

System documentation of a heating control system

This document has been produced, within the context of software engineering courses at the University of Kaiserslautern by Marco Heide, Wolfgang Wagenbichler, Andreas Schank, Angela Schmidt and Axel Seck, then reworked by Christiane Differding and Matthias Gutheil. It has been translated in english and adapted by Mario Negro Ponzi of Politecnico di Torino. It has subsequently undergone extensive rework by Vincenzo D'Elia, Alfredo Pironti and Massimiliano Schillaci of Politecnico di Torino.

V 3.1 march 2009

Abstract

A heating control system monitors and control temperature in a building composed of several rooms. It is a good example of a software system. The related documentation is divided in three parts: a problem-solution documentation, software-system documentation and software-component documentation. This document refers to the first part.

Table of contents

Part 1	Problem solution documentation	2
1.1	Problem description	2
1.1.1	Problem	2
1.1.2	Expected functionalities.....	2
1.1.3	Environment characteristics.....	3
1.2	Requirements / specifications	3
1.2.1	User requirements / specifications	3
1.2.2	User requirements analysis	5
1.2.3	Developer's Requirements/Specifications	9
1.2.4	Traceability matrix: user requirements <-> class diagrams.....	38

Part 1 Problem solution documentation

1.1 Problem description

The following problem description is an implementation and adjustment of a heating control system. A house is composed by many subsystems (e.g. rooms or devices as heaters), that are connected with doors and tubes. It is also connected with the outside world with doors and windows. A house is also made of a texture of subsystems that strictly interact with each other. A software system should be developed that controls and manages this complex system. It should monitor weather conditions and optimize energy consumption.

Below the idea of this system is described. A user interface for getting and setting room parameters to the system is an integral part of the system.

1.1.1 Problem

Up until today the temperature of every room is regulated by a thermostat in the heater. A target temperature for each room can be set. An automatic management of the temperature is based only on heaters. A window also is useful to control temperature in a single room. But it should be opened or closed by someone. Water in heaters has a fixed temperature set by a boiler.

Problems in temperature management are:

1. Heater's temperature cannot be differently set basing on heating demand or time. There is no regulation linked with people presence
2. Thermal regulation is limited to heating. The use of windows to cool down is based on user intervention
3. There is no optimization of boiler temperature. The boiler's temperature is set to the maximum value regardless whether the temperature is too low or too high. That produces an energy loss.

The above control system is not satisfying both because of energy loss and because it is not comfortable.

1.1.2 Expected functionalities

The user wants to be able to individually set temperature for each room using a terminal. So he should be able to provide described parameters.

First it should be possible to set a temperature that could be maintained when there is someone in the room. In doing so the system should not react immediately when someone comes in. It should be provided a time span that indicates how long a person should stay in the room before the system takes control. The user should also be able to set the time span in which the desired temperature should be reached. The provided time span can be maintained heating up heaters for a small period of time over the desired temperature.

Also the user should be able to set a minimum temperature that is acceptable when there is no one in the room. In this case, too, a time span should be provided, after everyone has left the room, before the system takes control.

There should also be default values available for every parameter.

Windows, too, should be controlled by the system. If a window has been completely opened by a user, the system should maintain only the minimal temperature. If the window is open and it starts raining the system should put it in tilt position. More, the windows should be usable by the system to cool down when the outside temperature is cooler, by tilting it.

1.1.3 Environment characteristics

Below are described the characteristics of the environment inside which the system will operate.

1.1.3.1 House

The house is composed by the following rooms:

- living room (40sqm)
- eating room (15sqm)
- sleeping room (15sqm)
- children room (20sqm)
- visitors room (15sqm)
- work room (10sqm)
- kitchen (15sqm)
- bathroom (5sqm)
- visitors bathroom (5sqm)
- hall (10sqm)

The house also has:

- an outside temperature sensor
- a rain switch (closed if it rains)

1.1.3.2 Room

Each room is so configured (as a result of the whole system concept):

- 1 window with two switches and two actuators. A switch gets opened when the window is completely open (open when the window is open / NC = normally closed). The other when is tilted (open when the window is tilted / NC = normally closed). When the window is completely open the tilt switch is closed. An actuator swings the window while the other tilts it.
- 1 heater (hot water) with a thermostat
- 1 presence switch (closed when someone is in the room)
- 1 room temperature sensor

1.1.3.3 Centralized heating room

The control of the heating system is an already deployed autonomous subsystem. The maximum temperature is fixed at 70 degrees.

The heating pump is controlled with impulses and it is so robust that there is no loss in the way between boiler and heaters.

1.2 Requirements / specifications

Below system requirements are depicted. They will be divided in User and Developer requirements. As notation UML will be used. The UML diagrams to describe requirements will be ordered in User-Requirements and Developer-Requirements. The ordering follows the comprehensibility degree of the model from the point of view of the user. To user requirements also belong Use Cases and Scenarios. The rest of the model is in the Developer requirements described in chapter 1.3.2. There will also be a modeling of dynamic views.

1.2.1 User requirements / specifications

Below will be described functional, not functional and inverse user Requirements. Then user's conceptual choices will be described. In chapter 1.2.2 Use Case diagrams and Scenarios will be depicted that are based on User Requirements.

1.2.1.1 Functional User Requirements

Functional User Requirements are originated by problem description (chapter 1.2) and from opinions that are collected in interviews with the user.

UR-F	Requirement's description
Temp-UR-F 1	The user shall be able to set <i>StandardPresenceTemp</i> in °C
Temp-UR-F 2	The user shall be able to set <i>StandardAwayTemp</i> in °C
Temp-UR-F 3	The user shall be able to set a time span <i>StandardHeatingTime</i> in minutes
Temp-UR-F 4	The user shall be able to set a time span <i>StandardAwayTime</i> in minutes
Temp-UR-F 5	The user shall be able to set a time span <i>StandardInsideTime</i> in minutes
Temp-UR-F 6	The temperature of Temp-UR-F 1 shall be maintained when there is someone in the room and the <i>StandardInsideTime</i> from Temp-UR-F 5 has elapsed
Temp-UR-F 7	The temperature of Temp-UR-F 2 should be maintained in the room when the room is empty and <i>StandardAwayTime</i> from Temp-UR-F 4 has elapsed
Temp-UR-F 8	The time span from Temp-UR-F 3 indicates after how many minutes the <i>StandardPresenceTemp</i> from Temp-UR-F 1 after the entrance of a person in the room shall be reached.
Temp-UR-F 9	The time span from Temp-UR-F 4 indicates for how many minutes the <i>StandardPresenceTemp</i> from Temp-UR-F 1 shall be maintained after the last person has left the room
Temp-UR-F 10	The time span from Temp-UR-F 5 indicates for how long shall someone stay in the room for the system to get in control
Temp-UR-F 11	The <i>StandardHeatingTime</i> from Temp-UR-F 3 must be at least 5 minutes longer than the <i>StandardInsideTime</i> from Temp-UR-F 5
Temp-UR-F 12	For each quantity from Temp-UR-F 1, Temp-UR-F 2, Temp-UR-F 3, Temp-UR-F 4, Temp-UR-F 5 default values shall exist.
Temp-UR-F 13	If a window gets opened the system maintains the room temperature at <i>StandardAwayTemp</i> , whether or not there is someone in the room.
Temp-UR-F 14	If it starts raining with the window open, the system will set the window in a rain secure position.
Temp-UR-F 15	When the temperature of a room with someone inside exceeds that from Temp-UR-F 1 and the outer temperature is lower than the room temperature, the window in this room gets tilted
Temp-UR-F 16	The control of the temperature is admitted with an error of +/-1 °C
Temp-UR-F 17	The system is planned for the room described on page 4
Temp-UR-F 18	The temperature of the boiler shall be set to the maximum required by any heater, not to waste energy.

Table 1

1.2.1.2 Non-functional user requirements

The system should complain also with the following requirements.

UR-NF	Requirement's description
UR-NF 1	The system should be easily changeable.

Table 2

1.2.1.3 Inverse user requirements

Following situations should be avoided.

UR-Inv	Requirement's description
--------	---------------------------

UR-Inv 1	When there is nobody in the room and the room is hotter than target temperature, the window does not get tilted to cool.
UR-Inv 2	If the user manually opens the window, the system shall not tilt it to the rain secure position.

Table 3

1.2.1.4 Design decisions

Following decisions have been taken.

DD	Requirement's description
DD 1	The system should be object oriented.

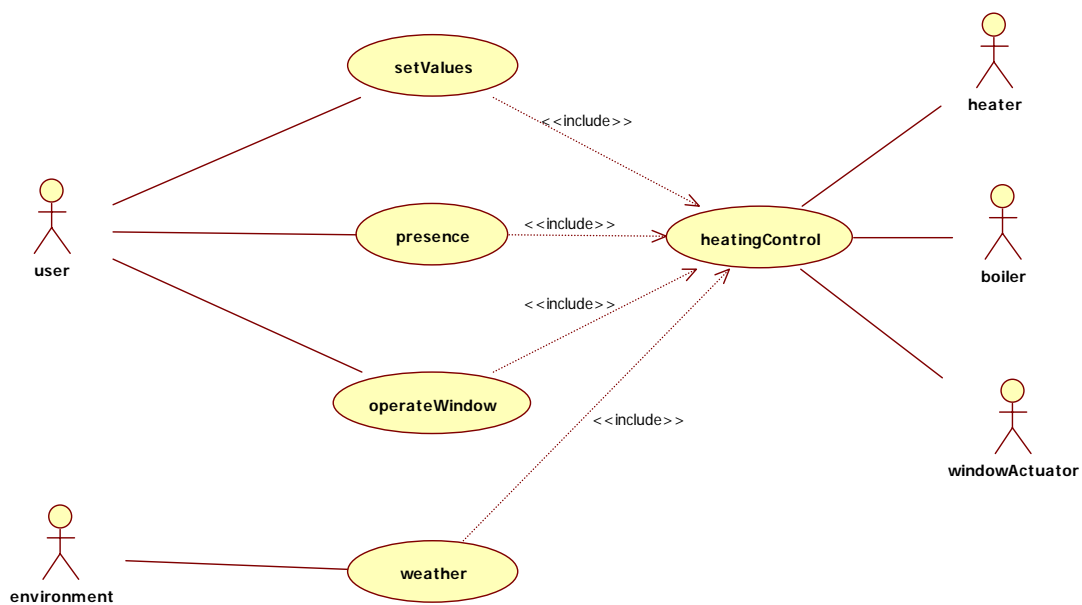
Table 4

1.2.2 User requirements analysis

In the following UML use case diagrams and Scenarios derived from user requirements are reported, as noted in the introduction of chapter 1.2.1.

1.2.2.1 Use case diagram

Use cases typically depict interactions between external actors and the system from the point of view of the user. The interaction among actors and the system starts with initial events triggered by the actor on the system and ends when the logical reaction of the system has finished.



Use Case: presence

The activating actor of this Use Case is the user. The presence of a person in the room is notified to the system by the presence switch.

This Use Case will be shown in the following scenarios:

- Scenario 2

- Scenario 3
- Scenario 4

Use Case: operateWindow

The activating actor of this Use Case is the user that opens or closes the window. The system detects the changed position through the open window switch and the tilted window switch.

This Use Case will be shown in the following scenario:

- Scenario 5

Use Case: heatingControl

This Use Case describes the recalculation of the temperature of the heater (to heat, maintain or cool). At the same time the new temperature of the boiler is notified.

This Use Case will be shown in the following scenarios:

- Scenario 7
- Scenario 9
- Scenario 10
- Scenario 11

Use Case: weather

The activating actor of this Use Case is the environment. The rain sensor is monitored by the system to discover whether it's raining on the house.

This Use Case will be shown in the following Scenario:

- Scenario 6

Use Case: setValues

The user sets various values. The activating actor is the user.

This Use Case will be shown in the following scenarios:

- Scenario 1
- Scenario 8

1.2.2.2 Scenarios for Use Cases

The following scenarios describe the system in common situations and realize the Use Cases. So they must show every functional and inverse user requirement.

Other notes (to the following table):

- (-) after a User Requirement indicates that a specific precondition (e.g. a time span elapsing) has not been met
- (+) after a User Requirement indicates that a specific precondition has been met
- Without any other specific note, in the following scenarios those following default values should be considered:
 - o StandardPresenceTemp: 20 °C
 - o StandardAwayTemp: 15 °C
 - o StandardHeatingTime: 10 minutes
 - o StandardInsideTime: 5 minutes
 - o StandardAwayTime: 15 minutes

#	Step	Scenario's description	User requirement
Scenario 1		<i>Scenario for Use Case setValues. User changed StandardHeatingTime, StandardInsideTime and StandardPresenceTemp. The room will be heated up.</i>	

	1.	Presence switch: closed Outside Temperature: 21 °C Window: closed Rain switch: open Target temperature: 19 °C (StandardPresenceTemp) Room temperature: 18 °C	
	2.	The user sets StandardInsideTime to 5 minutes	Temp-UR-F5
	3.	The user sets StandardHeatingTime to 9 minutes	Temp-UR-F3
	4.	The system ignores new time set, because it is just 4 minutes over StandardInsideTime and throws an error message	Temp-UR-F11
	5.	The user sets StandardHeatingTime to 10 minutes	Temp-UR-F3
	6.	The user sets StandardAwayTemp to 16 °C	Temp-UR-F2
	7.	The user sets StandardAwayTime to 10 minutes	Temp-UR-F4
	8.	The user sets StandardPresenceTemp to 21 °C	Temp-UR-F1
	9.	The target temperature is set at 21 °C	
		-> Use Case heating control	
Scenario 2	<i>Scenario for Use Case presence</i> <i>The user leaves the room before the system activates</i>		
	1.	Presence switch: open Outside temperature: 21 °C Window: closed Rain sensor: open Target temperature: 15 °C (StandardAwayTemp) Room temperature: 18 °C	
	2.	A user enters in the room	
	3.	The user leaves the room before the StandardInsideTime has elapsed	Temp-UR-F10 (-)
	4.	The target temperature does not change	Temp-UR-F7
Scenario 3	<i>Scenario for Use Case presence</i> <i>The user leaves the room for a long time, affecting system behavior</i>		
	1.	Presence switch: closed Outside temperature: 16 °C Window: closed Rain sensor: open Target temperature: 20 °C (StandardPresenceTemp) Room temperature: 19 °C	
	2.	The user leaves the room, which becomes empty	
	3.	After StandardAwayTime the target temperature is set to StandardAwayTemp.	Temp-UR-F7
		-> Use Case Heating control	
4.	The window does not get opened because there is no one inside the room.	UR-Inv 1	
Scenario 4	<i>Scenario for Use Case presence</i> <i>There is only one user in the room. This user leaves for a short time the room so that it does not have any effect on the system behavior</i>		
	1.	Presence switch: closed Outside temperature: 19 °C Window: closed Rain sensor: open Target temperature: 20 °C (StandardPresenceTemp) Room temperature: 19 °C	
	2.	The user leaves the room, which becomes empty	

	3.	The user comes back in the room before StandardAwayTime has elapsed.	Temp-UR-F7 (-)
	4.	The temperature does not change	Temp-UR-F6
Scenario 5	<i>Scenario for use Case operateWindow</i> <i>The window is open and a user is present. Target temperature is set to StandardAwayTemp</i>		
	1.	Presence switch: closed Outside temperature: 19 °C Window: closed Rain sensor: open Target temperature: 20 °C (StandardPresenceTemp) Room temperature: 20 °C	
	2.	The user opens the window	
	3.	Target temperature is set to StandardAwayTemp	Temp-UR-F13
	-> Use Case Heating Control		
Scenario 6	<i>Scenario for Use Case weather</i> <i>It rains with the window open</i>		
	1.	Presence switch: closed Outside temperature: 17 °C Window: completely open Rain sensor: open Target temperature: 15 °C (StandardAwayTemp) Room temperature: 18 °C	
	2.	It starts raining	
	3.	The system closes the window	Temp-UR-F14
	4.	Target temperature is set to StandardPresenceTemp	Temp-UR-F6
	-> Use Case Heating Control		
	5.	The user opens again the window	
	6.	Since the window has been opened manually, it doesn't get closed by the system	Temp-UR-F14 UR-Inv2
	7.	Target temperature is set to StandardAwayTemp	Temp-UR-F13
-> Use Case Heating Control			
Scenario 7	<i>Scenario for Use Case heatingControl</i> <i>The window gets controlled by the system to cool</i>		
	1.	Presence switch: closed Outside temperature: 17 °C Window: closed Rain sensor: open Target temperature: 20 °C (StandardPresenceTemp) Room temperature: 21 °C	
	2.	The room temperature raise to 22 °C and so is more that 1 °C over target temperature.	Temp-UR-F16
	-> Use Case Heating Control		
	3.	The system controls the window to cool down	Temp-UR-F15
Scenario 8	<i>Scenario for Use Case setValues</i> <i>The user sets all values to default</i>		
	1.	The user sets StandardPresenceTemp to default	Temp-UR-F12
	2.	The user sets StandardAwayTemp to default	
	3.	The user sets StandardInsideTime to default	
	4.	The user sets StandardAwayTime to default	
	5.	The user sets StandardHeatingTime to default	
Scenario 9	<i>Scenario for Use Case heatingControl</i> <i>Target Temperature of a room is reached</i>		

	1.	Presence switch: closed Outside temperature: 14 °C Window: closed Rain sensor: open Target temperature: 20 °C (StandardPresenceTemp) Room temperature: 18 °C	
	2.	Room temperature reach 19 °C	
	3.	Stop temperature has been reached, since room temperature is just 1 °C below target temperature.	Temp-UR-F6 Temp-UR-F16
	4.	Required new boiler temperature is computed	Temp-UR-F18
Scenario 10	<i>Scenario for Use Case Heating Control</i> <i>The room must be heated up</i>		
	1.	Presence switch: closed Outside temperature: 23 °C Window: closed Rain sensor: open Target temperature: 20 °C (StandardPresenceTemp) Room temperature: 19 °C	
	2.	Room temperature drops to 18 °C	
	3.	Heating temperature gets calculated because room temperature has dropped below 1 °C under target temperature.	Temp-UR-F6 Temp-UR-F8 Temp-UR-F16
	4.	Heater temperature is set to the necessary value	
	5.	Required new boiler temperature is computed	Temp-UR-F18
Scenario 11	<i>Scenario for Use Case Heating Control</i> <i>Room temperature is above target temperature</i>		
	1.	Presence switch: closed Outside temperature: 23 °C Window: closed Rain sensor: open Target temperature: 20 °C (StandardPresenceTemp) Room temperature: 21 °C	
	2.	Room temperature raises to 22 °C	
	3.	As the temperature of the room is more than 1 °C over target temperature, heater temperature is set to 0 °C	
	4.	Required new boiler temperature is computed	Temp-UR-F18
	5.	The window is not opened to cool down room temperature.	Temp-UR-F15(-)

1.2.3 Developer's Requirements/Specifications

Here UML diagrams that describe requirements follow. These include a class diagram with Data-Dictionary, Sequence Diagrams and State Diagrams.

1.2.3.1 UML class diagrams

In class diagrams the static structure of the system is described, composed of the classes and their relationships. It is further detailed in the Data-Dictionary.

1.2.3.1.1 Package structure

This diagram shows the partitioning of the system into subsystems (Packages). Different packages all have their own classes. In a package methods and attributes of classes aren't defined.

Package: hcs.interfaces

Includes abstract classes that capture the common behavior of parts of the heating control system.

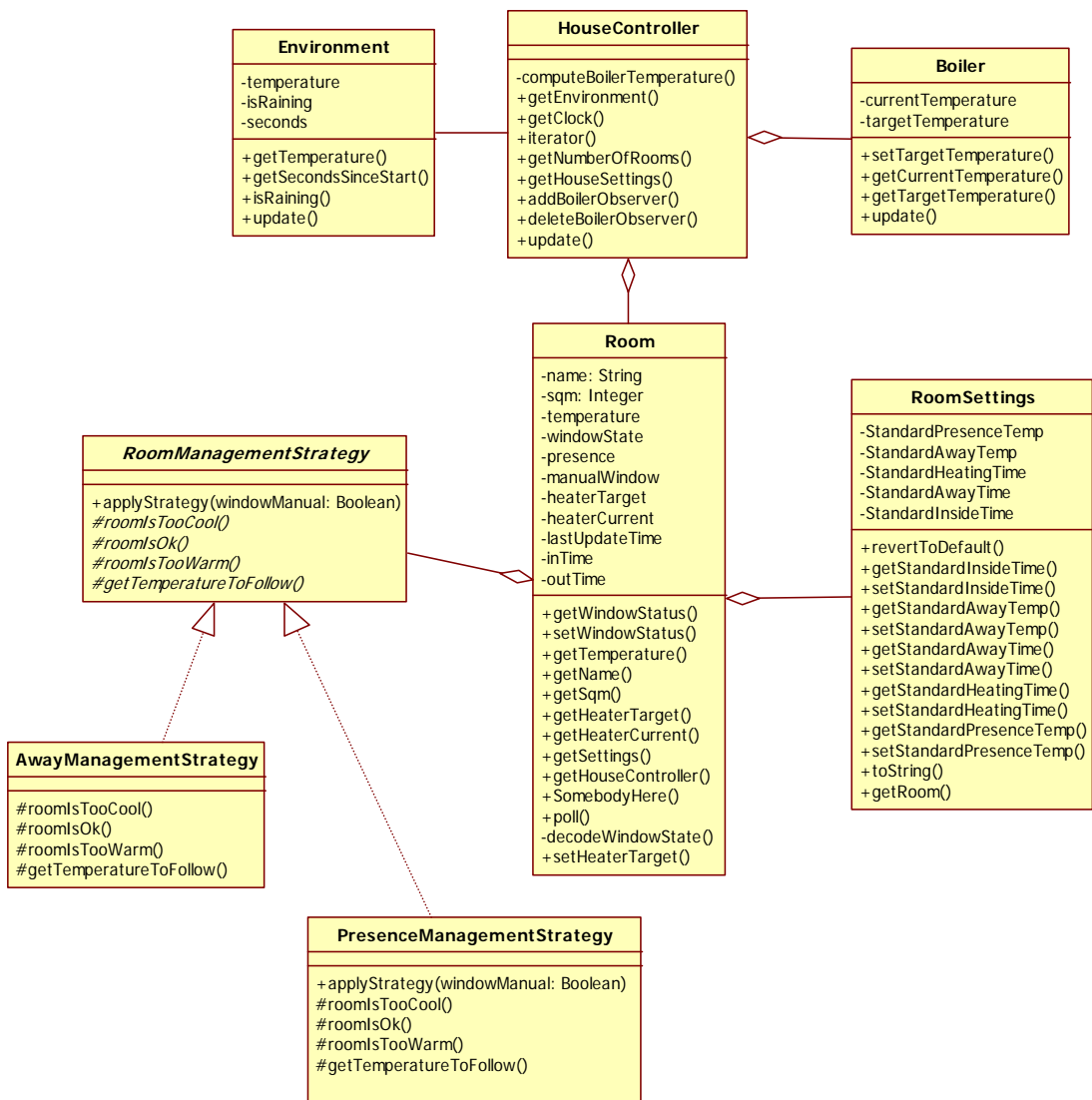
Package: hcs.settings

Is composed of classes that handle the global settings of the system.

1.2.3.1.2 Temperature control

This diagram shows the main classes that participate in controlling temperature. For clarity the classes that only contribute to the hardware wrapper are omitted, as well as the classes that handle the global system settings.

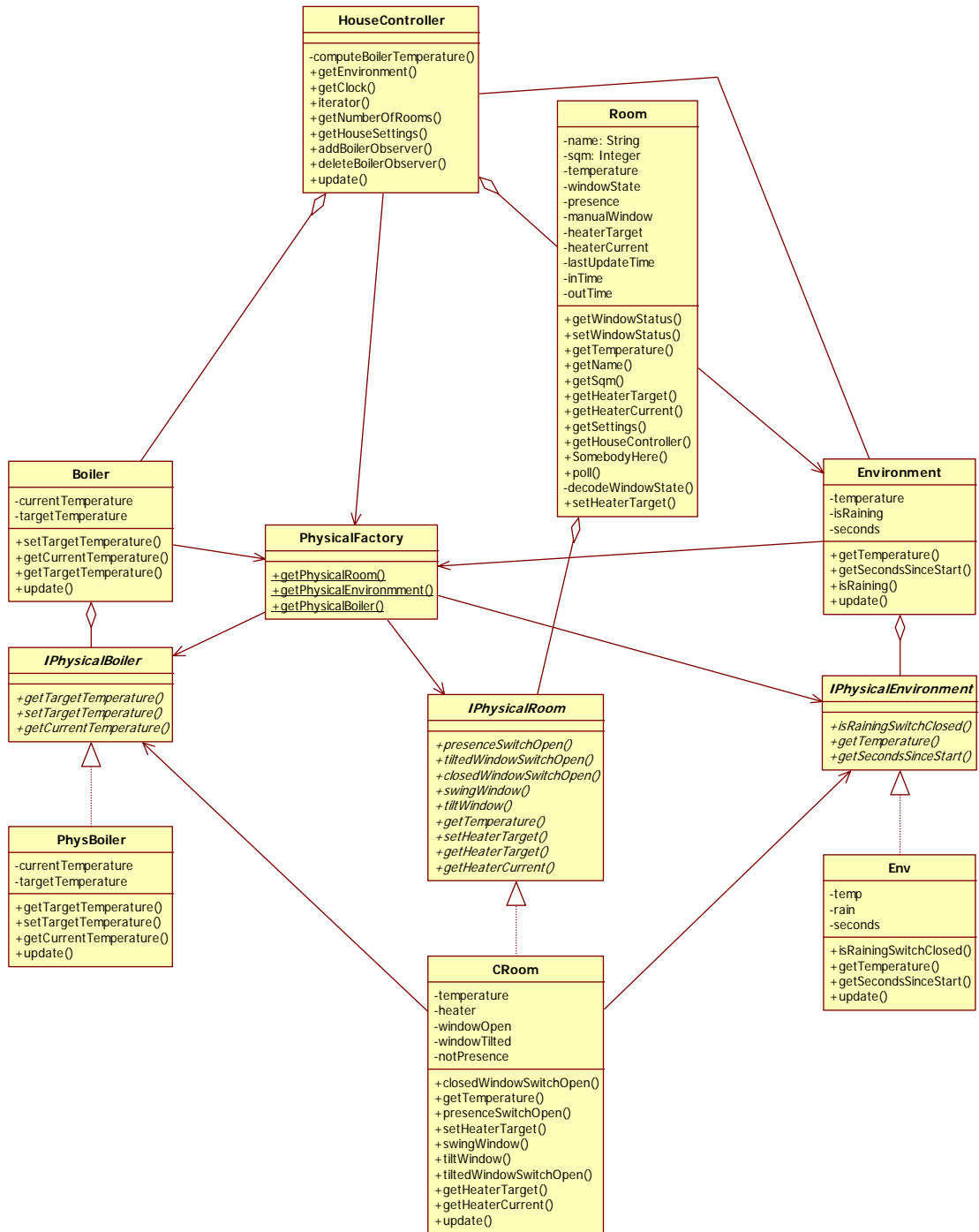
Class diagram



1.2.3.1.3 Hardware_wrapper

This diagram shows all the classes that represent the logical and abstract view of the hardware

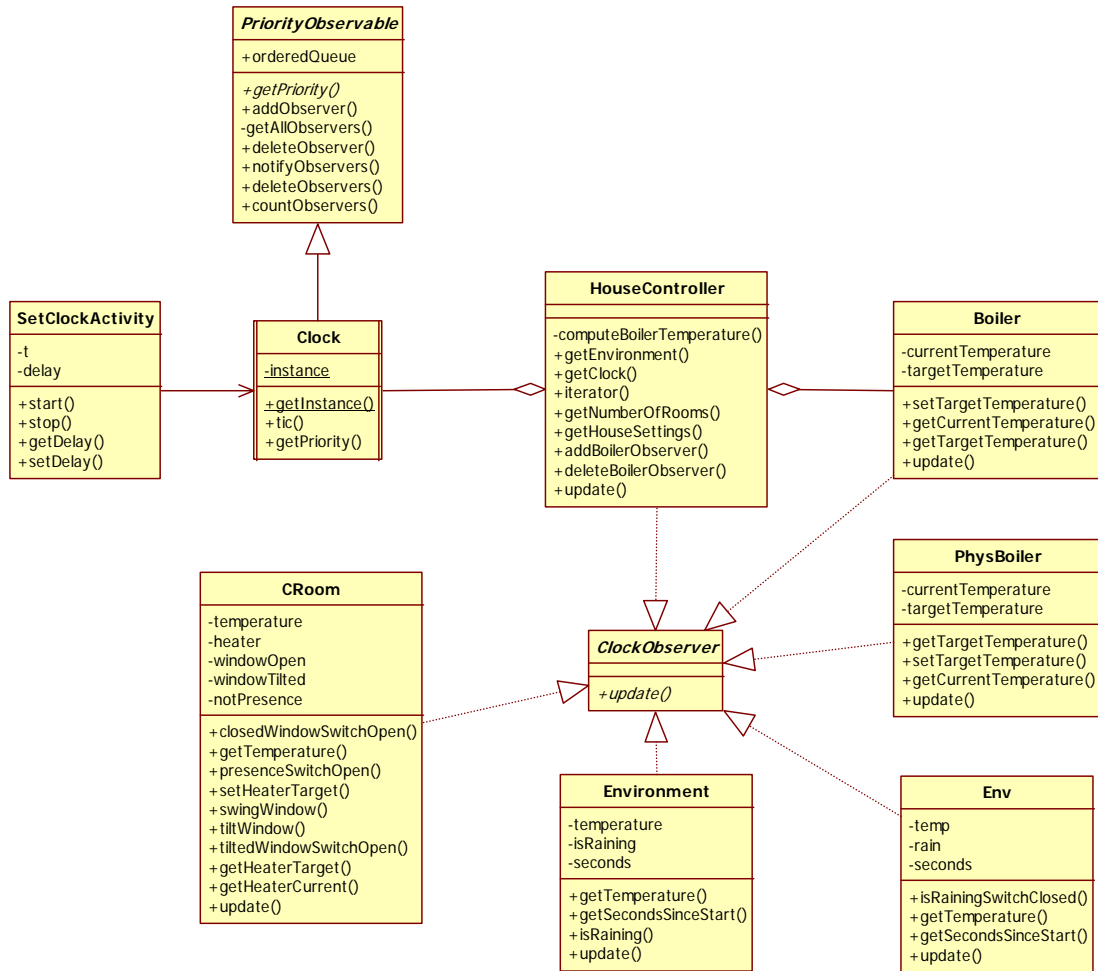
Class diagram



1.2.3.1.4 Event model

This diagram shows how the generation and propagation of events is handled.

Class diagram

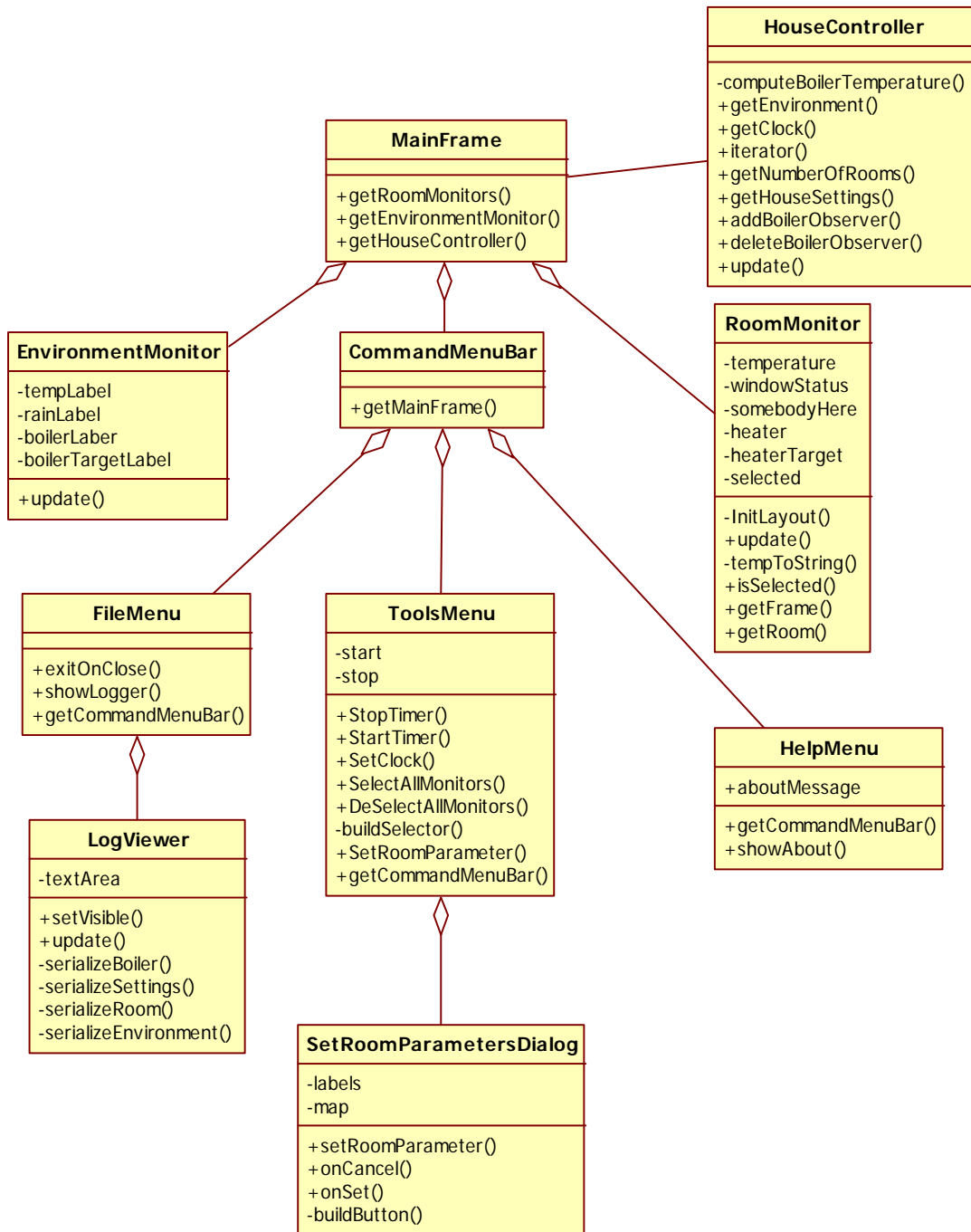


The *Clock* class generates an event every second; the *update()* method of every class that inherits from the *ClockObserver* is called as a result of that event.

1.2.3.1.5 User interface

This diagram shows a simplified view of the user interface subsystem.

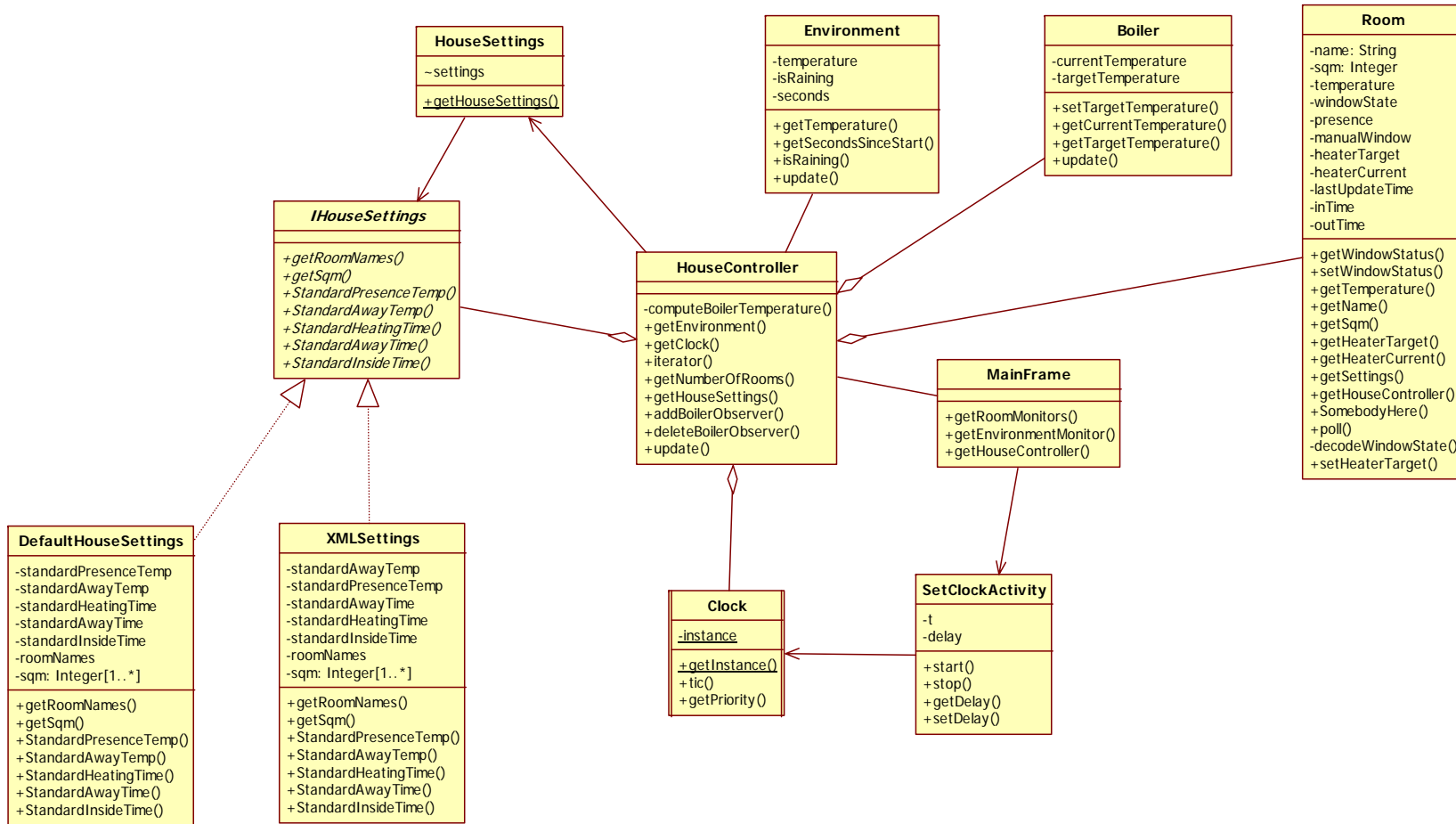
Class diagram



1.2.3.1.6 Initialization

This diagram shows the classes involved in the initialization of the entire system.

Class diagram



1.2.3.2 Data-dictionary

Into the data-dictionary are described all the classes of the UML-analysis together with their attributes, methods and relationships.

Object to be described	Attribute, Method, Relationship	Description
AwayManagementStrategy	Controls the room management strategy when the user is away or the window is manually opened	
	getTemperatureToFollow()	Returns the target temperature
	roomIsOK()	Manages the room when the temperature is correct
	roomIsTooCool()	Manages the room when the temperature is below the target
	roomIsTooWarm()	Manages the room when the temperature is above the target
Boiler	Models the logical behavior of the central boiler.	
	currentTemperature	The current measured water temperature
	targetTemperature	The desired water temperature
	getCurrentTemperature()	Returns the current measured water temperature
	setTargetTemperature()	Sets the desired water temperature
	getTargetTemperature()	Returns the desired water temperature
	update()	Computes the next logical state of the boiler
Clock	Measures wall-clock time and distributes events at fixed intervals. Only one instance of the class can exist at any given time.	
	instance	The current class instance
	getInstance()	Returns the current class instance, or generates one if it does not exist
	tic()	Executes one clock tick, notifying all observers
	getPriority()	Returns the priority of a given observer
ClockObserver	This interface describes all classes that must autonomously update their status	
	update()	Computes the next state of the object
CommandMenuBar	The application menu bar	
	getMainFrame()	Navigates to the containing MainFrame
CRoom	This class is a simplified model of the physical room. It also includes a simple thermal model.	
	temperature	The room temperature
	heater	The current heater temperature
	windowOpen	The open/closed state of the window
	windowTilted	The tilted/not tilted state of the window
	notPresence	The presence state of the user
	closedWindowSwitchOpen()	Returns the state of the open window switch
	getTemperature()	Returns the measured temperature
	presenceSwitchOpen()	Returns the state of the presence switch
	setHeaterTarget()	Sets the target heater temperature
	swingWindow()	Opens the window if it is closed, or vice versa. Does not care of the tilt state
	tiltWindow()	Tilts the window if it is closed, or vice versa. Does not care of the open state
	tiltedWindowSwitchOpen()	Returns the state of the tilted window switch
	update()	Computes the next state of the room

	getHeaterTarget()	Returns the target heater temperature
	getHeaterCurrent()	Returns the current heater temperature
DefaultHouseSettings	Represents the defaults the user can use for values.	
	standardPresenceTemp	Default temperature when the user is in the room
	standardAwayTemp	Default temperature when the room is empty
	standardHeatingTime	Default time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	standardAwayTime	Default time in minutes that should pass, after the user has left the room, before the room is managed as empty
	standardInsideTime	Default time in minutes that should pass, after the user has entered the room, before the room is managed as non empty
	roomNames	The names of all the rooms
	sqm	The areas of all the rooms in square meters
	getRoomNames()	Returns the names of all the rooms
	getSqm()	Returns the areas of all the rooms in square meters
	StandardPresenceTemp()	Returns the default temperature when the user is in the room
	StandardAwayTemp()	Returns the default temperature when the room is empty
	StandardHeatingTime()	Returns the default time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	StandardAwayTime()	Returns the default time in minutes that should pass, after the user has left the room, before the room is managed as empty
	StandardInsideTime()	Returns the default time in minutes that should pass, after the user has entered the room, before the room is managed as non empty
Env	This class provides a simplified model of the physical environment.	
	temp	The outside temperature
	rain	It is raining
	seconds	The time in seconds elapsed since application start
	getTemperature()	Returns the measured temperature
	isRainSwitchClosed()	Returns the state of the rain switch
	update()	Computes the next state of the environment
	getSecondsSinceStart()	Returns the time in seconds elapsed since application start
Environment	Represents the logical model of the environment.	
	temperature	The measured outside temperature
	isRaining	The logical indication of rain, as given by the switch
	seconds	The measured time elapsed since application start, in seconds
	getTemperature()	Returns the measured outside temperature
	isRainSwitchClosed()	Returns the state of the rain switch

	update()	Computes the next logical state of the environment
	getSecondsSinceStart()	Returns the measured time elapsed since application start, in seconds
EnvironmentMonitor	Observes the state of the Environment and of Boiler, and reports them to the user.	
	tempLabel	The screen label for the measured temperature
	rainLabel	The screen label for the rain switch state
	boilerLabel	The screen label for the current measured boiler temperature
	boilerTargetLabel	The screen label for the desired boiler temperature
	update()	Updates the environment display
FileMenu	Provides the user with the options in the File menu.	
	ExitOnClose	A private action listener that makes the application exit when the Close option is selected. Semantically similar to a method
	ShowLogger	A private action listener that makes the application log appear on the screen when the Show Logger option is selected. Semantically similar to a method
	getCommandMenuBar()	Navigates to the containing CommandMenuBar
GuiUtilities	Provides comon functions to the classes implementing the interface.	
	fitMenuItem()	Adds a new item to an existing menu
	fitNewLabel()	Adds a new component to an existing interface window
HcsGuiMenu	It is the abstract class that all menus inherit from.	
	getCommandMenuBar()	Navigates to the containing CommandMenuBar
HelpMenu	Provides the user with the options in the Help Menu.	
	aboutMessage	The message to be displayed when the About option is selected
	ShowAbout	A private action listener that makes the about message appear on the screen when the About option is selected. Semantically similar to a method
	getCommandMenuBar()	Navigates to the containing CommandMenuBar
HouseController	The main class in the heating control system, it integrates the logical model of the various parts of the house and performs the high-level activities.	
	computeBoilerTemperature()	Computes the desired water temperature in the boiler
	getEnvironment()	Navigates to the logical model of the environment
	getClock()	Navigates to the Clock
	iterator()	Returns an iterator to the contained Rooms
	getNumberOfRooms()	Returns the number of rooms
	getHouseSettings()	Navigates to the current global settings
	update()	Computes the next logical state of the system
	addBoilerObserver()	Adds an observer to the Boiler

	deleteBoilerObserver()	Removes an object from the list of Boiler observers
HouseSettings	A factory that returns the global system settings (i.e. not the settings of any single room).	
	settings	The name of the XML file containing the global system settings
	getHouseSettings()	Returns an instance of either XMLSettings or, if the settings file could not be read, of DefaultHouseSettings
IHouseSettings	The interface that defines the global system settings.	
	getRoomNames()	Returns the names of all the rooms
	getSqm()	Returns the areas of all the rooms in square meters
	StandardPresenceTemp()	Returns the desired temperature when the user is in the room
	StandardAwayTemp()	Returns the desired temperature when the room is empty
	StandardHeatingTime()	Returns the desired time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	StandardAwayTime()	Returns the desired time in minutes that should pass, after the user has left the room, before the room is managed as empty
	StandardInsideTime()	Returns the desired time in minutes that should pass, after the user has entered the room, before the room is managed as non empty
InvalidTimeException	This exception is raised when the user specifies a StandardHeatingTime less than 5 minutes longer than StandardInsideTime	
	requestedTime	The StandardHeatingTime requested by the user
	standardInsideTime	The current StandardInsideTime
	serialVersionUID	The unique numerical identifier of the class version
	getRequestedTime()	Returns the StandardHeatingTime requested by the user
	getStandardInsideTime()	Returns the current StandardInsideTime
IPhysicalBoiler	The interface that defines the behavior of a physical boiler.	
	getTargetTemperature()	Returns the desired water temperature
	setTargetTemperature()	Sets the desired water temperature
	getCurrentTemperature()	Returns the current measured water temperature
IPhysicalEnvironment	The interface that defines the system behavior of the physical environment.	
	isRainingSwitchClosed()	Returns the state of the rain switch
	getTemperature()	Returns the measured temperature
	getSecondsSinceStart()	Returns the time in seconds elapsed since application start
IPhysicalRoom	The interface that defines the behavior of a physical room.	
	presenceSwitchOpen()	Returns the state of the presence switch
	tiltedWindowSwitchOpen()	Returns the state of the tilted window switch
	closedWindowSwitchOpen()	Returns the state of the open window switch
	swingWindow()	Opens the window if it is closed, or vice versa. Does not care of the tilt state

	tiltWindow()	Tilts the window if it is closed, or vice versa. Does not care of the open state
	getTemperature()	Returns the measured temperature
	setHeaterTarget()	Sets the target heater temperature
	getHeaterTarget()	Returns the target heater temperature
	getHeaterCurrent()	Returns the current heater temperature
LogViewer	Provides the user a view of the system log.	
	textArea	The text area used to hold log information
	setVisible()	Makes the log information visible on screen
	update()	Updates the log information
	serializeBoiler()	Transforms current Boiler information in string format
	serializeSettings()	Transforms current RoomSettings information in string format
	serializeRoom()	Transforms current Room information in string format
	serializeEnvironment()	Transforms current Environment information in string format
Main	The starting point for code execution.	
	main()	
	getMainFrame()	Navigates to the application main interface component
MainFrame	This class integrates the user interface components.	
	getRoomMonitors()	Navigates to the list of enclosed RoomMonitors
	getEnvironmentMonitor()	Navigates to the enclosed EnvironmentMonitor
	getHouseController()	Navigates to the HouseController
PhysBoiler	This class is a simplified model of a physical boiler.	
	currentTemperature	The current water temperature
	targetTemperature	The desired water temperature
	getCurrentTemperature()	Returns the desired water temperature
	setTargetTemperature()	Sets the desired water temperature
	update()	Computes the next boiler state
	getTargetTemperature()	Returns the current measured water temperature
PhysicalFactory	Factory class that returns instances of physical models.	
	getPhysicalRoom()	Returns an existing instance of a CRoom
	getPhysicalEnvironment()	Returns the existing instance of Env
	getPhysicalBoiler()	Returns the existing instance of PhysBoiler
PresenceManagementStrategy	Controls the room management strategy when the user is inside.	
	applyStrategy()	Manages the room supposing that the user is inside
	getTemperatureToFollow()	Returns the target temperature
	roomIsOK()	Manages the room when the temperature is correct
	roomIsTooCool()	Manages the room when the temperature is below the target
	roomIsTooWarm()	Manages the room when the temperature is above the target
PriorityObservable	Abstract class that defines the behavior of a class that is observable by other classes.	
	orderedQueue	The queue of all observers, ordered by their priority

	getPriority()	Returns the priority of a given observer
	addObserver()	Adds an observer to the ordered queue
	getAllObservers()	Returns a list of all observers, in priority order
	deleteObserver()	Removes an observer from the ordered queue
	notifyObservers()	Calls the update() method of all the observers in priority order
	deleteObservers()	Removes all observers from the queue
	countObservers()	Returns the current number of observers
Room	This class defines the logical model of a room. It also handles the relative switches and actuators.	
	name	The room name
	sqm	The room area in square meters
	temperature	The measured room temperature
	windowState	The last remembered window state
	presence	The last remembered user presence state
	manualWindow	The window management state. True if the window should not be controlled by the system, false otherwise
	heaterTarget	The desired heater temperature
	heaterCurrent	The current measured heater temperature
	lastUpdateTime	The last time the logical state of the room has been recomputed
	inTime	Elapsed time since the user last entered the room, if he is still in
	outTime	Elapsed time since the user last left the room, if he is still out
	decodeWindowState()	Returns a composite window state information by reading the switch states
	getWindowStatus()	Returns the last remembered window state
	setWindowStatus()	Operates the window
	getTemperature()	Returns the measured temperature
	poll()	Computes the next logical status of the room
	getName()	Returns the room name
	getSqm()	Returns the room area, in square meters
	SomebodyHere()	Returns the last remembered user presence state
	getHeaterTarget()	Returns the target heater temperature
	getHeaterCurrent()	Returns the current heater temperature
	setHeaterTarget()	Sets the target heater temperature
	getSettings()	Navigates to the RoomSettings for this Room
	getHouseController()	Navigates to the enclosing HouseController
RoomManagementStrategy	An abstract class that defines a generic management strategy	
	applyStrategy()	Manages the room in the general case
	getTemperatureToFollow()	Returns the target temperature
	roomIsOK()	Manages the room when the temperature is correct
	roomIsTooCool()	Manages the room when the temperature is below the target
	roomIsTooWarm()	Manages the room when the temperature is above the target
RoomMonitor	Observes the state of a Room, and reports it to the user.	

	temperature	The temperature to display
	windowStatus	The window status to display
	somebodyHere	The user presence status
	heater	The current heater temperature
	heaterTarget	The desired heater temperature
	selected	The selection state of the RoomMonitor. True if the user has selected the room
	initLayout()	Initializes the display of the room status
	update()	Updates the displayed room information
	tempToString()	Converts the heater temperature to a string
	isSelected()	Returns true if the RoomMonitor has been selected by the user
	getFrame()	Navigates to the enclosing MainFrame
	getRoom()	Navigates to the relative Room
RoomSettings	This class contains the settings for a Room.	
	StandardPresenceTemp	Desired temperature when the user is in the room
	StandardAwayTemp	Desired temperature when the room is empty
	StandardHeatingTime	Desired time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	StandardAwayTime	Desired time in minutes that should pass, after the user has left the room, before the room is managed as empty
	StandardInsideTime	Desired time in minutes that should pass, after the user has entered the room, before the room is managed as non empty
	revertToDefault()	Resets all settings to their default values
	getStandardPresenceTemp()	Returns the desired temperature when the user is in the room
	setStandardPresenceTemp()	Sets the desired temperature when the user is in the room
	getStandardAwayTemp()	Returns the desired temperature when the room is empty
	setStandardAwayTemp()	Sets the desired temperature when the room is empty
	getStandardHeatingTime()	Returns the desired time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	setStandardHeatingTime()	Sets the desired time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	getStandardAwayTime()	Returns the desired time in minutes that should pass, after the user has left the room, before the room is managed as empty
	setStandardAwayTime()	Sets the desired time in minutes that should pass, after the user has left the room, before the room is managed as empty
	getStandardInsideTime()	Returns the desired time in minutes that should pass, after the user has entered the room, before the room is managed as non empty

	setStandardInsideTime()	Sets the desired time in minutes that should pass, after the user has entered the room, before the room is managed as non empty
	toString()	Converts the setting values to a HTML string
	getRoom()	Navigates to the related Room
SetClockActivity	This class is used to set up the Clock, start it and stop it.	
	t	The actual timer
	delay	The delay between one timer tick and the next
	tic	A private action listener that calls the tic() method of Clock at every tick of the timer. Semantically similar to a method
	start()	Starts a new timer. If a running timer already exists, stops it first
	stop()	Stops the running timer
	getDelay() setDelay()	Returns the programmed timer delay Programs the timer delay
SetRoomParametersActivity	This class is used to set room parameters and get the defaults.	
	setRoomParameter()	Sets a single room parameter
	getDefaultParameter()	Returns the default for a single room parameter
SetRoomParametersDialog	This class provides the user interface to set room parameters.	
	labels	The names of the parameters for a given room
	map	The correspondence map between parameters and display components
	setRoomParameter()	Sets a single room parameter
	onCancel	A private action listener that makes the parameter dialog disappear without consequences when the Cancel option is selected. Semantically similar to a method
	onSet	A private action listener that makes the parameter dialog disappear and sets all the options to new values when the Set option is selected. Semantically similar to a method
StateObservable	This abstract class defines the interface to a class whose state is observable.	
	addObserver()	Adds an observer to the object
	deleteObserver()	Removes an object from the set of observers
StateObserver	This interface defines the behavior of a class that observes the state of another class.	
	update()	Computes the next state of the object
ToolsMenu	Provides the user with the options in the Tools Menu.	
	start	The Start option of the menu
	stop	The Stop option of the menu
	StopTimer	A private action listener that makes the application stop its timer when the Stop option is selected. Semantically similar to a method

	StartTimer	A private action listener that makes the application start a new timer when the Start option is selected. Semantically similar to a method
	SetClock	A private action listener that makes the application display a dialog box when the Set Clock option is selected and then program a new timer delay when the Ok option of the box is selected. Semantically similar to a method
	SelectAllMonitors	A private action listener that selects all rooms when the Select All option is selected. Semantically similar to a method
	DeSelectAllMonitors	A private action listener that deselects all rooms when the Deselect All option is selected. Semantically similar to a method
	buildSelector()	Returns an action listener that handles single selections
	SetRoomParameter	A private action listener that makes the application exit when the Close option is selected. Semantically similar to a method
	getCommandMenuBar()	Navigates to the enclosing CommandMenuBar
XMLSettings	Represents the setting values derived from the XML file.	
	standardPresenceTemp	Desired temperature when the user is in the room
	standardAwayTemp	Desired temperature when the room is empty
	standardHeatingTime	Desired time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	standardAwayTime	Desired time in minutes that should pass, after the user has left the room, before the room is managed as empty
	standardInsideTime	Desired time in minutes that should pass, after the user has entered the room, before the room is managed as non empty
	roomNames	The names of all the rooms
	sqm	The areas of all the rooms in square meters
	getRoomNames()	Returns the names of all the rooms
	getSqm()	Returns the areas of all the rooms in square meters
	StandardPresenceTemp()	Returns the desired temperature when the user is in the room
	StandardAwayTemp()	Returns the desired temperature when the room is empty
	StandardHeatingTime()	Returns the desired time in minutes, after an user has entered the room, within which the room should reach the StandardPresenceTemp
	StandardAwayTime()	Returns the desired time in minutes that should pass, after the user has left the room, before the room is managed as empty

	StandardInsideTime()	Returns the desired time in minutes that should pass, after the user has entered the room, before the room is managed as non empty
--	----------------------	--

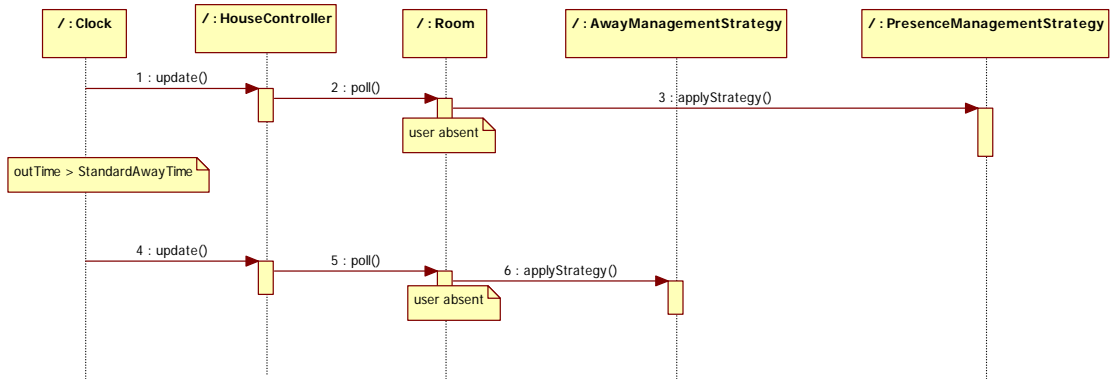
Table 5

1.2.3.3 UML Sequence Diagrams

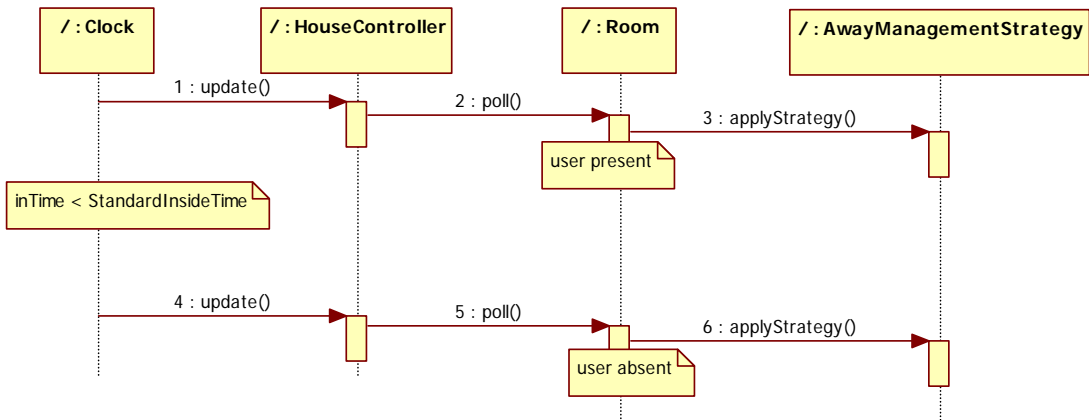
Use Cases of this system are detailed below with sequence diagrams.

1.2.3.3.1 Sequence diagram for Use Case Presence

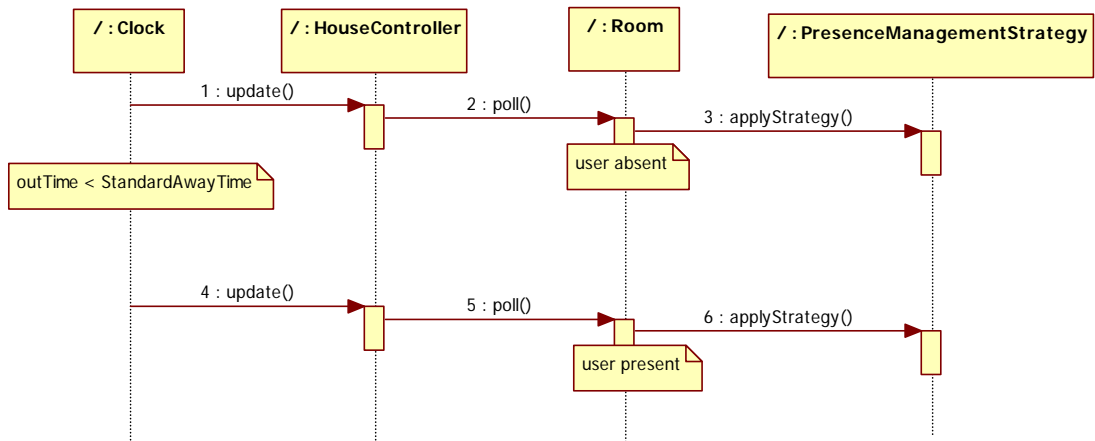
Sequence diagram for scenario 3:



Sequence diagram for scenario 2:

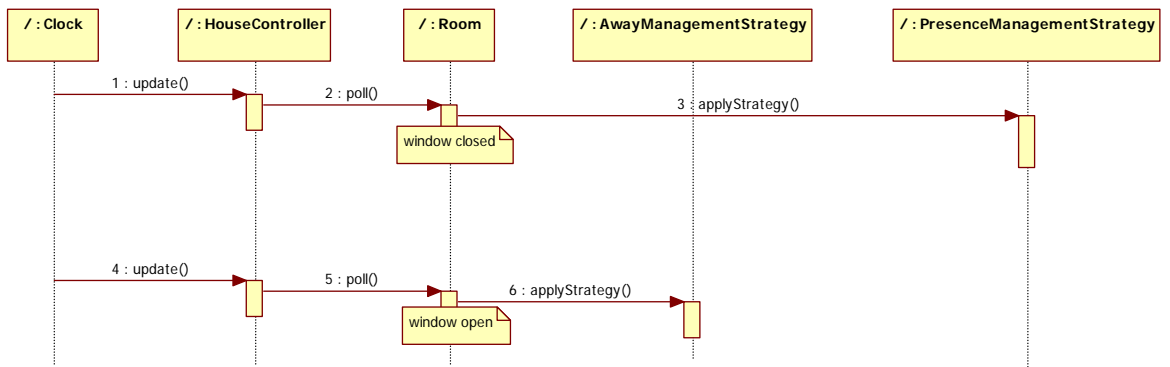


Sequence diagram for scenario 4:



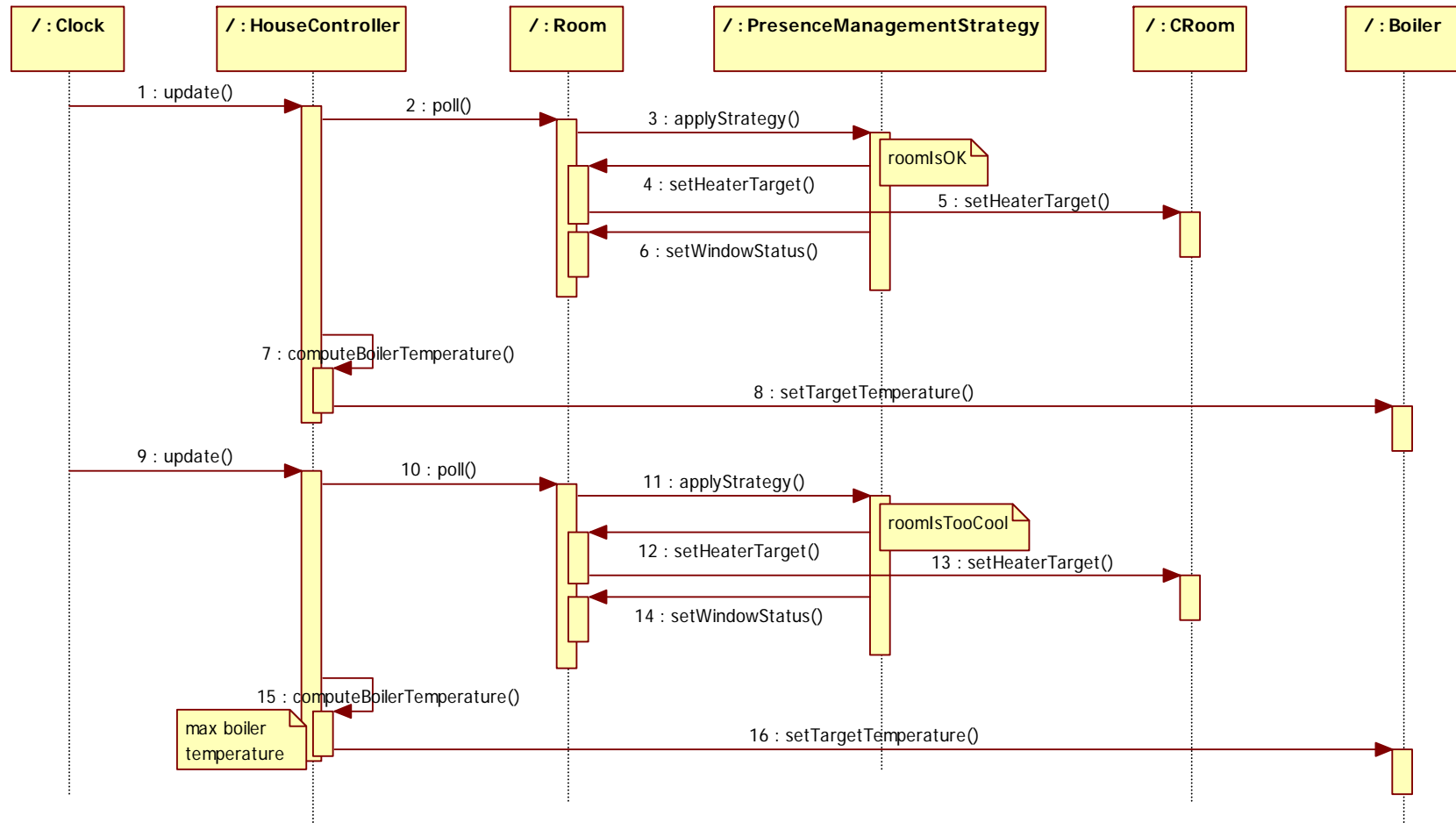
1.2.3.3.2 Sequence diagrams for Use Case WindowPosition

Sequence diagram for scenario 5:

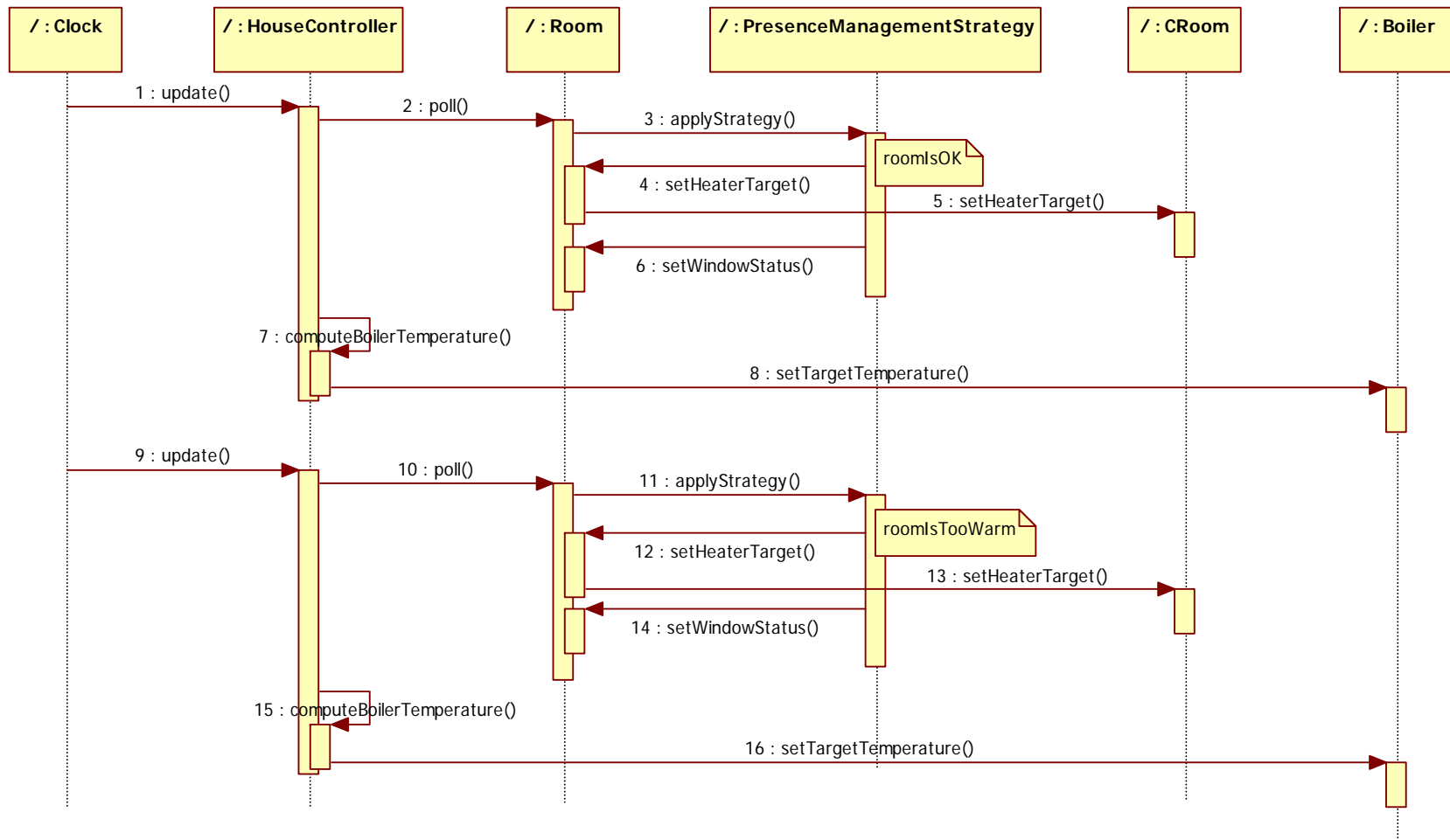


1.2.3.3.3 Sequence Diagram for Use Case HeatingControl

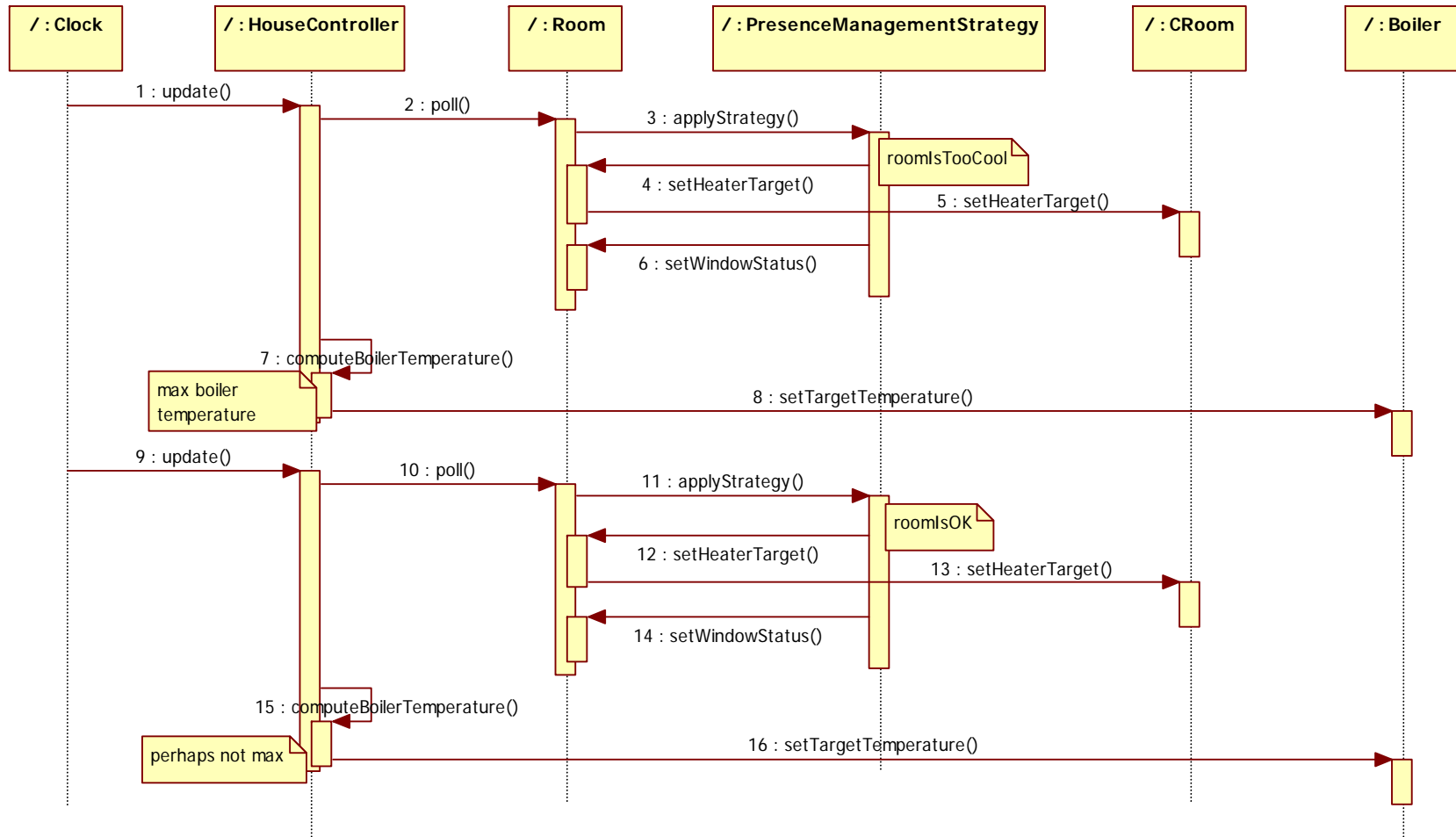
Sequence diagram for scenario 10:



Sequence diagram for scenario 11:

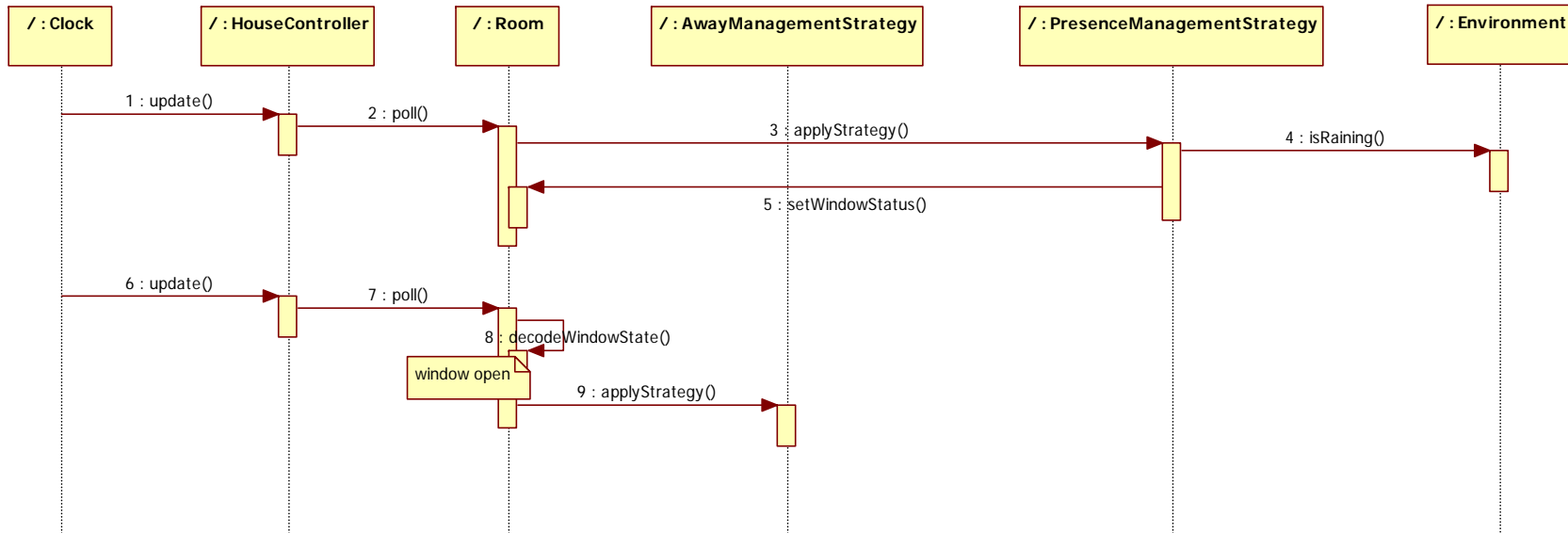


Sequence diagram for scenario 9:



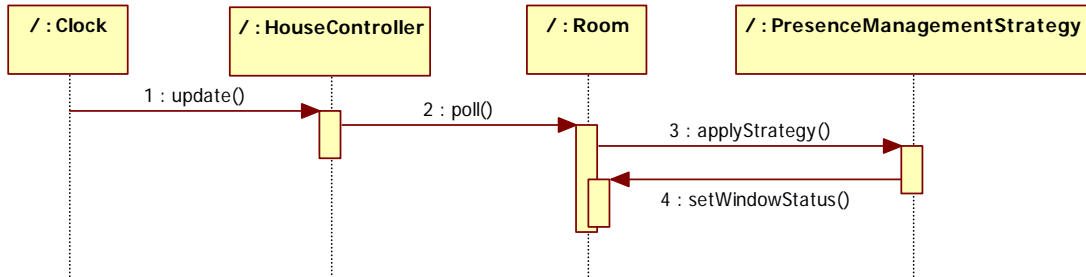
1.2.3.3.4 Sequence Diagrams for Use Case RainMonitoring

Sequence diagram for scenario 6:



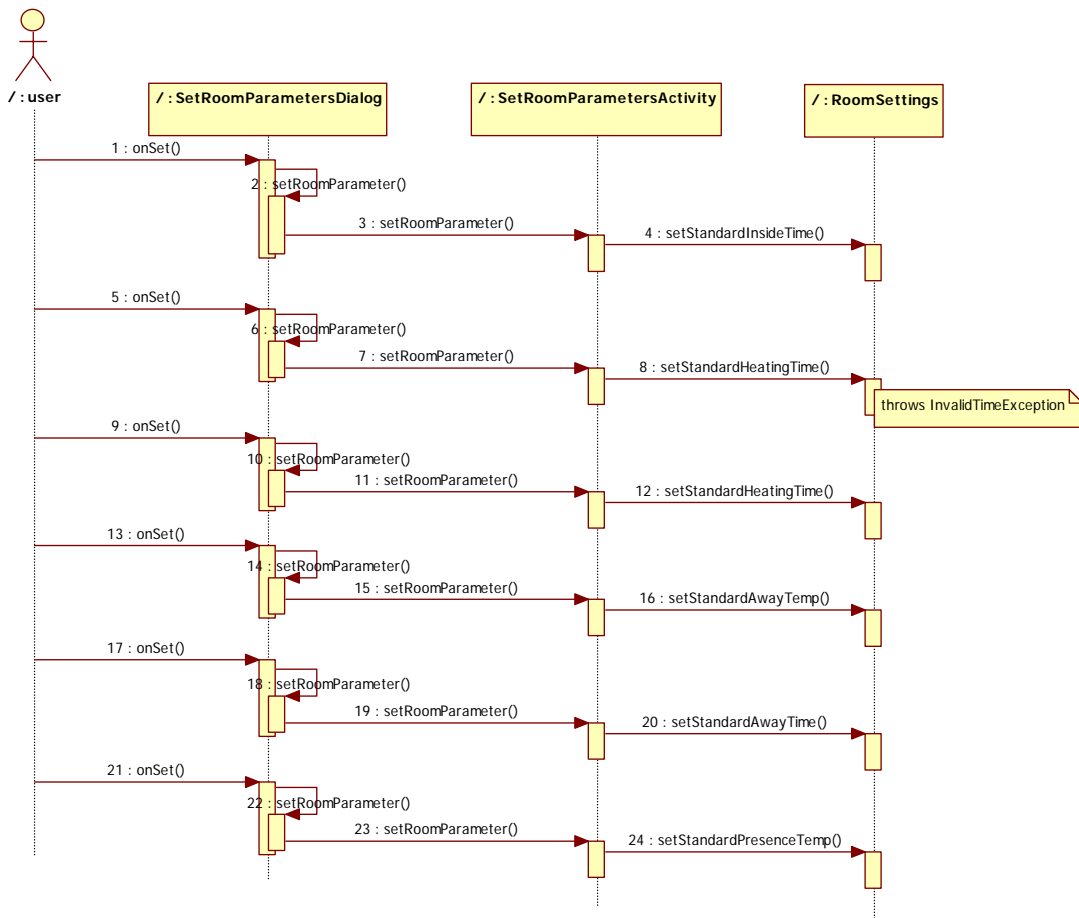
1.2.3.3.5 Sequence diagram for Use Case TemperatureTooHigh

Sequence diagram for scenario 7:

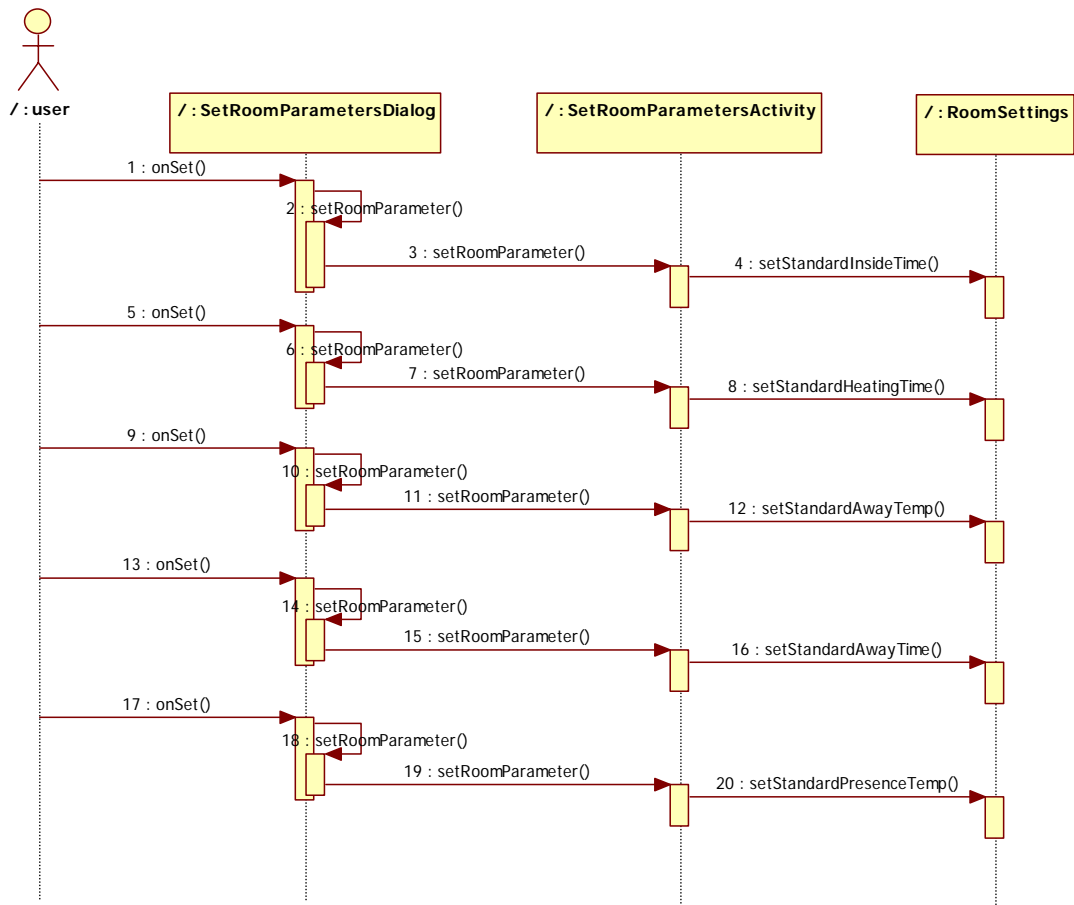


1.2.3.3.6 Sequence diagrams for Use Case ValuesSetting

Sequence diagram for scenario 1:



Sequence diagram for scenario 8:

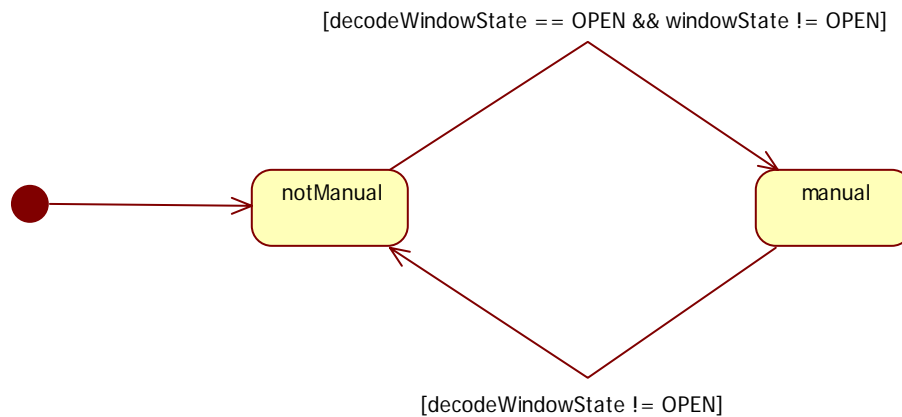


1.2.3.4 UML state diagrams

State diagrams depict dynamic behavior of classes. The behavior of each class will be depicted below with one or more specific state diagrams. Methods calls are unaccounted in the state diagrams, as they serve only for setting or getting values.

1.2.3.4.1 Manual Window (class Room)

This state diagram refers to the manual or automatic operation of the window. In manual state the system does not attempt to operate the window, whereas in notManual state the system may open, close or tilt the window as needed.

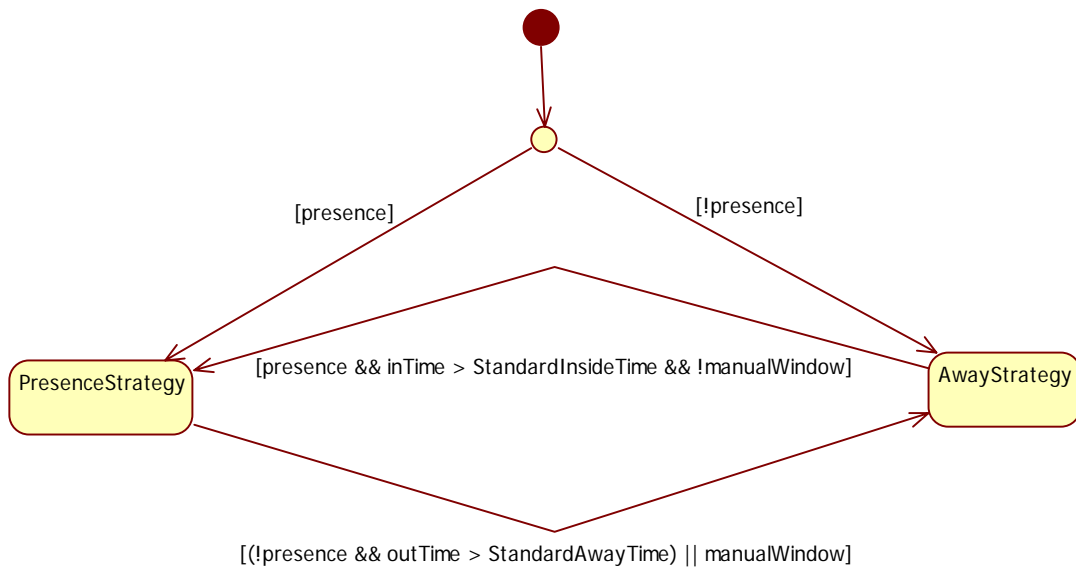


Do-activity for State: notManual
windowState := decodeWindowState()
presence := CRoom.presenceSwitchOpen()

Do-activity for State: manual
windowState := decodeWindowState()
presence := CRoom.presenceSwitchOpen()

1.2.3.4.2 Management Strategy (class Room)

This diagram depicts the choice of the temperature management strategy to use.

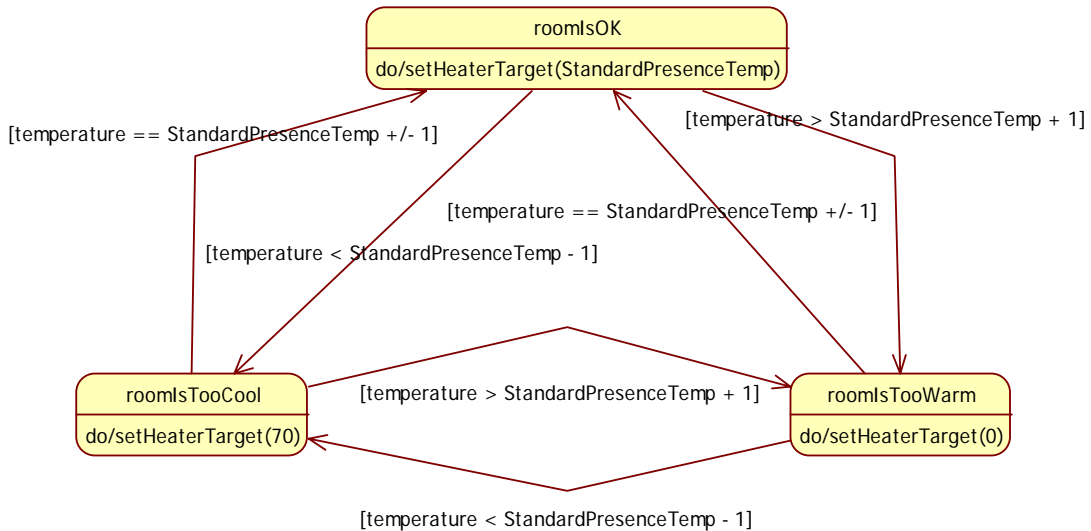


Do-activity for State: PresenceStrategy
`presence := Croom.presenceSwitchOpen()`

Do-activity for State: AwayStrategy
`presence := Croom.presenceSwitchOpen()`

1.2.3.4.3 Management strategy (class PresenceManagementStrategy)

Here the management strategy used when the user is present is depicted.



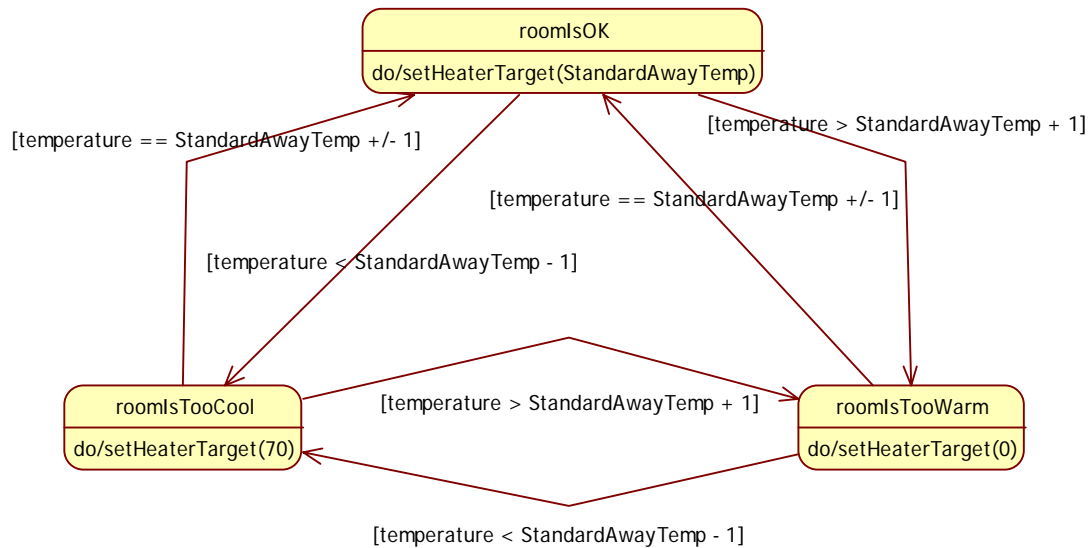
Do-activity for State: roomIsTooCool
`temperature := Room.getTemperature()`

Do-activity for State: roomIsOK
`temperature := Room.getTemperature()`

Do-activity for State: roomIsTooWarm
temperature := Room.getTemperature()

1.2.3.4.4 Management strategy (class AwayManagementStrategy)

Here is described the management strategy used when the user is not present or has manually opened the window.



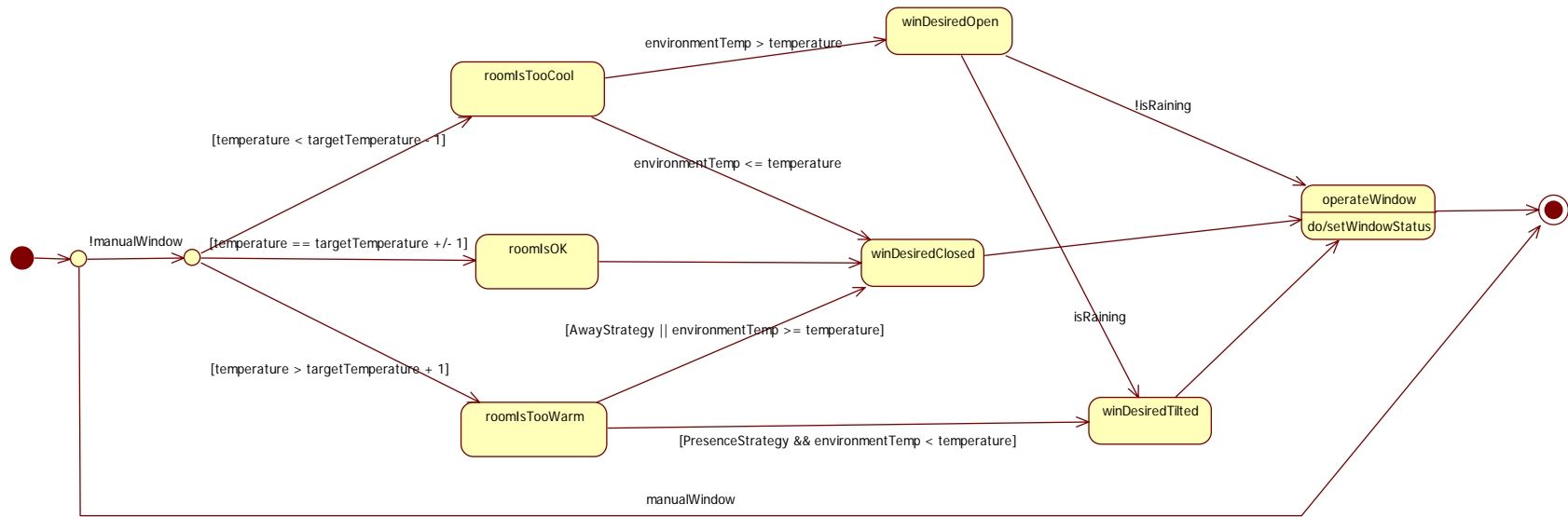
Do-activity for State: roomIsTooCool
temperature := Room.getTemperature()

Do-activity for State: roomIsOK
temperature := Room.getTemperature()

Do-activity for State: roomIsTooWarm
temperature := Room.getTemperature()

1.2.3.4.5 Window operation (class Room)

This diagram details the flow of computation that leads to system use of the window actuators.



Do-activity for State: roomIsTooCool
 temperature := Room.getTemperature()
 environmentTemp := Environment.getTemperature()

Do-activity for State: roomIsOK
 temperature := Room.getTemperature()
 environmentTemp := Environment.getTemperature()

Do-activity for State: roomIsTooWarm
 temperature := Room.getTemperature()
 environmentTemp := Environment.getTemperature()

Do-activity for State: winDesiredOpen
 isRaining := Environment.isRainSwitchClosed()

Do-activity for State: winDesiredClosed
 none

Do-activity for State: winDesiredTilted
 none

Do-activity for State: operateWindow
 setWindowStatus()

1.2.4 Traceability matrix: user requirements <-> class diagrams

The traceability matrix reports which classes user requirements have been mapped to. Inside it you'll find the classes, but not instances.

	AwayManagementStrategy	Boiler	CRoom	DefaultHouseSettings	Env	Environment	HouseController	InvalidTimeException	PhysBoiler	PresenceManagementStrategy	Room	RoomManagementStrategy	RoomSettings	SetRoomParametersActivity	SetRoomParametersDialog	XMLSettings
Temp-UR-F1											X		X	X	X	X
Temp-UR-F2											X		X	X	X	X
Temp-UR-F3											X		X	X	X	X
Temp-UR-F4											X		X	X	X	X
Temp-UR-F5											X		X	X	X	X
Temp-UR-F6		X	X		X	X	X		X	X	X	X				
Temp-UR-F7	X	X	X		X	X	X		X		X	X				
Temp-UR-F8		X	X		X	X	X		X	X	X	X				
Temp-UR-F9		X	X		X	X	X		X	X	X	X				
Temp-UR-F10	X	X	X		X	X	X		X		X	X				
Temp-UR-F11								X							X	
Temp-UR-F12				X									X	X		
Temp-UR-F13	X	X	X		X	X	X		X		X	X				
Temp-UR-F14	X		X		X	X	X			X	X	X				
Temp-UR-F15			X		X	X	X			X	X	X				
Temp-UR-F16	X									X						
Temp-UR-F17			X	X			X				X					X
Temp-UR-F18		X					X		X							
UR-Inv 1	X	X	X		X	X	X		X		X	X				
UR-Inv 2		X	X		X	X	X		X	X	X	X				