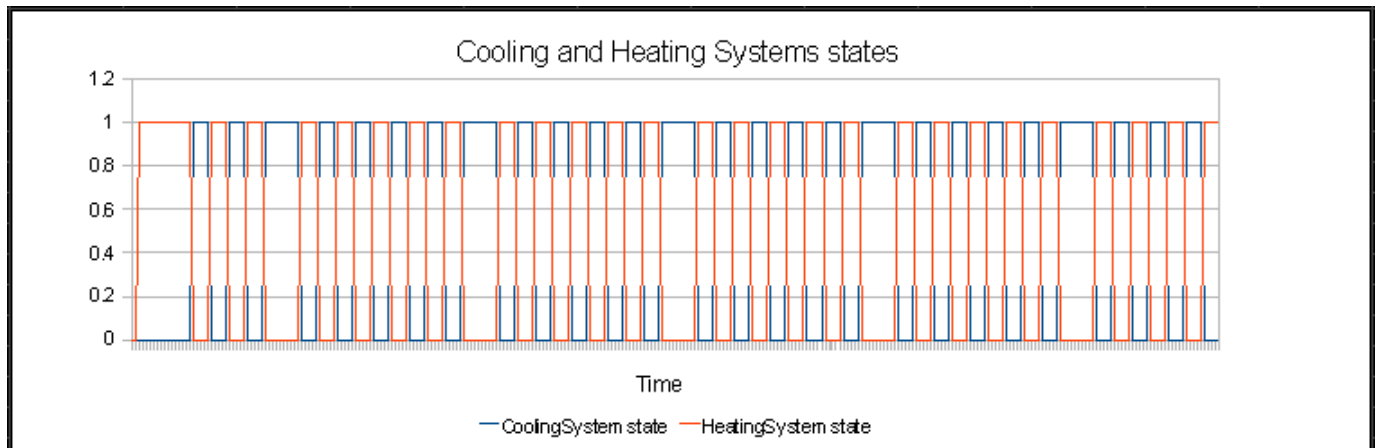


Figure 15: Simulation results of the CoolingSystem and HeatingSystem states

The red curve represents the state of the *HeatingSystem* and the blue curve represents the state of the *CoolingSystem*.

For this simulation the chosen temperature is 22 degrees Celsius, the initial temperature is 18 degrees Celsius, and the room volume is 40 cubic meters.

When the systems are ON, the *CoolingSystem* is defined to provide -10000 Watt second and the *HeatingSystem* is defined to provide +12000 Watt second (arbitrary values). These heat amounts represent the *Heat* flows that are added to the *Heat* present in the room and then the new current room temperature is calculated according to the previous room temperature and the *Heat* amount provided by the *HeatingSystem* and *CoolingSystem*. As previously said, it is also possible to use this system to simulate the loss or gain during the summer or winter season by setting a *Heat* value modeling the *Heat* loss or gain of the system in the *CoolingProfile/HeatingProfile* and by setting the position of the *CoolingSystem/HeatingSystem* to ON during all the simulations.

Once the new current room temperature is obtained, it is available to the *TemperatureSensor* and then sent to the *AllOrNothingRegulator*.

The temperature at time t is defined using the first thermodynamics law [6]:

$$Q = \rho_{Air} * Cp_{Air} * V_{Air} * \Delta\theta = \rho_{Air} * Cp_{Air} * V_{Air} * (\theta(t) - \theta(t-1))$$

The following equation is used to determine the room temperature at time t :

$$\theta(t) = \theta(t-1) + \frac{Q_c(t) + Q_h(t)}{\rho_{Air} * Cp_{Air} * V_{Room}}$$

Where:

$\theta(t)$ = temperature, at time t ;

$\Delta\theta$ = temperature variation;

ρ_{Air} = volumetric mass of the air;

Cp_{Air} = heat mass of the air;

$V_{Air} = V_{Room}$ = Volume of Air inside the Room

Q = heat quantity;

Q_c = heat quantity provided by the *CoolingSystem*

Q_h = heat quantity provided by the *HeatingSystem*

Figure 16 presents other simulation results obtained using the Java classes:

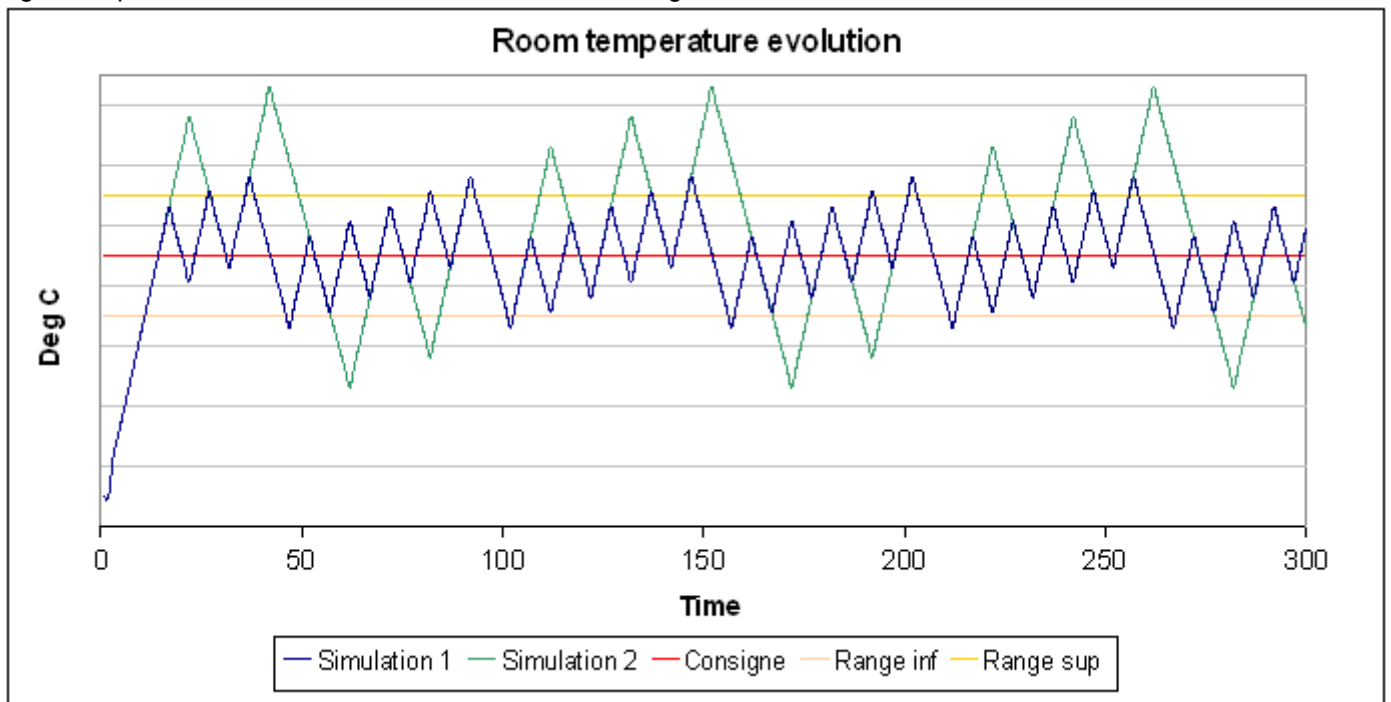


Figure 16: Simulations results obtained using the Java classes

The “Simulation 1” curve results as an update of the *AllOrNothingRegulator* every 5 step times, the “Simulation 2” curve results as an update of the *AllOrNothingRegulator* every 10 step times. The “Consigne” curve represents the temperature chosen by the user and the “Range inf” and “Range Sup” curves represent the area in which the room temperature should be included. The simulation conditions are the same as for the previous simulation results in Figure 14 and Figure 15.

According to Figure 16, it is clear that the update step time of the *AllOrNothingRegulator* impacts its behavior.

Moreover, these results show that the *AllOrNothingRegulator*, which is the “brain” of the regulation loop [7], does not comply with all the requirements. The *Accuracy* requirement (the room temperature should be included within a defined range of +/-1 degree Celsius, represented as the “range inf” and “range Sup” curves in Figure 16) and the *Comfort* requirement (the system should establish the chosen temperature with no pumping of the temperature), for instance, are not satisfied by this regulator, according to Figure 16.

5. Conclusions

This simple example scans the specific SysML features as well as functionality provided by the SysML plug-in and emphasizes how a SysML model can be built. In a system engineer perspective, SysML provides all the needed features for an accurate model and can also be customized to make it more suitable for a specific field of application. SysML also permits an improved communication among people who work on a project by reducing ambiguities and allowing a more complete representation of the system. Another benefit of SysML and UML is the architecture model reuse and maintainability. Furthermore, this plug-in is intuitive to use for someone knowledgeable of UML.

The SysML plug-in for MagicDraw UML is not yet able to run simulations. However, the code generation tool provided by MagicDraw allows fast creation of class stubs in several programming languages and allows a correspondence between MagicDraw UML and Integrated Development Environment software able to run the needed simulations. Future versions of the plug-in should consider adding such tools able to run simulations, as well as the ability to communicate with an engineering tool such as MathModelica® and 3D representation software such as CATIA® or AutoCAD®.

References

- [1] Régulation, principes; University of Le Mans, France:
<http://www.univ-lemans.fr/enseignements/physique/02/electro/regulation.html>, March 2008 [Online]
- [2] Energie+,
<http://www-energie.arch.ucl.ac.be/cdrom/Climatisation/theorie/clithPrincipeReguPID2.htm>, March 2008 [Online]
- [3] Ecole d'Ingénieurs de Genève, Switzer land,
http://www.eig.ch/fileadmin/laboratoires/systeme_asservis/SAth72.pdf, March 2008 [Online]
- [4] The OMG SysML™, Version 1.0
OMG Systems Modeling Language (OMG SysML™), V1.0
<http://www.omg.org/cgi-bin/doc?formal/2007-09-01>, March 2008 [Online]
- [5] Heat Transfer,
http://en.wikipedia.org/wiki/Heat_transfer, June 2008 [Online]
- [6] Edmond Julien and Jean-Noël Foussard from CNRS France, 2005
Thermodynamique Bases et applications
- [7] R. Rode, Régulation analogique,
http://www.educnet.education.fr/rnchimie/gen_chim/regulation/rhode/reg_ana.pdf, March 2008 [Online]

Contacts

Please contact us support@nomagic.com for getting more detailed information about No Magic services, training courses, scheduling possibilities, or if you have any specific needs for your company.



No Magic, Inc.

One Allen Center
700 Central Expressway South,
Suite 110
Allen, Texas 75013
Phone: +1-214-291-9100
Fax: +1-214-291-9099
www.nomagic.com

No Magic Europe

Savanoriu ave. 363, Kaunas
LT - 51480, Lithuania
Phone: +370-37-324032
Fax: +370-37-320670
www.nomagic.lt

No Magic Asia

719 KPN Tower, 22nd floor,
Rama IX Road, Bangkapi, Huaykwang,
Bangkok 10310, Thailand
Phone: +66-2-7170250
Fax: +66-2-7170251
www.nomagicasia.com