Software Modeling and Architecture

(CS 5373-001)

<u>Smart Home System Project (Team 6)</u>

Professor: Dr. Michael Shin

Students:
Cristiano E. Caon
Pavan Sairam Kollimarla
Ramya Sri Thota
Tyler Johnson
Yamini Pothuraju

Note: You can click on the diagram to get the clear version of it.

Term Project: Phase I

You are required to develop an Analysis model of the Smart Home System (SHS), which includes:
a) Develop a System Context Class Model depicted on a class diagram showing how the system interfaces to the external environment. (5 pts)

b) Develop a static model that describes entity classes, attributes, and relationships between entity classes. (5 pts)

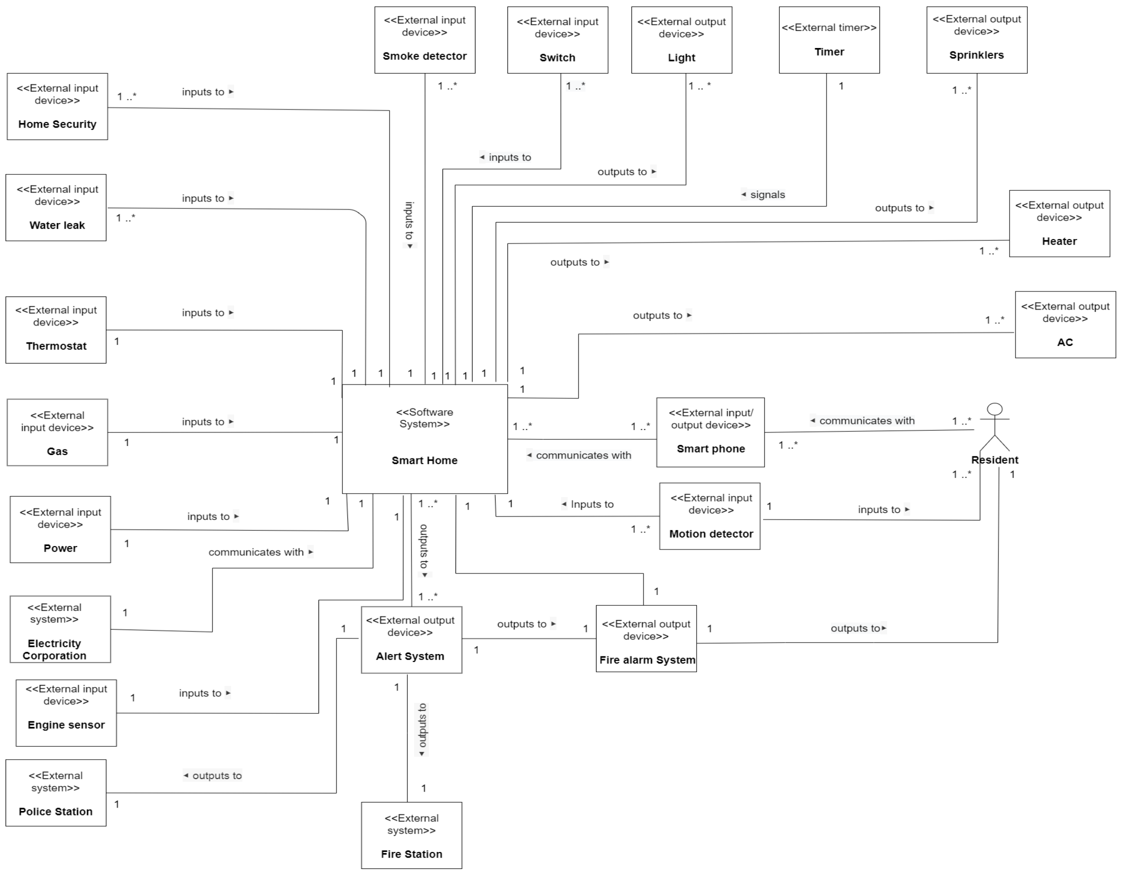
c) Develop sequence (or communication) diagrams (one for each use case), depicting the sequence of interactions among the objects participating in each use case. Define the object structuring criteria used. (12 pts)

d) Develop the statechart for the control object. The statechart needs to show the alternative paths on the sequence diagrams (or communication diagrams). Ensure that the statechart is consistent with the sequence (or communication) diagram for use cases. (8 pts)

Note: You can click on the diagram to get the clear version of it.

Context Class Model

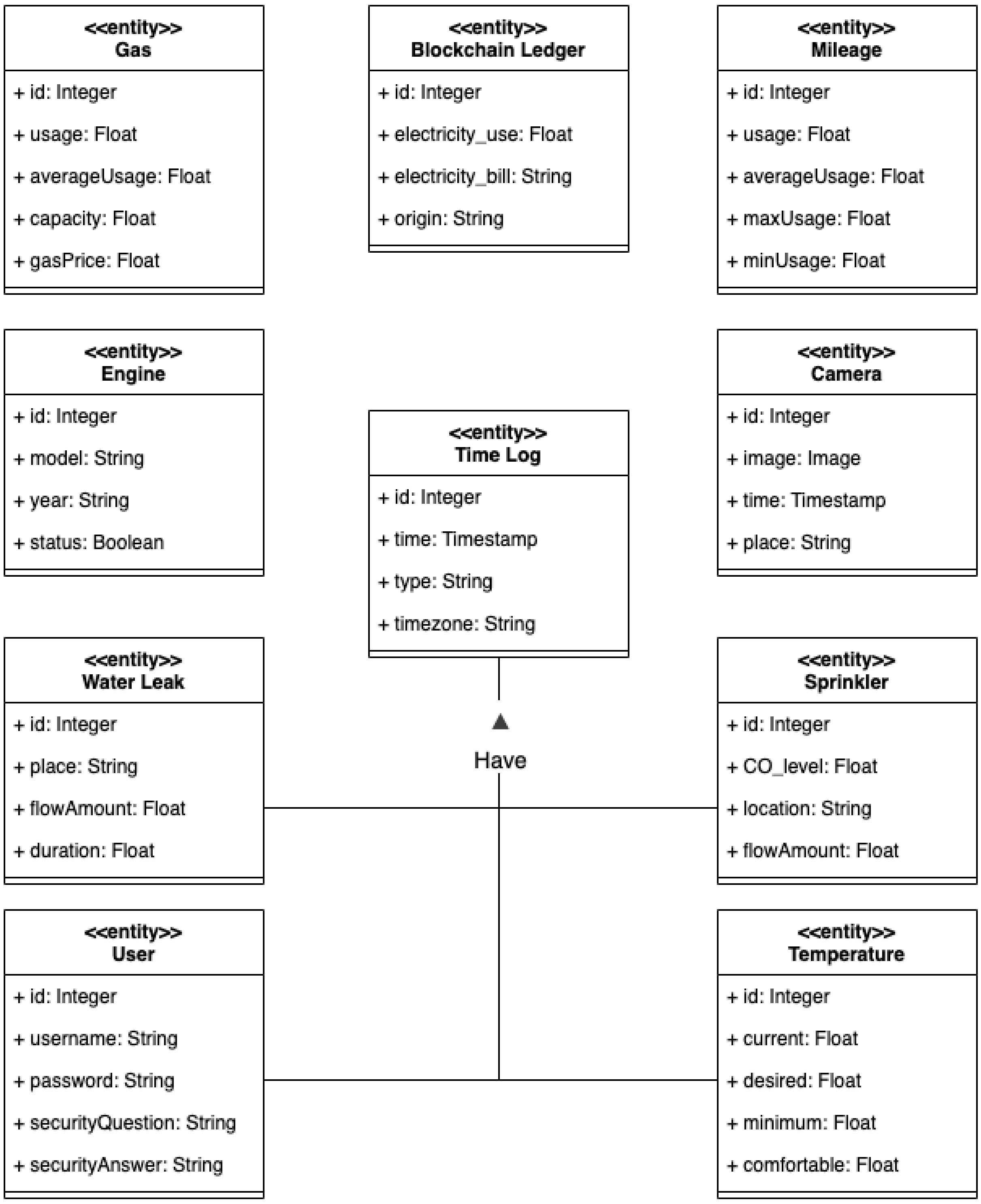
In the context class diagram there are eight classes categorized as the **External Input devices:** Smoke detector, Switch, Motion detector, Power, Engine sensor,Thermostat, Water Leak and Home security; six classes categorized as the **External output devices:** Light,Sprinklers, Heater, AC, Fire Alarm, Alert System; one classes categorized as the **External Input/Output device:** Smart phone; two class categorized as the **External Systems:** Fire Station, Police station; and there is one **External Timer.**



<<External input device>> Smoke detector
<<External input device>> Switch
<<External output device>> Light
<<External timer>> Timer
<<External output device>> Sprinklers
<<External input device>> Home Security
<<External input device>> Water leak
<<External output device>> Heater
<<External input device>> Thermostat
<<External output device>> AC
<<External input device>> Gas
<<Software System>> Smart Home
<<External input/ output device>> Smart phone
Resident
<<External input device>> Power
<<External input device>> Motion detector
<<External system>> Electricity Corporation
<<External output device>> Alert System
<<External output device>> Fire alarm System
<<External input device>> Engine sensor
<<External system>> Police Station
<<External system>> Fire Station
inputs to
outputs to
signals
communicates with
Inputs to
outputs to
1 ..*
1

Note: You can click on the diagram to get the clear version of it.

Static Model



Note: You can click on the diagram to get the clear version of it.

Communication Diagrams

CONTROL TEMPERATURE

Summary: The system controls the heater or air conditioning (AC) on or off to increase or decrease the home temperature to meet the desired temperature.

Precondition: The desired temperature for the home has been specified. The resident switched on Air Conditioning (AC)/heater thermostat, and the AC and heater are off.

Description and Assumption:

Main sequence:

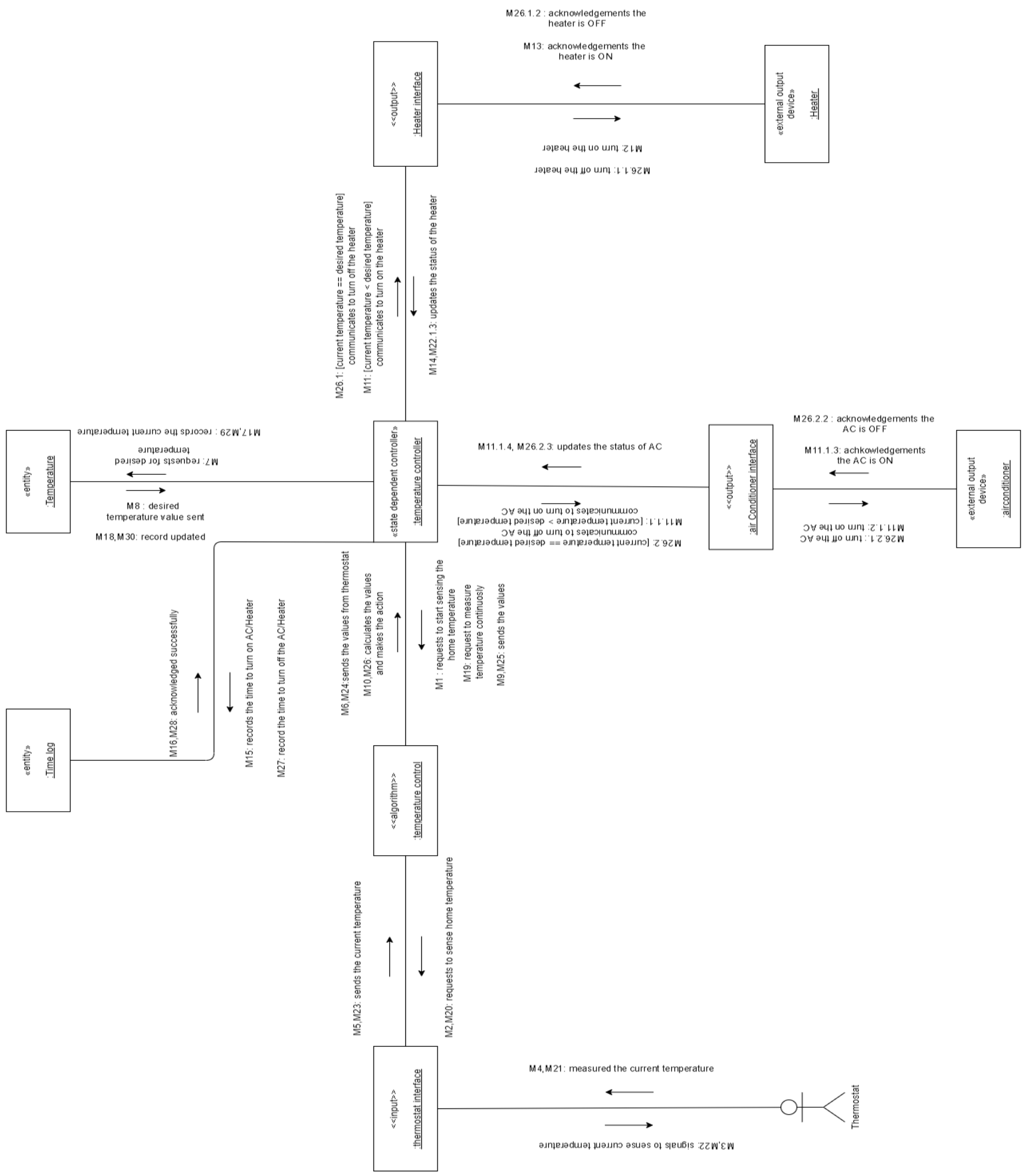
In this use case the system acts as a state-dependent controller. It requests the temperature control(<<algorithm>>) object to sense the home temperature. The algorithm object requests a thermostat interface to sense home temperature and it gives signals to the thermostat to sense the current temperature. The thermostat measures the current home temperature and updates it to the thermostat interface. The interface object sends the current temperature values to the algorithm object. The algorithm object updates the current temperature values to the system. The system requests Temperature(<<entity>>) for the desired temperature. The entity sends the values of the desired temperature to the system. The system sends both the current and desired temperature values to the algorithm object to calculate. It calculates the temperatures and updates them to the system to perform actions. If current temperature < desired temperature the system communicates with the heater interface to turn on the heater(<<external output device>>). The heater sends an acknowledgment to the system. If current temperature > desired temperature the system communicates with the air conditioner interface to turn on AC. The AC sends an acknowledgment to the interface and updates the status of the AC to the system. The system records the time to turn on Heater/AC. The system records the current temperature from the Temperature(<<entity>>) and requests a temperature control object to measure the temperature continuously. The algorithm object sends temperature values from the thermostat to the system. If the current temperature == desired temperature the system communicates with the interface to turn off the heater. Simultaneously, if current temperature == desired temperature the system communicates with the AC interface to turn off the AC. The heater interface updates the status of the heater and the AC interface updates the status of the AC to the system.

Assumptions:

In the Control Temperature Use Case we assumed that, after the temperature controller requests for the temperature values, the temperature control algorithm will perform the action of comparing the values of desired and current temperature and decide whether to turn AC on/off (or) to turn heater on/off. In the temperature entity all the values of temperature will be stored and in the time log entity the time of action will be stored in it. The temperature control algorithm will take care of the continuous loop of measuring the room temperature and performing the respective actions.

Note: You can click on the diagram to get the clear version of it.

Main sequence



<<output>> :Heater interface
«external output device» :Heater
M26.1.2 : acknowledgements the heater is OFF
M13: acknowledgements the heater is ON
M12: turn on the heater
M26.1.1: turn off the heater
M26.1: [current temperature == desired temperature] communicates to turn off the heater
M11: [current temperature < desired temperature] communicates to turn on the heater
M14,M22.1.3: updates the status of the heater
«entity» :Temperature
M17,M29 : records the current temperature
M7: requests for desired temperature
M8 : desired temperature value sent
M18,M30: record updated
«state dependent controller» :temperature controller
M11.1.4, M26.2.3: updates the status of AC
M11.1.1: [current temperature > desired temperature] communicates to turn on the AC
M26.2: [current temperature == desired temperature] communicates to turn off the AC
<<output>> :air Conditioner interface
M26.2.2 : acknowledgements the AC is OFF
M11.1.3: achkowledgements the AC is ON
M11.1.2: turn on the AC
M26.2.1.: turn off the AC
«external output device» :airconditioner
«entity» :Time log
M16,M28: acknowledged successfully
M15: records the time to turn on AC/Heater
M27: record the time to turn off the AC/Heater
M6,M24:sends the values from thermostat
M10,M26: calculates the values and makes the action
M1 : requests to start sensing the home temperature
M19: request to measure temperature continuosly
M9,M25: sends the values
<<algorithm>> :temperature control
M5,M23: sends the current temperature
M2,M20: requests to sense home temperature
<<input>> :thermostat interface
M4,M21: measured the current temperature
M3,M22: signals to sense current temperature
Thermostat

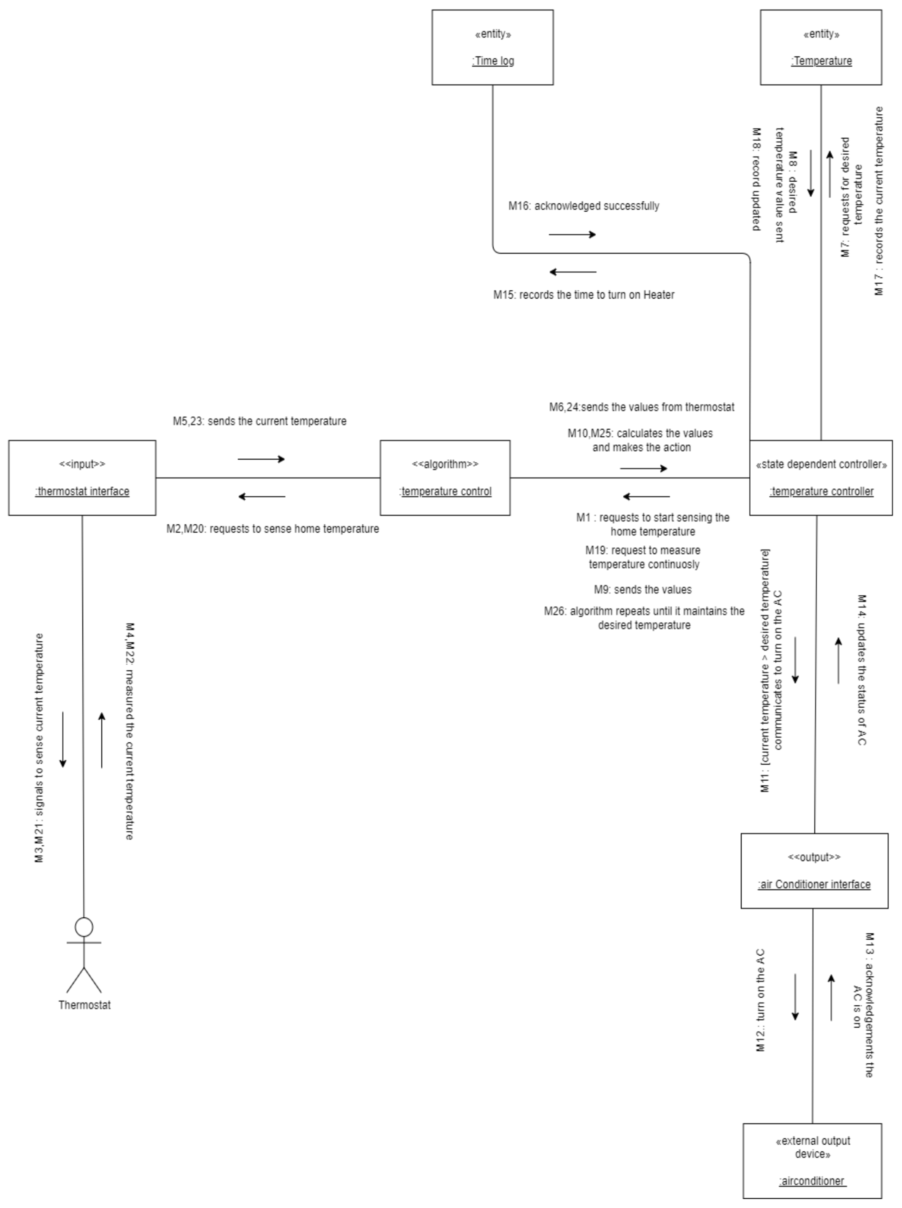
Note: You can click on the diagram to get the clear version of it.

Alternative sequence:

The system requests the temperature control(<<algorithm>>) object to sense the home temperature. The algorithm object requests a thermostat interface to sense home temperature and it gives signals to the thermostat to sense the current temperature. The thermostat measures the current home temperature and updates it to the thermostat interface, The interface object sends the current temperature values to the algorithm object. The algorithm object updates the current temperature values to the system. The system requests Temperature(<<entity>>) for the desired temperature. The entity sends the values of the desired temperature to the system. The system sends both the current and desired temperature values to calculate. It calculates the temperatures and updates them to the system to perform actions. If current temperature > desired temperature the system communicates with the air conditioner interface to turn on AC. The AC sends an acknowledgment to the interface and updates the status of the AC to the system. The system records the time to turn on AC. The system records the current temperature from the Temperature(<<entity>>) and requests a temperature control object to measure the temperature continuously. The algorithm object sends temperature values from the thermostat to the system.

Note: You can click on the diagram to get the clear version of it.

Alternative sequence


«entity»
:Time log
«entity»
:Temperature
M8 : desired temperature value sent
M18 : record updated
M7: requests for desired temperature
M17 : records the current temperature
M16: acknowledged successfully
M15: records the time to turn on Heater
M6,24:sends the values from thermostat
M10,M25: calculates the values and makes the action
M5,23: sends the current temperature
<<input>>
:thermostat interface
<<algorithm>>
:temperature control
«state dependent controller»
:temperature controller
M2,M20: requests to sense home temperature
M1 : requests to start sensing the home temperature
M19: request to measure temperature continuosly
M9: sends the values
M26: algorithm repeats until it maintains the desired temperature
M11 : [current temperature > desired temperature] communicates to turn on the AC
M14: updates the status of AC
M3,M21: signals to sense current temperature
M4,M22: measured the current temperature
<<output>>
:air Conditioner interface
Thermostat
M12:: turn on the AC
M13 : acknowledgements the AC is on
«external output device»
:airconditioner

Postcondition: The system has maintained the desired temperature for the home.

Note: You can click on the diagram to get the clear version of it.

SET DESIRED TEMPERATURE

Summary: The system sets the desired temperature to a comfortable temperature for residents or a minimum safe temperature for a vacant house.

Precondition: The desired temperature has been set to a minimum safe temperature.

Description and Assumption:

Main sequence:

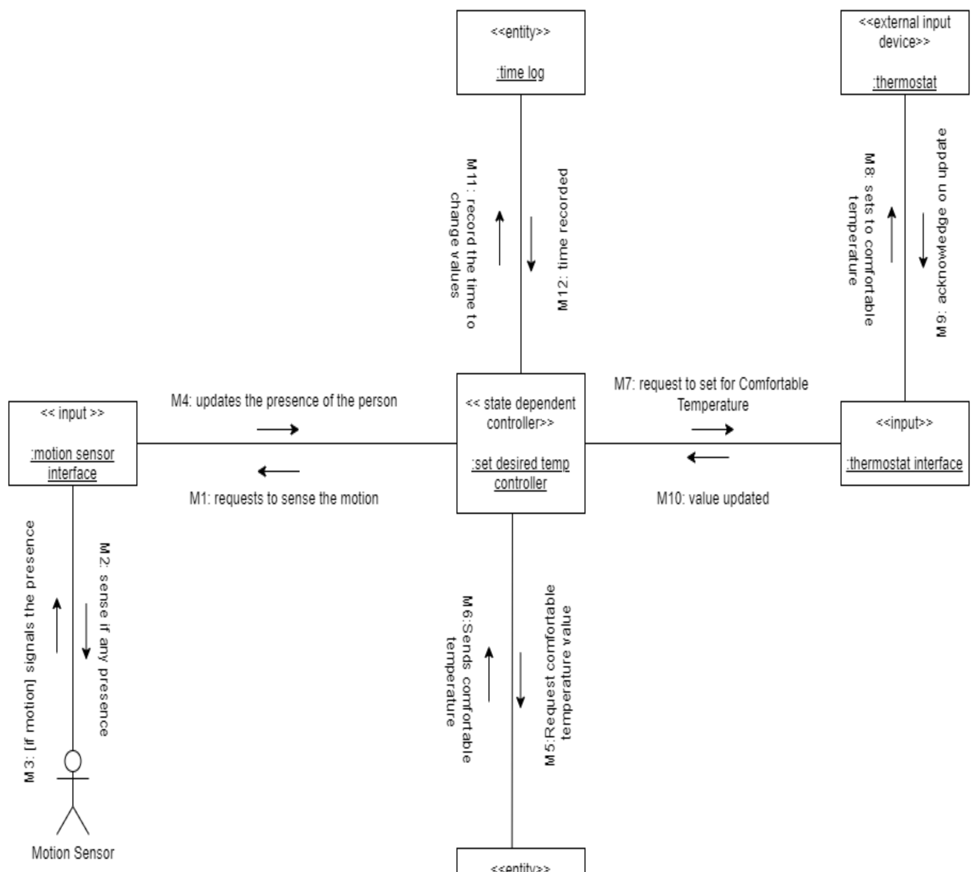
In this use case the system acts as a state-dependent controller. The system requests the motion sensor interface to sense the motion. The interface signals to the motion sensor which is an external input device to sense the motion. If it detects any motion it sends signals to the motion sensor interface. The motion sensor interface updates the status of the motion to the system. The system requests the Temperature(<<entity>>) object for a comfortable temperature value. The temperature object sends the comfortable temperature to the system. The system requests the thermostat interface to be set for a comfortable temperature. The interface sets the comfortable temperature and updates the values to the system. The system records the time to change the desired temperature to a comfortable temperature.

Assumptions:

In the set desired temperature use case the assumptions we included are, the temperature entity will store the values of temperature that has been updated by the system and the time log is used to record the time of action in the system.

Note: You can click on the diagram to get the clear version of it.

Main sequence



<<entity>>
:time log
<<external input device>>
:thermostat
M11: record the time to change values
M12: time recorded
M8: sets to comfortable temperature
M9: acknowledge on update
M4: updates the presence of the person
M7: request to set for Comfortable Temperature
<< input >>
:motion sensor interface
<< state dependent controller>>
:set desired temp controller
<<input>>
:thermostat interface
M1: requests to sense the motion
M10: value updated
M3: [if motion] signals the presence
M2: sense if any presence
M6:Sends comfortable temperature
M5:Request comfortable temperature value
Motion Sensor
<<entity>>
:temperature

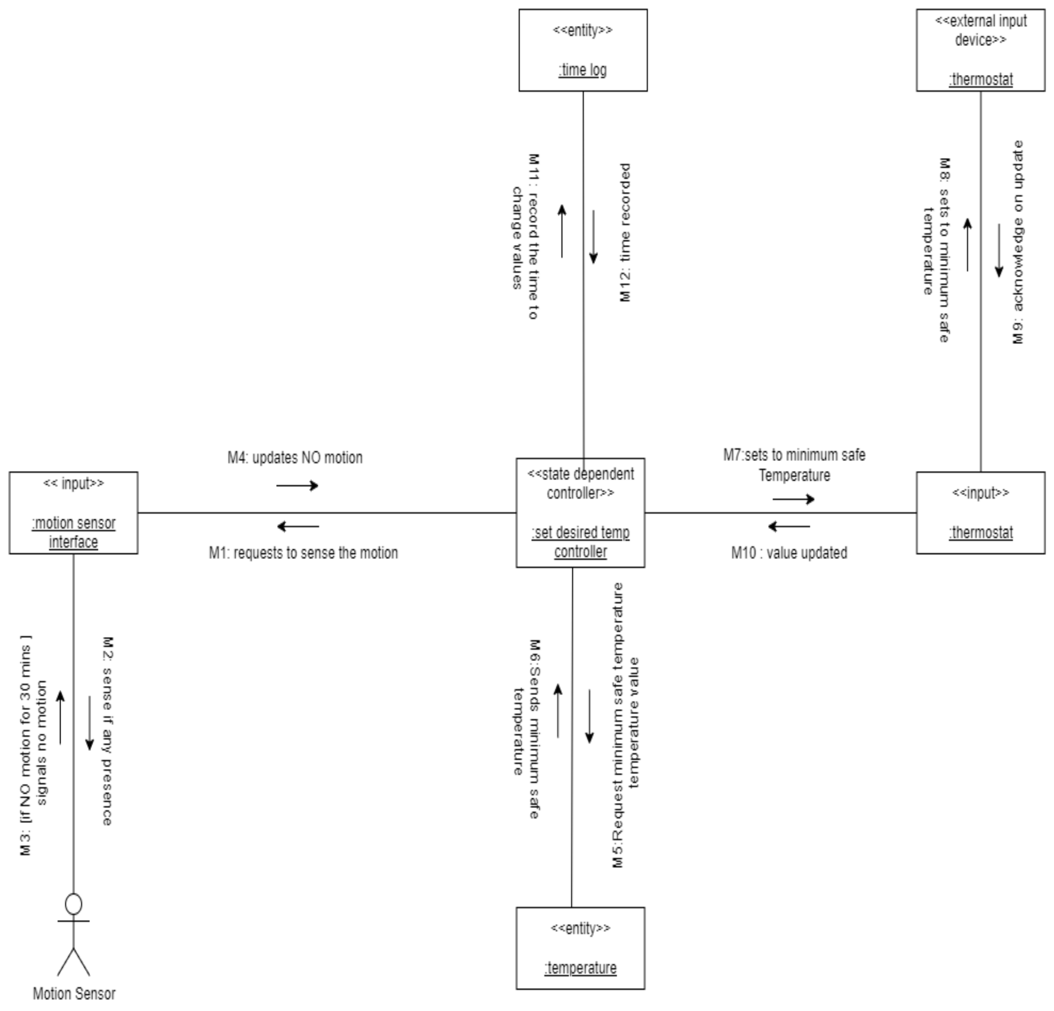
Note: You can click on the diagram to get the clear version of it.

Alternative sequence:

The system requests the motion sensor interface to sense the motion. The interface signals to the motion sensor which is an external input device to sense the motion. The motion sensor sends signals that there is no motion to the motion sensor interface. The motion sensor interface updates the status that there is no motion to the system. The system requests the temperature(<<entity>>) for minimum safe temperature values. The temperature(<<entity>>) sends temperature values to the system. The system communicates with the thermostat interface to set to minimum safe temperature values. The thermostat interface sets the minimum safe temperature and updates the values to the system. The system records the time to change the minimum safe temperature.

Note: You can click on the diagram to get the clear version of it.

Alternate sequence



<<entity>>
:time log
<<external input device>>
:thermostat
M11 : record the time to change values
M12: time recorded
M8: sets to minimum safe temperature
M9: acknowledge on update
<< input>>
:motion sensor interface
M4: updates NO motion
M1: requests to sense the motion
<<state dependent controller>>
:set desired temp controller
M7:sets to minimum safe Temperature
M10 : value updated
<<input>>
:thermostat
M3: [if NO motion for 30 mins] signals no motion
M2: sense if any presence
M6:Sends minimum safe temperature
M5:Request minimum safe temperature temperature value
<<entity>>
:temperature
Motion Sensor

Postcondition: The system has set the desired temperature.

Note: You can click on the diagram to get the clear version of it.

RECORD AUTOMOBILE GAS USE

Summary: The system records the automobile gas amount used.

Pre condition: None.

Description and Assumption

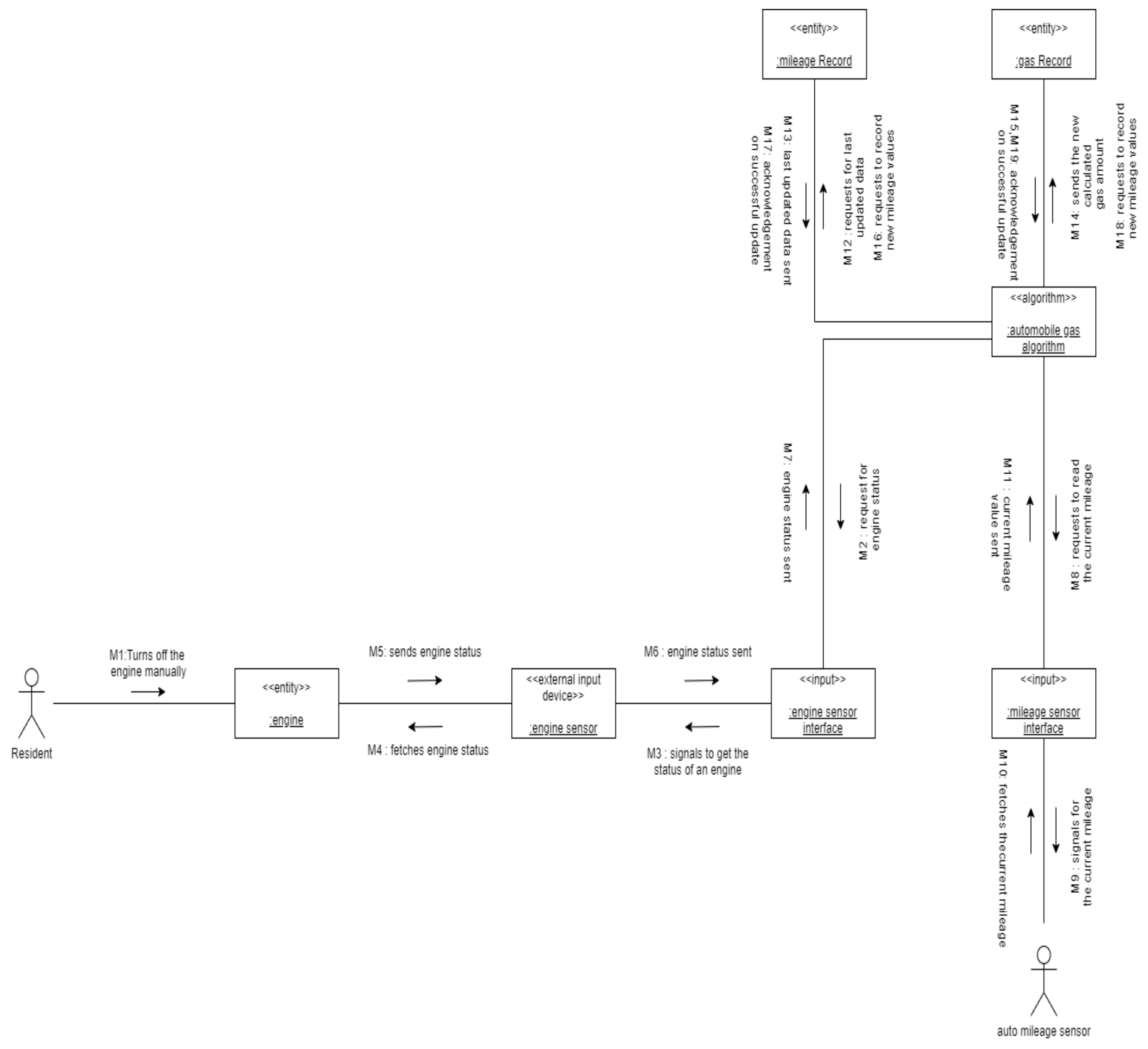
Main sequence:

In this use case there is no use of a state-dependent controller or any other controller. We need only the algorithm object to calculate the usage of gas amount. The resident turns off the engine manually. The automobile gas (<<algorithm>>) object requests the engine sensor interface for the engine status. The engine sensor interface signals the engine sensor(<<external input device>>) to get the status of an engine. The engine sensor fetches the status of an engine. The engine (<<entity>>) sends the status to the sensor. The sensor updates the status to the interface. The interface sends the engine status to the algorithm object. The algorithm object requests the mileage sensor interface to read the current mileage value. The interface signals the auto mileage sensor for the mileage value. The mileage sensor fetches the current mileage and sends it to the interface. The interface updates the current mileage value to the algorithm object. The algorithm object requests the mileage record(<<entity>>) for the updated mileage value. The entity updated the last mileage value. The automobile gas algorithm sends the new calculated gas amount to the gas record(<<entity>>). The entity gives acknowledgement on successful updates. The algorithm object requests the mileage record entity for the new mileage value. The entity acknowledged successfully the automobile gas algorithm object. The automobile gas algorithm requests to record a new mileage value to the gas record(<<entity>>). The entity gives acknowledgment of a successful update.

Assumptions:

In the record automobile gas use, we assumed when the resident turns off the engine, the engine sensor will automatically detect the engine status and it sends the update to the algorithm, then algorithm will work with the flow of description.

Note: You can click on the diagram to get the clear version of it.



<<entity>> :mileage Record
<<entity>> :gas Record
M13: last updated data sent
M17: acknowledgement on successful update
M12 : requests for last updated data
M16: requests to record new mileage values
M15,M19: acknowledgement on successful update
M14: sends the new calculated gas amount
M18: requests to record new mileage values
<<algorithm>> :automobile gas algorithm
M7: engine status sent
M2 : request for engine status
M11 : current mileage value sent
M8 : requests to read the current mileage
M1:Turns off the engine manually
M5: sends engine status
M6 : engine status sent
Resident
<<entity>> :engine
<<external input device>> :engine sensor
<<input>> :engine sensor interface
<<input>> :mileage sensor interface
M4 : fetches engine status
M3 : signals to get the status of an engine
M10: fetches thecurrent mileage
M9 : signals for the current mileage
auto mileage sensor

Alternative sequence: None

Postcondition: The system has recorded the automobile gas use.

Note: You can click on the diagram to get the clear version of it.

LIGHTS ON/OFF

Summary: The system turns on/off the light depending on the resident staying or not in a room.

Pre condition: None.

Description and Assumptions

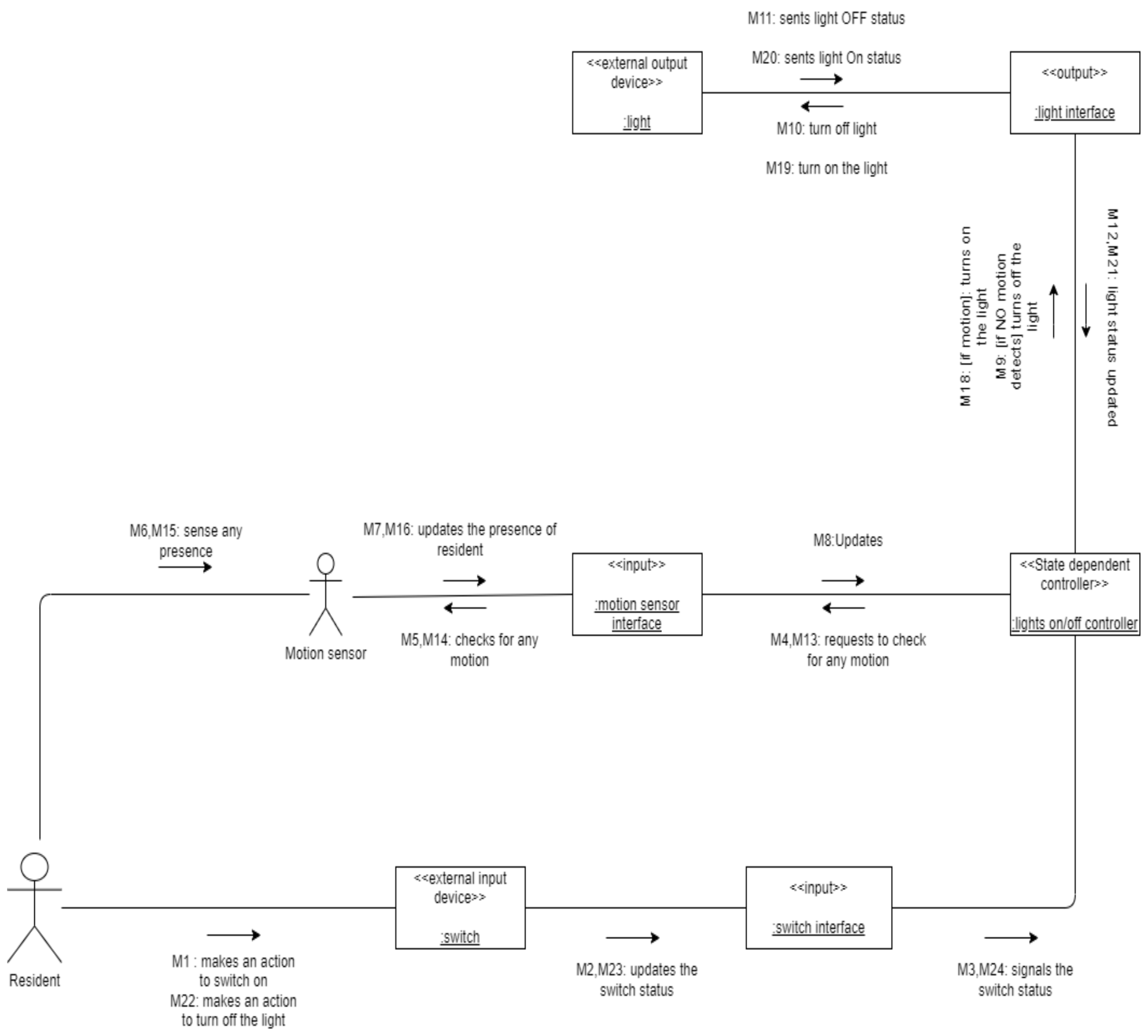
Main sequence:

In this use case the system acts as a state-dependent controller. The resident makes an action to switch on (<<external input device>>). The switch object updates the switch status to the switch interface. The interface signals the switch status to the system. The system requests the motion sensor interface for the presence of any motion. The interface signals the motion sensor for the motion. If the motion sensor detects the person’s presence updates the status that there is a motion to the interface. The interface updates the status to the system. If there is no motion the system communicates with the light interface to turn off the light. The interface turns off the light and sends the light status to the system. The system again requests the motion sensor interface for the motion. The motion sensor detects the presence of the resident and updates the status to the motion sensor interface. The interface gives the status of the motion to the system. The resident makes an action to turn off the light. The switch interface updates the switch status to the system.

Assumptions:

In the lights on/off use case, we assumed that when the resident turns on the switch, the switch interface will communicate the status of switch to the controller. The controller performs the continuous action of sensing the motion in the room and acts accordingly, and finally when resident turns the switch off, it will be directly reported to the controller by the switch interface.

Note: You can click on the diagram to get the clear version of it.


M11: sents light OFF status
M20: sents light On status
M10: turn off light
M19: turn on the light
<<external output device>>
:light
<<output>>
:light interface
M18: [if motion]: turns on the light
M9: [if NO motion detects] turns off the light
M12,M21: light status updated
M6,M15: sense any presence
M7,M16: updates the presence of resident
M8:Updates
Motion sensor
<<input>>
:motion sensor interface
<<State dependent controller>>
:lights on/off controller
M5,M14: checks for any motion
M4,M13: requests to check for any motion
Resident
<<external input device>>
:switch
<<input>>
:switch interface
M1 : makes an action to switch on
M22: makes an action to turn off the light
M2,M23: updates the switch status
M3,M24: signals the switch status

Alternative sequence: None

Postcondition: The system has lit on/off the light.

Note: You can click on the diagram to get the clear version of it.

MONITOR WATER LEAK

Summary: The system alerts the resident of water leaks.

Pre condition: None.

Description and Assumptions

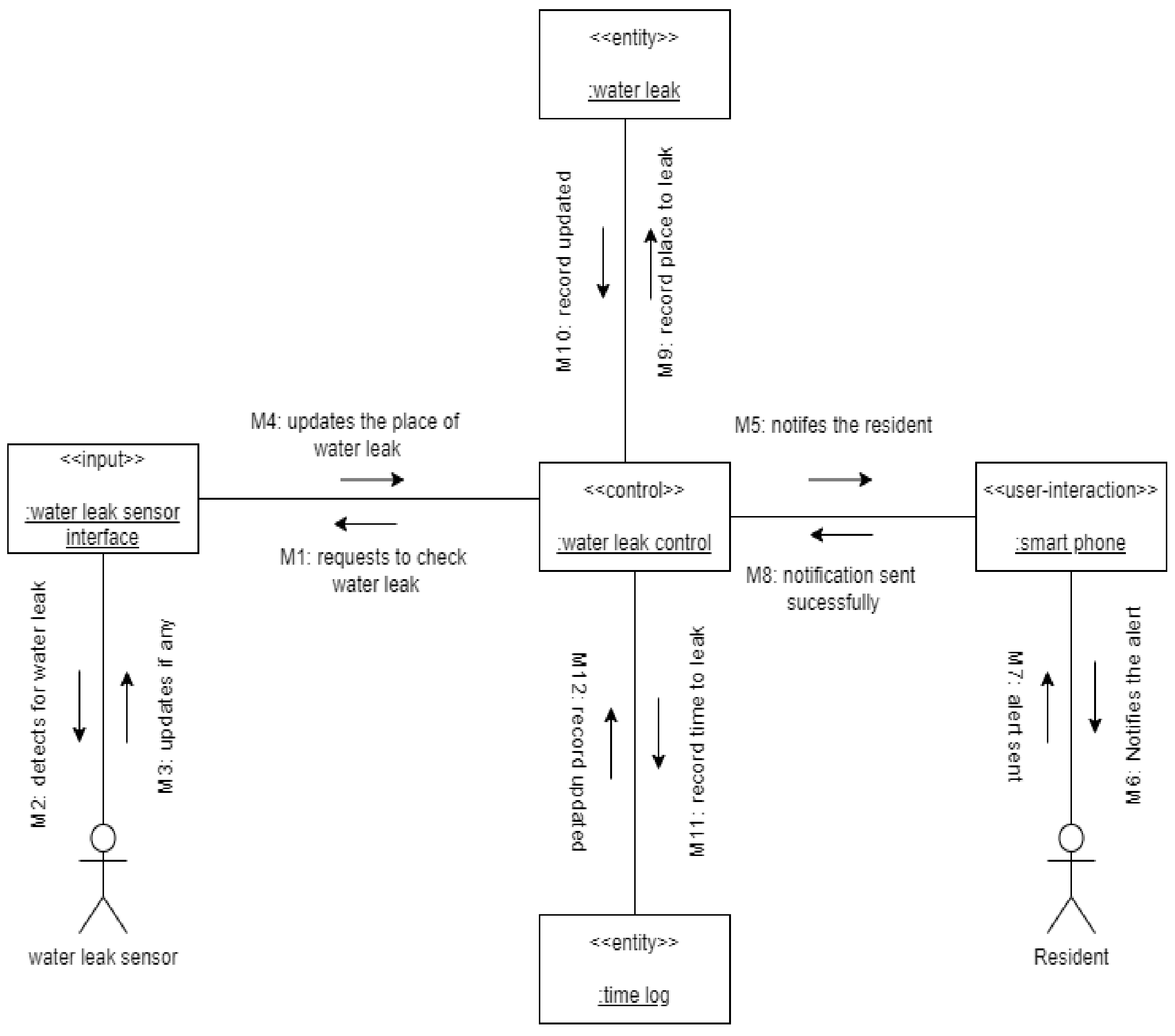
Main sequence:

In this use case the system acts as a control object. The control object requests the water leak sensor interface to check for water leaks. The water leak sensor updates if there is any water leak to the interface. The interface sends the place of the water leak to the system. The system notifies the resident via smartphone. The resident acknowledged successfully. The system records the place of a leak in a water leak object(<<entity>>). The system records the time to leak in the time log object(<<entity>>).

Assumptions:

In the monitor water leak use case, we assumed that the water leak control will take care of the flow. In the use case, water leak control will requests to check for water leak in the house continuously with the help of interface and notifies the user by sending any message or notification with the help of user interaction smartphone. The time of action is recorded in the time log. This time log entity is used to record time in every action of the entire system

Note: You can click on the diagram to get the clear version of it.



Alternative sequence: None

Postcondition: The system has sent the water leak alert to the resident.

Note: You can click on the diagram to get the clear version of it.

DETECT FIRE

Summary: The system detects smoke or fire and extinguishes it.

Pre condition: None.

Description and Assumptions

Main sequence:

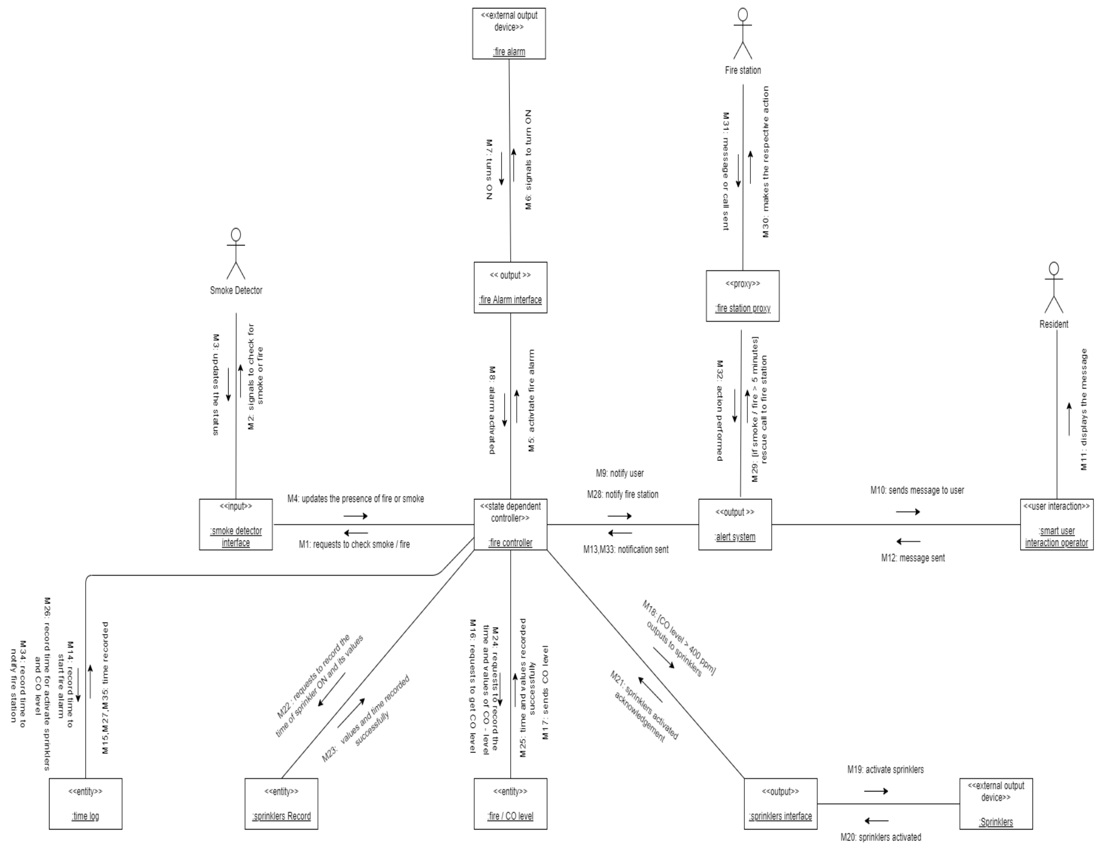
In this use case the system acts as a state-dependent controller. The system requests a smoke detector interface to check if there is a smoke/fire. The smoke detector interface signals to the smoke detector to check for smoke/fire. The smoke detector updates the status on the interface. The interface updates the existence of a fire/smoke to the system. The system communicates with the fire alarm interface to activate the fire alarm. The interface signals the fire alarm to turn on and sends the status to the interface. The interface sends the acknowledgment to the system that the fire alarm is activated. The system notifies the user about fire via smartphone and records the time to activate the fire alarm. The system requests fire/co level(<<entity>>) to get CO level. The entity sends CO level to the system. If CO level > 400 ppm the system outputs to the sprinkler interface. The sprinkler interface communicates with the sprinkler to turn on. The sprinkler sends the status to the interface. The interface updates the status to the system. The fire controller requests the sprinklers record(<<entity>>) to record the time of sprinkler on and its values. The sprinkler's record sends the status to the system. The system requests a fire record entity to store the time and the values of the CO level. The fire record entity sends the CO level values to the system. The system records the time to activate the sprinklers and CO level. If Smoke/fire exists > 5 minutes the system notifies the fire station. The fire station proxy (<<proxy>>) object makes the respective action and sends the acknowledgment to the system.

Assumptions:

In the detect fire use case, we assumed that the fire controller uses the object alert system to notify both the resident and the fire station. And every external device uses the interface to interact with the system and respective entities to store their values.

Note: You can click on the diagram to get the clear version of it.

Main Sequence



<<external output device>> :fire alarm
M7: turns ON
M6: signals to turn ON
Fire station
M31: message or call sent
M30: makes the respective action
Smoke Detector
M3: updates the status
M2: signals to check for smoke or fire
<< output >> :fire Alarm interface
M8: alarm activated
M5: activate fire alarm
<<proxy>> :fire station proxy
M32: action performed
M29: [if smoke / fire > 5 minutes] rescue call to fire station
Resident
M11: displays the message
M9: notify user
M28: notify fire station
M13,M33: notification sent
<<input>> :smoke detector interface
M4: updates the presence of fire or smoke
M1: requests to check smoke / fire
<<state dependent controller>> :fire controller
<<output >> :alert system
M10: sends message to user
M12: message sent
<<user interaction>> :smart user interaction operator
M26: record time for activate sprinklers and CO level
M34: record time to notify fire station
M14: record time to start fire alarm
M15,M27,M35: time recorded
M22 : requests to record the time of sprinkler ON and its values
M23: values and time recorded successfully
M16: requests to get CO level
M24: requests to record the time and values of CO - level
M25: time and values recorded successfully
M17: sends CO level
M18: [CO level > 400 ppm] outputs to sprinklers
M21: sprinklers activated acknowledgement
<<entity>> :time log
<<entity>> :sprinklers Record
<<entity>> :fire / CO level
<<output>> :sprinklers interface
M19: activate sprinklers
M20: sprinklers activated
<<external output device>> :Sprinklers

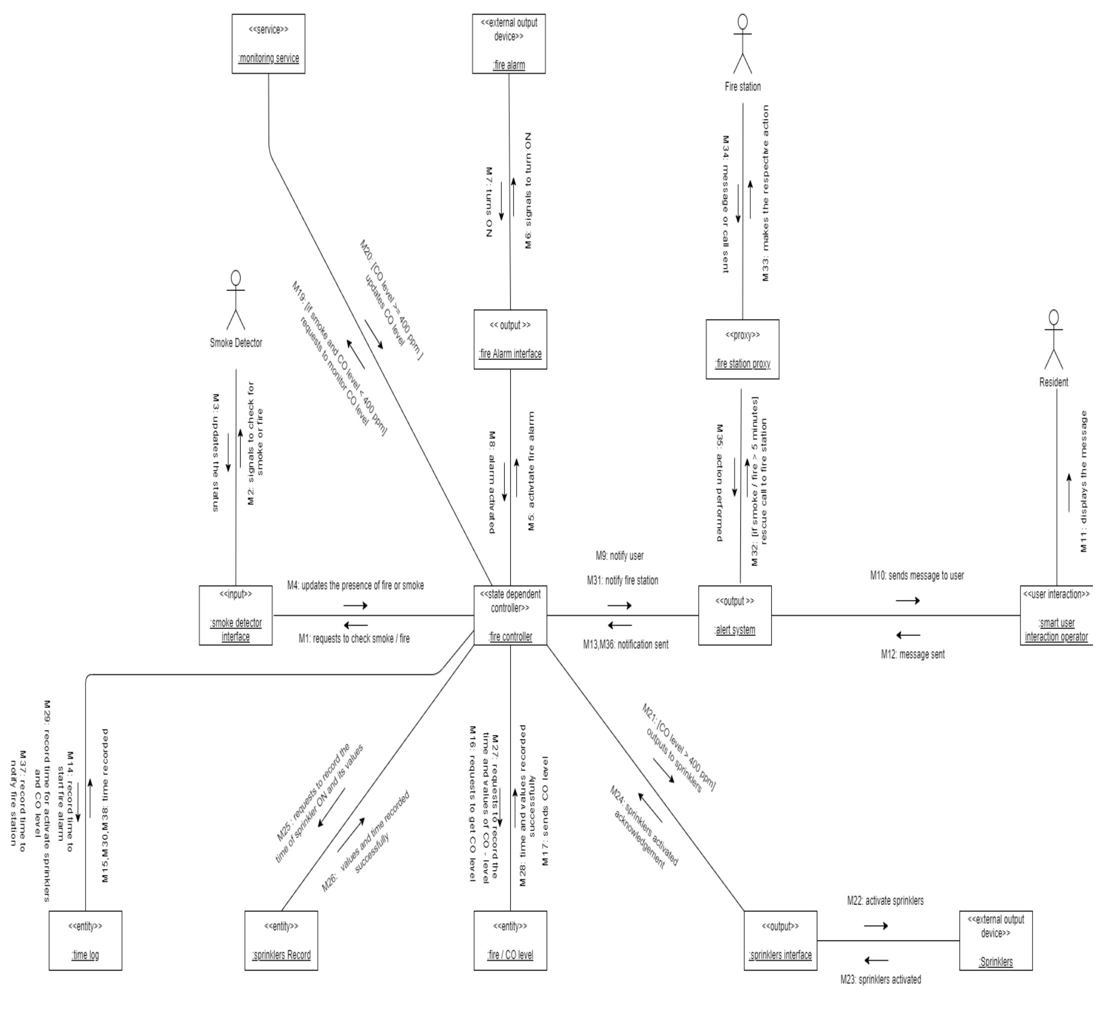
Note: You can click on the diagram to get the clear version of it.

Alternative sequence:

The system requests a smoke detector interface to check if there is a smoke/fire. The smoke detector interface signals to the smoke detector to check for smoke/fire. The smoke detector updates the status on the interface. The interface updates the existence of a fire/smoke to the system. The system communicates with the fire alarm interface to activate the fire alarm. The interface signals to fire alarm to turn on and sends the status to the interface. The interface sends the acknowledgment to the system that the fire alarm is activated. The system notifies the user about fire via smartphone and records the time to activate the fire alarm. The system requests fire/co level(<<entity>>) to get the CO level. The entity sends CO level to the system. If smoke and CO level are < 400 ppm the system requests the monitoring service to monitor CO levels. If the CO level is > 400 ppm the monitoring service updates the CO level to the system and the system output to the sprinklers interface. The sprinkler interface communicates with the sprinkler to turn on. The sprinkler sends the status to the interface. The interface updates the status to the system. The fire controller requests the sprinklers record(<<entity>>) to record the time of sprinkler on and its values. The sprinkler's record sends the status to the system. The system requests a fire record entity to store the time and the values of the CO level. The fire record entity sends the CO level values to the system. The system records the time to activate the sprinklers and CO level. If Smoke/fire exists > 5 minutes the system notifies to fire station. The fire station proxy (<<proxy>>) object makes the respective action and sends the acknowledgment to the system.

Note: You can click on the diagram to get the clear version of it.

Alternative Sequence


<<service>>
:monitoring service
<<external output device>>
:fire alarm
Fire station
M34: message or call sent
M33: makes the respective action
M7: turns ON
M6: signals to turn ON
M20: [CO level >= 400 ppm] updates CO level
M19: [if smoke and CO level < 400 ppm] requests to monitor CO level
Smoke Detector
<< output >>
:fire Alarm interface
<<proxy>>
:fire station proxy
Resident
M3: updates the status
M2: signals to check for smoke or fire
M8: alarm activated
M5: activate fire alarm
M35: action performed
M32: [if smoke / fire > 5 minutes] rescue call to fire station
M11: displays the message
M9: notify user
M31: notify fire station
<<input>>
:smoke detector interface
M4: updates the presence of fire or smoke
M1: requests to check smoke / fire
<<state dependent controller>>
:fire controller
M13,M36: notification sent
<<output>>
:alert system
M10: sends message to user
M12: message sent
<<user interaction>>
:smart user interaction operator
M29: record time for activate sprinklers and CO level
M37: record time to notify fire station
M14: record time to start fire alarm
M15,M30,M38: time recorded
M25: requests to record the time of sprinkler ON and its values
M26: values and time recorded successfully
M27: requests to record the time and values of CO - level
M16: requests to get CO level
M28: time and values recorded successfully
M17: sends CO level
M21: [CO level > 400 ppm] outputs to sprinklers
M24: sprinklers activated acknowledgement
<<entity>>
:time log
<<entity>>
:sprinklers Record
<<entity>>
:fire / CO level
<<output>>
:sprinklers interface
M22: activate sprinklers
M23: sprinklers activated
<<external output device>>
:Sprinklers

Postcondition: The system has detected smoke or fire.

Note: You can click on the diagram to get the clear version of it.

HOME SECURITY

Summary: The system guards the home against a suspicious person.

Pre condition: None.

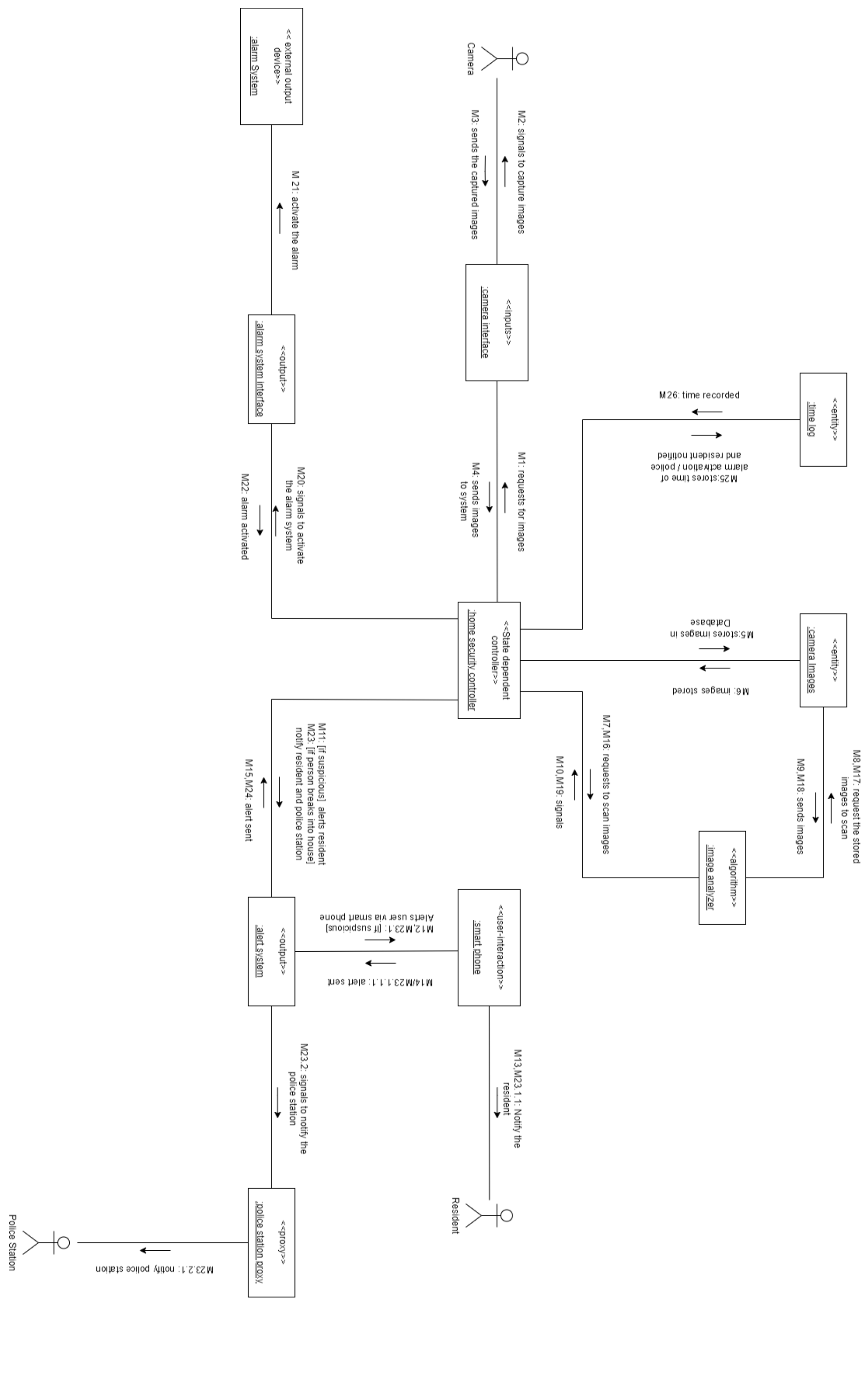
Main sequence:

In this use case the system acts as a state-dependent controller. The system requests the camera interface to capture images. The interface signals the camera to capture the images. The camera captures the images and sends them to the interface and the interface sends them to the system. The system stores the captured images in the database(camera images <<entity>>). The entity was acknowledged successfully to the system. The system requests an image analyzer(<<algorithm>>) object to scan images. The analyzer requests the entity to scan the stored images. The camera image object sends the scanned images to the analyzer and the analyzer sends the images to the system. The system notifies residents via smartphone if the images are suspicious. The system again requests an image analyzer(<<algorithm>>) object to scan images. The analyzer requests the entity to scan the stored images. The camera image object sends the scanned images to the analyzer and the analyzer sends the images to the system. The system signals the alarm interface to activate the alarm system. The interface activates the alarm and sends the status to the system. If the suspicious person breaks into the house the system notifies the resident and police station.

Assumptions:

In the home security use case, we assumed that the fire controller uses the object alert system to notify both the resident and the police station. And every external device uses the interface to interact with the system and respective entities to store their values. The image analyzer is the object we used to analyze the images and informs the threat to the controller.

Note: You can click on the diagram to get the clear version of it.


<< external output device>>
:alarm System
Camera
M2: signals to capture images
M3: sends the captured images
M 21: activate the alarm
<<inputs>>
:camera interface
<<output>>
:alarm system interface
M26: time recorded
<<entity>>
:time log
M25:stores time of alarm actwation / police and resident notified
M1: requests for images
M4: sends images to system
M20: signals to activate the alarm system
M22: alarm activated
<<State dependent controller>>
:home security controller
<<entity>>
:camera images
M5:stores images in Database
M6: images stored
M7,M16: requests to scan images
M10,M19: signals
M8,M17: request the stored images to scan
M9,M18: sends images
<<algorithm>>
:image analyzer
M11: [if suspicious] alerts resident
M23: [if person breaks into house] notify resident and police station
M15,M24: alert sent
<<output>>
:alert system
<<user-interaction>>
:smart phone
M12,M23.1: [if suspicious] Alerts user via smart phone
M14/M23.1.1.1: alert sent
M23.2: signals to notify the police station
M13,M23.1.1: Notify the resident
Resident
<<proxy>>
:police station proxy
M23.2.1: notify police station
Police Station

Alternative sequence: None

Postcondition: The system has guarded the house.

Note: You can click on the diagram to get the clear version of it.

LOGIN

Summary: The resident logs into the system.

Pre condition: None.

Description and Assumptions

Main sequence:

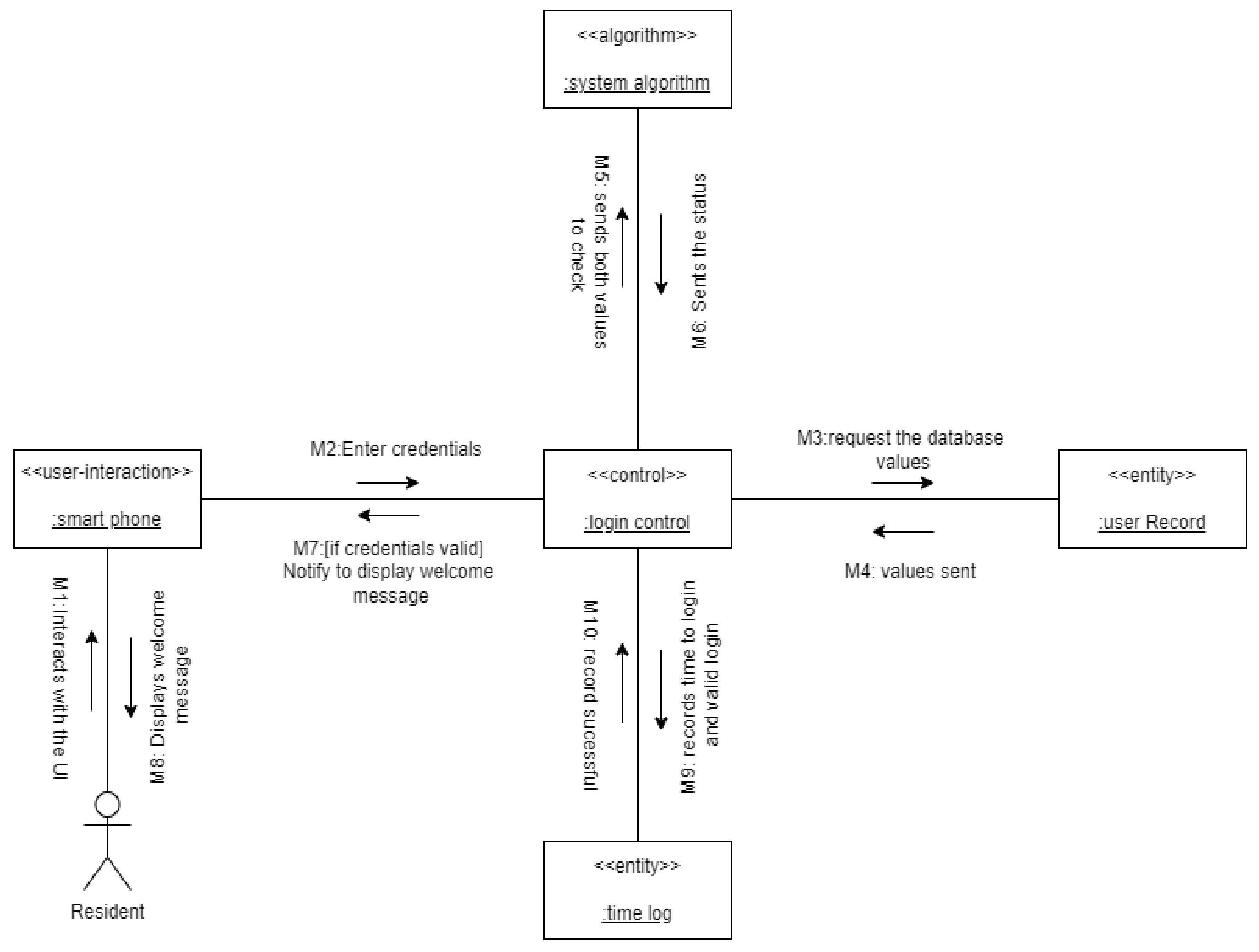
In this use case the system acts as a controller. The resident interacts with the UI via smartphone to log in to the system by entering credentials(ID and password). The control object requests the user record(<<entity>>) for the database values. The user record object sends the values to the system. The system sends both the values(ID and password) to the algorithm object to check whether the credentials are valid or not. If the credentials are valid the system is notified to display a welcome message. The smartphone displays the welcome message to the user. The system records the time and valid login.

Assumptions:

In the login use case, we assumed that the login control will take care of the flow in the use case and the object system algorithm is used to validate the username and password in the system. User Record will store the values of user credentials and time log stores the action of time.

Note: You can click on the diagram to get the clear version of it.

Main sequence



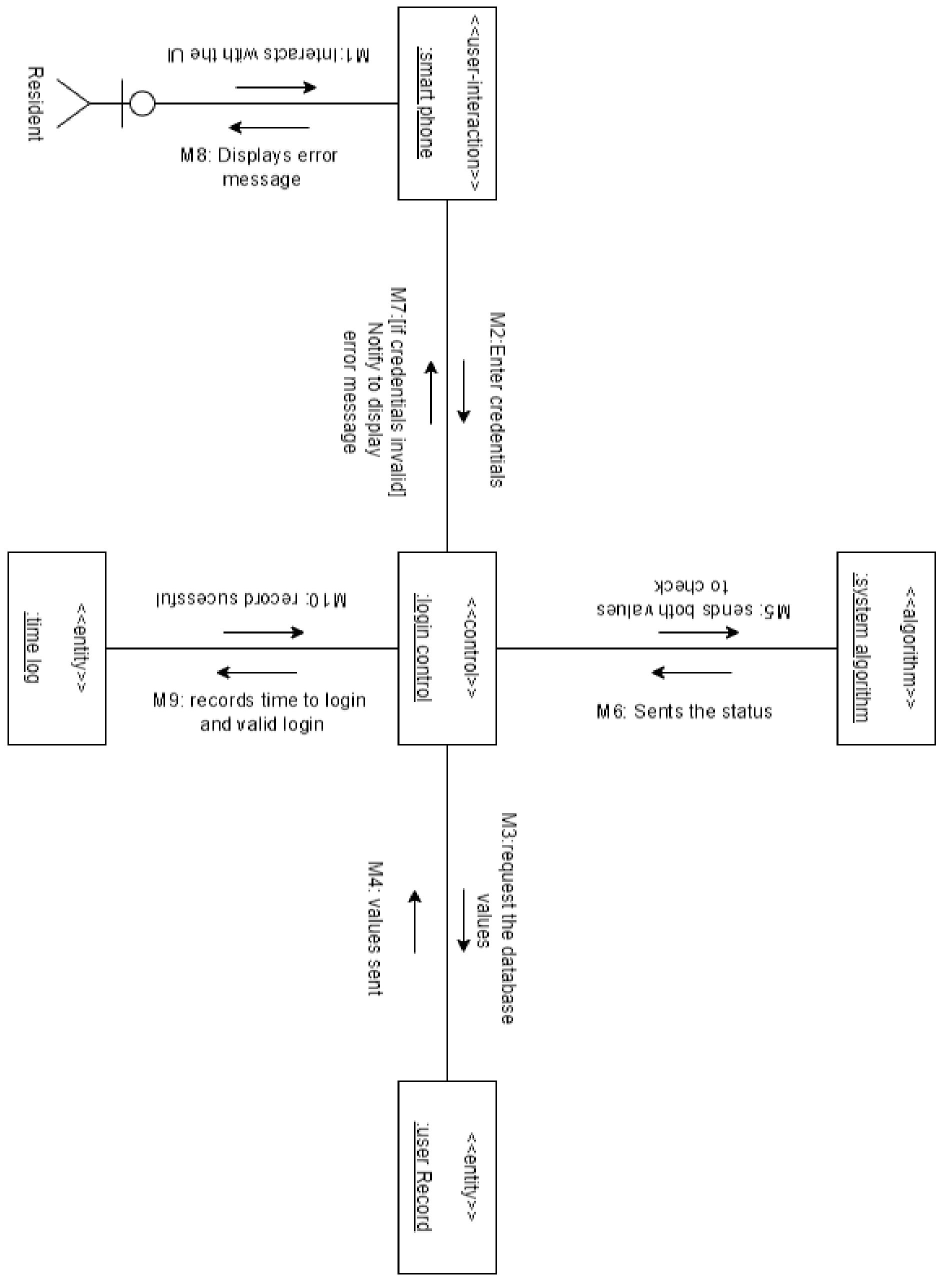
Note: You can click on the diagram to get the clear version of it.

Alternative sequence:

The resident interacts with the UI via smartphone to log in to the system by entering credentials(ID and password). The control object requests the user record(<<entity>>) for the database values. The user record object sends the values to the system. The system sends both the values(ID and password) to the algorithm object to check whether the credentials are valid or not. If the credentials are invalid the system notifies to display an error message. The smartphone displays the error message to the user. The system records the time and valid login.

Note: You can click on the diagram to get the clear version of it.

Alternative Sequence



Postcondition: The resident has logged into the system.

Note: You can click on the diagram to get the clear version of it.

VIEW/ CHANGE VALUES

Summary: The resident specifies the comfortable temperature and a minimum safe temperature.

Precondition: The resident has logged into the system.

Description and Assumption

Main sequence:

In this use case the system acts as a state-dependent controller. The resident selects the view status on the smartphone. The smartphone requests the system to view the status. The system grants the user request and displays the home status edit/logout menu. The residents select the edit menu on the smartphone. The smartphone requests the system to edit the menu. The system grants the request and displays the page to edit a comfortable or minimum safe temperature. The resident enters the comfortable or minimum safe temperature. The user interaction object (smartphone) sends the new updated values to the system. The system sets the desired temperature values. The thermostat interface signals the thermostat to update the desired temperature values. The thermostat sends the acknowledgment on the update of desired temperature values to the system. The system records the values of the temperature and also records the time to alter. The resident selects the logout menu and the system has logged out the resident.

Alternative sequence:

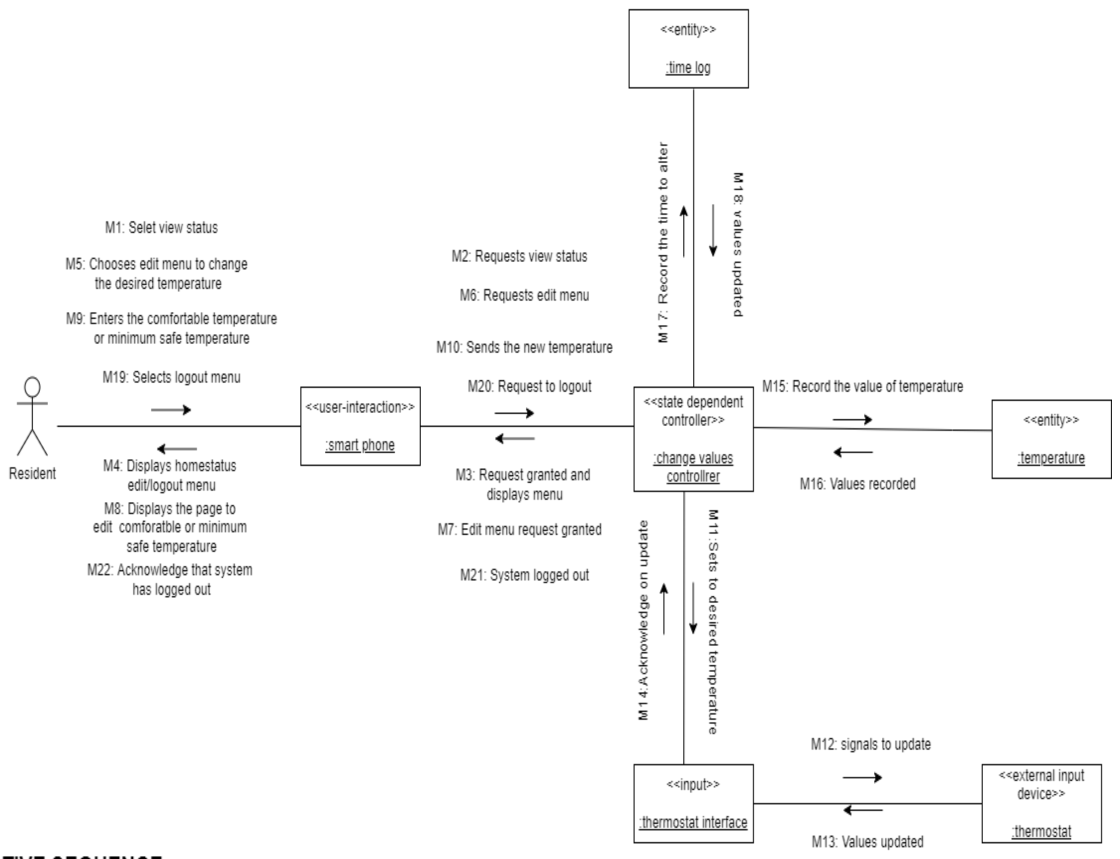
The resident selects the view status on the smartphone. The smartphone requests the system to view the status. The system grants the user request and displays the home status edit/logout menu. The residents select the logout menu on the smartphone. The smartphone requests the system to log out. The system has logged out the resident.

Assumptions:

In the View/change values use case we did not assume anything, the system goes with the description flow.

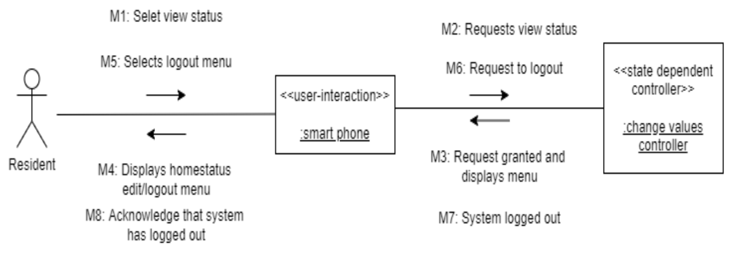
Note: You can click on the diagram to get the clear version of it.

MAIN SEQUENCE



<<entity>>
:time log
M17: Record the time to alter
M18: values updated
M1: Selet view status
M5: Chooses edit menu to change the desired temperature
M9: Enters the comfortable temperature or minimum safe temperature
M19: Selects logout menu
M2: Requests view status
M6: Requests edit menu
M10: Sends the new temperature
M20: Request to logout
M15: Record the value of temperature
Resident
<<user-interaction>>
:smart phone
<<state dependent controller>>
:change values controllrer
<<entity>>
:temperature
M4: Displays homestatus edit/logout menu
M8: Displays the page to edit comforatble or minimum safe temperature
M22: Acknowledge that system has logged out
M3: Request granted and displays menu
M7: Edit menu request granted
M21: System logged out
M16: Values recorded
M14:Acknowledge on update
M11 :Sets to desired temperature
M12: signals to update
<<input>>
:thermostat interface
<<external input device>>
:thermostat
M13: Values updated

ALTERNATIVE SEQUENCE



M1: Selet view status
M5: Selects logout menu
M2: Requests view status
M6: Request to logout
Resident
<<user-interaction>>
:smart phone
<<state dependent controller>>
:change values controller
M4: Displays homestatus edit/logout menu
M8: Acknowledge that system has logged out
M3: Request granted and displays menu
M7: System logged out

Postcondition: The resident has viewed the home status and changed the comfortable or minimum safe temperature.

Note: You can click on the diagram to get the clear version of it.

RECORD ELECTRICITY USE AND BILL

Summary: The system records the resident's electricity use and monthly bill on blockchain ledgers

Precondition: A smart meter has regularly recorded the amount of electricity used by the resident in the system and EC. (Suppose that the LC initiated an incentive program to provide discounts on electricity bills to customers who use lower electricity monthly. The discount rates had been specified.)

Description and Assumptions

Main sequence:

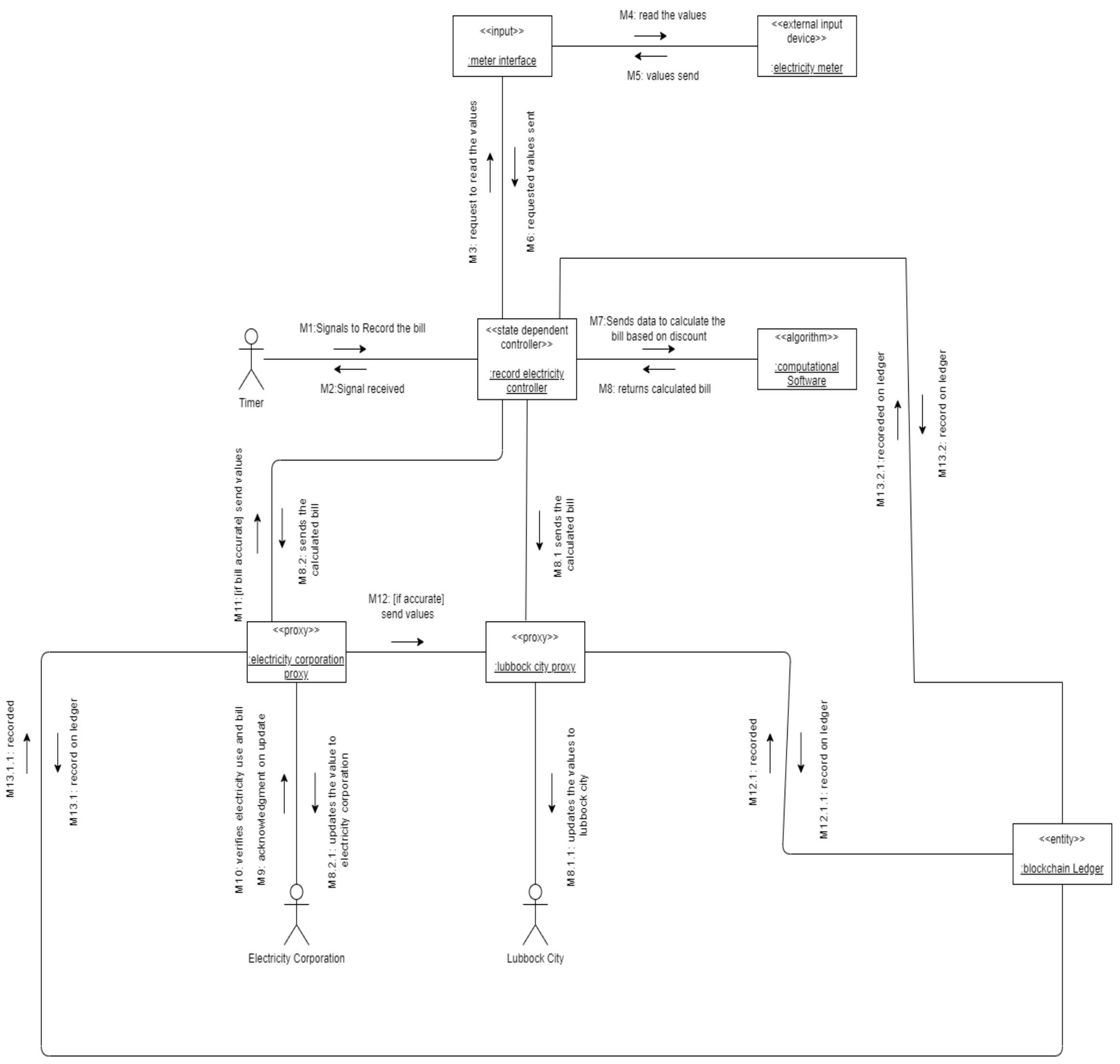
In this use case the system acts as a state-dependent controller. The timer sends signals to the system to record the bill. The system sends the acknowledgement to the timer. The system requests the meter interface to read the values. The meter interface signals the electric meter to read the values. The interface sends the requested values to the system. The system sends the data to the computational software(<<algorithm>>) object to calculate the bill based on the discount. The algorithm object returns the calculated bill to the system. The system sends the calculated bill to the electricity proxy object(<<proxy>>) and Lubbock city proxy object(<<proxy>>) simultaneously. The proxy objects update the values to the electricity corporation and Lubbock city. The electricity corporation sends acknowledgment on updates and verifies the electricity use and bill. If the electricity bill is accurate the electricity proxy object sends the values to the system and records the values in the electricity corporation blockchain ledger. Simultaneously, the electricity proxy object sends the values to the Lubbock city proxy object to record the values on the LC blockchain ledger. The system records the bill on the blockchain ledger(<<entity>>).

Assumptions:

In the record electricity, we assumed that there is a common block chain ledger for every thing in the system and the electricity corporation and Lubbock city also stores the values in the system ledger. We assumed a computational software algorithm in the use case to calculate the bill based on the discounts and send it to the controller.

Note: You can click on the diagram to get the clear version of it.

Main Sequence



<<input>>
:meter interface
M4: read the values
M5: values send
<<external input device>>
:electricity meter
M3: request to read the values
M6: requested values sent
Timer
M1:Signals to Record the bill
M2:Signal received
<<state dependent controller>>
:record electricity controller
M7:Sends data to calculate the bill based on discount
M8: returns calculated bill
<<algorithm>>
:computational Software
M13.2.1:recoreded on ledger
M13.2: record on ledger
M11:[if bill accurate] send values
M8.2: sends the calculated bill
M8.1 sends the calculated bill
M12: [if accurate] send values
<<proxy>>
:electricity corporation proxy
<<proxy>>
:lubbock city proxy
M13.1.1: recorded
M13.1: record on ledger
M10: verifies electricity use and bill
M9: acknowledgment on update
M8.2.1: updates the value to electricity corporation
M8.1.1: updates the values to lubbock city
M12.1: recorded
M12.1.1: record on ledger
<<entity>>
:blockchain Ledger
Electricity Corporation
Lubbock City

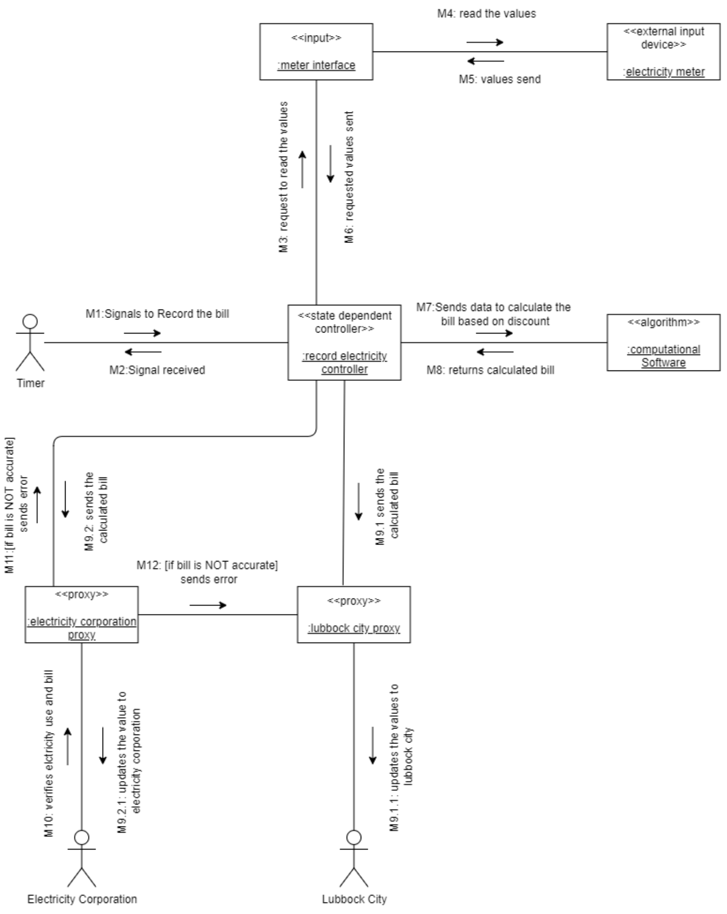
Note: You can click on the diagram to get the clear version of it.

Alternative sequence:

The timer sends signals to the system to record the bill. The system sends the acknowledgement to the timer. The system requests the meter interface to read the values. The meter interface signals the electric meter to read the values. The interface sends the requested values to the system. The system sends the data to the computational software(<<algorithm>>) object to calculate the bill based on the discount. The algorithm object returns the calculated bill to the system. The system sends the calculated bill to the electricity proxy object(<<proxy>>) and Lubbock city proxy object(<<proxy>>) simultaneously. The proxy objects update the values to the electricity corporation and Lubbock city. The electricity corporation sends acknowledgement on update and sends an error message if the calculated bill is inaccurate to the system.

Note: You can click on the diagram to get the clear version of it.

Alternative sequence


M4: read the values
<<input>>
:meter interface
<<external input device>>
:electricity meter
M5: values send
M3: request to read the values
M6: requested values sent
M1:Signals to Record the bill
<<state dependent controller>>
:record electricity controller
M7:Sends data to calculate the bill based on discount
<<algorithm>>
:computational Software
Timer
M2:Signal received
M8: returns calculated bill
M11:[if bill is NOT accurate] sends error
M9.2: sends the calculated bill
M9.1 sends the calculated bill
M12: [if bill is NOT accurate] sends error
<<proxy>>
:electricity corporation proxy
<<proxy>>
:lubbock city proxy
M10: verifies elctricity use and bill
M9.2.1: updates the value to electricity corporation
M9.1.1: updates the values to lubbock city
Electricity Corporation
Lubbock City

Postcondition: The system has recorded the monthly electricity use and bill on the blockchain ledger.

Note: You can click on the diagram to get the clear version of it.

VIEW ELECTRICITY USE AND BIL

Summary: The resident views the monthly electricity use and bill history.

Description and Assumption

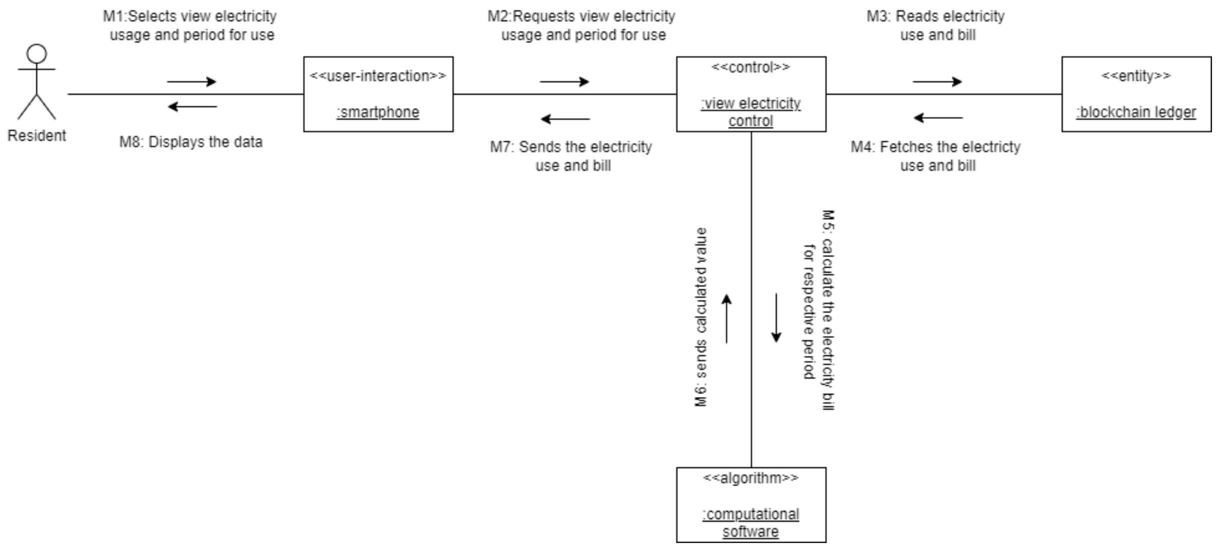
Main sequence:

In this use case the system acts as a controller. The resident selects the view electricity bill and usage on the smartphone. The user interaction object (smartphone) requests the system to view the electricity bill and period of usage. The system reads the electricity use and bill from the block chain ledger. The blockchain ledger sends the bill and use age to the system. The system sends the data to the computational software object(<,algorithm>>) to calculate. The algorithm object sends the calculated bill to the system. The system sends the electricity use and bill to the resident via smartphone.

Assumptions:

In the view of electricity use and bill use case, we assumed a computational software algorithm in the use case to calculate the bill based on the period and send it to the control. The rest of the flow goes with the description.

Note: You can click on the diagram to get the clear version of it.


M1:Selects view electricity usage and period for use
M2:Requests view electricity usage and period for use
M3: Reads electricity use and bill
Resident
<<user-interaction>>
:smartphone
<<control>>
:view electricity control
<<entity>>
:blockchain ledger
M8: Displays the data
M7: Sends the electricity use and bill
M4: Fetches the electricty use and bill
M6: sends calculated value
M5: calculate the electricity bill for respective period
<<algorithm>>
:computational software

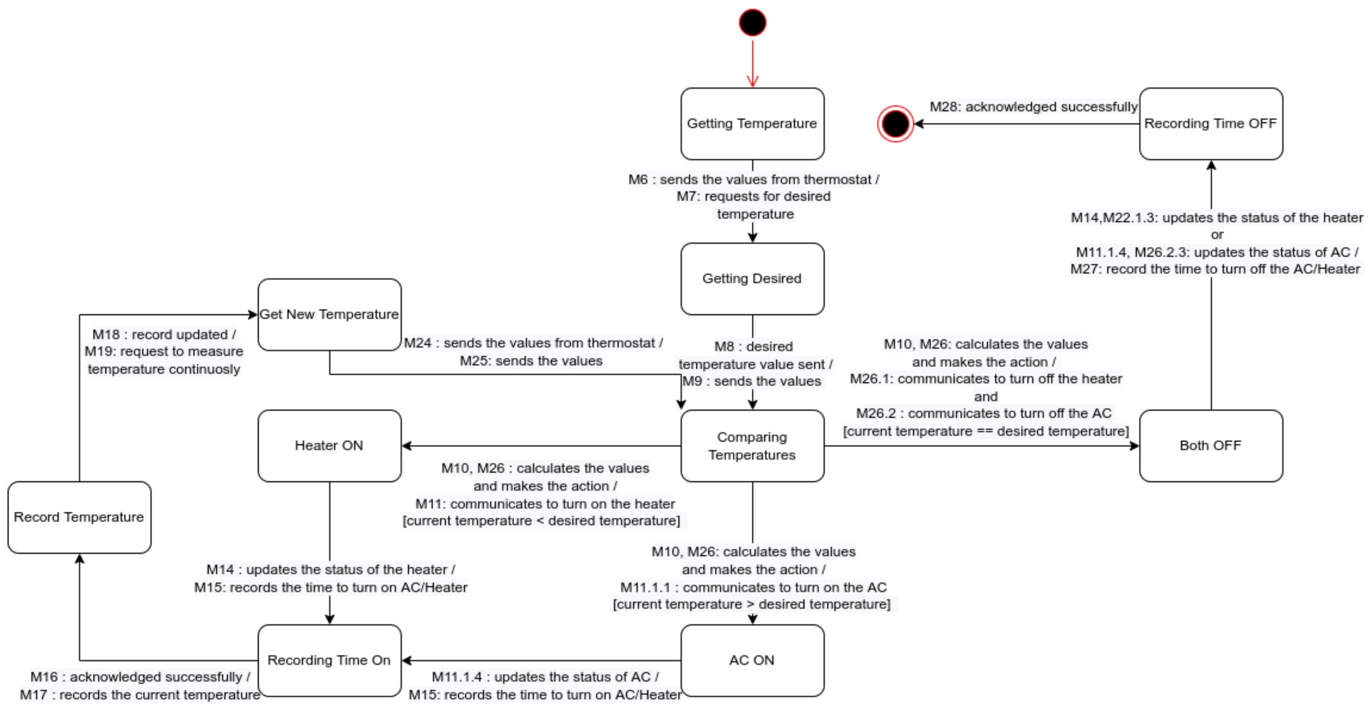
Alternative sequence: None

Postcondition: The resident has viewed the electricity use and bill for a period.

Note: You can click on the diagram to get the clear version of it.

Statechart Diagrams

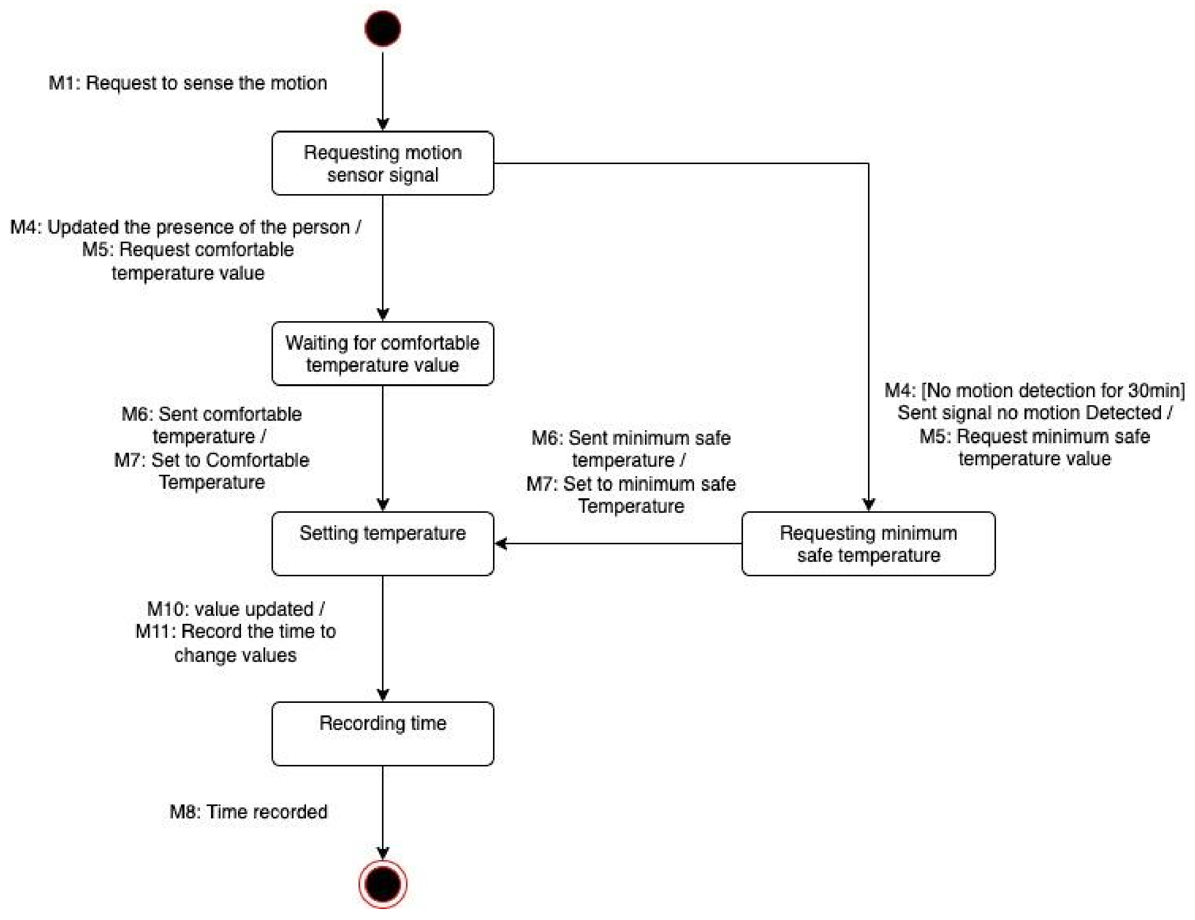
CONTROL TEMPERATURE



Getting Temperature
Recording Time OFF
M28: acknowledged successfully
M6 : sends the values from thermostat / M7: requests for desired temperature
M14,M22.1.3: updates the status of the heater or M11.1.4, M26.2.3: updates the status of AC / M27: record the time to turn off the AC/Heater
Getting Desired
Get New Temperature
M18 : record updated / M19: request to measure temperature continuosly
M24 : sends the values from thermostat / M25: sends the values
M8 : desired temperature value sent / M9 : sends the values
M10, M26: calculates the values and makes the action / M26.1: communicates to turn off the heater and M26.2 : communicates to turn off the AC [current temperature == desired temperature]
Heater ON
Comparing Temperatures
Both OFF
M10, M26 : calculates the values and makes the action / M11: communicates to turn on the heater [current temperature < desired temperature]
Record Temperature
M14 : updates the status of the heater / M15: records the time to turn on AC/Heater
M10, M26: calculates the values and makes the action / M11.1.1 : communicates to turn on the AC [current temperature > desired temperature]
Recording Time On
AC ON
M16 : acknowledged successfully / M17 : records the current temperature
M11.1.4 : updates the status of AC / M15: records the time to turn on AC/Heater

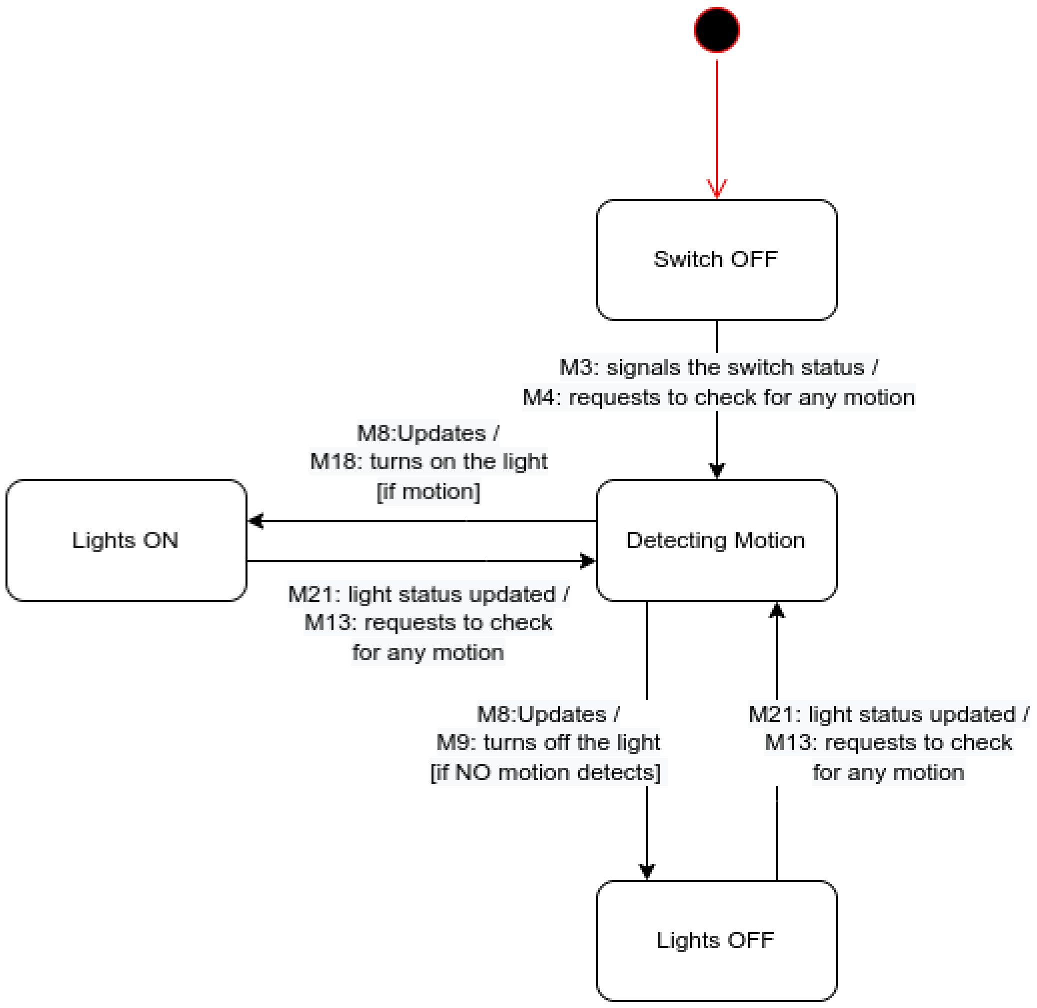
Note: You can click on the diagram to get the clear version of it.

SET DESIRED TEMPERATURE

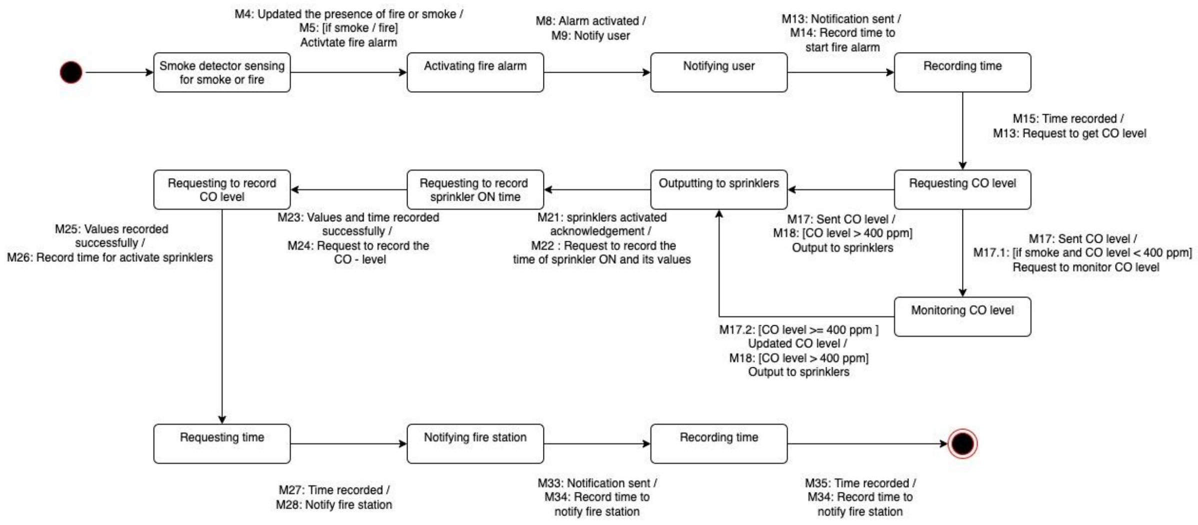

M1: Request to sense the motion
Requesting motion sensor signal
M4: Updated the presence of the person / M5: Request comfortable temperature value
Waiting for comfortable temperature value
M6: Sent comfortable temperature / M7: Set to Comfortable Temperature
M4: [No motion detection for 30min] Sent signal no motion Detected / M5: Request minimum safe temperature value
M6: Sent minimum safe temperature / M7: Set to minimum safe Temperature
Setting temperature
Requesting minimum safe temperature
M10: value updated / M11: Record the time to change values
Recording time
M8: Time recorded

Note: You can click on the diagram to get the clear version of it.

LIGHT ON/OFF

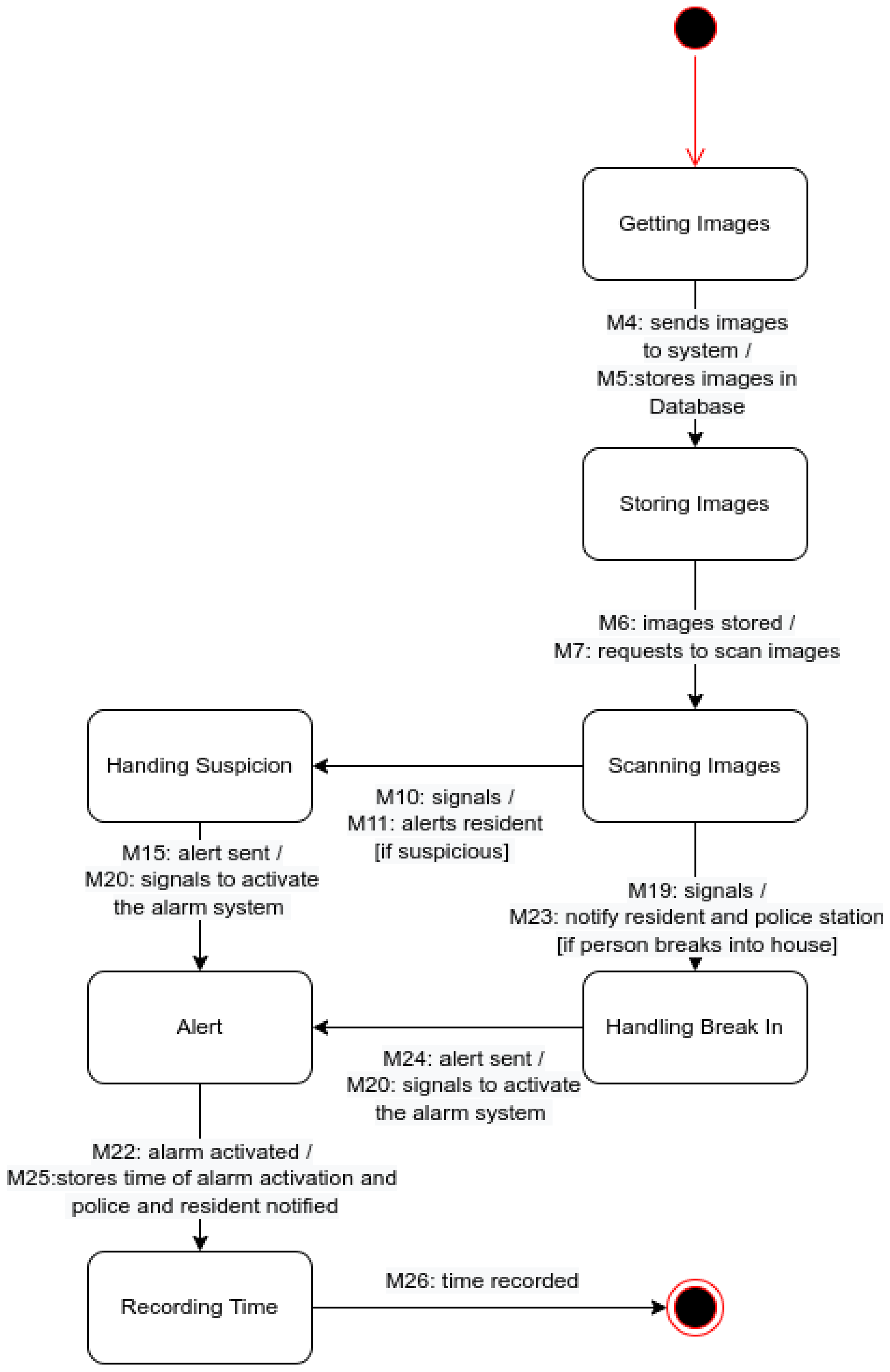


DETECT FIRE



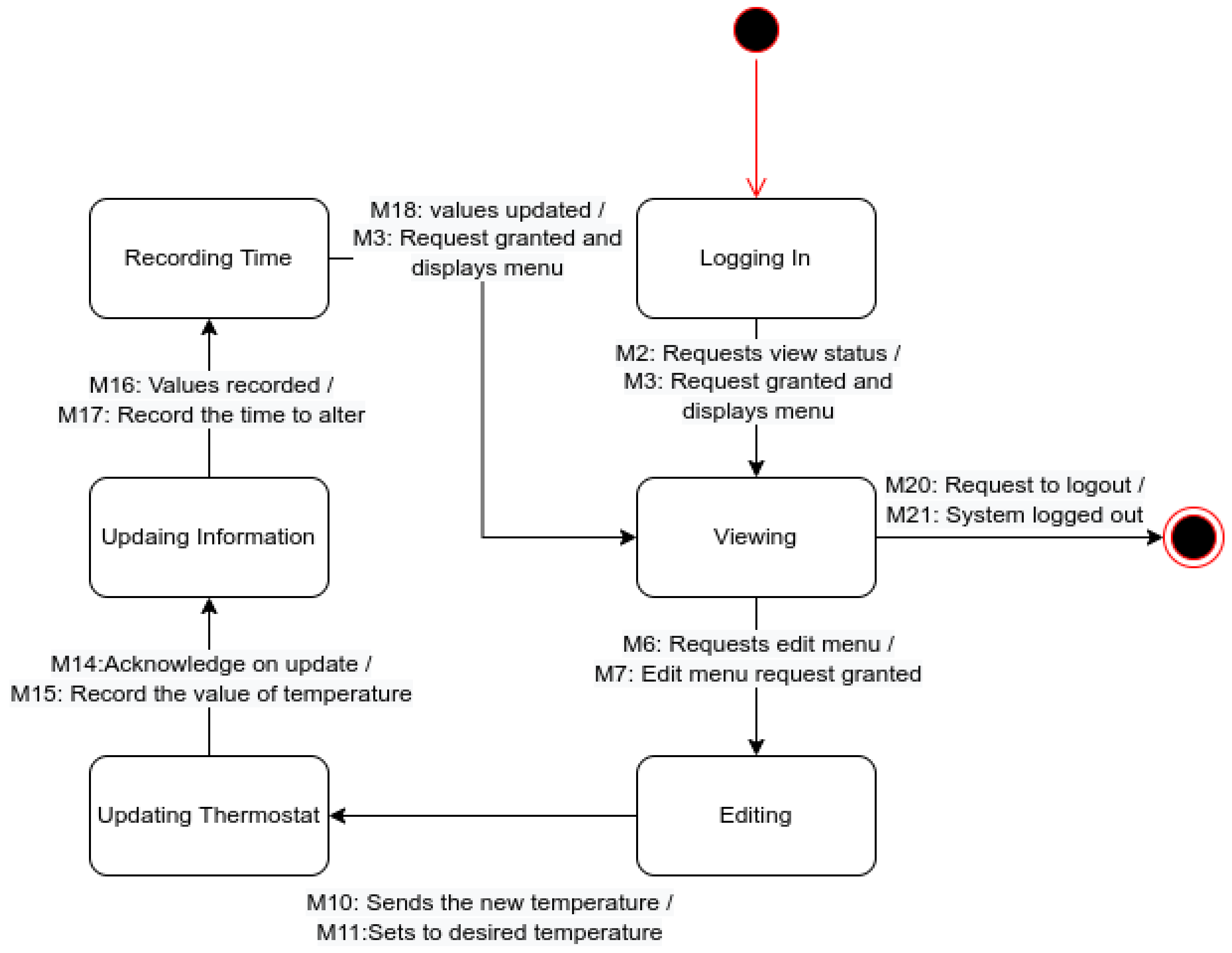
Note: You can click on the diagram to get the clear version of it.

HOME SECURITY

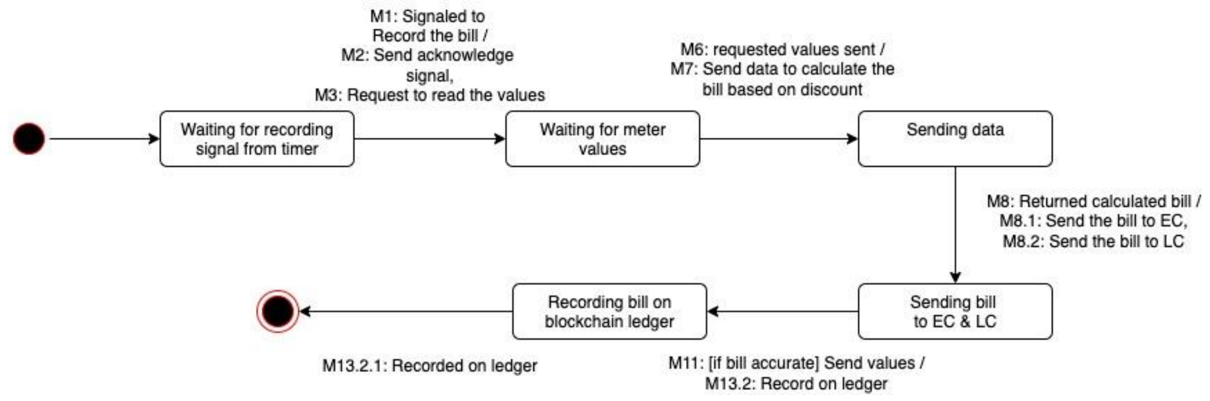

Getting Images
M4: sends images to system / M5:stores images in Database
Storing Images
M6: images stored / M7: requests to scan images
Scanning Images
Handing Suspicion
M10: signals / M11: alerts resident [if suspicious]
M15: alert sent / M20: signals to activate the alarm system
M19: signals / M23: notify resident and police station [if person breaks into house]
Alert
Handling Break In
M24: alert sent / M20: signals to activate the alarm system
M22: alarm activated / M25:stores time of alarm activation and police and resident notified
Recording Time
M26: time recorded

Note: You can click on the diagram to get the clear version of it.

VIEW/CHANGE VALUES



RECORD ELECTRICITY USE AND BILL



Note: You can click on the diagram to get the clear version of it.

Note: You can click on the diagram to get the clear version of it.

Term Project: Phase II

For Phase II of the project, you need to develop a Design Model for the SHS. In particular:

a) Develop an integrated (consolidated) communication diagram(s) showing all the objects and messages (or abstract messages) in the system. (4 pts)

b) Define the subsystem architecture (depicted on a concurrent communication diagram), showing the subsystems and their interactions. Describe the criteria used for subsystem structuring. Define the message communication interfaces between the subsystems. (6 pts)

c) Define the task architecture (depicted on concurrent communication diagrams), showing each subsystem's concurrent tasks and interfaces. Describe the criteria used for task structuring. Define the message communication interfaces between tasks (6 pts)

d) Define the information hiding classes in the system. (4 pts)

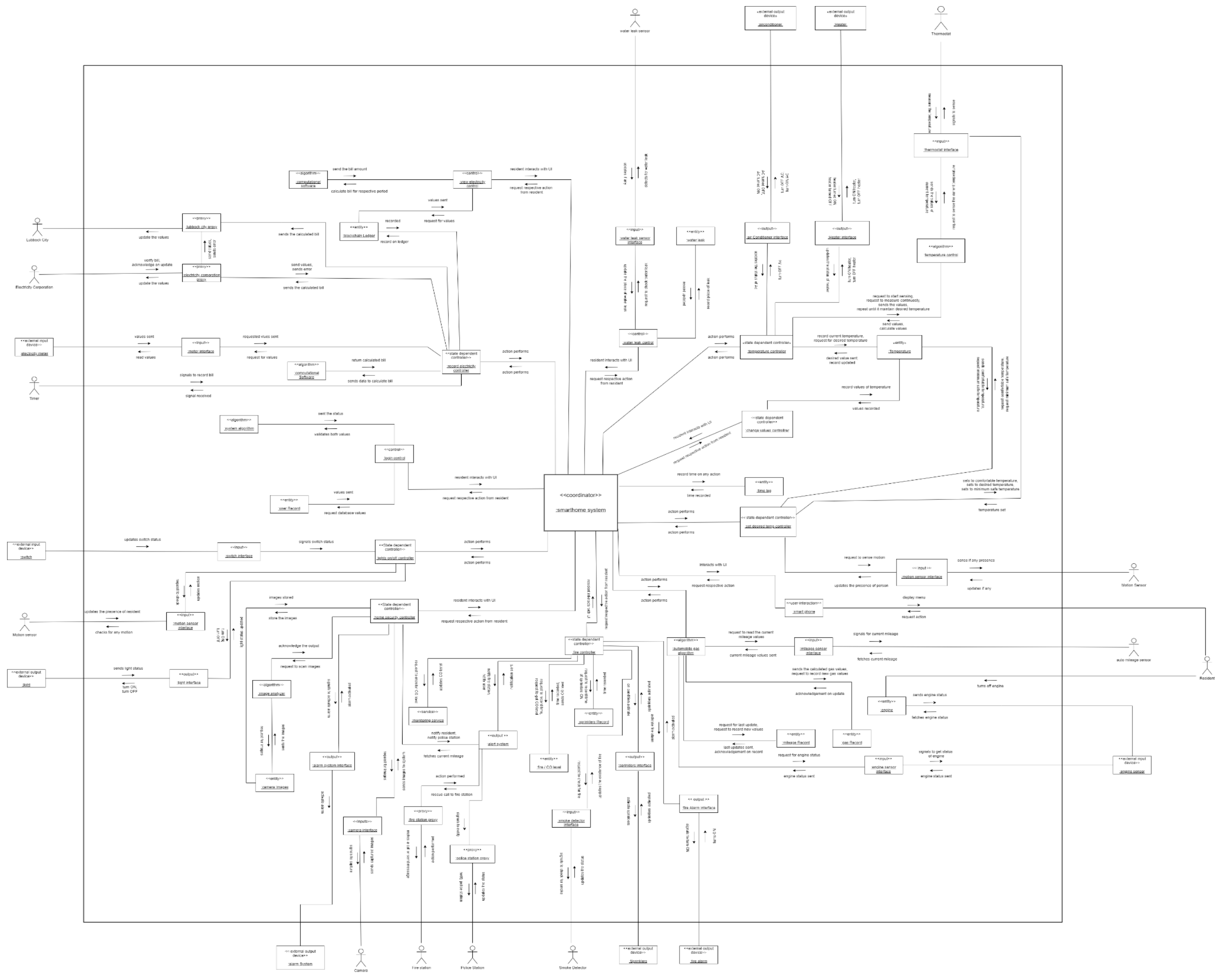
Note: You can click on the diagram to get the clear version of it.

INTEGRATED COMMUNICATION DIAGRAM

ASSUMPTIONS AND THE FLOW:

The integrated communication diagram is the combination of all the individual use cases. All the actors and external input/output devices are kept outside the boundary of the system, as they are not related to the system and they can interact with the system only through user interaction and interfaces assigned to them. System is enclosed in the boundary box where all the operations are held inside it. We assumed the main smart home system as the coordinator and from the coordinator resident can choose different or different use cases to perform the respective action. Each use case has its own flow. Some of the use cases have the state dependent controller object and others have control objects. Every state dependent controller or the control object is assigned to the coordinator object. If the resident sends any message or wants to interact with the system, he interacts through the user interaction smart phone and sends a message to the coordinator, from there coordinator selects the respective option and continues with the flow. As there is a lot of use of time logs<<entity>> and smart phone <<user interaction>> in the system, we have given a direct link to the coordinator and in order to understand it clearly. Each individual use case uses many objects and some use cases share the objects, in such cases we have taken the single object and combined it to all the use cases to which it is required. The flow is resident takes an action and sends a message to the coordinator, it chooses the respective controller or control object and performs it action, if any return message is required in the respective use case it updates it to the respective system like fire station, police station etc.

Note: You can click on the diagram to get the clear version of it.



Note: You can click on the diagram to get the clear version of it.

SUBSYSTEM ARCHITECTURE

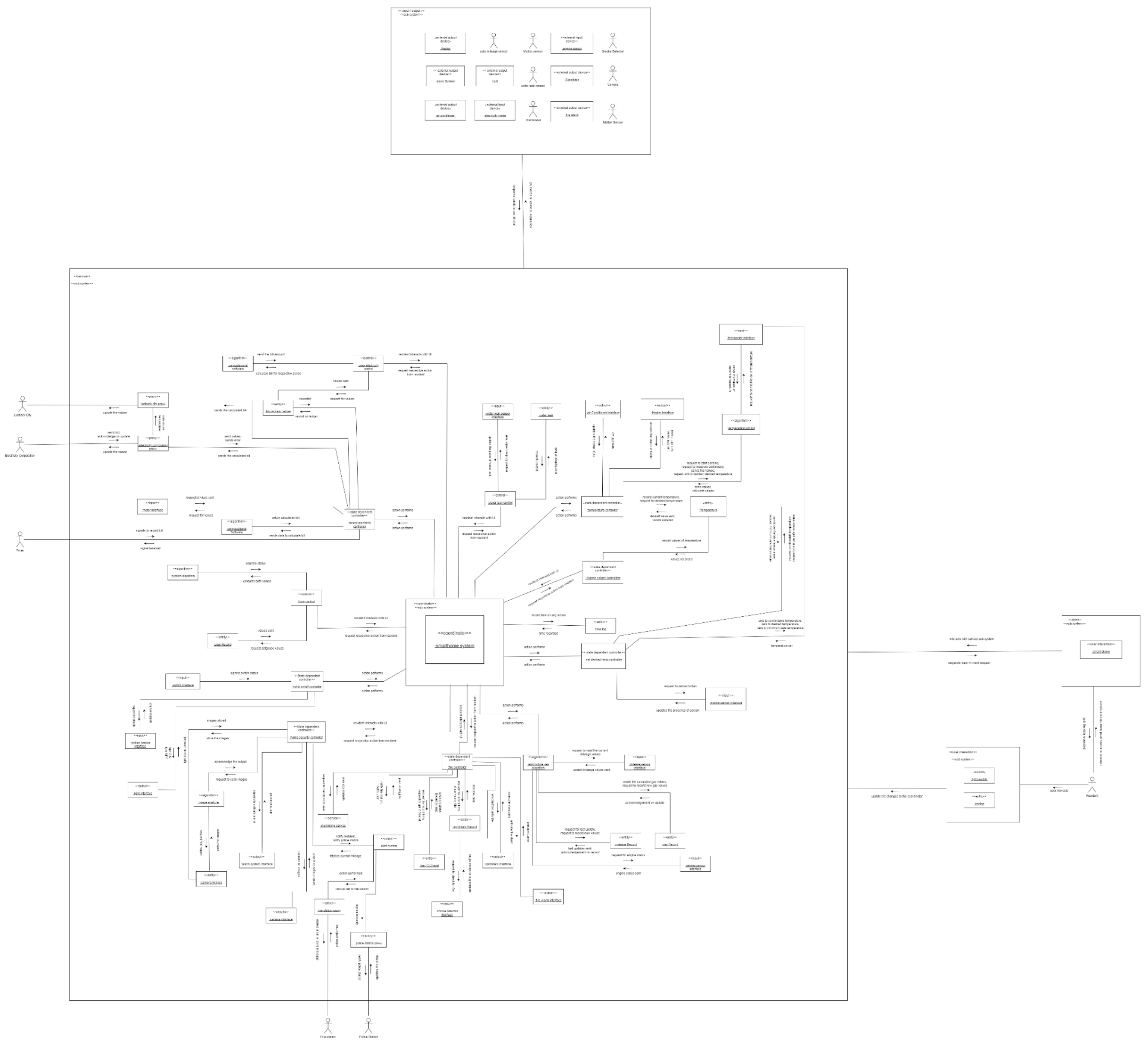
ASSUMPTIONS AND DESCRIPTION

1. After developing the integrated communication diagram the objects which are having the tight bond or perform the same operations could be grouped together into a subsystem.
2. In this smart home system we assumed the whole diagram into four subsystems i.e., I/O subsystem, client subsystem, user interaction subsystem and service subsystem.
3. In the user interaction subsystem we included the object's like light switch and engine because the resident interacts with the switch and the engine directly.
4. In the I/O subsystem we include the objects which perform input output operations like sensors, external input devices and external output devices. These objects perform input output operations on the behalf of other subsystems.
5. We assumed a smartphone as a client subsystem because it requests one or more services from the service subsystem.
6. The use cases which provide a service to the client subsystem could be grouped together and made a service subsystem. The service subsystem provides a service for the client subsystem and it responds to requests from client subsystems and it also provides access to a database which means we are having the entities which store the data. The data could be stored in a database for future access.
7. We assumed a smart home system as a coordinator subsystem because the smart home system coordinates the execution of all the other subsystems such as service subsystems and coordination subsystem decides the execution sequence of multiple service subsystems.

MESSAGE COMMUNICATION INTERFACES BETWEEN THE SUBSYSTEMS

1. The most important thing related to the subsystem is message communication. After the categorization of subsystems we need to define the communication between the subsystems, because we can have a clear idea of how the subsystems are communicating to each other, whether it's a synchronous communication or asynchronous communication or bidirectional message communication.
2. There is a synchronous message communication between the client and service subsystems because the client subsystem requests the services. The client subsystem interacts with the service subsystem and the service subsystem provides services for the client requests.
3. The user interaction subsystem communicates with the service asynchronously because it only updates the values to the service subsystem i.e., updates the status of a switch and engine through interfaces.
4. The coordinator subsystem requests the I/O subsystem to access I/O devices. The I/O subsystem sends the response to the coordinator subsystem. The coordinator subsystem(smart home system) coordinates all the actions which are performed by all the objects. Simply, it executes all the actions coming from every object.

Note: You can click on the diagram to get the clear version of it.



Note: You can click on the diagram to get the clear version of it.

TASK STRUCTURING

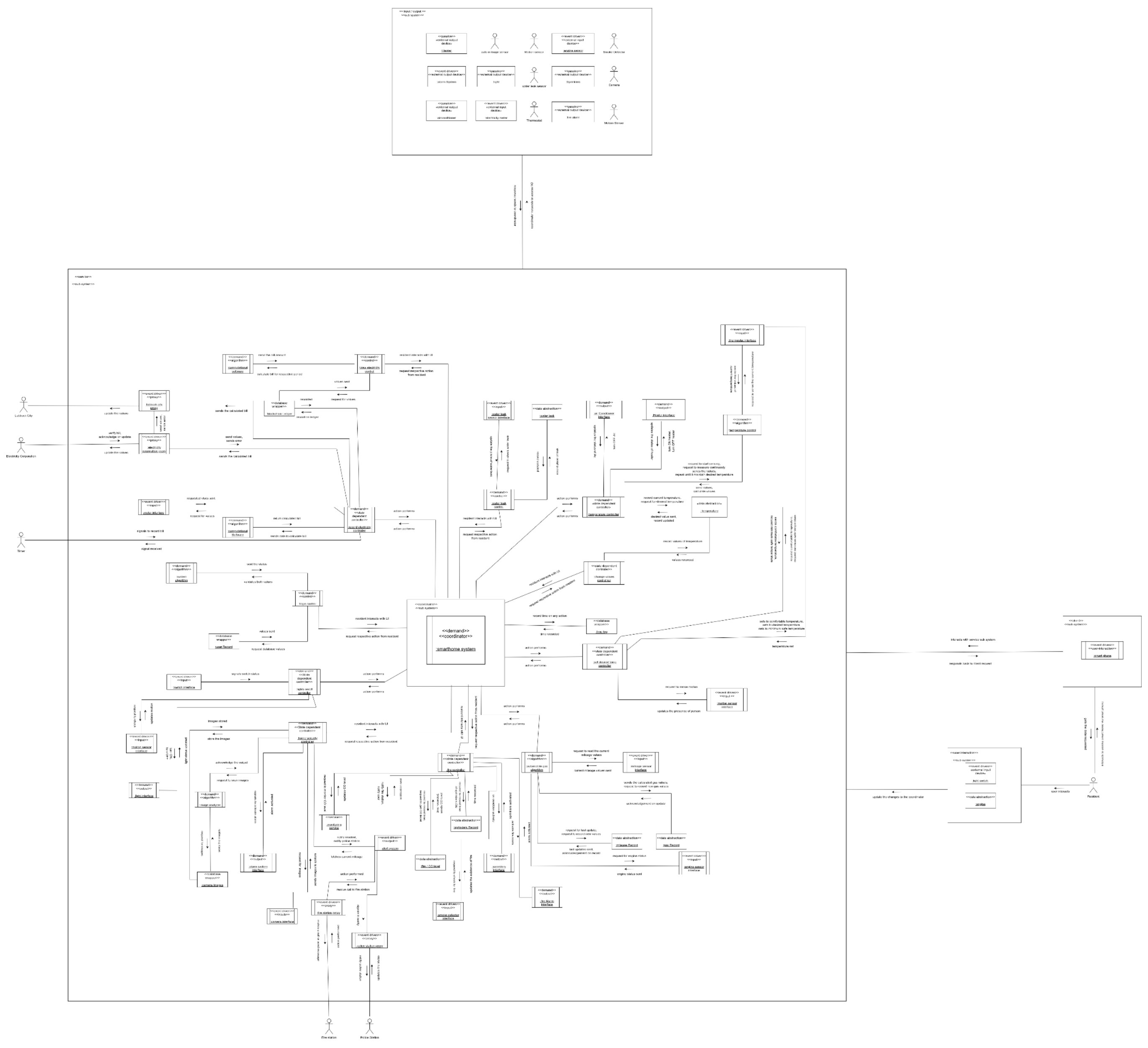
CRITERIA

1. We split the definitions of each task between passive, event driven, and demand.
2. Passive tasks within the system were defined by their use being solely for output. Controlling a passive task was then held by a demand which would send a signal to the passive to activate or deactivate. All of the passives are found in the input / output submodule and receive messages from the smart home coordinator.
3. When a device was declared to provide input and output it was defined as event driven. Any event driven devices send their interrupt to another event driven interface within the system.
4. Lastly, any additional tasks within the system were controllers, coordinators, and algorithms, which all were defined as demands. This is due to their jobs being to control and request the data flow through the entire system.

MESSAGE COMMUNICATION INTERFACES BETWEEN THE SUBSYSTEMS

1. Flow through the system originates from either a controller or an event driven input device. When an event driven device gets an input such as a user interacting with the system through their smartphone. It will send its signal to an interface and into the given controller for that usecase. If further information is required it will send its signal further to the coordinator to interact with other parts of the system.
2. During certain use cases such as heating and cooling a room the controller for that use case is also able to request information from event driven input tasks or send signals out to passive output tasks. This happens in the case when the temperature is requested and then a following signal is sent to either the AC or the heater to turn the device on or off.
3. When an algorithm is activated it is also able to send a request to a data entity in order to gain any information it may need to complete its computation.

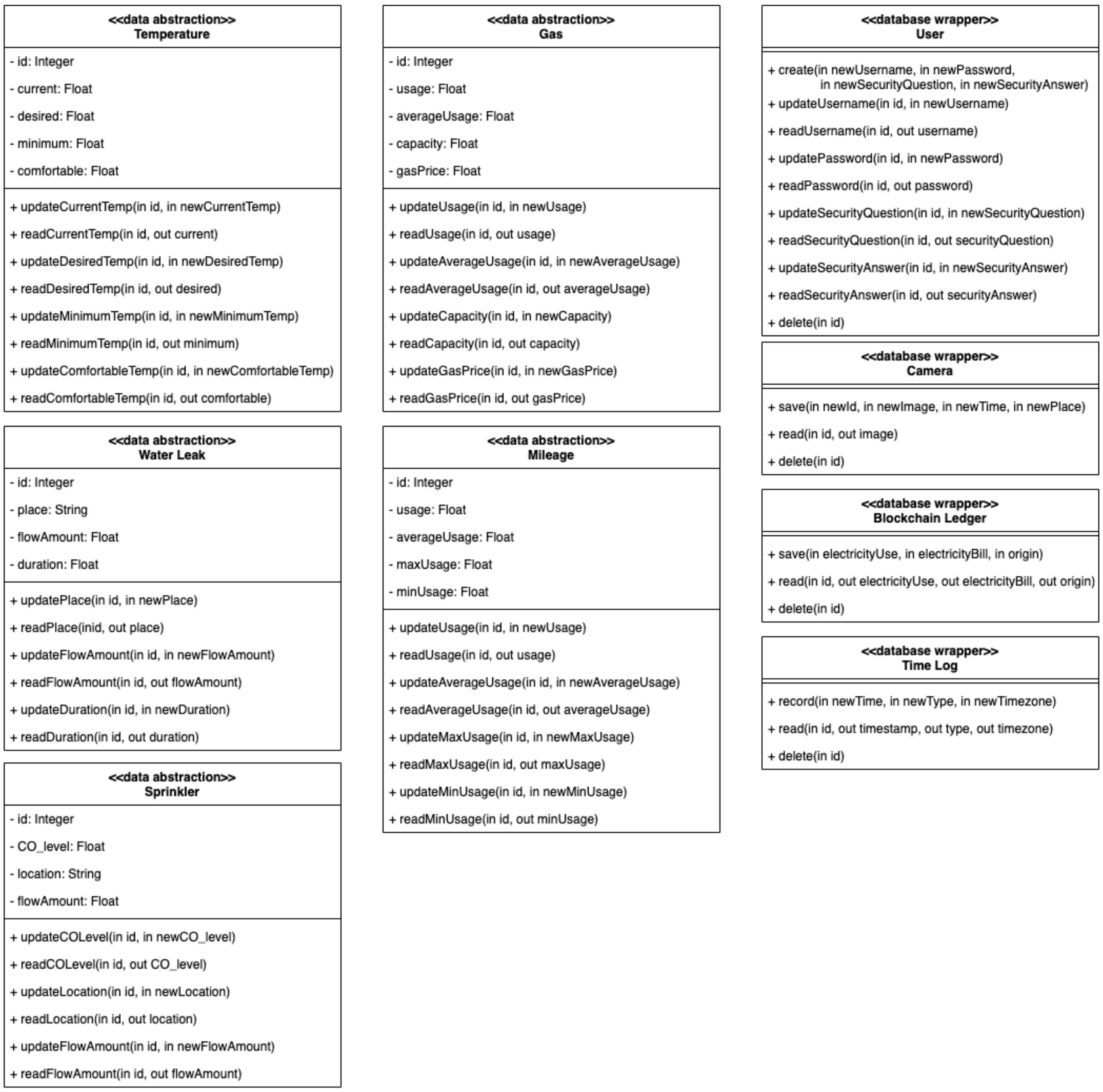
Note: You can click on the diagram to get the clear version of it.



Note: You can click on the diagram to get the clear version of it.

INFORMATION HIDING

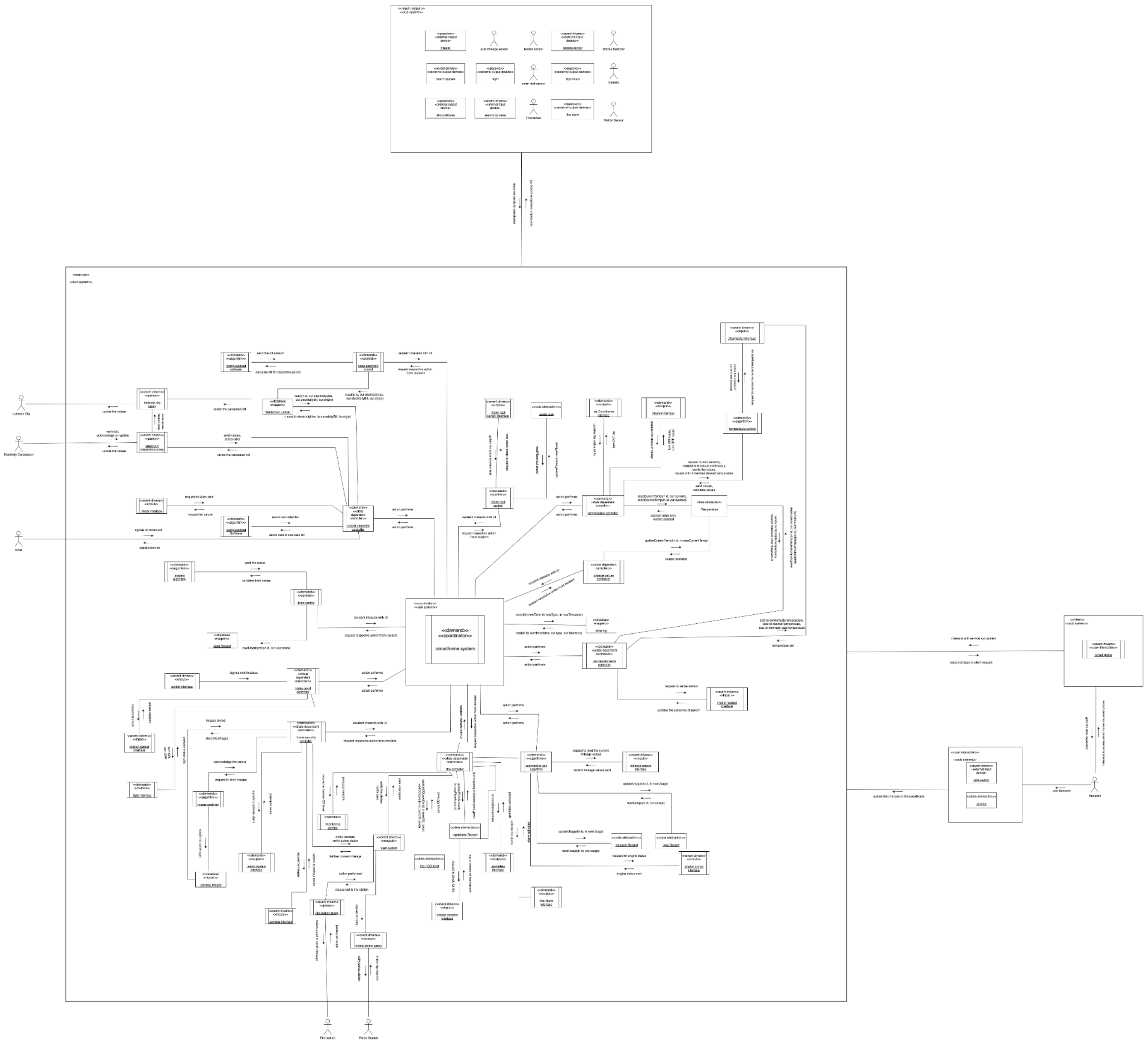
CLASSES



```
<<data abstraction>>
Temperature
- id: Integer
- current: Float
- desired: Float
- minimum: Float
- comfortable: Float
+ updateCurrentTemp(in id, in newCurrentTemp)
+ readCurrentTemp(in id, out current)
+ updateDesiredTemp(in id, in newDesiredTemp)
+ readDesiredTemp(in id, out desired)
+ updateMinimumTemp(in id, in newMinimumTemp)
+ readMinimumTemp(in id, out minimum)
+ updateComfortableTemp(in id, in newComfortableTemp)
+ readComfortableTemp(in id, out comfortable)

<<data abstraction>>
Water Leak
- id: Integer
- place: String
- flowAmount: Float
- duration: Float
+ updatePlace(in id, in newPlace)
+ readPlace(inid, out place)
+ updateFlowAmount(in id, in newFlowAmount)
+ readFlowAmount(in id, out flowAmount)
+ updateDuration(in id, in newDuration)
+ readDuration(in id, out duration)

<<data abstraction>>
Sprinkler
- id: Integer
- CO_level: Float
- location: String
- flowAmount: Float
+ updateCOLevel(in id, in newCO_level)
+ readCOLevel(in id, out CO_level)
+ updateLocation(in id, in newLocation)
+ readLocation(in id, out location)
+ updateFlowAmount(in id, in newFlowAmount)
+ readFlowAmount(in id, out flowAmount)

<<data abstraction>>
Gas
- id: Integer
- usage: Float
- averageUsage: Float
- capacity: Float
- gasPrice: Float
+ updateUsage(in id, in newUsage)
+ readUsage(in id, out usage)
+ updateAverageUsage(in id, in newAverageUsage)
+ readAverageUsage(in id, out averageUsage)
+ updateCapacity(in id, in newCapacity)
+ readCapacity(in id, out capacity)
+ updateGasPrice(in id, in newGasPrice)
+ readGasPrice(in id, out gasPrice)

<<data abstraction>>
Mileage
- id: Integer
- usage: Float
- averageUsage: Float
- maxUsage: Float
- minUsage: Float
+ updateUsage(in id, in newUsage)
+ readUsage(in id, out usage)
+ updateAverageUsage(in id, in newAverageUsage)
+ readAverageUsage(in id, out averageUsage)
+ updateMaxUsage(in id, in newMaxUsage)
+ readMaxUsage(in id, out maxUsage)
+ updateMinUsage(in id, in newMinUsage)
+ readMinUsage(in id, out minUsage)

<<database wrapper>>
User
+ create(in newUsername, in newPassword,
         in newSecurityQuestion, in newSecurityAnswer)
+ updateUsername(in id, in newUsername)
+ readUsername(in id, out username)
+ updatePassword(in id, in newPassword)
+ readPassword(in id, out password)
+ updateSecurityQuestion(in id, in newSecurityQuestion)
+ readSecurityQuestion(in id, out securityQuestion)
+ updateSecurityAnswer(in id, in newSecurityAnswer)
+ readSecurityAnswer(in id, out securityAnswer)
+ delete(in id)

<<database wrapper>>
Camera
+ save(in newId, in newImage, in newTime, in newPlace)
+ read(in id, out image)
+ delete(in id)

<<database wrapper>>
Blockchain Ledger
+ save(in electricityUse, in electricityBill, in origin)
+ read(in id, out electricityUse, out electricityBill, out origin)
+ delete(in id)

<<database wrapper>>
Time Log
+ record(in newTime, in newType, in newTimezone)
+ read(in id, out timestamp, out type, out timezone)
+ delete(in id)
```

Note: You can click on the diagram to get the clear version of it.

DIAGRAM



Note: You can click on the diagram to get the clear version of it.