**Smart Home Design**

Ansgar RADERMACHER, Asma SMAOUI  
(based on work of Jérémie Tatibouët and Shuai Li)

# Objective

Your task is in the domain of Internet of Things (IoT). An IoT is a network of (traditional) physical objects with embedded sensors, actuators, electronics, software, etc. The objects are thus connected and exchange data to improve real world productivity and quality of life. IoT systems can be found in general home appliances (e.g. window shield controller) to large-scale industrial systems (e.g. agricultural irrigation regulator).

You will design a Smart Home IoT system capable of:

1. Controlling the internal temperature of a home.
2. Detecting the presence of smoke and / or the presence of carbon monoxide in the home.

It will be possible to control the system thanks to a dedicated controller available in the home.

# Setup

Copy folder ExamStudentModel from the git

1. Import the project ExamStudentModel
2. Open the model in SmartHome.

The model is divided into a top-level class and three main packages:

1. **SmartHome**: this top level class represents the system component to design
2. **Hardware**: this package shall contain the definition of hardware components of your system.
3. **Software**: this package shall contain the definition of software components of your systems.
4. **Communications**: this package shall contain the definition of the communications protocols. (signals, interfaces, etc…)

# Structure

## System

### System design requirements:

The system component is the **SmartHome**. This component shall contain all parts of your system.

A **SmartHome** is composed of the following hardware components:

1. A **temperature sensor**. The purpose of this component is to provide the capability to measure the temperature at the sensor location.
2. A **smoke detection sensor**. The purpose of this component is to provide the capability to detect the presence of smoke at the sensor location. It can also be used to detect the presence of carbon monoxide.
3. An **alarm**. The purpose of this component is to provide the capability to emit a sound signal during a defined period of time.
4. The **heating actuators system (ACS)**. The purpose of this component is to translate orders of a software component into physical actions. In the context of the smart home system, the heating actuator system allows to physically control the heating system located in the home.

A **SmartHome** is composed of the following **software** components

1. A **temperature manager**. The purpose of this component is to read the temperature sensor and to take decisions (based on the read value) regarding the usage of the heating system.
2. A **smoke detection manager**. The purpose of this component is to read the smoke detection sensor and to take decisions (based on the read values) regarding the activation of the alarm.
3. A **home controller**. The purpose of this component is to offer to the user to turn on or off the temperature manager and smoke detection manager.

### System design Papyrus instructions

Complete the **Architecture (BDD)** diagram so that the **SmartHome** is composed of *Properties* typed with the hardware and software components.

Hints: You may use the *Palette* to create *Composite Association (Directed)* specify a composition relationship (one component owns a *Property* typed by the other component).

## Hardware

### Hardware design requirements

Make sure your hardware components design conforms to the following requirements:

|  |  |
| --- | --- |
| **Components** | **Requirements** |
| Temperature Sensor | The sensor can receive the operation call **getTemperature() : Real** |
| The sensor is directly connected with the **Temperature Manager** through a port of the latter. |
| Smoke Detection Sensor | The sensor can receive the operation calls **isSmokeDetected() : Boolean** and **isCODetected() : Boolean** |
| The sensor is directly connected with the **Smoke Detection Manager** through a port of the latter. |
| Heating ACS | The **heating ACS** can receive the operation calls **startHeating()** and **stopHeating().** |
| The **heating ACS** is directly connected with the **Temperature Manager** through a port of the latter. |
| Alarm | The alarm can receive the operation calls **trigger()** and **stop()**. |
| The alarm is directly connected with the **Smoke Detection Manager** through a port of the latter. |

### Hardware design Papyrus instructions:

In the *Model Explorer*, complete the *Operations* of the hardware components (*Classes*) in the **Hardware** package.

Complete the **Architecture (IBD)** diagram by displaying the parts (i.e. *Properties*) of the **SmartHome**, and the eventual *Ports* of the parts, and connecting them according to the software design requirements above. At this moment, it is possible that some *Ports* are not typed (you will do it during communication protocols design).

Hints: Usually *Ports* are connected together but it is also possible to connect a *Property* directly to a *Port* (e.g. this is the case for connections between hardware and software parts).

The use of MARTE HRM is a bonus, but standard SysML IBD mechanism are also acceptable

## Software

### Software design requirements:

Make sure your software components design conforms to the following requirements:

|  |  |
| --- | --- |
| **Components** | **Requirements** |
| Temperature Manager | This manager stores the **target temperature** and the **current temperature**. |
| This manager can receive signals **ON**, **OFF**, **TEMPERATURE**. |
| The manager is connected to the **Home Controller** through one of its ports. |
| Smoke Detection Manager | This manager stores the values of **smoke detected** and **CO detected**. |
| This manager can receive signals **ON**, **OFF**. |
| The manager is connected to the **Home Controller** through one of its ports. |
| Home Controller | This manager can receive signals **ON**, **OFF**, **RESET**, **CONFIGURATION**. |
| The manager is connected to the **Temperature Manager** and **Smoke Detection Manager** through a port each. |

Note: Signals received by the **Home** **Controller** represent the interactions performed by a habitant while using the physical interface provided for that controller. For instance, a habitant can push the OFF button and as result the controller will make sure the **Temperature Manager** and the **Smoke Detection Manager** are turned off. Capability of the **Home Controller** to receive interactions from its environment (i.e., from a habitant) is intentionally not modelled here in order to keep the model simpler and only focus on internal components interactions.

### Software design Papyrus instructions:

In the *Model Explorer*, complete the *Properties*, *Ports*, and *Signal Receptions* of the software components (*Classes*), in the **Software** package.

Complete the **Architecture (IBD)** diagram by displaying the parts (i.e. *Properties*) of the **SmartHome**, and the eventual *Ports* of the parts, and connecting them according to the software design requirements above. At this moment, it is possible that some *Ports* are not typed (you will do it during the design of the communication protocols).

## Communication protocol

### Communication protocol design requirements:

Make sure your communication protocols design conforms to the following requirements:

|  |  |
| --- | --- |
| **Components** | **Requirements** |
| Sensors Protocol | Components using these protocols can call operations of the sensors. |
| The protocols are realized by the sensors. |
| HeatingACS Protocol | Components using these protocols can call operations of the **Heating ACS**. |
| This protocol is realized by the **Heating ACS**. |
| Alarm Protocol | Components using these protocols can call operations of the **Alarm**. |
| This protocol is realized by the **Alarm**. |
| Managers Protocol | Components using these protocols can send signals to the managers. |
| These protocols are realized by the managers. |

### Communication protocol design Papyrus instructions:

In the **Communications** package, you have sub-packages each containing a diagram corresponding to a set of protocols described above (ignore the **Internal Events** sub-package). Complete these diagrams with *Interfaces* and the required relationships between these interfaces and the hardware and software components.

In the **Architecture (IBD)** diagram, type the *Ports* with the *Interfaces* you specified previously. Make sure the provided and required interfaces of a *Port* is consistent with its connection.

Hints: A *Port* typed with an *Interface* provides that *Interface*. You can specify instead that the port requires the *Interface* by setting *isConjugated* of the *Port* to *true* in the *Properties* view.

# Behavior

The **Home Controller**, the **Temperature Manager** and the **Smoke Detection Manager** are all software components. These components are defined as active classes whose behaviors are specified using UML state machines.

## Home Controller

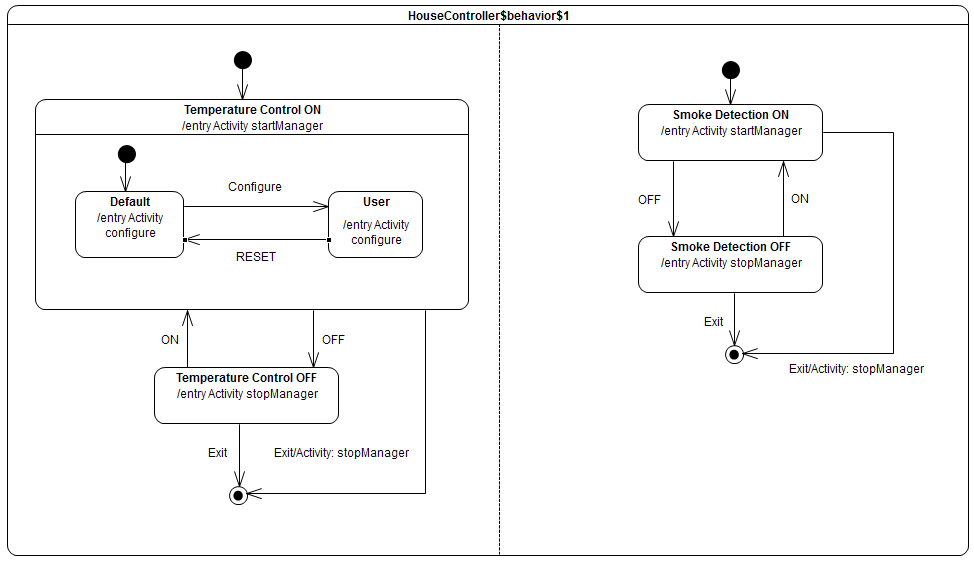


Figure 1 - Home Controller Classifier Behavior

The **Home Controller** is responsible for controlling both the temperature and smoke detection managers. The behavior of the **Home Controller** is shown in Figure 1 and meets the following requirements:

|  |  |
| --- | --- |
| **ID** | **Description** |
| #1 | The **Home Controller** controls both the **Temperature** and **Smoke Detection** manager in parallel. This implies that both managers can be ON, both can be OFF, the **Temperature Manager** can be ON while the **Smoke Detection Manager** is OFF and vice versa. |
| #2 | If the ON button of the **Temperature Manager** is pushed, then the **Temperature Manager** shall get started (an ON signal occurrence is sent to this manager) and the **target temperature** shall be set to the default temperature: 20.0°C. This temperature shall be sent to the **Temperature Manager** as an occurrence of the **Temperature** signal. |
| #3 | If the Configure button of the **Temperature Manager** is pushed, then the user shall be able to set a new **target temperature**. This temperature shall be sent to the **Temperature Manager** as an occurrence of the **Temperature** signal. |
| #4 | It shall always be possible to switch from a user defined **target temperature** to the default one. This happens when the **Temperature Manager** Reset button is pressed. |
| #5 | If the OFF button of the **Temperature Manager** is pressed while this manager is ON then this manager shall be turned off. |
| #6 | If the Exit button is pressed, then the **Home Controller** shall get shutdown. This implies the both the **Temperature** and **Smoke Detection Manager** are previously stopped. Stopping a manager implies that an OFF signal is sent to this latter. |
| #7 | If the ON button of the **Smoke Detection Manager** is pushed, then this manager shall get started. This implies an ON signal occurrence shall be sent to the **Smoke Detection** **Manager**. |
| #8 | If the OFF button of the **Smoke Detection Manager** is pushed, then this manager shall get turned off. This implies and OFF signal occurrence shall be sent to the **Smoke Detection Manager**. |

**Note**: you do not have to change / update the state machine implementing the behavior of the home controller. However, it is very important that you understand what is the specified behavior in order to be able to specify the behaviors of the **Temperature** and **Smoke Detection Manager**.

## Temperature Manager

### Temperature manager behavior requirements

This table presents the requirements that shall be met by the **Temperature Manager**.

|  |  |
| --- | --- |
| **ID** | **Requirements** |
| #1 | The manager can be either in mode *ON* or *OFF*. |
| #2 | When the manager is in the *ON* mode it shall read the temperature sensor. |
| #3 | When the sensor was read the manager shall either decide to start the heating or to keep this latter in standby mode. The reading of the temperature sensor always occurs by calling the **getTemperature()** operation on the temperature sensor. |
| #4 | The manager shall trigger the heating system if the measured temperature is lower than the target temperature. Triggering the heating system implies to call the **startHeating()** operation on the **HeatingACS** component. |
| #5 | The manager shall keep the heating in the standby mode if the measured temperature is greater than or equal to the target temperature. Moving to the standby move implies to stop the heating system. Stopping the heating system can performed by calling the **stopHeating()** operation |
| #6 | The temperature measured from the sensor shall be read every 50ms by the manager. |
| #7 | The **Temperature Manager** shall have the capability to account for any target temperature change received from the home controller. If such event is received, then the target temperature shall be updated and the manager shall be forced to read the temperature sensor value. |

### Temperature manager behavior design Papyrus instructions

In the model, you must complete the state machine of the **Temperature Manager** available under the **Software** package. This state machine shall meet the abovementioned requirements.

To complete the state machine, you will contribute to the diagram **TemperatureManager$behavior$1**. While completing the state machine, do not forget to add any required state entry / exit behaviors as well as effect on transitions. These behaviors are expected to be implemented as activities.

Hints: in the context of the **Temperature Manager** behavior you will likely have to add guards on transitions. To create a guard, you must:

1. Create a constraint playing the role of guard.
2. Create a specification for that constraint. The specification will be an Opaque Expression.
3. Select the language in which you specify the guard.
4. Write the piece of code corresponding to the guard specification.

In the context of the **Temperature Manager** behavior you will also have to defined triggers for time events. All time events required to be used in this model are already defined in the **InternalEvents** package.

## Smoke Detection Manager

### Smoke detection manager behavior requirements

This table presents the requirements that shall be met by the **Smoke Detection Manager**.

|  |  |
| --- | --- |
| **ID** | **Requirements** |
| #1 | The manager can be either in mode *ON* or *OFF*. |
| #2 | When the manager is in the *ON* mode it shall read the smoke detection sensor. The reading of the sensor consists into calls to operations **isSmokeDetected()** and **isCODetected().** |
| #3 | As soon as the sensor values were read, the manager shall either decide to trigger the **alarm** or not. The **alarm** shall be triggered if either carbon monoxide or smoke is detected. |
| #4 | If the manager decides to trigger the **alarm** then the operation **trigger()** shall be called on the **alarm** component. |
| #5 | The **alarm** gets shutdown automatically 300s after it was triggered. This implies the operation **stop()** shall be called on the **alarm** component. |
| #6 | When the **alarm** is shutdown then the manager shall return in the mode in which it can read the sensor values. |
| #7 | The sensor values shall be read every 50ms. |

### Smoke detection manager behavior design Papyrus instructions

In the model, you must complete the state machine of the **Smoke Detection Manager** available under the **Software** package. This state machine shall meet the above mentioned requirements.

To complete the state machine, you need to contribute to the **SmokeDetectionManager$behavior$1** diagram. While completing the state machine, do not forget to add any required state entry / exit behaviors as well as effect on transitions.