Protecc

Brett Walton – Architect – bmw0045@auburn.edu
Christopher Burger – Requirements Engineer – cdb0076@auburn.edu
Christopher Moorhead – Project Manager – cam0071@auburn.edu
Collins Hess – Designer – cmh0098@auburn.edu
Grant Diamant – Prototype Designer – gmd0012@auburn.edu
Hunter Donald – QA Engineer – hzd0011@auburn.edu

Table of Contents

Phase I

Phase II

Phase I

Domain Analysis

Concept Statement

The Protecc Command Center is a home security system that permits the home-owner to monitor the exterior and interior of their property either directly through a PC or remotely via the internet. Command Center incorporates various devices that a home-owner can install in order to ensure the prevention of theft and the safety of their family. Among these devices are external cameras, smoke and gas leak detectors (these will always be on), motion sensors for doors and windows, and motorized locks for doors and windows. Command Center offers two different armed modes: home and away.

Home mode is meant to be used when the house is occupied. Away mode is meant to be used when the homeowner leaves the house for any amount of time. Each mode specifies the appropriate emergency service for a situation. If the mode is set to home and an alarm is triggered, then the emergency services are called and know to expect people at home and should be prepared to protect them. In away mode, if the home owner is returning home or if they are expecting a visitor while they are gone, then they have a few seconds to enter the correct password on the control panel inside the home or through the online application to disable the alarm before emergency services are called. Emergency services should expect that somebody is breaking in if they are contacted in away mode. Therefore, the only difference between home and away mode is that emergency services are called immediately if an alarm is triggered in home mode, but a few seconds are given before emergency services are called in away mode to allow time for deactivation.

The cameras are installed around home entry points and provide constant surveillance. The Command Center allows direct access to any of the camera feeds for the home-owner. When cameras detect motion**,** they immediately notify the user and send a picture of the intrusion.

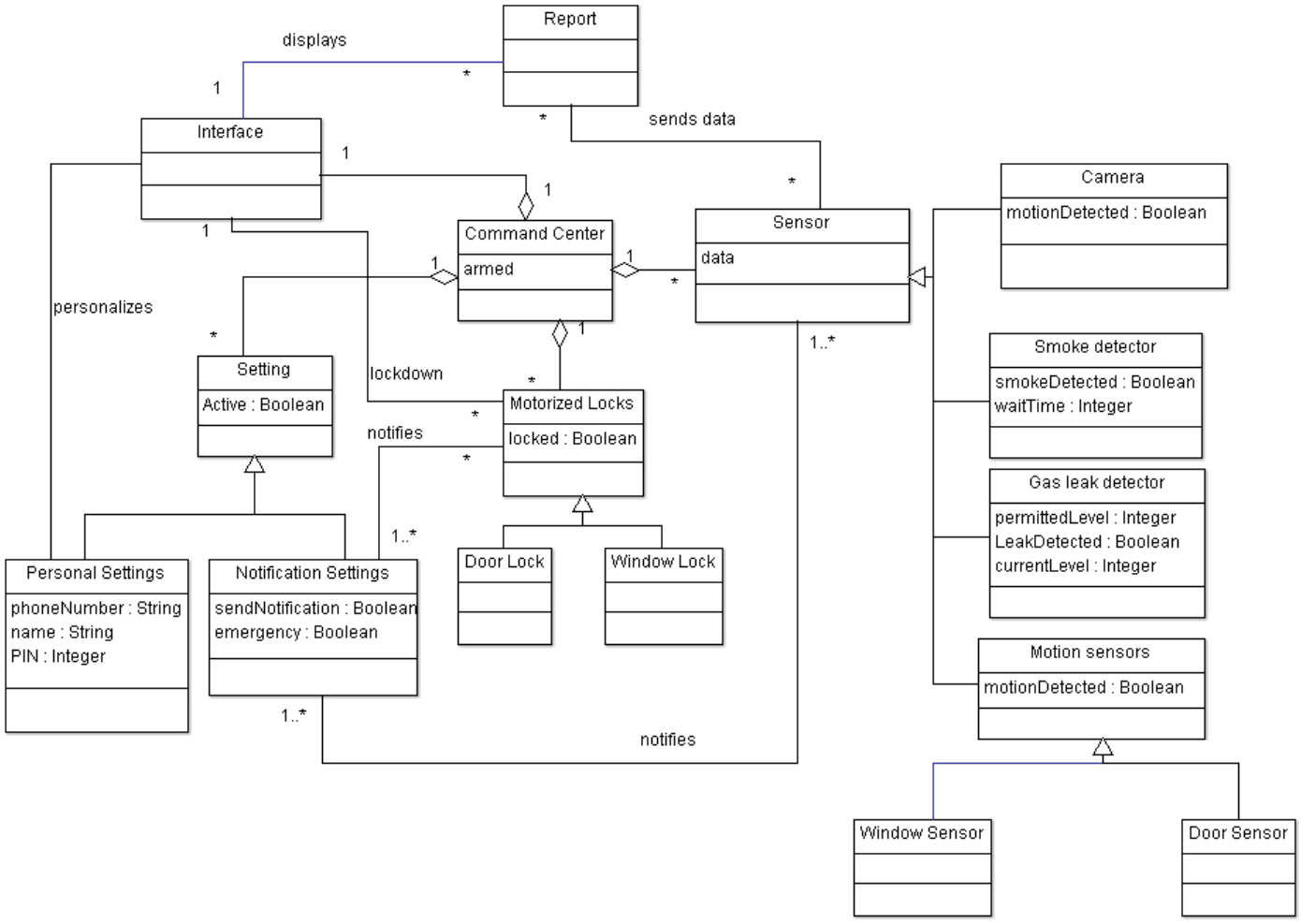
Command Center utilizes motion sensors which detect if a door or window is open or if there is movement coming through either. Abnormal readings for these sensors default to sending notifications to the user through system notifications. If the house is in home or away mode, these motion sensors will be armed.

The Command Center also includes motorized locks for doors and windows. Through the Command Center interface, the home-owner can see what doors and windows are locked or unlocked. In away and home mode, all doors and windows are locked. If the house is disarmed, the homeowner can still choose to do any of the following: lock them all through a “Lock” command, unlock them all through an “Unlock” command, or lock/unlock them all individually. The doors can be manually unlocked as well without the application but the home-owner will be notified if any doors or windows open without using the application.

The Command Center application gives the home-owner control of what they see and when they see it. Upon entering the application, the home-owner is shown the most recent and important notifications. If there are images associated with the notification, such as via the video feed, then the home-owner can easily expand the notification to view it. From the main menu, the home-owner can access settings or view a live feed of sensor data.

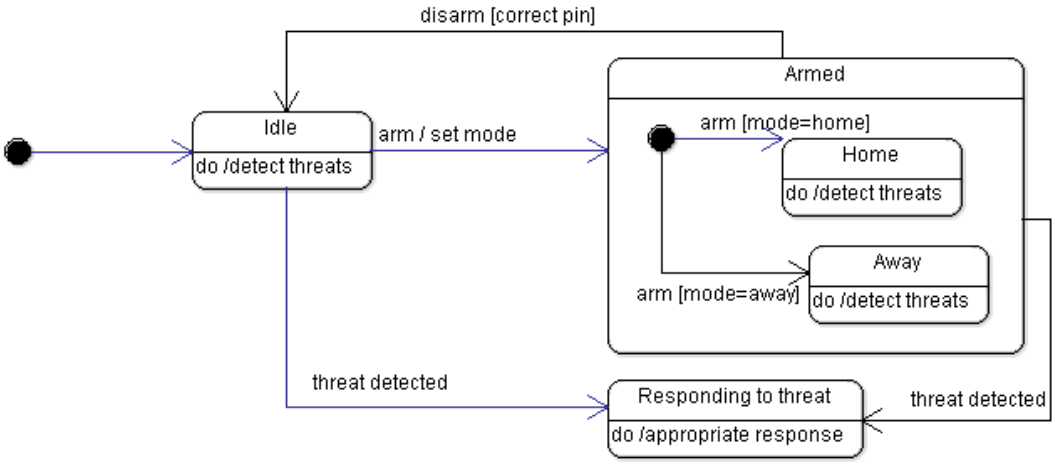
There are many settings within the Command Center that the home-owner can adjust. The primary setting is how they want to receive notifications and which ones they prioritize. Similarly, the home-owner can set up the system to automatically notify the authorities for specific sensor readings. If a gas leak is detected or a smoke alarm stays on for a certain duration of time, then the fire department can be alerted. Or if a person is detected attempting to break into the home then the police department can be immediately notified.

Concept Domain Model

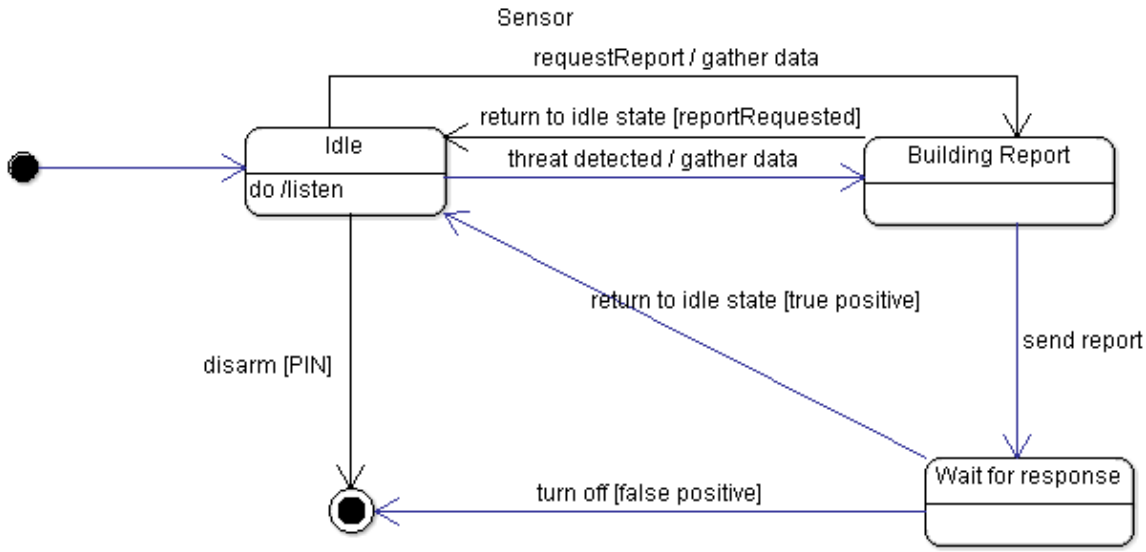


Domain State Model

System



Sensor



Application Analysis

Application Interaction Analysis

Use Cases

Phase II

Use Case: View Live Feed

Essential Use Case:

Participants: Home-owner

Pre-conditions: Home-owner has devices set up.

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Access application	
	2. Show interface
3. Choose option to view feed	
	4. List devices with feeds to show
5. Select device	
	6. Show live feed of device
7. View feed	
8. Exit feed	
	9. List devices with feeds to show

Exceptions:

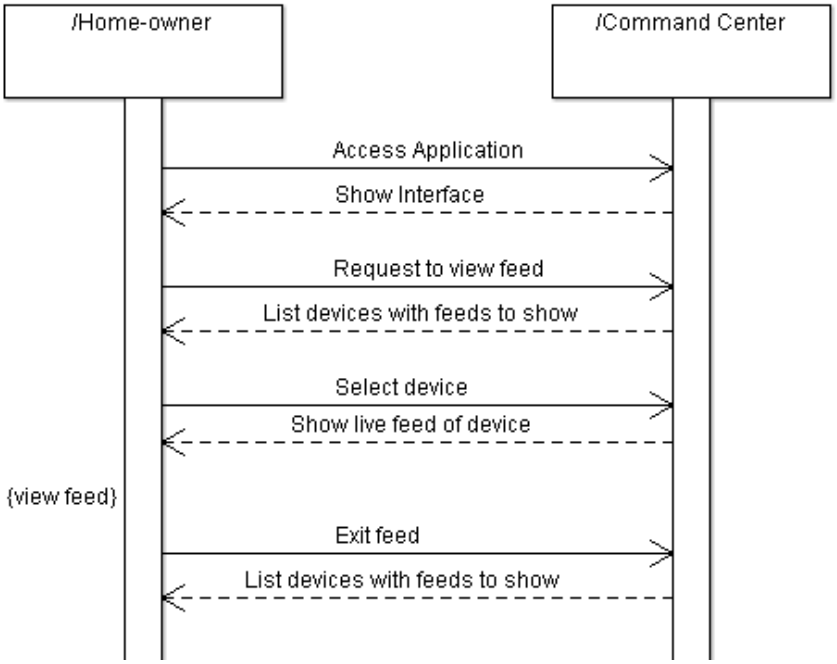
Step 4: No devices to show. Show error message and return to initial interface.

Post-condition: Home-owner has viewed feed (if available) and exited the feed.

Scenario:

Home-owner access application.
System shows interface.
Home-owner chooses to view live feed.
System shows a list of devices with feeds to show.
Home-owner selects camera feed.
System shows the live camera footage.
Home-owner views footage.
Home-owner exits feed.
System returns to list of devices.

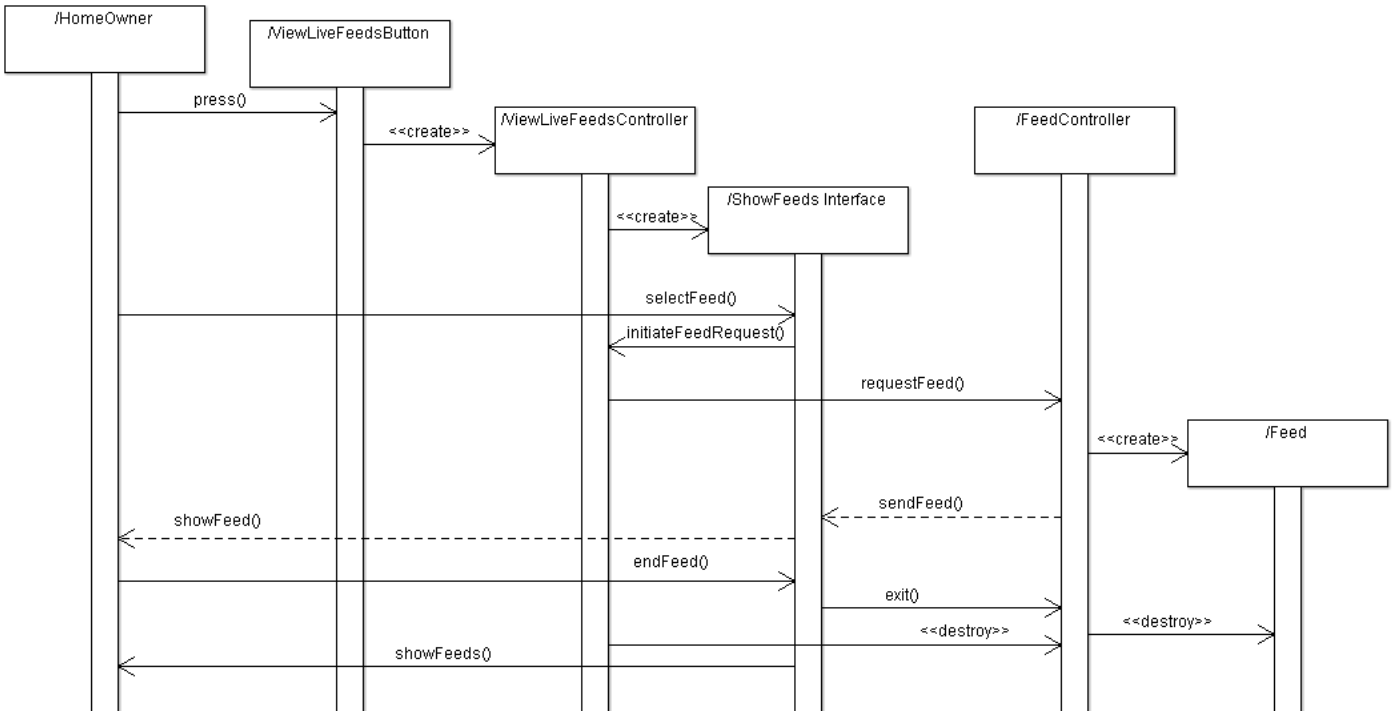
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	ViewLiveFeed
Entry condition	1. The HomeOwner accesses the CommandCenter and presses the "View Live Feeds" button.
Flow of events	2. The CommandCenter displays an interface that lists the available devices to view the live feeds for. There is also an option to just view all current readings. 3. The HomeOwner selects the devices that they would like to view the feed for by pressing its respective button. The device's live results are then accessed by the CommandCenter. 4. The CommandCenter displays an interface that shows the live feed for the device. If the device is a camera, then the camera feed is shown. If it is a sensor monitoring some levels, it shows the current reading, expected reading, and a graph showing the level variation over the last few hours. If the user chooses the option to view all current readings, a similar display to this is shown that includes the data for all of the monitoring sensors.
Exit condition	5. The HomeOwner exits the live feed.

Detailed System Sequence Diagram:



Use Case: Contact Emergency Services

Essential Use Case:

Participants: Home-owner, Emergency Services

Pre-conditions: System sends a notification to the home-owner about a potential reason to contact emergency services.

Typical Course of Events:

Actor Intention	**System Responsibility**
	1. A sensor detects an issue that requires attention.
	2. Notify the home-owner of the problem and recommend action.
3. Review the notification.	
4. Tell system to contact emergency services.	
	5. Contact emergency services and send a report to assist them.
	6. Tell home-owner help is on the way.

Exceptions:

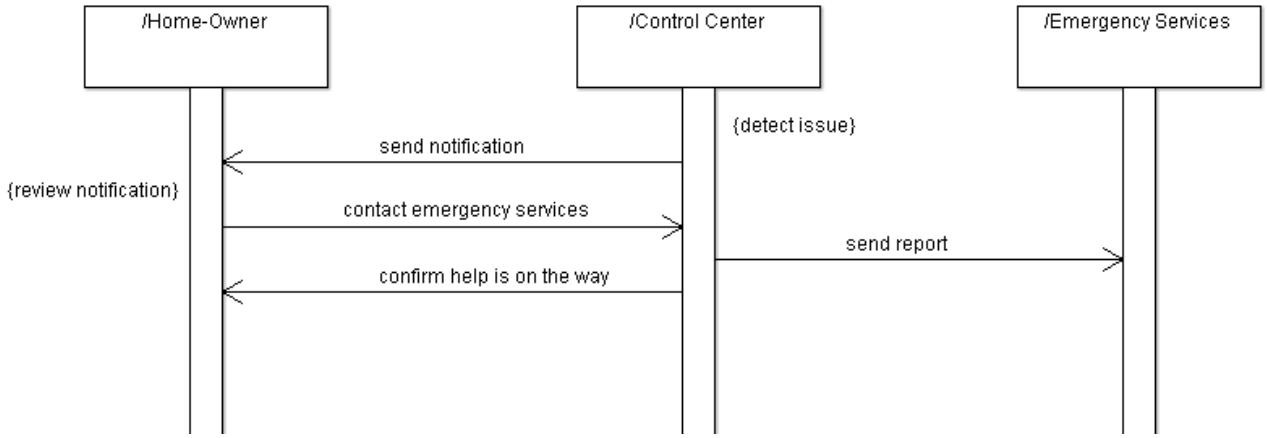
- **Step 4:** Decide no actions are required. Do not contact emergency services. Skip step 5-6.
- **Step 5:** Fail to contact emergency services. Tell home-owner.

Post-condition: Emergency services have been called.

Scenarios:

System detects an issue that requires attention. System notifies home-owner of the problem and delivers a recommended action. Home-owner reviews the notification. Home-owner tells the system to contact emergency services. System contacts emergency services and sends a report to assist them. System confirms to home-owner that help is on the way.

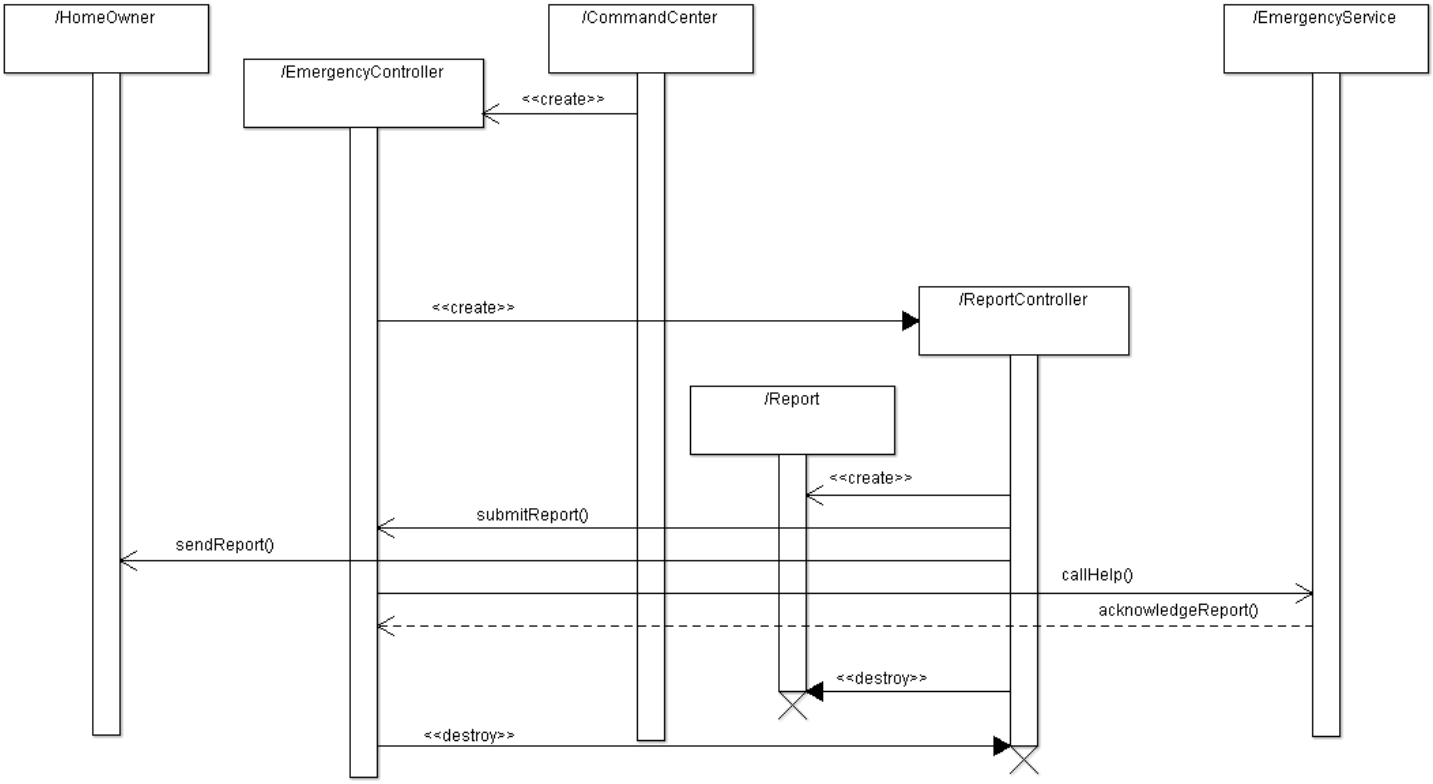
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	ContactEmergencyServices
Entry condition	1. The CommandCenter identifies an issue that may require emergency services to be called.
Flow of events	2. The CommandCenter notifies the HomeOwner of the issues that it identified. 3. The HomeOwner reviews the issue and decides whether to call for help. Typically, the HomeOwner will review the issue and decide that emergency services should be called. 4. The CommandCenter contacts EmergencyService and sends an EmergencyReport that lists the issue and provides as much assistance to the EmergencyService as needed. The EmergencyService then confirms that help is being sent to CommandCenter. 5. CommandCenter notifies HomeOwner that help is on the way.
Exit condition	6. EmergencyService is on the way.

Detailed System Sequence Diagram:



Use Case: Lockdown

Essential Use Case:

Participants: Home-owner

Pre-conditions: Mechanical locks installed.

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Access application.	
	2. Display interface.
3. Initiate lockdown.	
	4. Activate mechanical locks.
	5. Once locks have stopped, check that all entries have successfully locked.
	6. All entries are locked, success notification sent to home-owner.
7. Exit application.	

Exceptions:

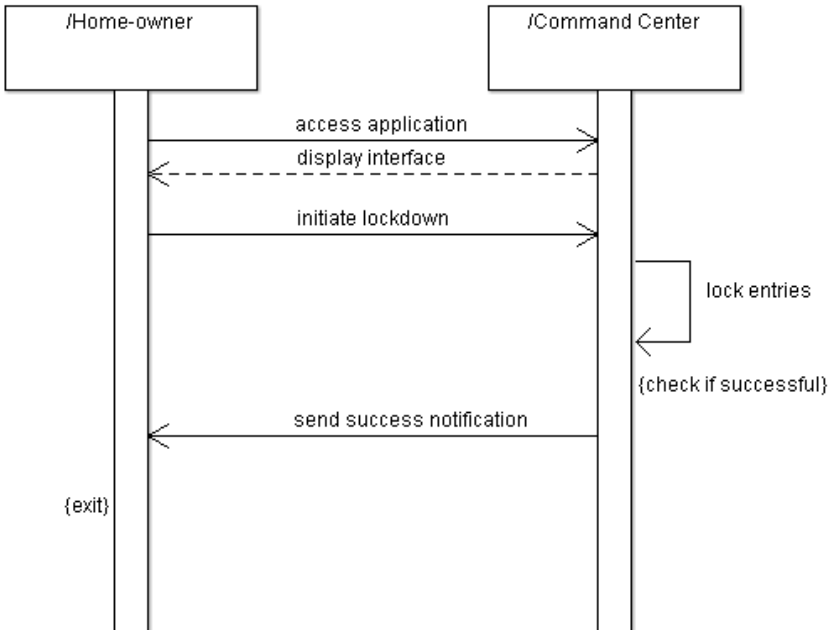
- **Step 5:** An entry failed to lock. The process stops and the home-owner is notified of which entry was not successfully locked.

Post-condition: Report saved.

Scenario:

Home-owner accesses application. System displays main interface. Home-owner initiates lockdown. System activates all mechanical locks not currently in a locked state. Once locks have stopped, system confirms that the entries have successfully locked. System sends home-owner a confirmation message. Home-owner exits application.

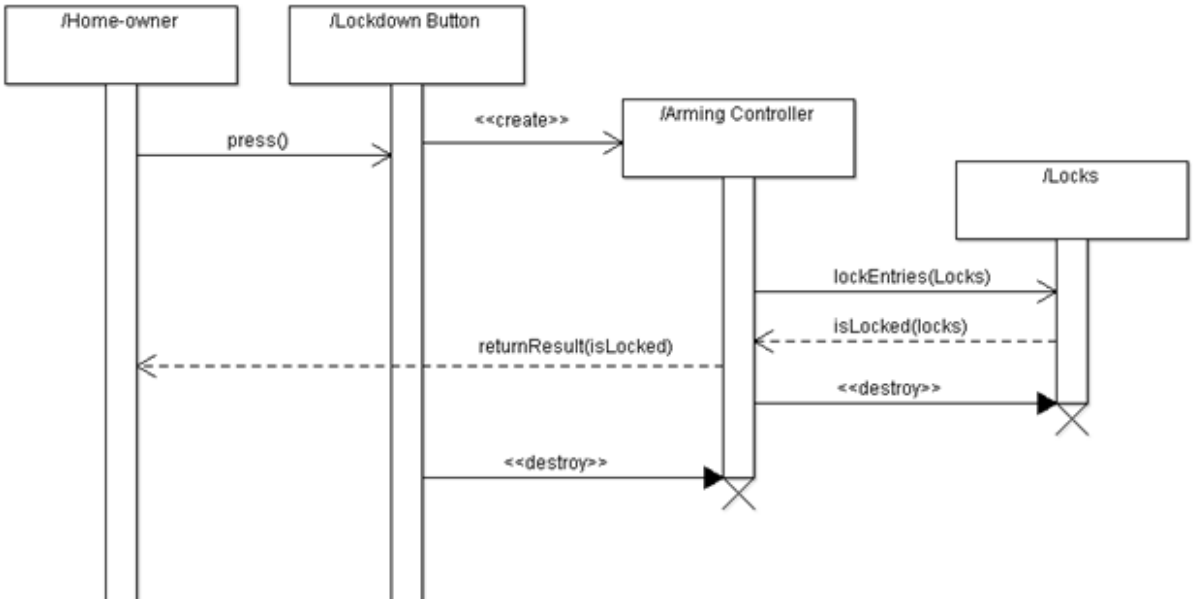
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	Lockdown
Entry condition	1. HomeOwner initiates lockdown.
Flow of events	2. CommandCenter initiates lockdown by activating the mechanical locks on the entries. 3. CommandCenter waits until the locks stop moving. 4. CommandCenter checks the status of all the locks. 5. All entries are confirmed as locked.
Exit condition	6. CommandCenter sends a success message to HomeOwner.

Detailed System Sequence Diagram:



Use Case: View Report

Essential Use Case:

Participants: Home-owner (User)

Pre-conditions: Protecc system is running

Typical Course of Events:

Actor Intention	System Responsibility
1. Access application.	
	2. Display Interface
3. User selects View Report	
	4. Notification Report is displayed
5. Review the report log	
6. Exit Report	

Exceptions:

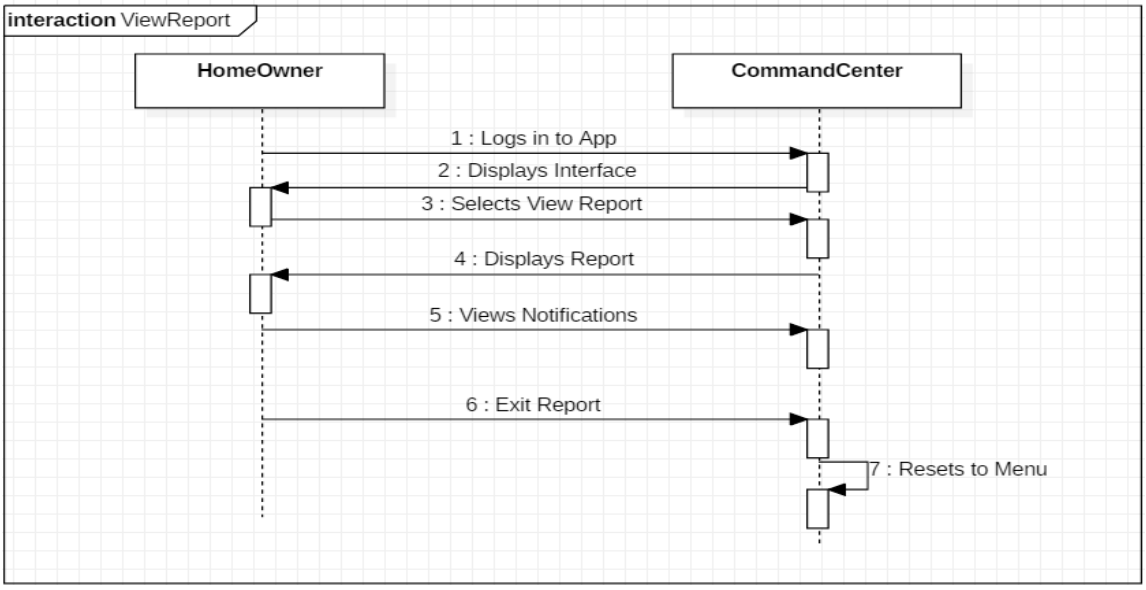
- **Step 5:** User selects notification, where it can be seen before exiting back to report.

Post-condition: User has viewed and exited report log.

Scenario

User logs in to application.
The Command Center pulls up the interface and options.
User chooses to view report.
The Command Center displays the report for the user.
The User views and selects any notifications they deem important.
The User selects to exit the report.
The Command Center exits to the main menu screen.

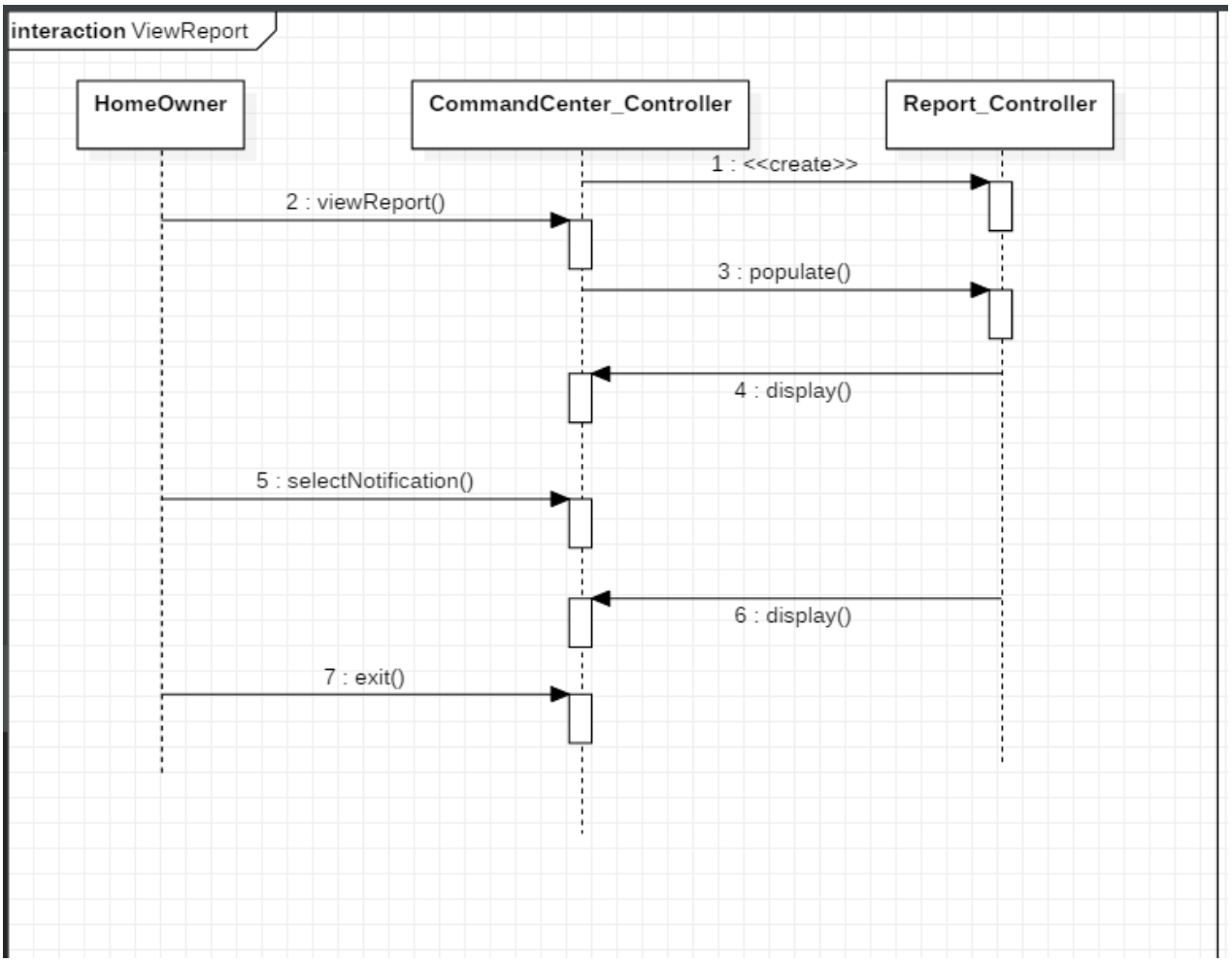
High Level System Sequence Diagram



Concrete Use Case:

Use case name	View Report
Entry condition	1. A user is logged in to the system
Flow of events	2. The Command Center displays the interface to the user 3. The User selects the option to View Report 4. The Command Center displays a report of notifications 5. The User views the notifications as requested 6. The User selects to exit the report
Exit condition	7. The Command Center displays the main menu

Detailed System Sequence Diagram:



Use Case: Trigger Alarm

Essential Use Case:

Participants: Homeowner, Command Center

Pre-conditions: The system is armed.

Actor Intentions	**System Responsibility**
	1. A sensor detects an issue that requires attention.
	2. Determines system is in away mode.
	3. Responds with correct alarm timing
	4. Asks for safety verification
5. Enter safety verification	
	6. Contact Emergency Services

Exceptions:

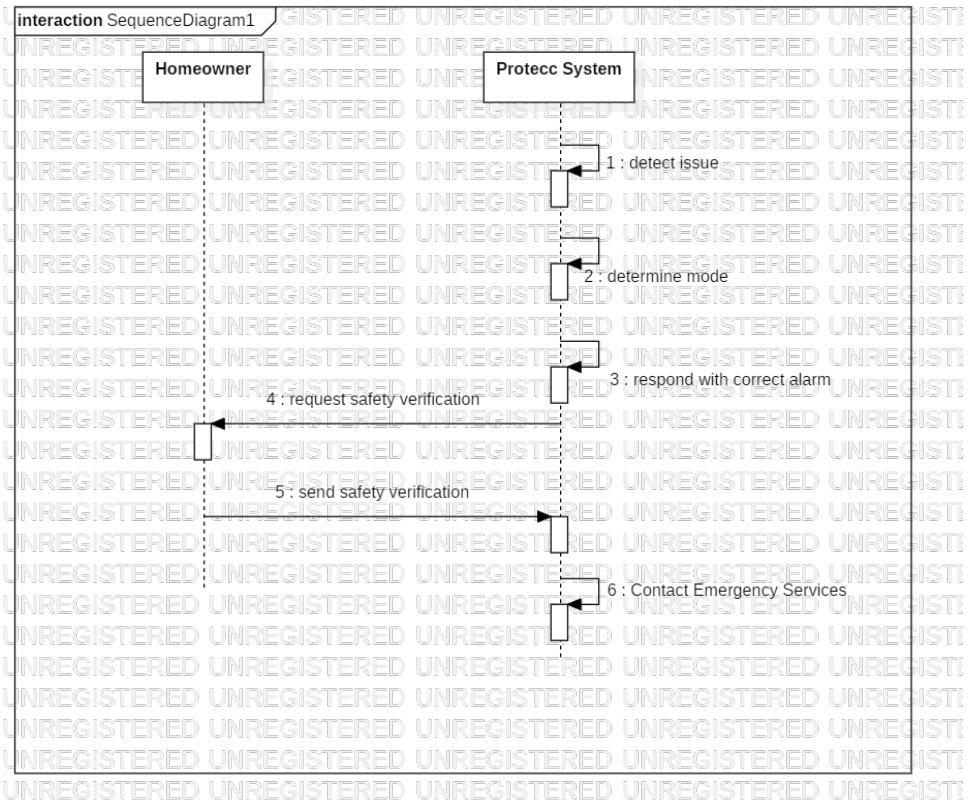
2. Determines system is in home mode and immediately triggers the alarm.
5. The user does not enter the correct verification and the alarm is immediately triggered.

Post Condition: The alarm is correctly triggered.

Scenario:

The system detects an issue from a sensor and begins to trigger the alarm. If in away mode, the user will have a short amount of time to enter their information before the alarm is triggered. Otherwise, the alarm is triggered, and the Contact Emergency Services sequence begins.

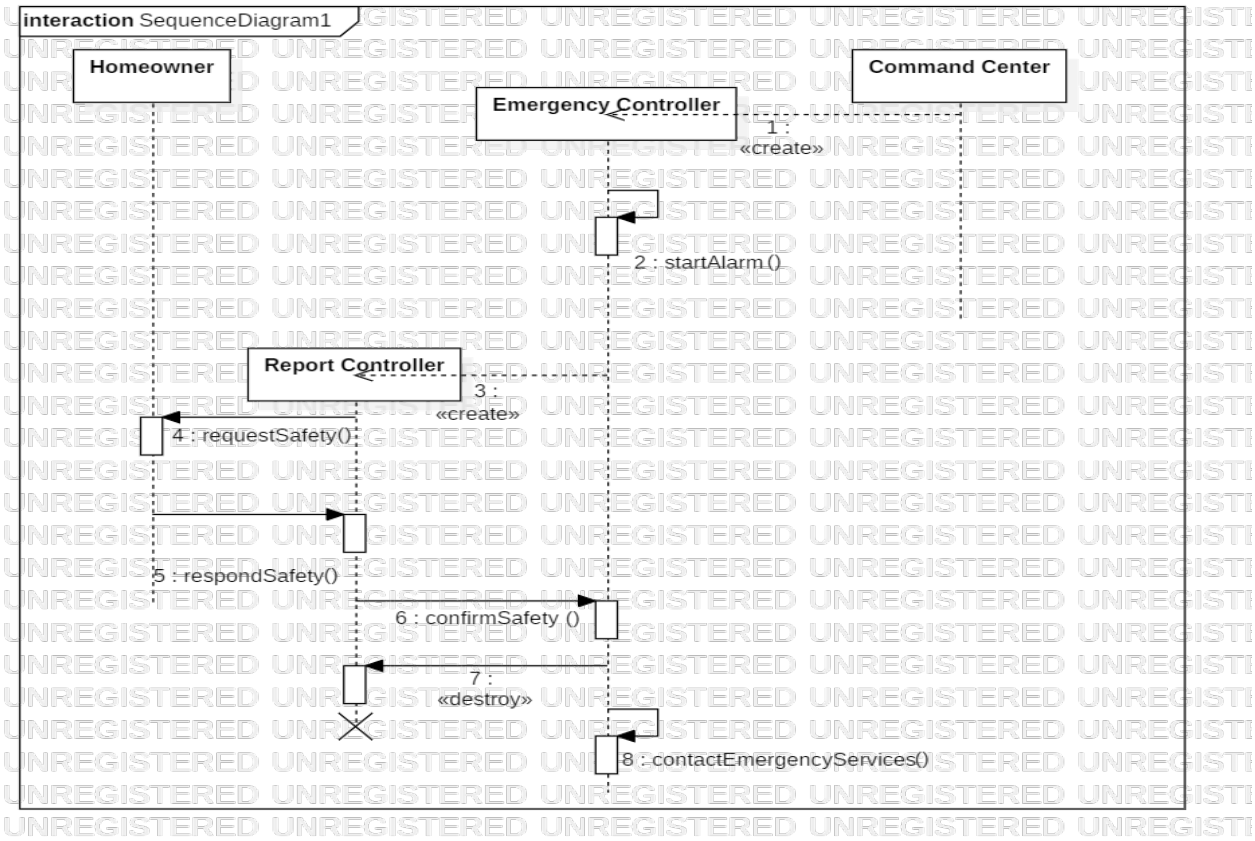
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	Trigger Alarm
Entry condition	1. A detector is tripped while the system is activated.
Flow of events	2. The CommandCenter determines that the system is armed and releases a warning that the alarm is about to be triggered. 3. The system asks for the homeowner's personal information to verify the homeowner is safe and not in danger 4. The homeowner does not enter their correct information within the allotted amount of time. 5. The system triggers the alarm and waits for further instructions.
Exit condition	6. The system starts the Contact Emergency Services sequence.

Detailed System Sequence Diagram:



Use Case: Turn off Alarm

Essential Use Case:

Participants: Homeowner, Command Center

Pre-conditions: The alarm is triggering.

Actor Intentions	**System Responsibility**
	Displays current alarm status.
3. Replies that the home is safe.	Requests verification of safety
	Alarm is deactivated.

Exceptions:

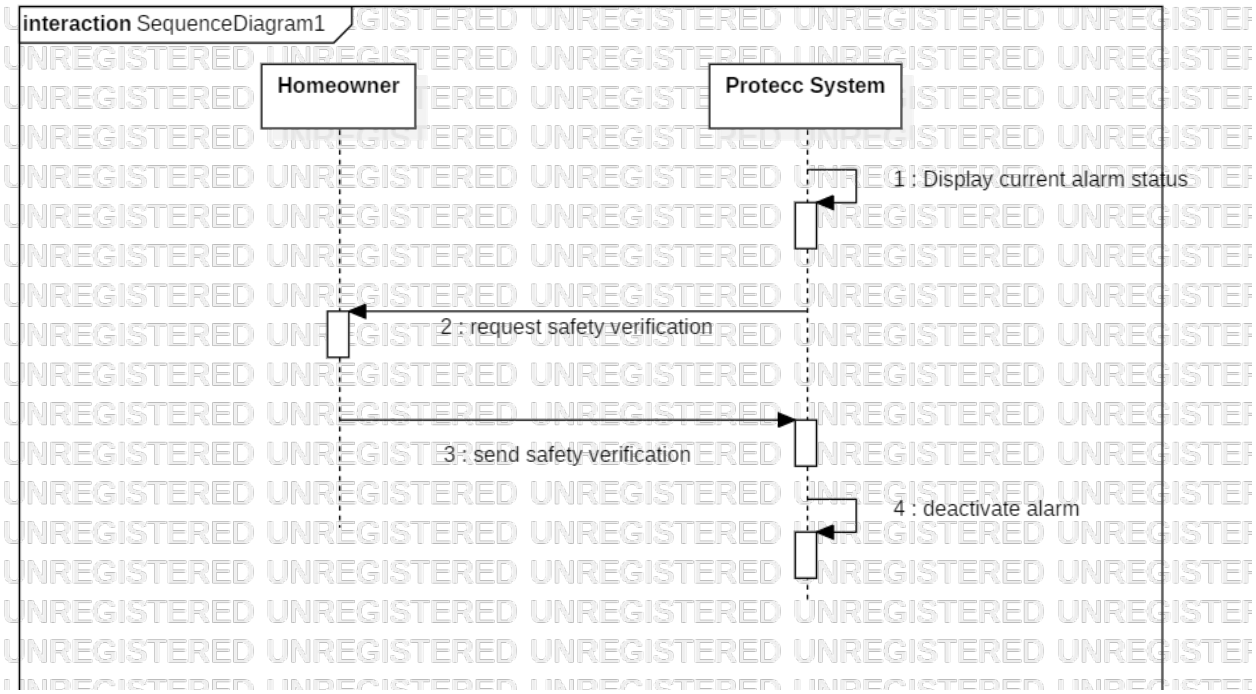
- **Step 3:** Homeowner does not reply that they are safe, and the alarm is not deactivated.

Post Condition: Alarm is turned off.

Scenarios:

If the alarm is triggered and the system is in away mode, the system will warn the homeowner about the alarm. The homeowner has a short amount of time to disable the alarm otherwise it will be set off. The homeowner then has to correctly enter their information in order to deactivate the alarm.

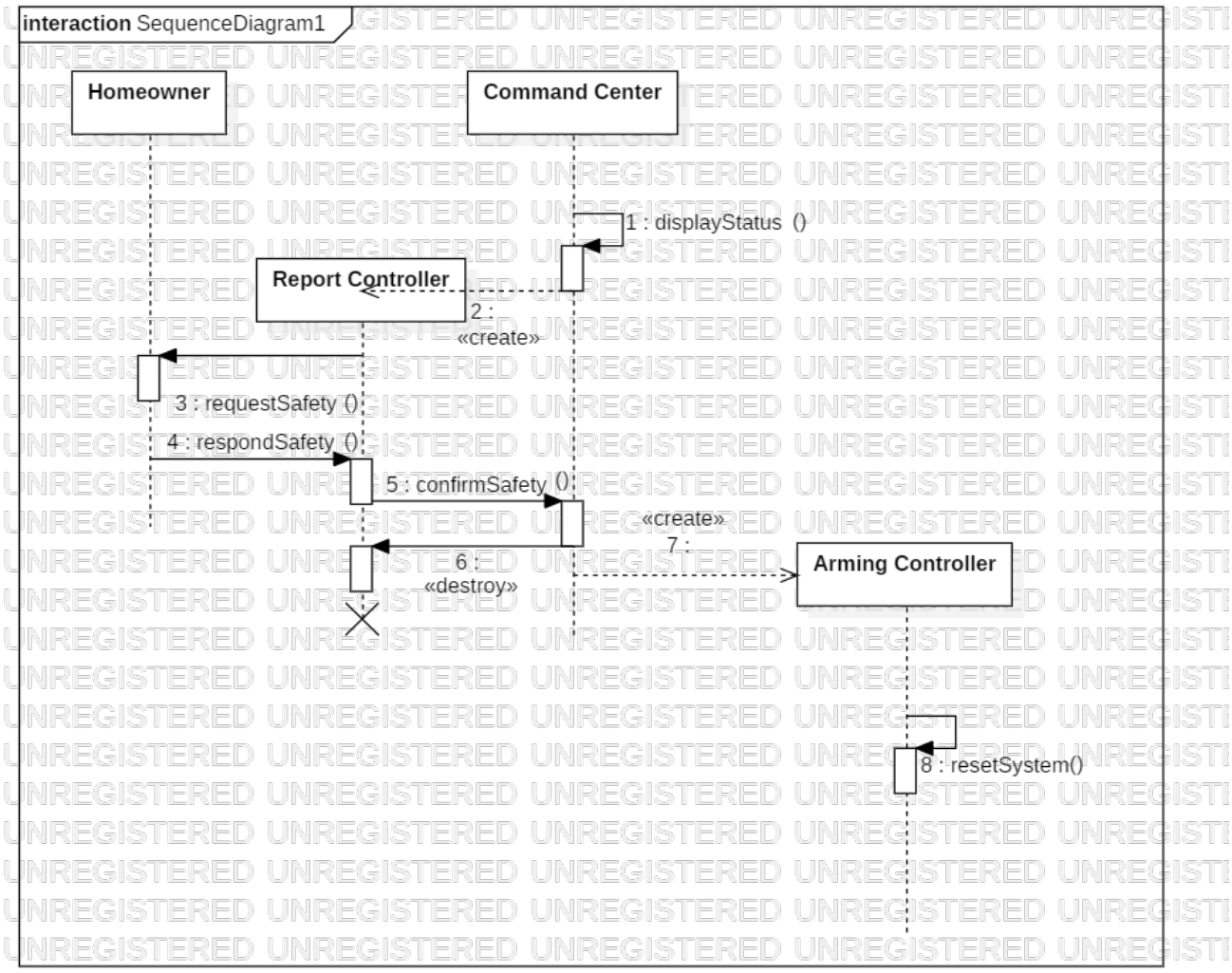
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	TurnOffAlarm
Entry condition	1. A detector is triggered while the system is activated. Alarm is on.
Flow of events	2. The CommandCenter displays current status and that the alarm is triggered. 3. The system asks for the homeowner's personal information to verify the homeowner is safe and not in danger 4. The homeowner enters the correct PIN. 5. The system disables the alarm.
Exit condition	6. The system resets to an unarmed state.

Detailed System Sequence Diagram:



Use Case: Add Sensor

Essential Use Case:

Participants: Home-owner

Pre-conditions: The Protecc system is set up and running

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Access Command Center	
	2. Show Interface
3. Choose option to add sensor	
	4. Show which type of sensor you would like to add
5. Choose sensor type	
	6. Show message enter serial number
7. Enter serial number of the sensor	
	8. Adds new sensor
	9. Show message successfully added
10. Exits to main menu	

Exceptions:

-Step 5. If the user tries to add the wrong type of sensor it will not pair.

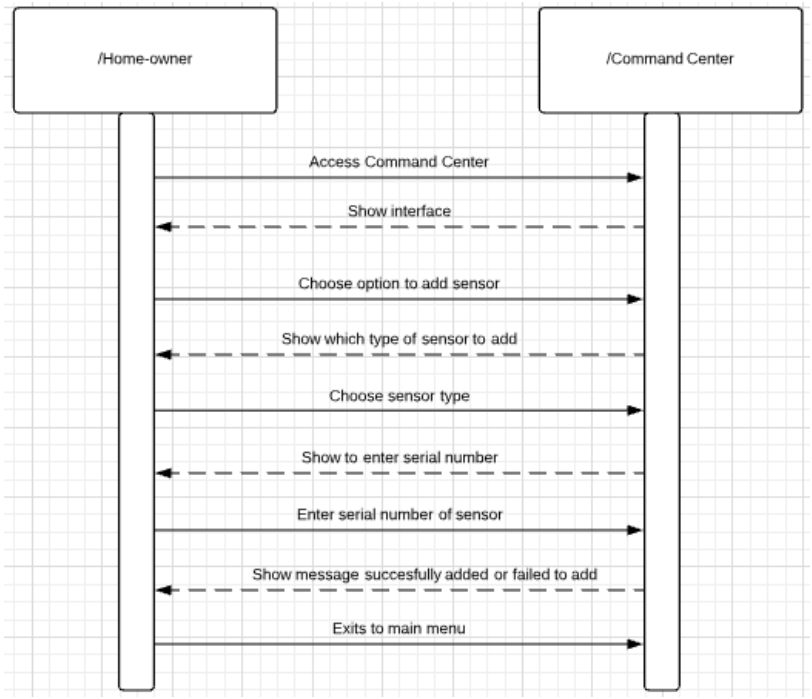
-Step 7**.** If the user enters the wrong serial number, the sensor will not pair with the system.

Post-conditions: The serial numbers are visible.

Scenario:

Home-owner accesses Command Center.
The Command Center pulls up an interface with options.
Home-owner choses add sensor.
The Command Center asks the user what type of sensor they would like to add.
The Home-owner then choses sensor type.
The Command Center then tell user to enter serial number of the sensor.
The Home-owner enters serial number of the sensor.
The Command center adds the sensor.
The Command Center then displays whether the pairing of the sensor is successful or failed.
The Home-owner then exits to the main menu.

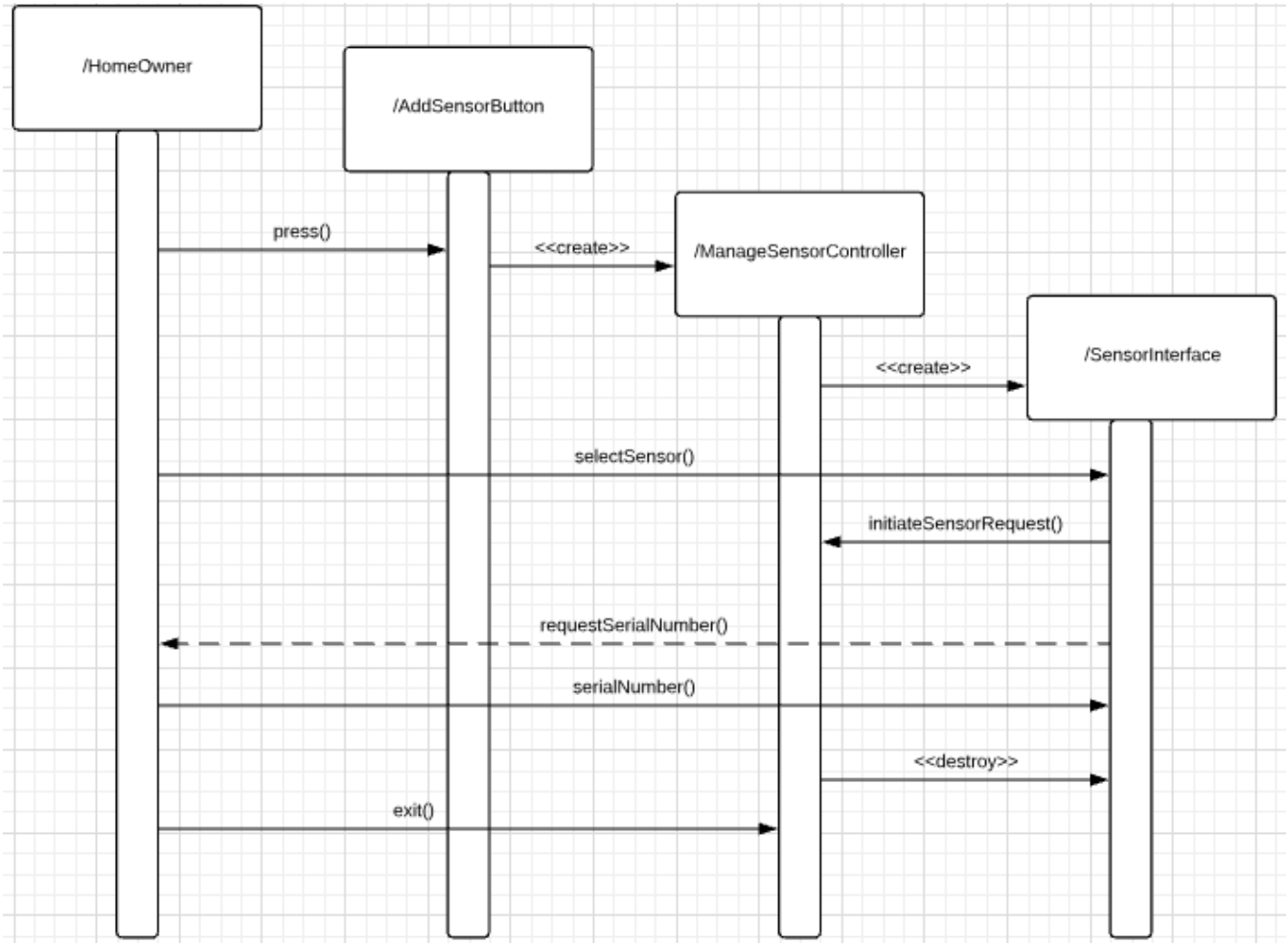
High Level System Sequence Diagram



Concrete Use Case:

Use Case	Add Sensor
Entry Conditions	1. The Home-owner accesses the Command Center and presses the "Add Sensor" button.
Flow of Events	2. The Command Center displays an interface asking what type if sensor you want to add. 3. The Home-owner selects the type of sensor they are adding. 4. The command center displays a message asking them to enter the serial number of the sensor. 5. The Home-owner then enters the serial number of the sensor for pairing. 6. The Command Center then displays a message saying whether the pairing of the sensor was successful or failed.
Exit conditions	7. The Home-owner then exits to the main menu of the Command Center.

Detailed System Sequence Diagram:



Use Case: Remove Sensor

Essential Use Case

Participants: Home-owner

Pre-conditions: The Protecc system is set up and running

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Access Command Center	
	2. Show interface
3. Choose option to remove sensor	
	4. Displays message select which sensor you want to remove
5. Select the sensor you want to remove	
	6. Prompts home-owner to enter password
7. Enters password	
	8.Removes selected sensor
	9. Displays message sensor removed
10. Exits to main menu	

Exceptions:

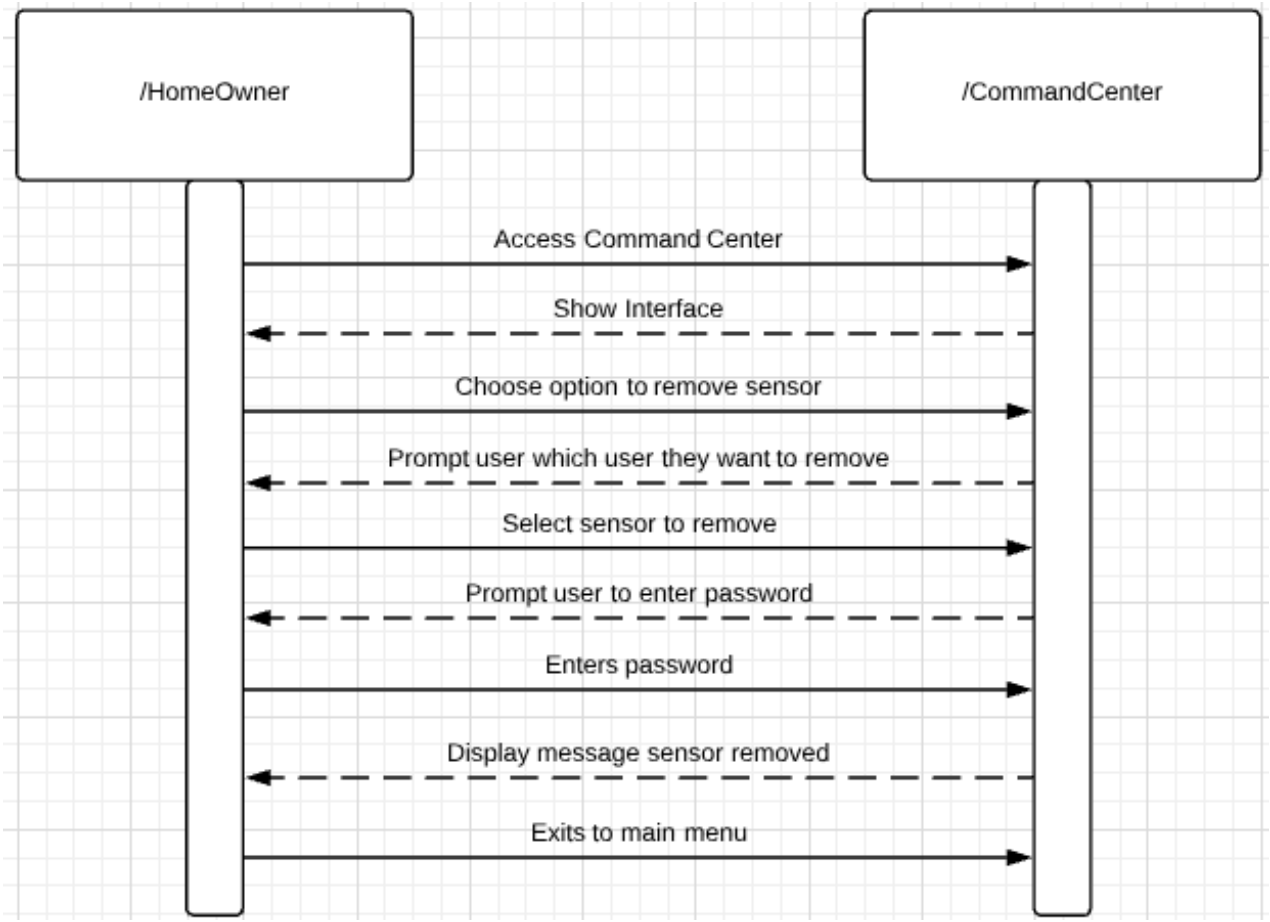
-Step 7**.** If the home-owner enters the wrong password the sensor will not be removed.

Post-conditions: Removed sensor must be deleted from system.

Scenario:

Home-owner accesses Command Center.
The Command Center pulls up interface with options.
Home-owner choses remove sensor.
The Command Center prompts home-owner to select which sensor they would like to remove
The home-owner then selects which sensor to remove.
The Command Center prompts home-owner for password
The home-owner then enters the password.
The Command Center then removes the selected sensor.
The Command Center displays a message that the sensor was removed.
The home-owner exits to main menu.

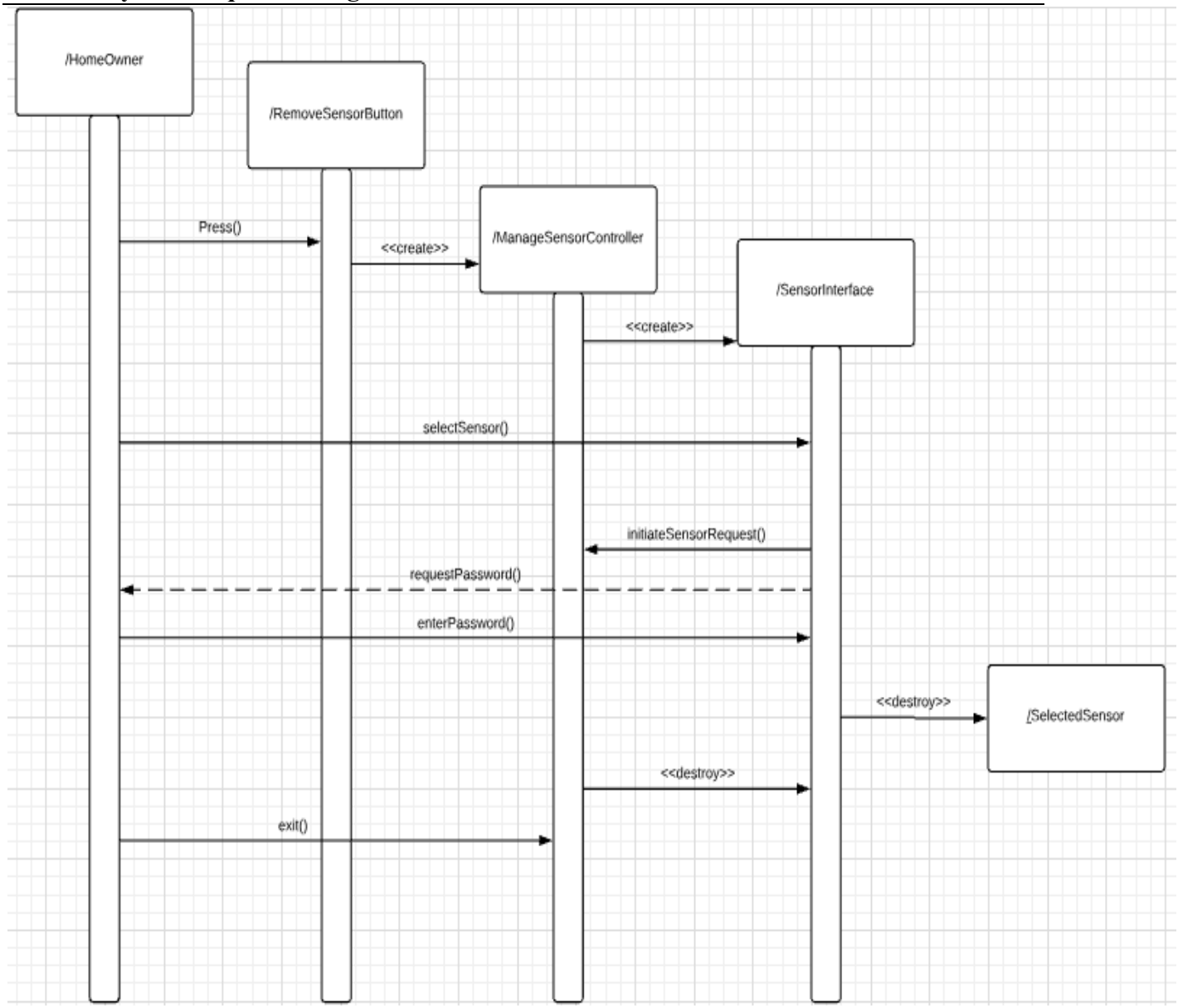
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	Remove Sensor
Entry Condition	1. The Home-owner accesses the Command Center and presses the "Remove Sensor" button.
Flow of Events	2. The Command Center prompts what sensor would you like to remove. 3.The Home-owner then selects which sensor he wants to remove. 4. The command center then asks for a password. 5. The home-owner enters the password. 6. The command center displays message the sensor was removed.
Exit conditions	7. The home-owner exits to the main menu

Detailed System Sequence Diagram:



Use Case: Arm System

Essential Use Case:

Participants: Home-owner, Command Center

Pre-conditions: Homeowner decides to arm system to away mode.

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Accesses application.	
	2. Displays interface
3. User selects away mode.	
	4. System arms house in away mode. 5. Motion sensors and doors/windows are locked. 6. Display to user that house has been put in away mode.
7. Exits application.	

Exceptions:

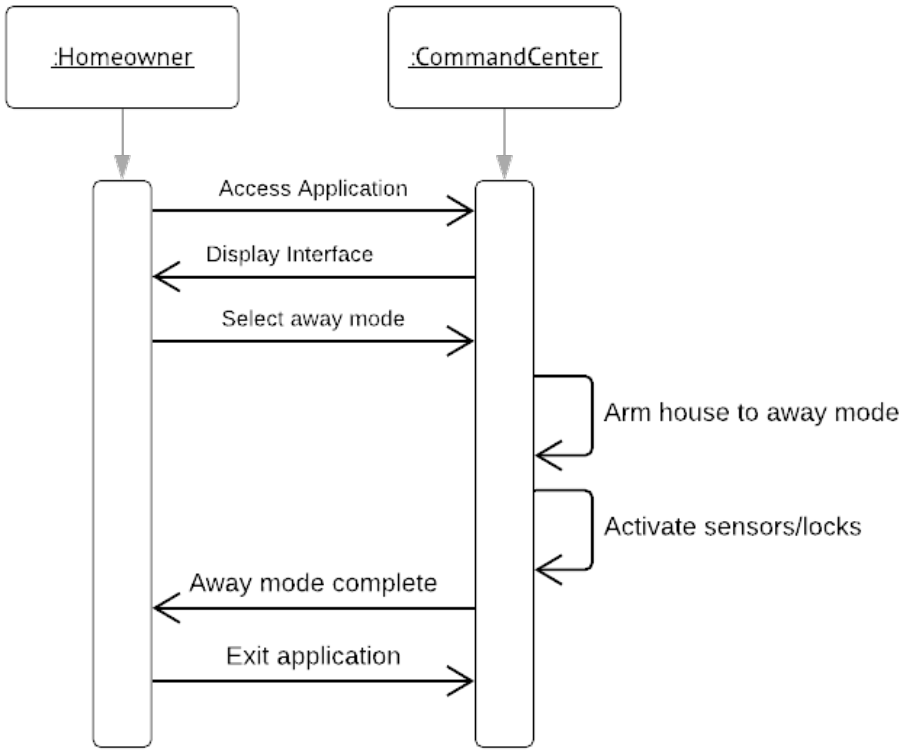
- **Step 1:** Home mode is activated.

Post-condition: Away mode has been activated

Scenarios:

The user decides to arm the house in away mode while he/she is out of town.
User opens application.
The system displays the available modes to put the house in.
User selects to put the house in away mode.
System arms house in away mode.
System activates motion sensors and locks windows/doors.
User closes application.

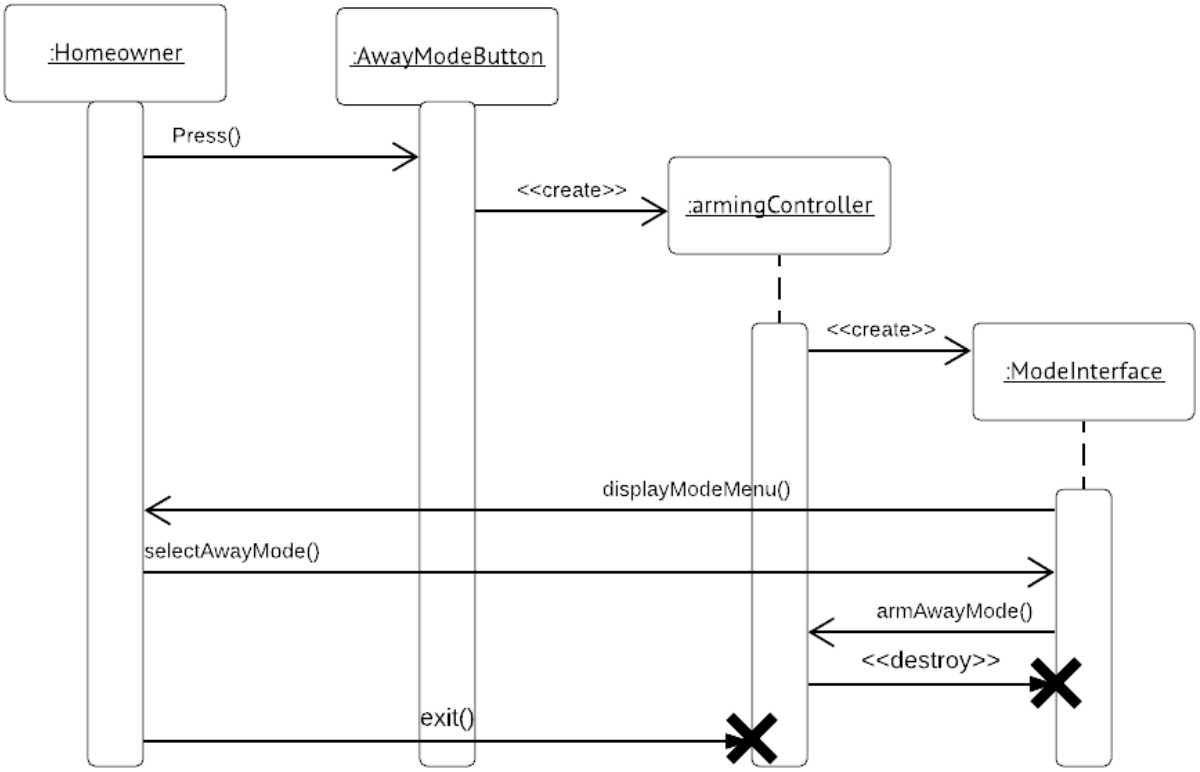
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	ArmHouseToAwayMode
Entry condition	1. The homeowner wishes to arm their house to away mode while on vacation.
Flow of events	2. The homeowner opens the Command Center application via the internet. 3. The Command Center displays a main menu with options to arm house. Within the arm house menu is where user can select away mode. 4. Homeowner selects the “Away Mode” button. 5. Command Center changes house to away mode, which means all sensors are activated and all locks are locked. 6. Command Center notifies user that house is in away mode. 7. User exits the online application
Exit condition	6. House is in away mode.

Detailed System Sequence Diagram:



Use Case: Disarm System

Essential Use Case:

Participants: Home-owner

Pre-conditions: System is armed.

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Access interface.	
2. Select to disarm system.	
	3. Prompt for PIN.
4. Present PIN.	
	5. Disarm.

Exceptions:

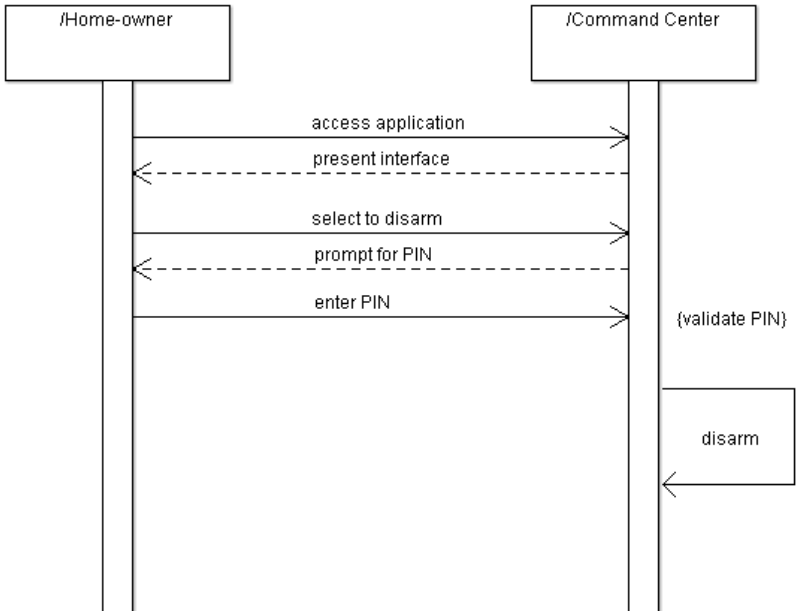
- **Step 5:** Presented PIN is incorrect. Fails to disarm and will signal an alarm and create an emergency report if it is not fixed.

Post-condition: System disarmed.

Scenarios:

Home-owner comes home to an armed system.
Walks into the house, alarm tripped.
Home-owner goes to Command Center house terminal.
Home-owner presses disarm button.
CommandCenter prompts home-owner for a PIN.
Home-owner enters correct PIN.
Alarm stops, system is disarmed.

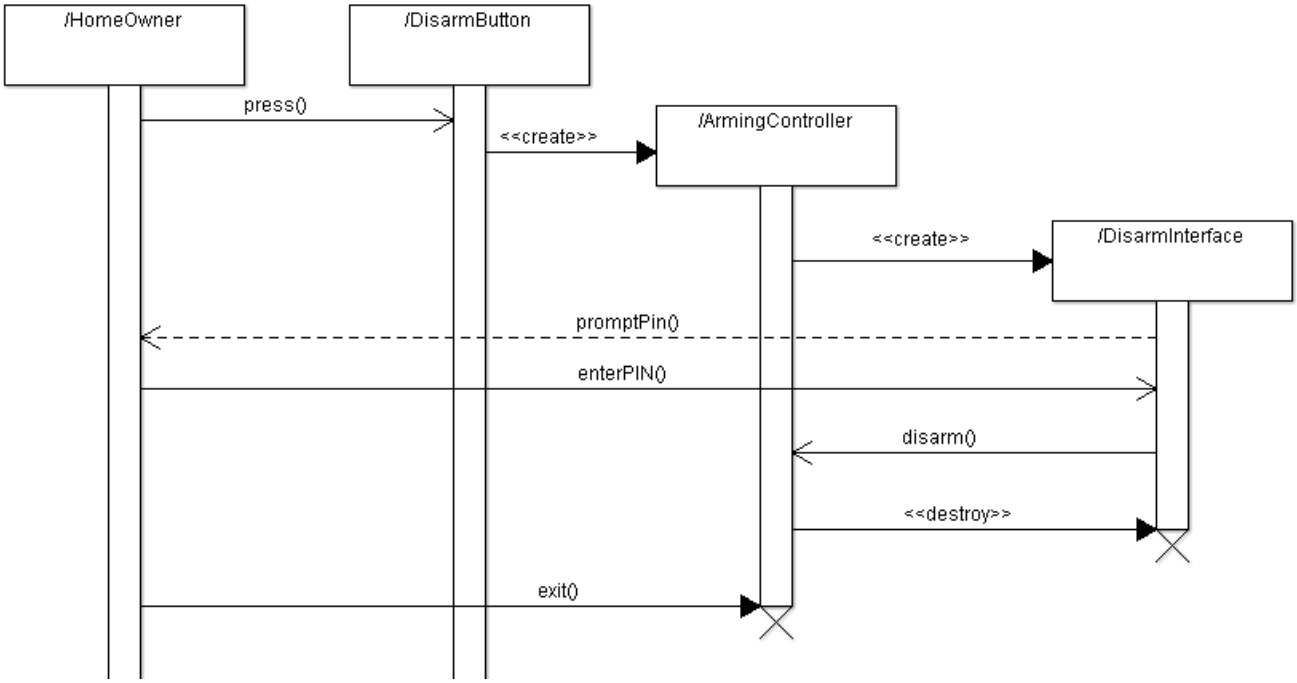
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	DisarmSystem
Entry condition	1. CommandCenter is armed to home or away mode.
Flow of events	2. HomeOwner accesses CommandCenter via the online interface or the home terminal. 3. HomeOwner hits the "Disarm" button. 4. CommandCenter prompts HomeOwner for a PIN. 5. HomeOwner inputs correct PIN. 6. CommandCenter disarms the system and deactivates any alarms. 7. User exits the online application
Exit condition	6. House is disarmed.

Detailed System Sequence Diagram:



Use Case: Adjust Settings

Essential Use Case:

Participants: Home-owner, Command Center

Pre-conditions: Homeowner changes notification settings.

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Accesses application.	
	2. Displays interface
3. Select settings.	
	4. Displays list of settings options.
5. Selects appropriate setting option to change.	
	6. Changes settings as requested by user.
7. Exits application.	

Exceptions:

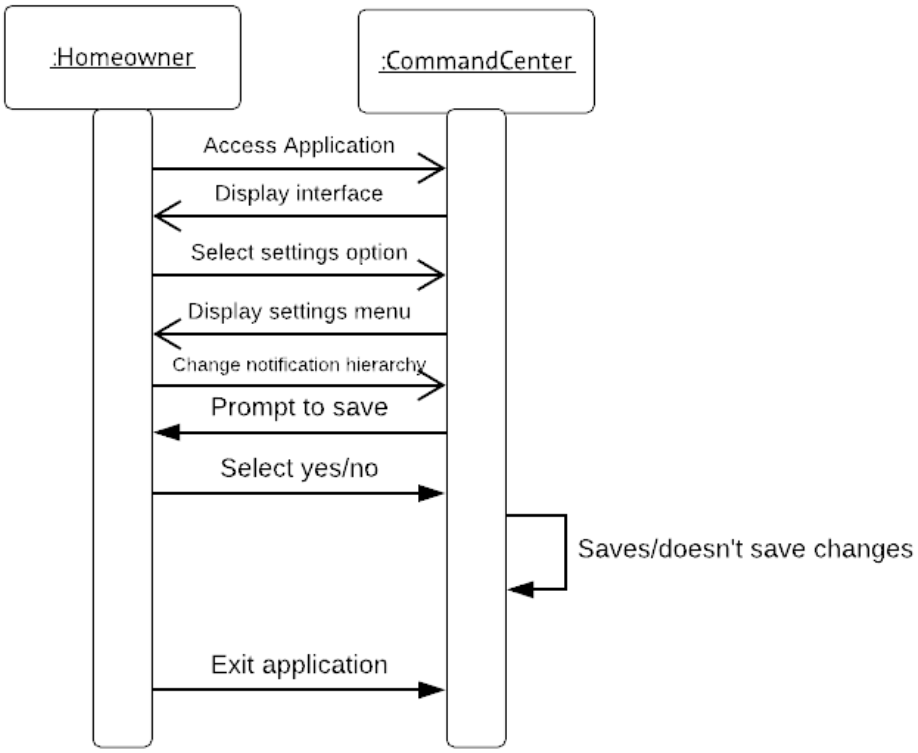
- **Step 5:** Setting options include notification hierarchy and when emergency services should be called.

Post-condition: Settings have been changed or remain the same.

Scenarios:

The user decides to change the notification settings
User opens application.
The system displays a main menu.
User selects the settings option.
A settings menu appears.
User chooses to change notification hierarchy.
User is prompted to save changes to settings.
User selects yes and closes the application

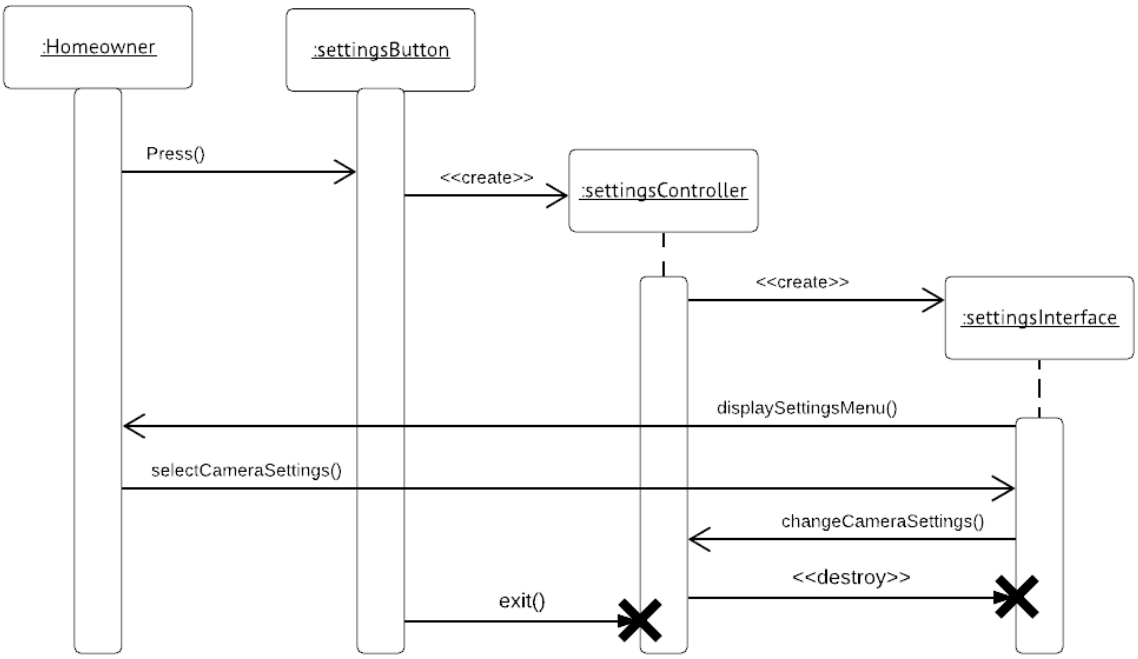
High Level System Sequence Diagram:



Concrete Use Case:

Use case name	AdjustCameraSettings
Entry condition	1. The homeowner wishes to not be notified every time a camera captures a person outside of the house except when in home/away mode.
Flow of events	2. The homeowner opens the Command Center application via the internet. 3. The Command Center displays a main menu. 4. Homeowner selects the “Settings” option button. 5. Homeowner selects “Camera Settings” within settings. 6. Homeowner changes notification settings for the camera. 7. System prompts user to save changes. 7. User saves changes and exits the online application.
Exit condition	6. Camera settings are changed.

Detailed System Sequence Diagram:



Use Case: Add to Report

Essential Use Case:

Participants: Home-owner

Pre-conditions: Home-owner chooses to add a description to the report.

Typical Course of Events:

Actor Intention	**System Responsibility**
1. Access application	
	2. Display interface
3. Choose the add to report button	
	4. Display add to report interface
5. Fill contents of report description	
	6. Prompt home-owner to save or not
7. Choose to save	
	8. Update report description
9. Exit application	

Exceptions:

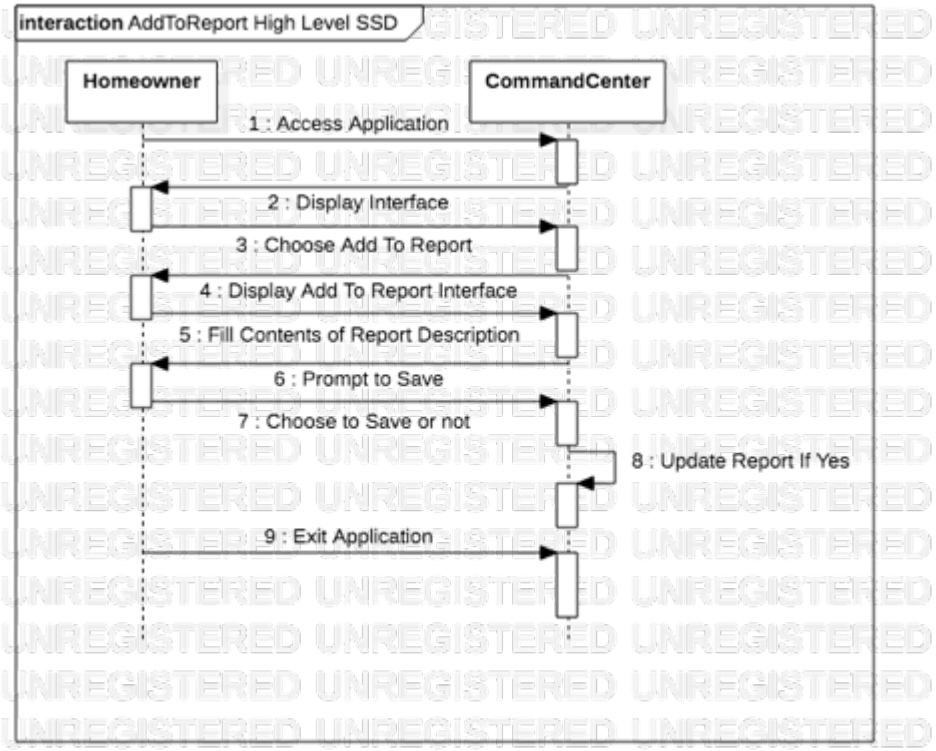
- **Step 7:** If the homeowner chooses not to save, the system will not update the report description.

Post-condition: Report description is updated if the user saved, and nothing changes if not.

Scenarios:

Home-owner accesses command center.
The command center displays the interface.
Home-owner chooses to add to the report.
The command center displays the add to report interface.
Home-owner fills the contents of the report description.
The command center prompts the homeowner to save or not.
The home-owner chooses to save the report description.
The command center updates the report description.
The home-owner exits the application.

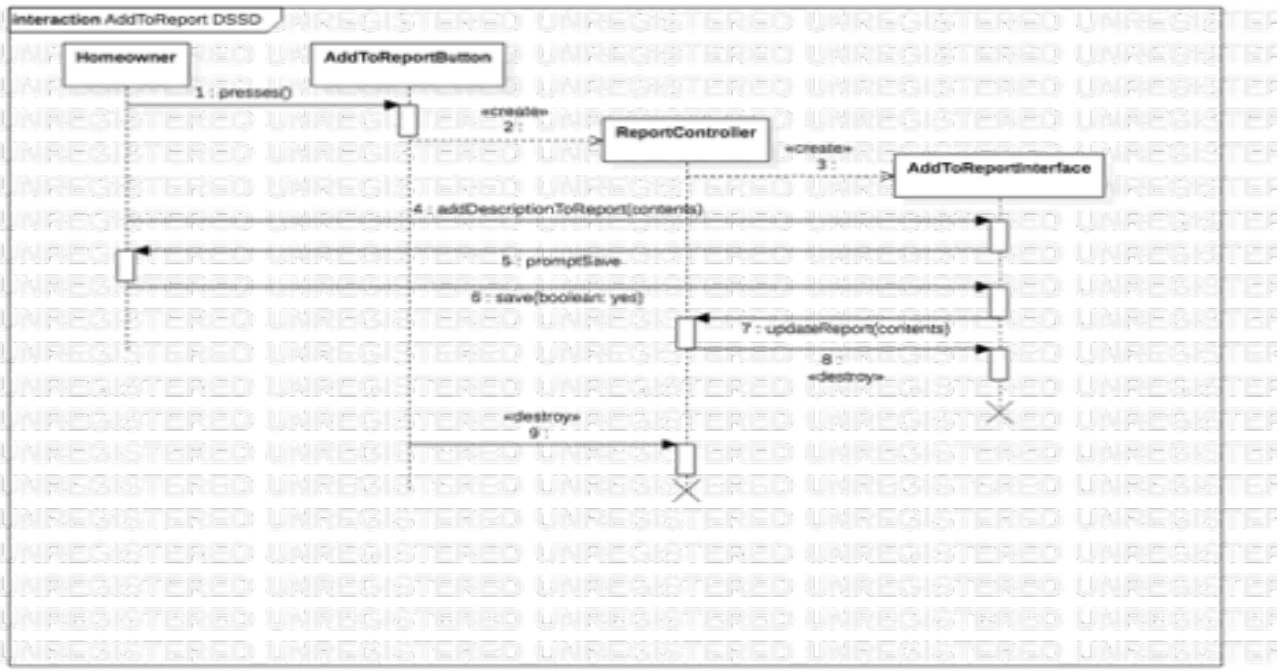
High Level System Sequence Diagram:



Concrete Use Case:

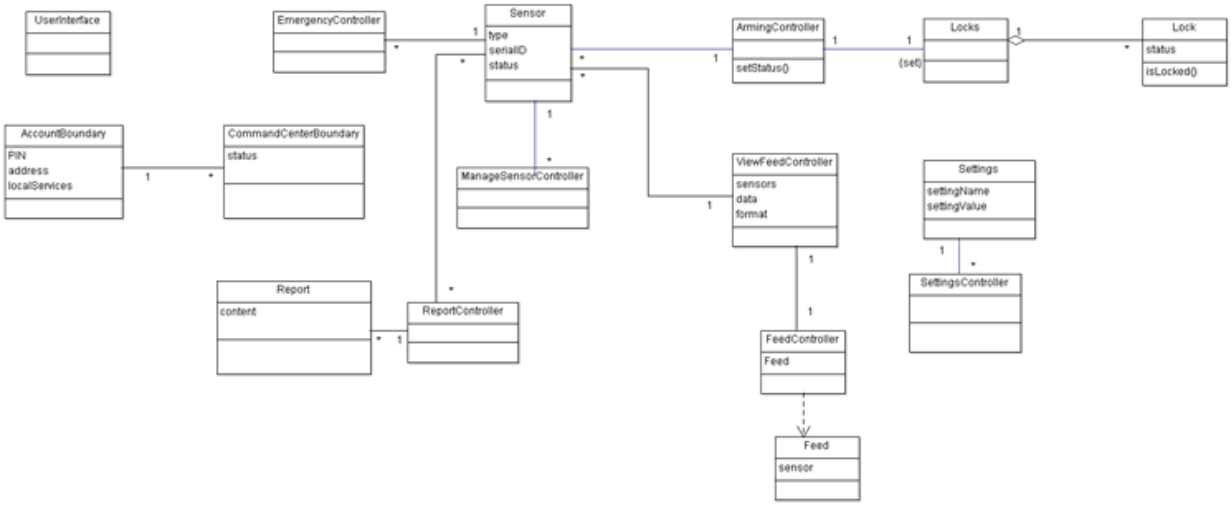
Use case name	AddToReport
Entry condition	1. The homeowner wishes to add a description to an event in the report.
Flow of events	2. The homeowner accesses the command center application. 3. The command center displays the main menu interface. 4. The homeowner selects the Add to Report option. 5. The command center displays the Add to Report interface. 6. The homeowner fills in a description of an event in the report. 7. The command center prompts the user to save the description. 8. The homeowner chooses to save the description. 9. The command center saves the report description. 10. The homeowner exits the application.
Exit condition	The report has been saved.

Detailed System Sequence Diagram:



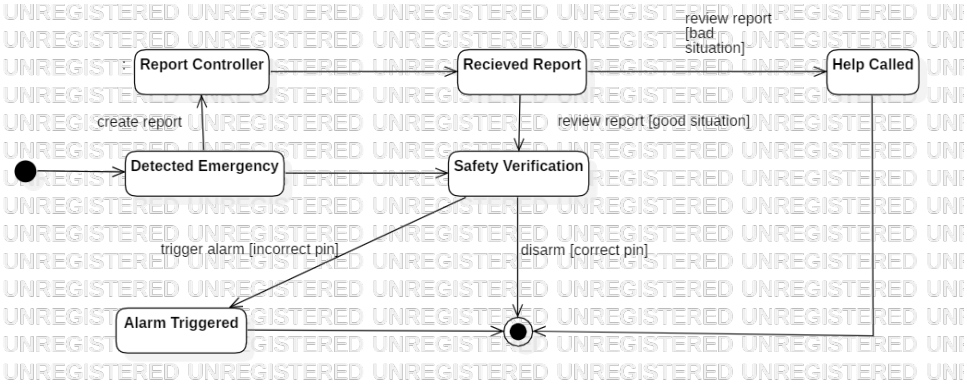
Application Design

Application Class Model:

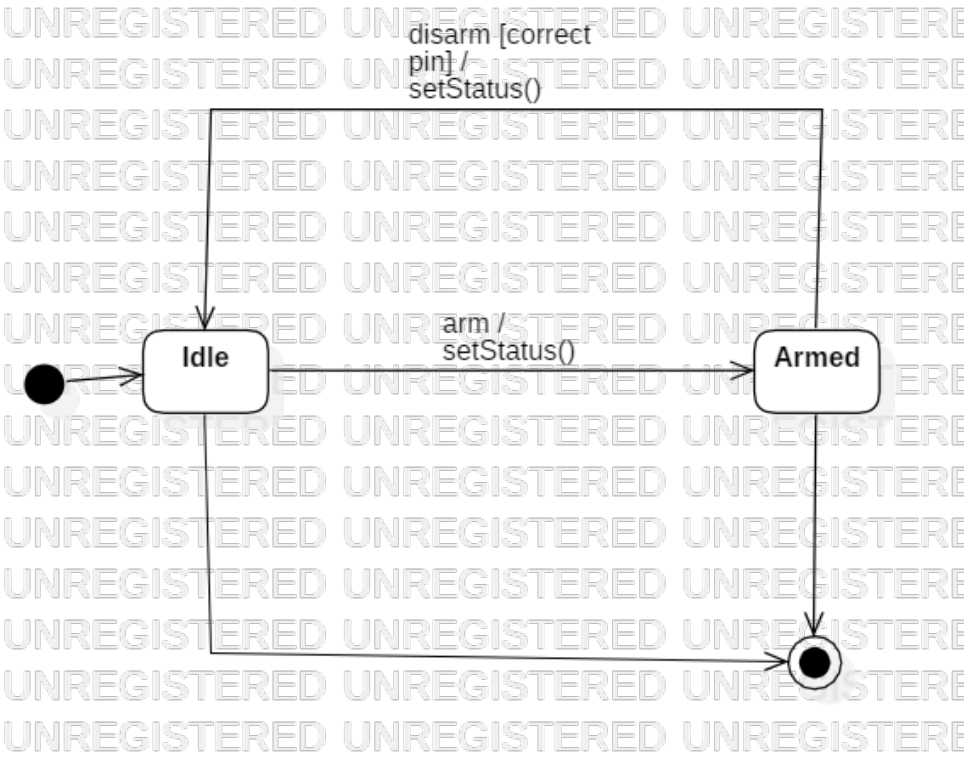


Application State Model:

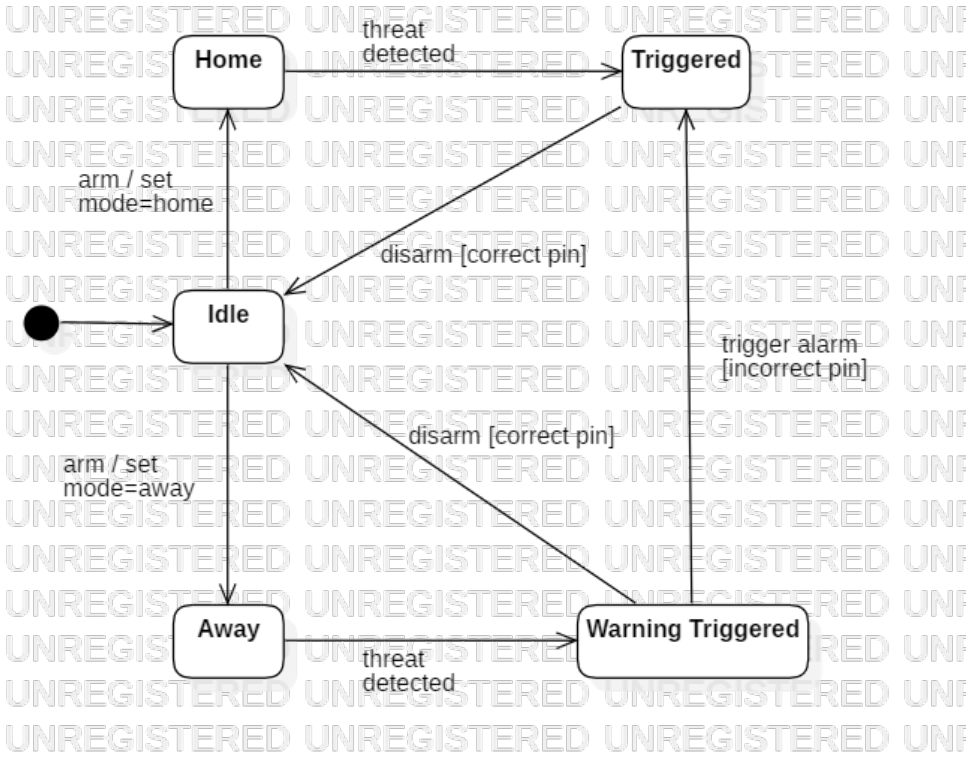
Emergency Controller:



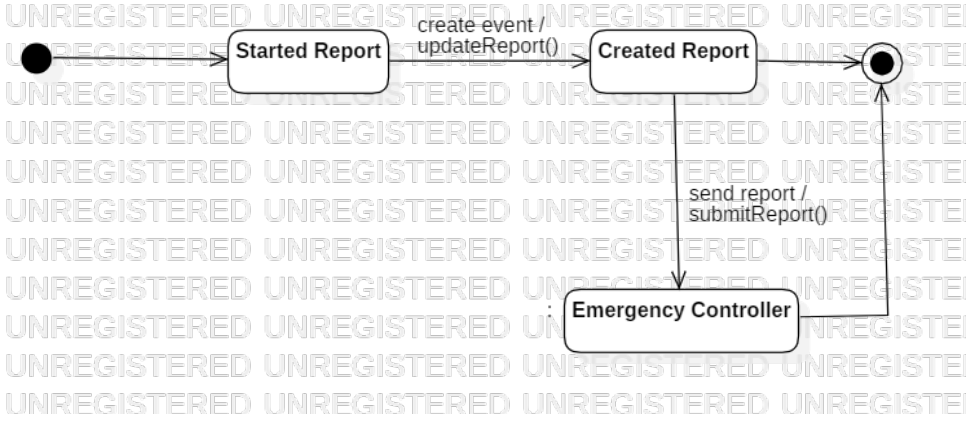
Arming Controller:

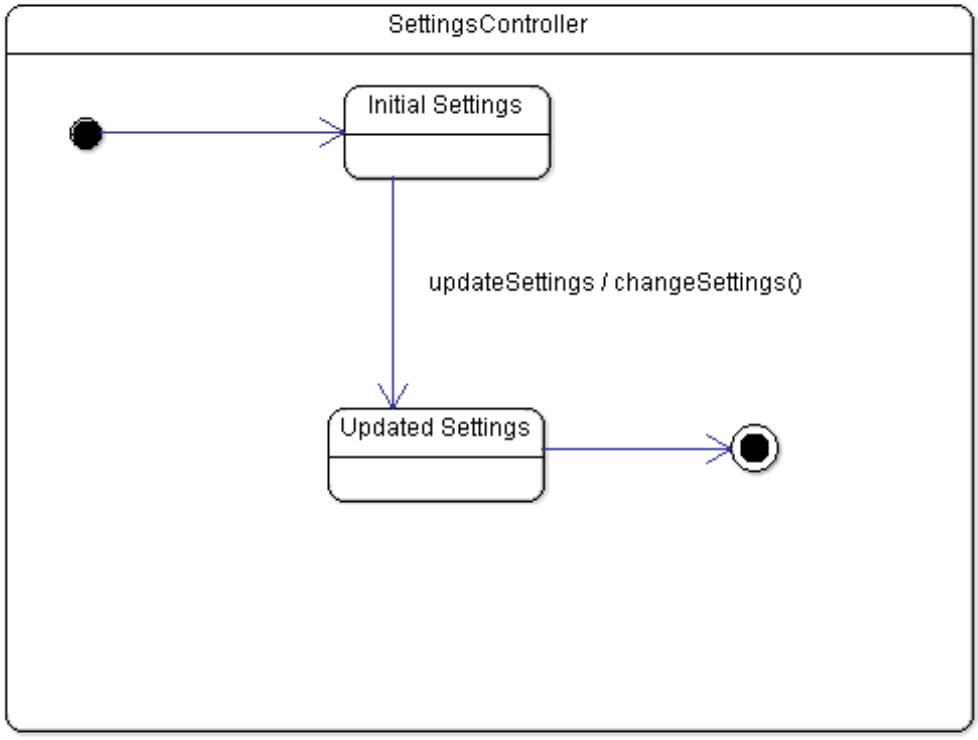


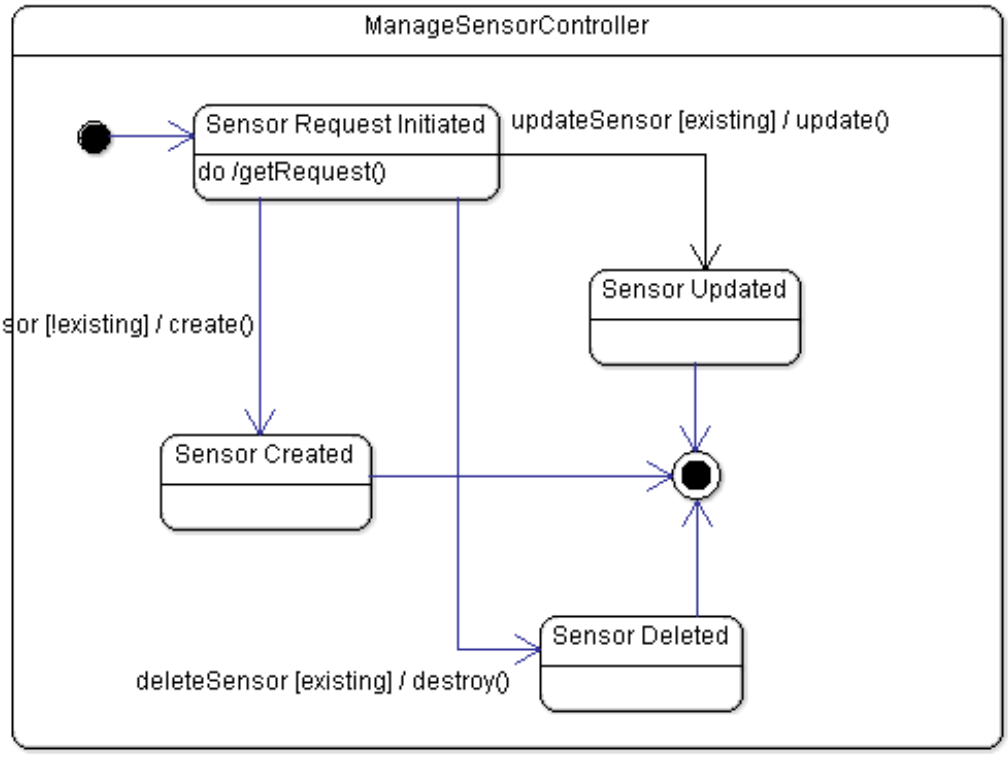
System:

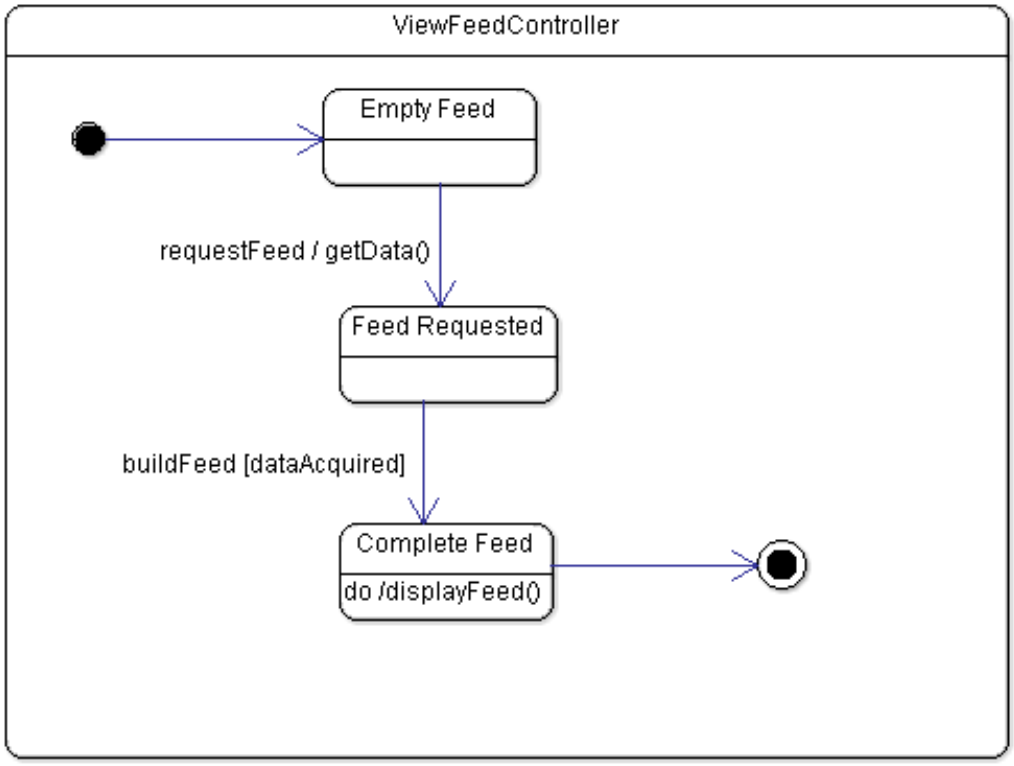


Report Controller:

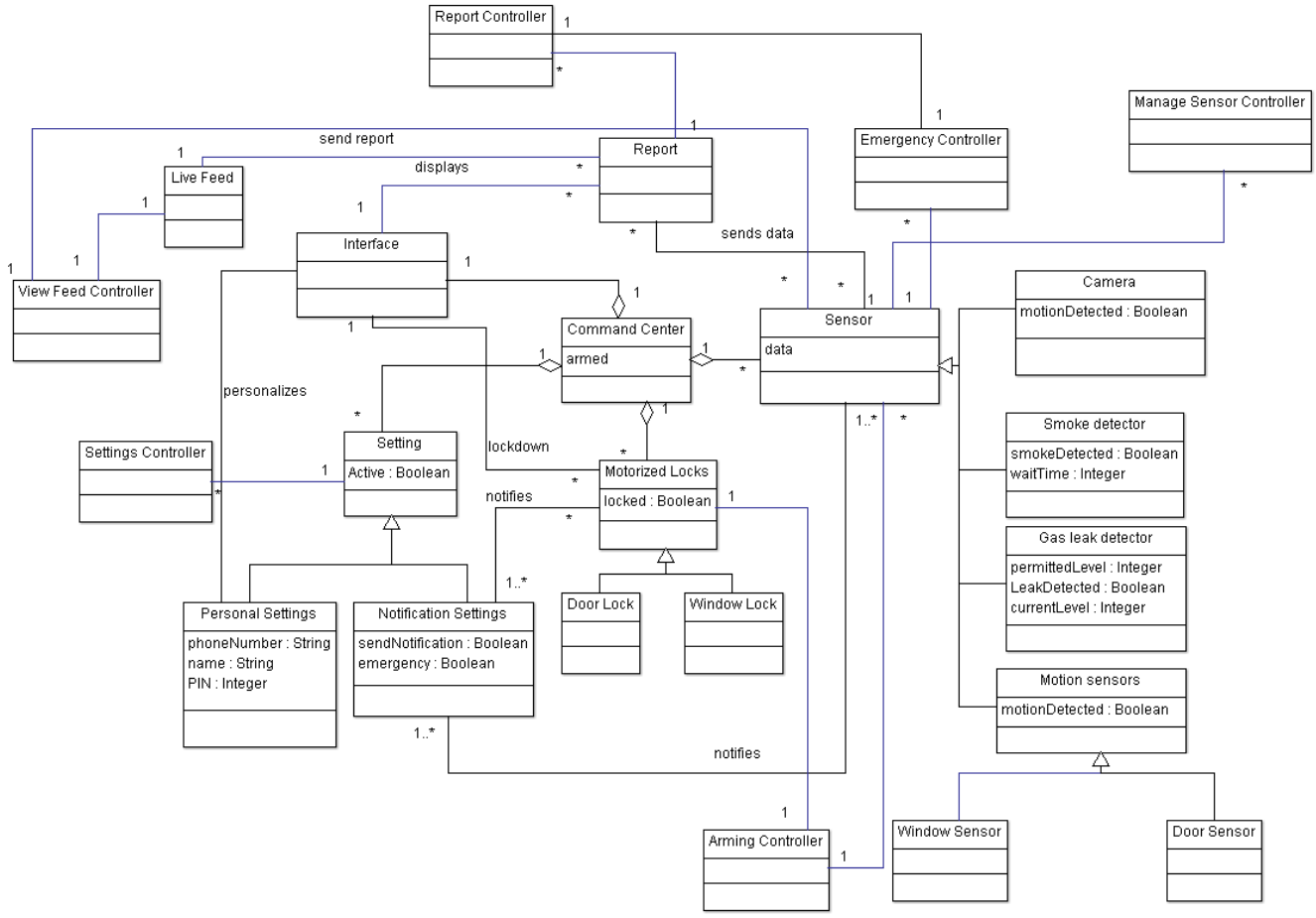



SettingsController
Initial Settings
updateSettings / changeSettings()
Updated Settings


ManageSensorController
Sensor Request Initiated
do /getRequest()
updateSensor [existing] / update()
Sensor Updated
newSensor [!existing] / create()
Sensor Created
Sensor Deleted
deleteSensor [existing] / destroy()


ViewFeedController
Empty Feed
requestFeed / getData()
Feed Requested
buildFeed [dataAcquired]
Complete Feed
do /displayFeed()

Consolidated Class Model:



Model Review

Class Domain Model

- Completeness
    - To ensure the model was completed, we used linguistic analysis on our concept statement and highlighted all the important classes and attributes to put into our model. This ensured that every part of our system was covered.
- Consistency
    - To ensure consistency, we covered all of our concept statement and made sure every important detail was represented in our domain model.
- Correctness
    - The concept statement and class domain model both model the entire system, therefore making them correct.

Domain State Model

- Completeness
    - The state model covers all possible states that the system and the sensor could be at, therefore it is complete.
- Consistency
    - When compared to the concept statement and class model, the domain state model does not contain any new information and can only be one state at a time.
- Correctness
    - The state model covers all options for the states to be in and covers all the entry conditions and events.

Application Interaction Models

- Completeness
    - The use cases elaborate for each scenario and completely cover all of the scenario's options.
- Consistency
    - To ensure consistency, we checked all of our use cases with our concept statement and made sure every important detail was represented in both.
- Correctness
    - The use cases directly apply to the situations presented by our system and do not create any new situations previously unpresented.

Application Class Model

- Completeness
    - The application class model accurately translates the domain model to a form that can be built with software and follows the application interaction model. Hence it is complete.
- Consistency
    - The application class model is consistent with the domain model and the concept statement, as well as the application interaction model.
- Correctness
    - The application class model correctly solves the problem that the product is designed to solve and designs a correct software solution.

Application State Model

- Completeness
    - Our state models covered every controller we implemented and included their actions between states.
- Consistency
    - Our state models represented the information previously presented in the class model without adding anything new that would affect them.
- Correctness
    - Our state models represented the information in the same way that they were previously represented while featuring all the required information about them.

Phase II

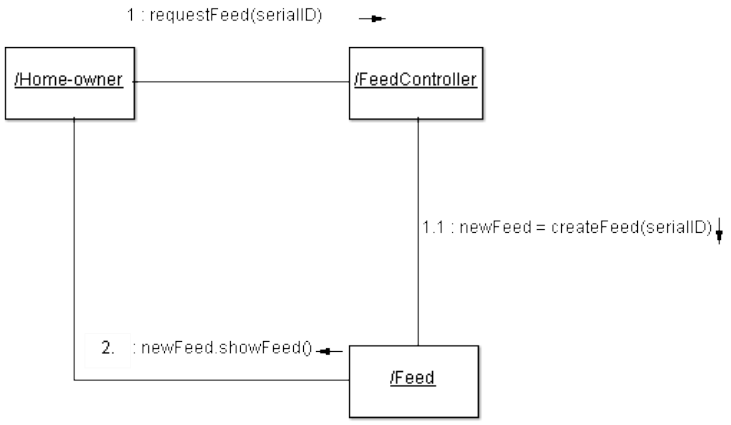
Architectural Design

Architectural Design

Interaction Design - Collaboration Diagrams

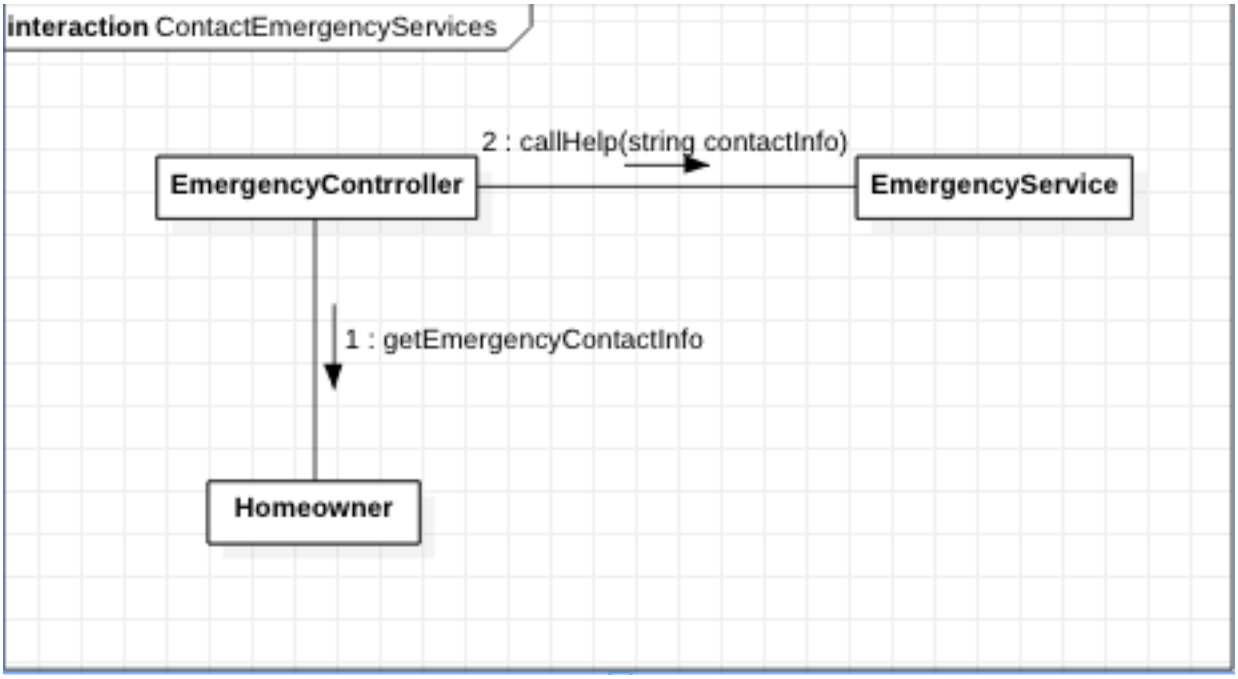
Object Design

Collaboration Diagram: View Live Feed



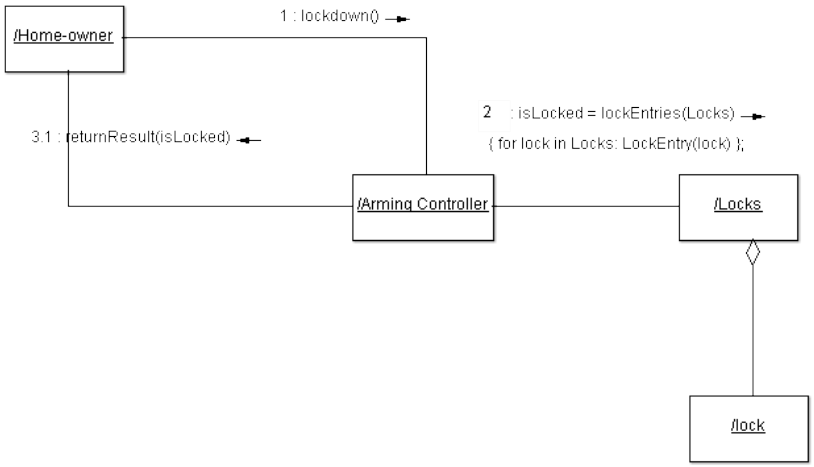
The primary GRASP guideline used in this collaboration diagram is **controller**. The Feed Controller is used to bridge the gap between the home-owner's interface and the feed itself. The controller is also a **creator** as without it the feed does not actually exist. Once the feed is created, it is an information **expert.** It has sole knowledge of how the feed structure is designed. It gathers the data required to build said feed from the database then displays the structure to the home-owner.

Collaboration Diagram: Contact Emergency Services



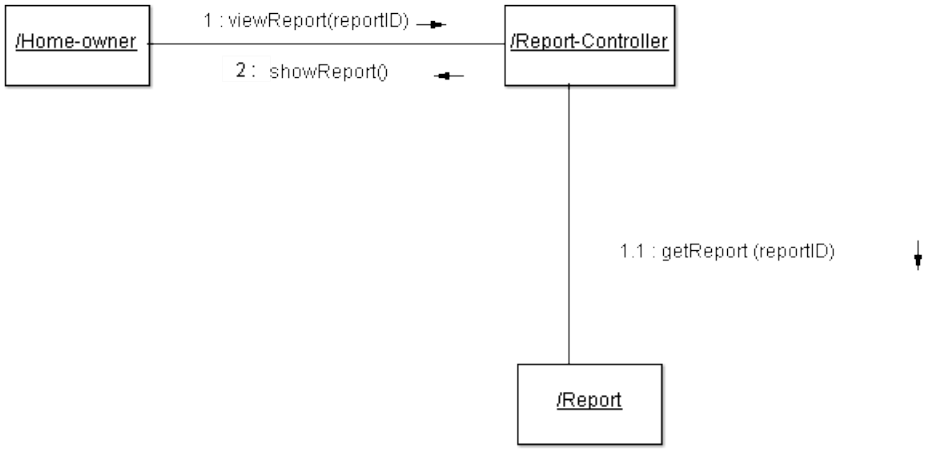
The EmergencyController object acts as an intermediary between the Homeowner and the EmergencyServices, so the **Don't Talk to Strangers** guideline is used. Since EmergencyController acts as a controller object, the **Controller** GRASP guideline is also used. This diagram also exhibits the **Low Coupling** GRASP guideline because each object is only focused on one message.

Collaboration Diagram: Lockdown



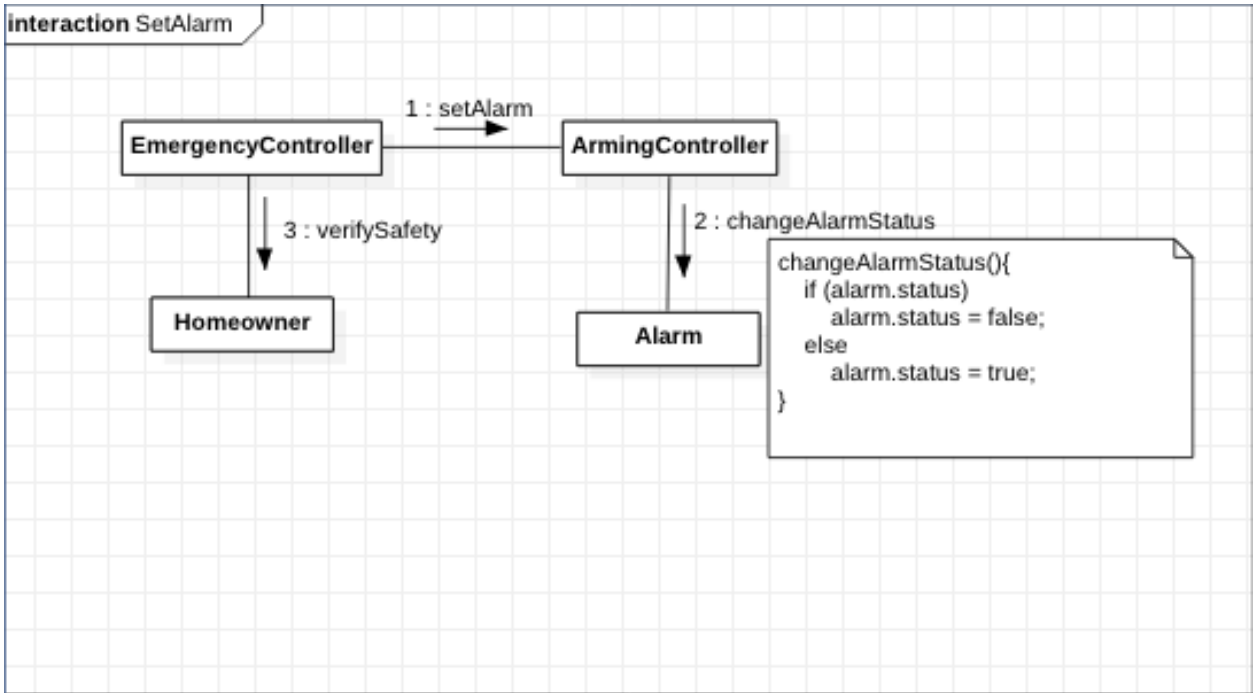
The Arming Controller separates the user's interface from the actual lockdown procedure. Hence it follows the **controller** guideline. Since the locks are physical locks, they do not carry any information on board. The Locks class follows the **pure fabrication** guideline since it carries the information for the set of locks, but it is not a device itself. Letting the Arming Controller be the information expert would cause cohesion issues since it is in charge of simply arming and disarming devices.

Collaboration Diagram: View Report



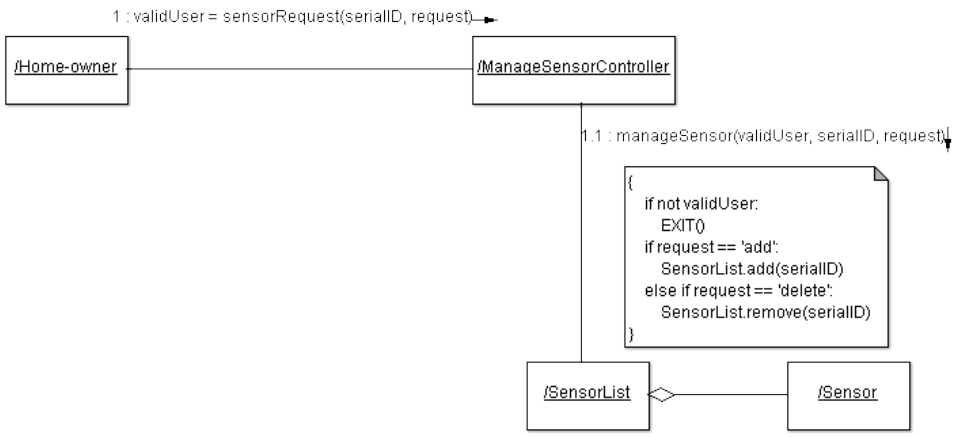
The Report Controller separates the user action from the view report procedure. The Report is an **expert** of its contents, which it passes off to the report controller to handle the transfer of the data to the home-owner. Since the Report Controller handles all of the communication between the Report and the home-owner, it has high cohesion within its own functions.

Collaboration Diagram: Trigger/Turn Off Alarm



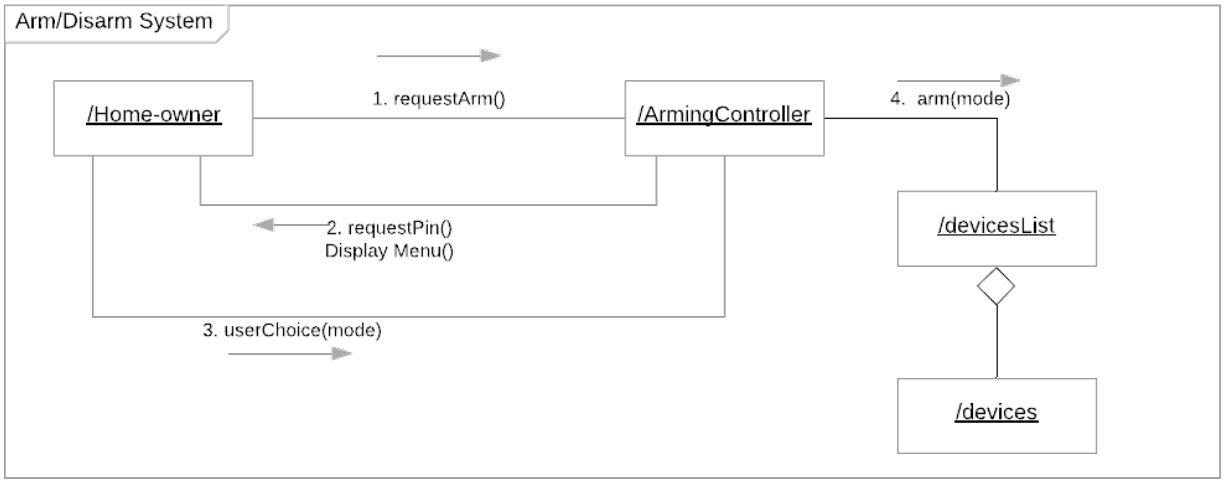
The EmergencyController object acts as a **Controller** that interacts with the ArmingController. Similarly, the ArmingContoller object acts as a **Controller** which changes the status of the Alarm object whenever the EmergencyController contacts it with the setAlarm message. The ArmingController object also acts as an **Expert** between EmergencyController and Alarm, which follows the **Expert** as well as **Don't Talk to Strangers** guidelines. This model also shows **Low Coupling** and **High Cohesion** because each object is only responsible for one message between two objects.

Collaboration Diagram: Add/Remove Sensor



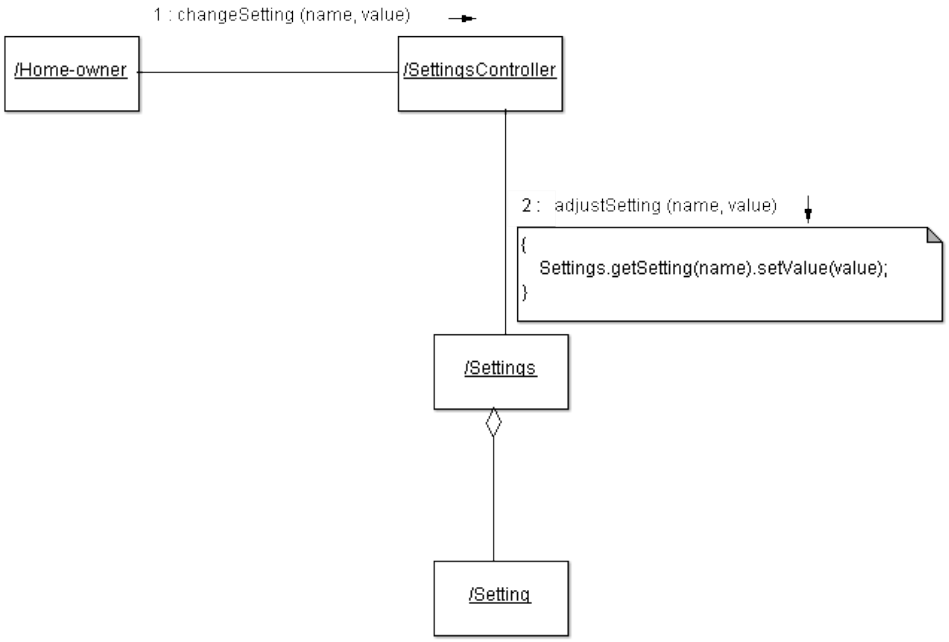
The Manage Sensor Controller is a **controller** between the user's interface and the list of sensors. It verifies that the user is authorized to make the request. Then the controller manages the sensor list. The sensor list is an information **expert** for all the sensors within the system. This prevents individual sensors from **talking to strangers** by giving a specific line of communication.

Collaboration Diagram: Arm/Disarm System



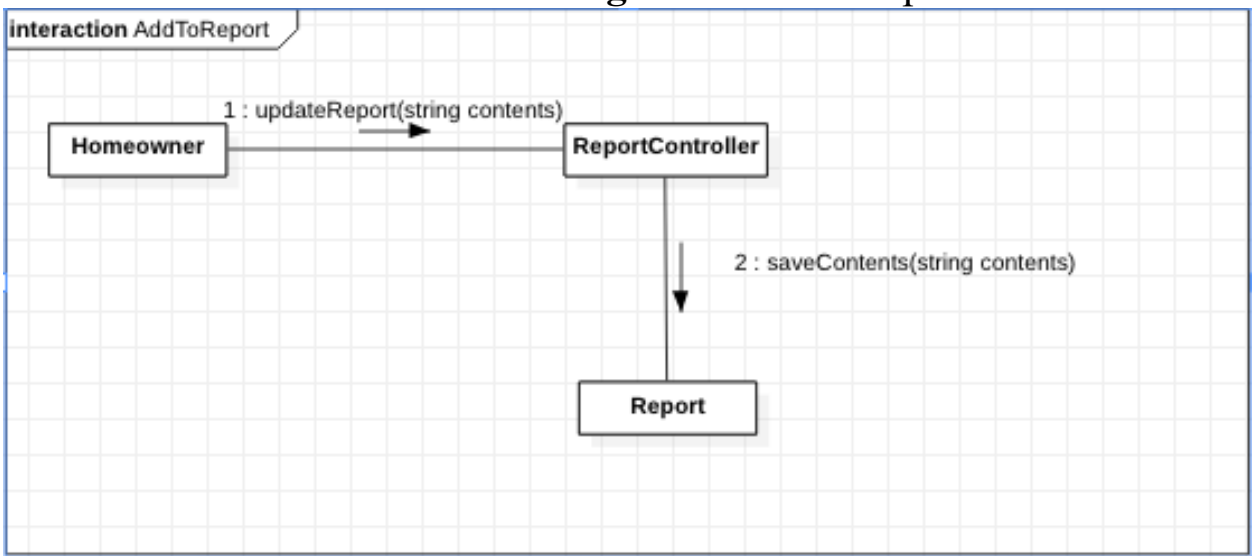
The Arming Controller follows the **controller** GRASP guideline since its name reflects the use case, arming the security system. It acts as the user's interface from the actual arming procedure. It also carries the responsibilities of the devicesList class, which includes arming certain devices depending on which mode the user chooses. The devicesList class follows the **pure fabrication** guideline since it is an abstraction and artificial class of the devices class. It represents all of the devices that the arming controller would be communicating with to arm the house in the specified mode.

Collaboration Diagram: Adjust Settings



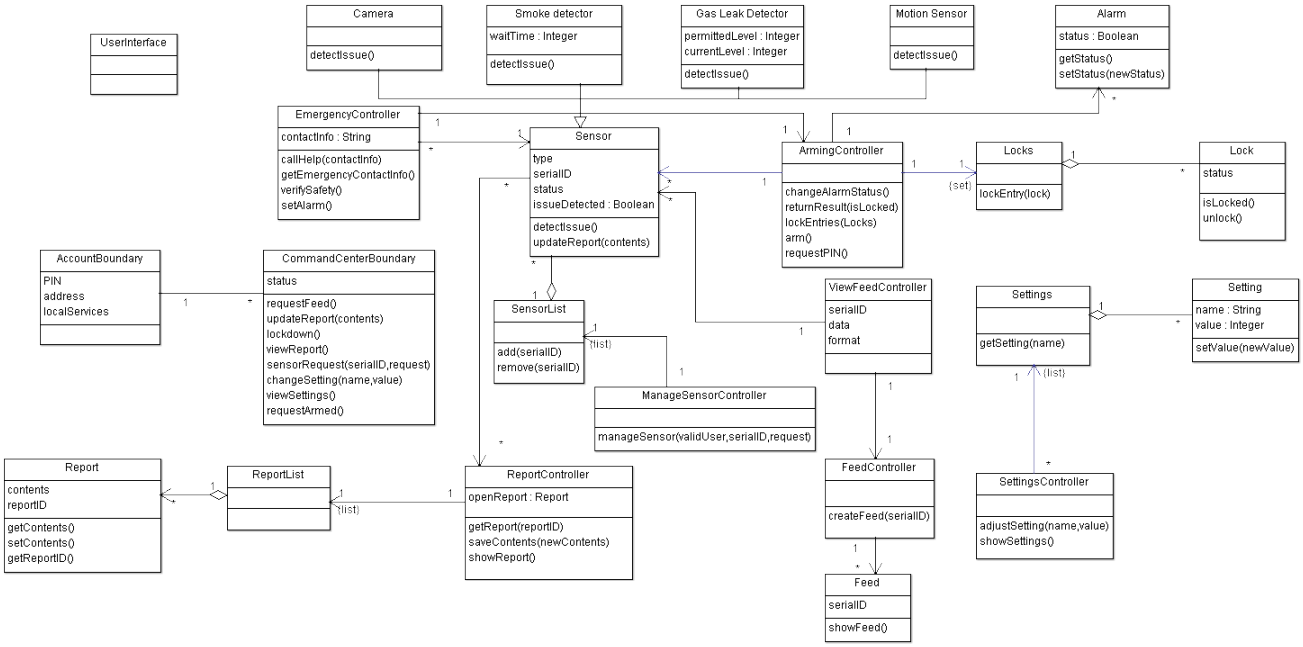
The Settings Controller takes in the information from the interface that the user wants to do with the settings and changes the settings with it. Settings follows the **pure fabrication** guideline as it is a collection of Setting objects used to abstract each setting. This allows each setting to adhere to the **don't talk to strangers** GRASP guideline.

Collaboration Diagram: Add to Report



The GRASP guidelines used in this diagram are **Controller**, **Creator**, **Don't Talk to Strangers**, and **Expert**. The ReportController object is used to create a Report as well as update the contents and save the contents which follows the **Creator** guideline. This diagram also follows the **Expert** and **Don't Talk to Strangers** guidelines because the ReportController is used to manage and save contents of the report instead of the Homeowner directly accessing the report object.

Design Class Diagram (DCD)



Object Design

Class: Sensor

Interface Contract:

- Invariant: The sensors are continuously listening to detect an issue.

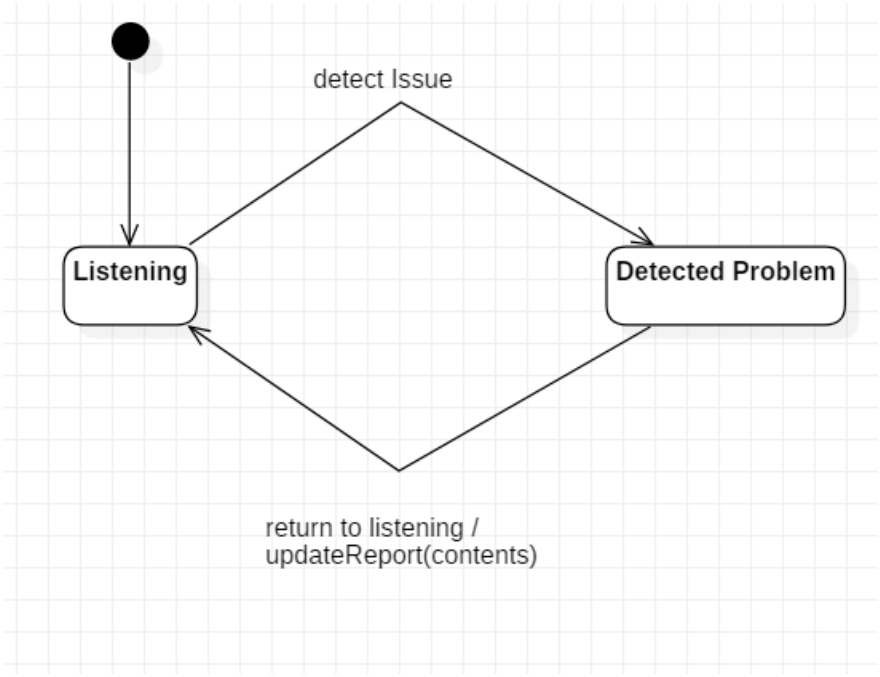
detectIssue()

- Precondition: A sensor is given a variation of its data.
- Postcondition: A sensor returns that it has detected an issue during its listening.

updateReport(contents)

- Precondition: There exists a report that has available contents.
- Postcondition: The report now includes the new contents in it.

UML State-chart:



Procedural Behavioral Specification of Methods:

```
updateReport(contents) {
        openReport.contents = contents;
}
```

Class: Alarm

Interface Contract:

- Invariant: The Alarm is installed and connected to the system.

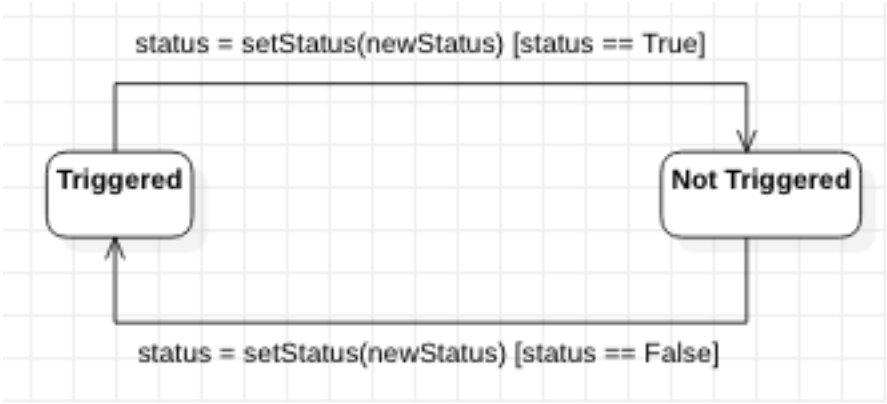
getStatus()

- Precondition: The Alarm has a current status.
- Postcondition: Returns the current status of the Alarm.

setStatus(newStatus)

- Precondition: The Alarm has a current status.
- Postcondition: The Alarm's status is updated to the newStatus.

UML State-chart:



Procedural Behavioral Specification of Methods:

```
SetStatus(newStatus)
{
        status = newStatus;
}
```

Class: Lock

Interface Contract:

- Invariant: The Lock is installed and connected to the system.

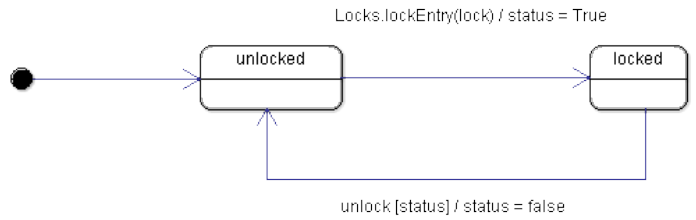
isLocked()

- Precondition: The Lock has a current status of its lock status.
- Postcondition: The Lock's current status is returned.

unlock()

- Precondition: status is currently true.
- Postcondition: status is now set to false.

UML State-chart:



Procedural Behavioral Specification of Methods:

```
unlock():
        if (status):
                status = False
```

Class: ReportController

Interface Contract:

- Invariant: There is an open Report.

getReport(reportID)

- Precondition: A Report exists with the given reportID.
- Postcondition: The openReport object is equal to the Report with the given reportID.

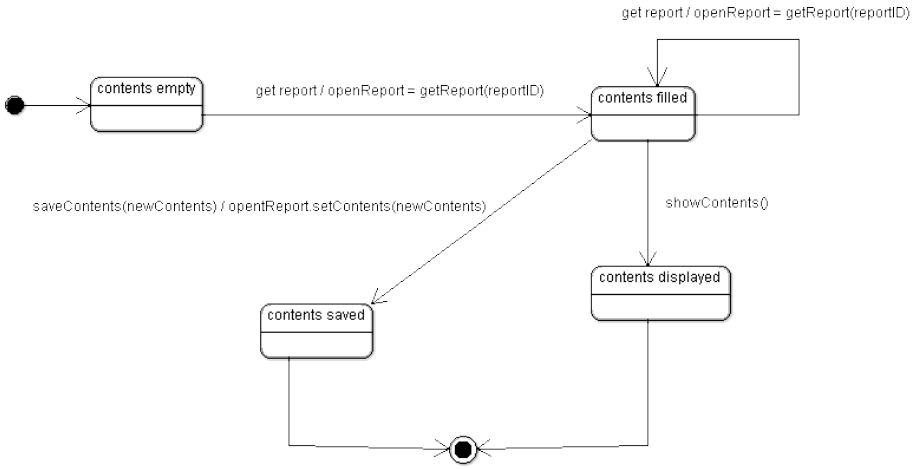
showReport()

- Precondition: There is a Report loaded.
- Postcondition: The contests are displayed to the interface.

saveContents(newContents)

- Precondition: The contents in the controller differ from contents of the Report with the given reportID.
- Postconditions: The contents in the Report have been updated to the new contents.

UML State-chart:



Procedural Behavioral Specification of Methods:

```
getReport(ReportID)
{
        for(report : reportList)
        {
                if (report.ReportID == ReportID)
                {
                        return report;
                }
        }
}
```

Class: Setting

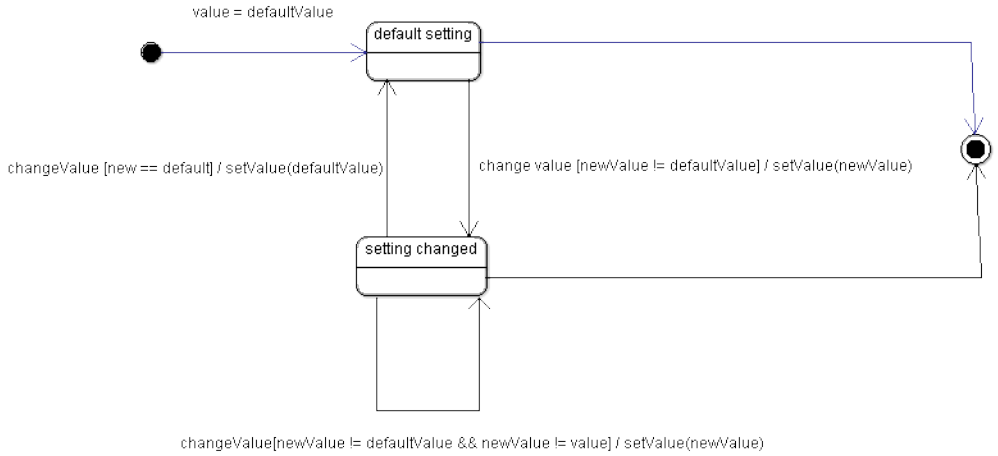
Interface Contract:

- Invariant: The Setting is being applied to the system.

setValue(newValue)

- Precondition: The Setting has a current value.
- Postcondition: The Setting's value has been changed to the newValue.

UML State-chart:



Procedural Behavioral Specification of Methods:

setValue(newValue):

value = newValue

Completeness, Consistency, and Quality

Completeness

Arming – there is an arming controller handling the arming of the system. It contains a changeAlarmStatus method that will activate an alarm if it is listening for one. Similarly, when armed the controller will initiate an entry locking sequence.
Locking – The arming controller has a lockEntries method that runs through each lock in the Locks set and locks it. Locks can be locked or unlocked individually through manual turning of the lock. Otherwise, through lockEntries, all of the locks are handled together.
Sensor readings – every sensor has an issue it is constantly trying to detect via a looping detectIssue method. Gas leak sensors are checking that the currentLevel stays below a permitted level, motion sensors listen for motion, smoke detectors wait until smoke is detected, and cameras wait until something shows up on its feed.
Report – Reports written for issues (or for status updates) by sensors should be stored and readable by the home-owner. These reports should be viewable and editable. The report controller allows these functions. The reports are stored in the report list.
Call for help – when an issue is detected, after appropriate steps are followed, an emergency service should be called to assist the home-owner. The sensor has a detectIssue method, and the emergency controller uses this to call for help after verifying that help is needed by the home-owner. After calling for help, the emergency controller verifies that help has arrived.
Feeds – When the home-owner wants to see what a sensor is reading; they can choose to view a feed for the sensor. Through the interface, the home-owner requests the feed for a sensor. The feed controller creates a feed for a specific sensor. Then, the feed itself has a showFeed method that sends the created feed to the interface for the user to view.

Consistency

Arming – The system arming is consistent behavior. The alarm has highly cohesive behavior. It cannot be changed by anything besides the changeAlarmStatus method. This is directly handled by the arming controller. The changeAlarmStatus will always flip the status of the alarm. This reduces variability within the arming protocol.
Locking – Similarly, the locking system is consistent. The locks can only be unlocked manually. Physically, if the lock is already unlocked, then it cannot be unlocked. The lock just detects that it is being unlocked and changes the status. For the locking protocol, since it is always handled the same way regardless of the initial status of the locks. Since all the locks can behave as a single unit, variability is reduced in this case as well.
Sensor readings – Each sensor has its own way of ensuring that they work consistently. The smoke detector has a wait time that allows it to be sure that there is an actual issue and it is not a false reading. This prevents problems from being caused by emergency services being called too soon. The gas leak detector is constantly checking the current level with a standard permitted level. This allows there to be a constant that controls when the issues are detected and thus when statuses change. Camera and motion detectors work similarly. Whenever there

is movement detected for them, they detect issues. However, since they will detect more issues than the other sensors, that does not automatically call for help. Instead, they consistently report it. Then, if the home-owner decides that help should be called, the emergency controller handles the rest.

Report – Report consistency is ensured through reportID's. Since each report has a reportID, this allows a specific report to be stored uniquely. Altering a report (through the saveContents method) edits in in place. The sensors call the updateReport method which utilizes this functionality. Hence there is only one, consistent method for editing the contents of a report.

Call for help – Since emergency contacts are saved under account data, there is a single store for the emergency contact information. Thus, the same contacts will be called each time a specific issue comes up.

Feeds – Feeds are dependent on the sensor that they are made for (using a specific serialID). For a specific type of sensor, a feed will be built the same. The only thing that will change between instances of a feed for the same sensor are the data that the feed displays. This allows consistency for all feeds of the same sensor.

For all these system intentions, we ensured that the methods in the collaboration diagrams is in the design class diagram. As the model has evolved, we checked that there was strong transformational accuracy between components. These methods from the collaboration diagram come from messages that we used in our detailed system sequence diagrams for the use cases that we designed. To help guarantee this accuracy, statecharts were built to express the states and activities corresponding to the methods.

Design Quality Criteria

For most of our classes, we have a very low number of **methods per class (MPC)**. This reduces complexity and reduces the amount of testing required to ensure correctness. Also, the **average method complexity (AMC)** is low. Most methods are simple accessors and mutators. This keeps the system easy while allows for good object-oriented practice through encapsulation.

Since the deepest inheritance tree level is one, the class behaviors are much more predictable due to the **Depth of Inheritance Tree (DIT)** and **Inheritance Dependencies** criteria. Since the only tree is not deep, this reduces design complexity considerably.

We used many of the size metrics to allow reduced complexity in many classes. Most methods only require zero or one parameters. Also, since many of them are for accessing and mutating, the number of operators and operands used per method is low. Most classes have few member variables to simplify them and keep them cohesive.

The system as a whole is designed to reduce coupling by adding controllers and following pure fabrication guidelines. Doing this allows good design quality as it reduces complexity. These controllers help split up system responsibility to the classes that best represent them. This helps reduce the risk of failure due to coupling issues.