SafeHome System Architectural Model

By Timothy C. Lethbridge, in conjunction with Roger Pressman

Version 7.01, July 2004

1. Introduction

This document describes the design of the SafeHome system. Many parts of the SafeHome design, and the process used to develop it, are covered as a running example in Roger Pressman's book "Software Engineering: A Practitioner's Approach, 6th Edition" © McGraw-Hill, 2004. We will refer to the book simply as *SEPA*.

The purpose of this document is to assist those who are reading SEPA and wish to see a more detailed design document describing SafeHome. It can, for example, be used by faculty who wish to explore the ideas in SafeHome more deeply with their students or by students who would like to study a more detailed example of the system architectural model.

Accompanying this document is a file containing a UML model that can be loaded into the ArgoUML, an open source modeling tool. The version used to develop the model is ArgoUML 0.16, which is based on UML 1.3[1]. ArgoUML also exports XMI, a format for interchange of UML models among tools, so an XMI version of the model is also provided. ArgoUML is available at argouml.tigris.org and runs on any platform that has a Java implementation. We chose ArgoUML simply because it is easy for anyone to obtain – it is not as complete and as bug-free as certain commercial tools, but is good enough for our purposes (and we have found that no tool behaves exactly the way we would like).

A software model can be produced at many different levels of abstraction. The model presented here is at an intermediate level: It includes diagrams relating the important hardware units, classes, and states.

The model presented does not correspond completely with the version of SafeHome described in SEPA. SEPA describes snapshots of a design in progress, whereas this document represents a subsequent and more complete iteration of SafeHome. We have, for example, added details that were not discussed in SEPA, but are necessary to explain how the system as a whole works.

In accordance with modern software engineering practice, the design presented here should be considered an iteration, and *not* the 'final release'. We invite critique and suggestions for improvement.

[1] See appendix 1 for a discussion of the version of UML used.

2. Quick overview of SafeHome

SEPA describes much of the background of the SafeHome product line; we suggest you read the relevant sections of the book before proceeding. We will, however, quickly set the scene:

The SafeHome company has developed an innovative hardware box that implements wireless Internet (802.11) connectivity in a very small form factor (the size of a matchbook). The idea is to use this technology to develop and market a comprehensive home automation product line. This would ultimately provide not only security functions, but also would enable control over telephone answering machines, lights, heating, air conditioning, and home entertainment devices. The first generation of the system, described here, will only focus on home security since that is a market the public readily understands.

3. Key actors and use cases

When fully developed the system envisioned by the SafeHome marketing team will implement hundreds of use cases. As a first step in development, however, we will identify the use cases that should be available in the first few releases – these will provide very basic security functionality.

Figure 1 shows that there are there are two main actors (roles played by users of the system): HouseholdUser and ConfigurationManager. The latter is a sub-actor of the former since ConfigurationManagers can do everything that HouseholdUsers can.

The Arm System and Disarm System use cases both require the actor to specify an activation code. The latter is shown as an *inclusion* use case. The use cases performed by the Configuration manager require a more sophisticated Log-In use case to be performed.

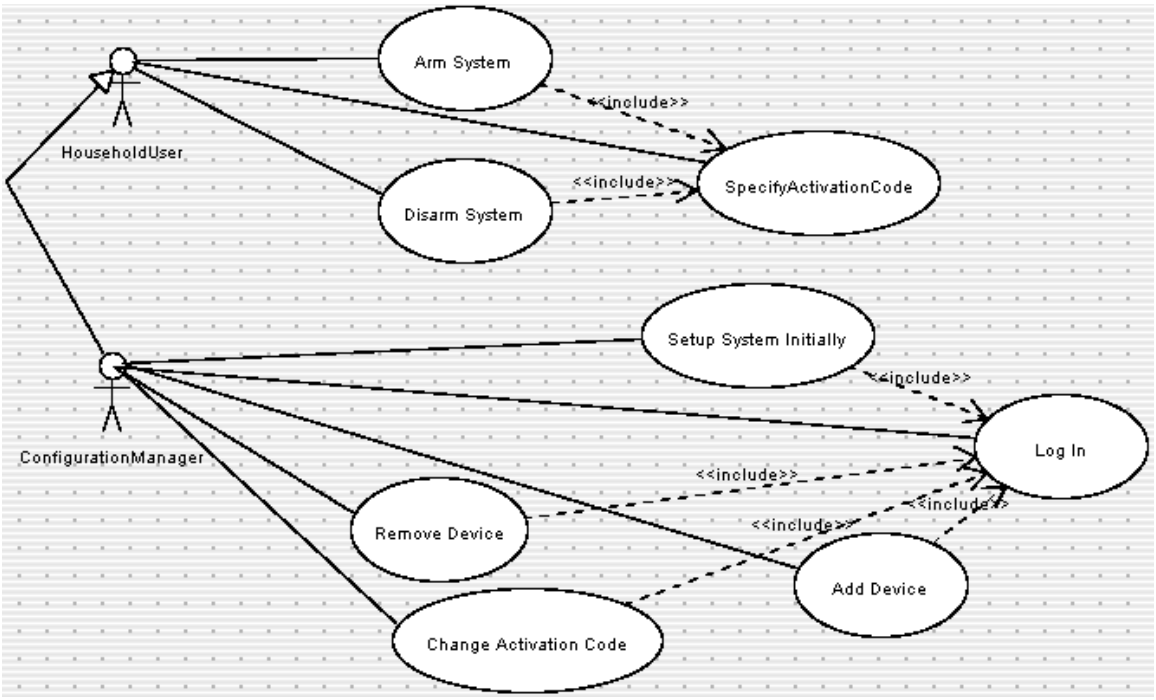

Figure 1: Use cases for release 1 of the system

Figure 2 shows additional use cases that will need to be implemented in order to provide the functionality of changing configurations of the system. For simplicity we have not shown the fact that each of these also includes the Log-In use case.

There will also be a set of use cases for designing a room layout. These are not included for the time being.

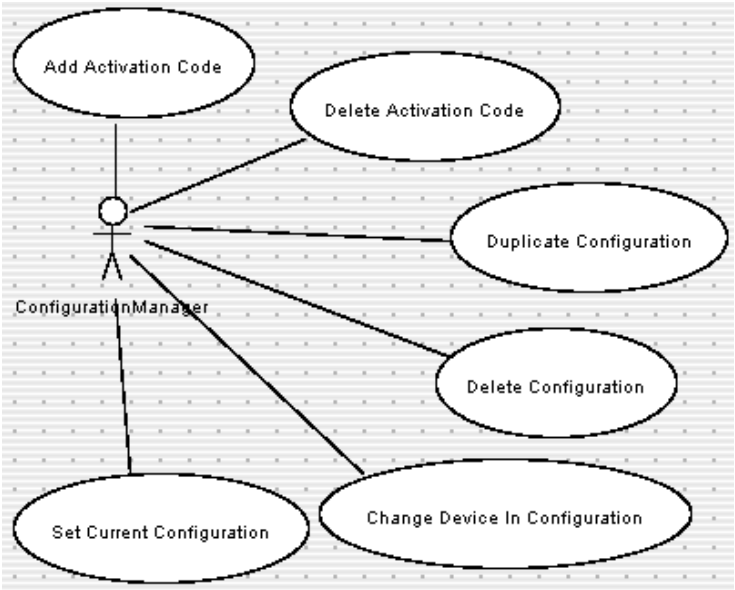

Figure 2: Use cases for release 2 of the system

4. Packages for organizing the model

The SafeHome model is arranged into a hierarchy of packages to help organize it. Figure 3 shows the packages. Each of the elements in the remaining sections of this document is arranged into one of the packages shown. There is also a conceptual outer package, simply called SafeHome, that includes all of them.

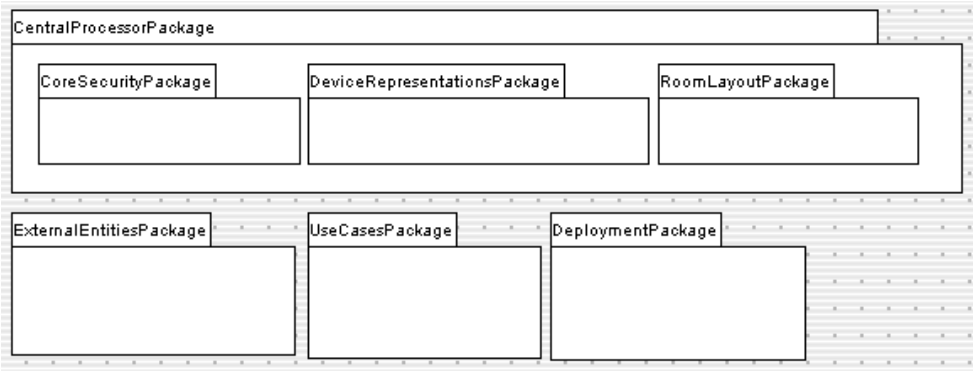

Figure 3: Package diagram for the SafeHome model

5. Hardware description.

Before understanding the software architecture it is often best to first understand the hardware on which the software will run and with which it will interact. The hardware environment for SafeHome is as follows:

- **Central processor**: There is a central processor (CP) located on the customer's premises. The CP software can run on any computer with an external broadband Internet connection or a modem, as well as an 802.11b wireless card. The CP uses its broadband Internet connection and/or modem to communicate with the SafeHome corporate site, and with a monitoring company (that can call emergency services as needed). If both broadband and modem are installed, they can act as backups for each other. The CP also acts as a wireless Internet base station for communication with the devices described below. The CP could run on any home PC; we assume however, that to ensure the integrity of the security system, the software would actually run on a dedicated computer – so crashes of other programs, viruses, etc. cannot circumvent the security. This dedicated computer would have an uninterruptible power supply, but would lack such things as a video card, keyboard and screen, commonly found in a home PC.
- **Sensor and actuator devices**: There are a variety of devices that communicate with the CP. Some of these are *sensors* (e.g. motion sensors placed internally and externally, sensors that detect whether a door or window is open, fire detectors, smoke detectors, carbon monoxide detectors, basement water detectors, etc.).

Some are alarm signalers (siren, flashing light, etc.). Some are cameras that can send digital pictures to the CP, and can be panned or zoomed. It is the intent of the company that the line of devices would be expanded in the future, so the interface with the CP needs to be flexible to allow for innovation. The devices need a power supply, and may have battery backups – details such as that are out of the scope of this document. The main thing to know at this point is that a significant number of devices can be purchased, installed physically and then configured with the software system so that the software system can communicate with them.

- **Special Hardware Control panels**: These are hardware devices that provide a simple user interface to the system. Like the devices described above, they communicate with the CP wirelessly. They allow for such basic functions as arming and disarming the system. Normally there would be one in the home near the front door, but there could be others (e.g. at other exterior doors or in a bedroom).
- **Web browser**: The control panel described above is used for the simplest of operations. However, in order to access the full functionality of SafeHome, its users must connect to the CP via a web browser. The CP runs a web server, accessible over its wireless connection (or through the SafeHome corporate web site); this provides the full user interface (UI) capability for the system. The web UI is a superset of the capability available on the hardware control panels.
- **SafeHome corporate web site**. Users who are traveling can have full access to their home system by connecting to the SafeHome corporate web site. This site routes all its communication to the CP located in the client's home. Direct external connection to the CP is not allowed to help prevent denial-of-service attacks and other forms of hacking.

Communication among all the above components is heavily encrypted to enhance security. Details of the encryption layers are not discussed here.

Figure 4 below (Deployment Diagram 1 in the model) illustrates these hardware elements. It also illustrates the main software components that will exist in the system. The web server and web browsers are generic; the main software components then are the Control Software (described below), and the device control software on each device (not described here).

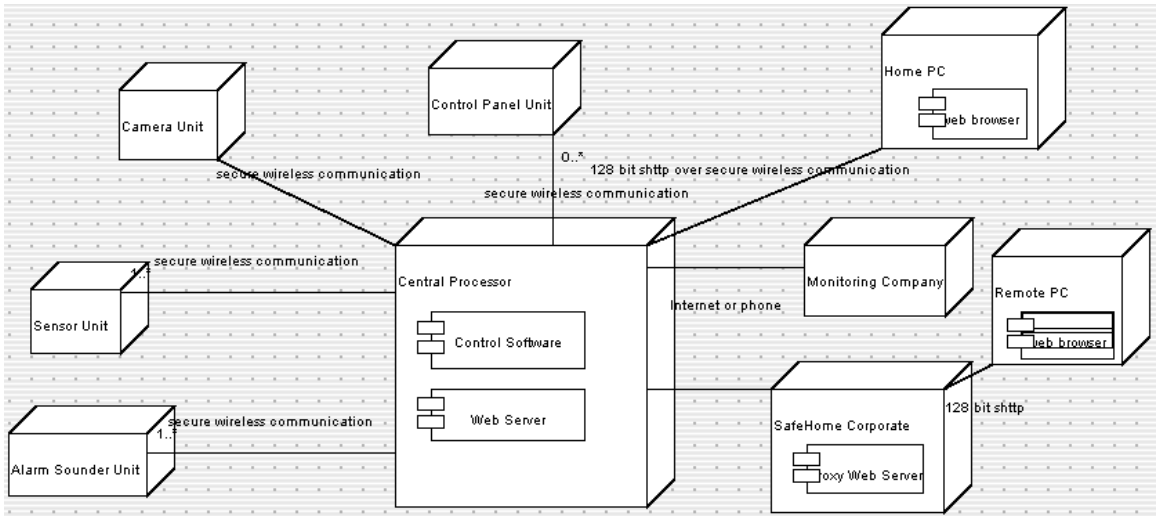

Figure 4: Deployment Diagram for SafeHome

4. Control Software Classes

Figure 5 shows the core class diagram of the Control Software component. The classes in this diagram are described below.

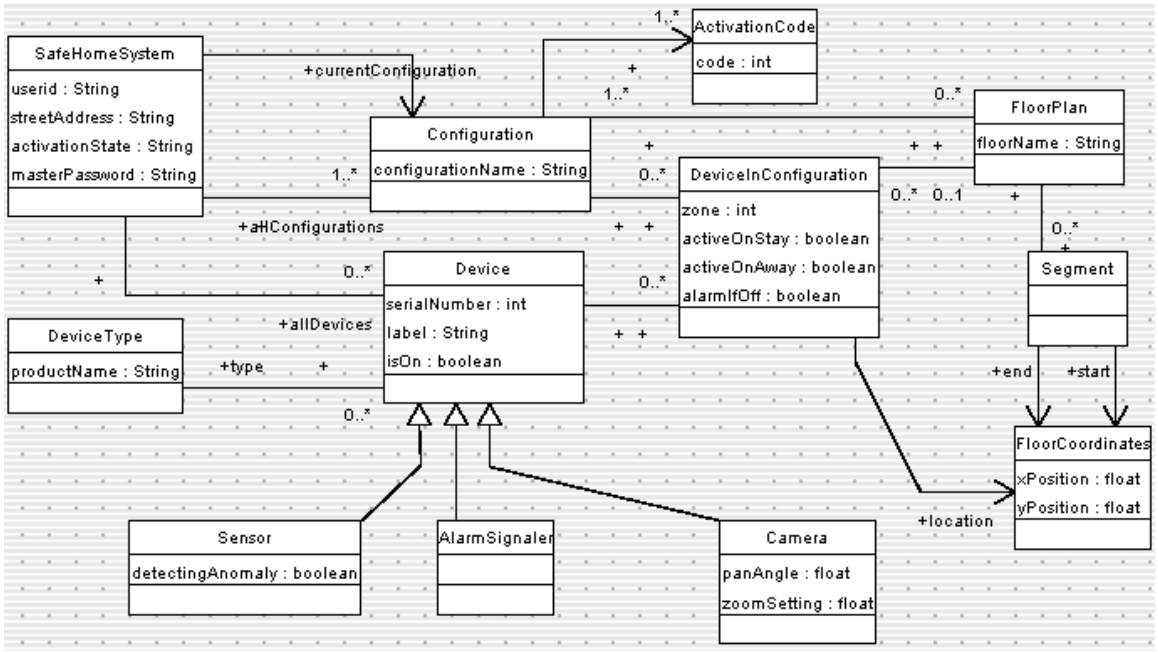

Figure 5: Class diagram for the Control Software of the SafeHome Central Processor

SafeHomeSystem: The core singleton class (only one instance exists)

- **userid**: Entered when logging into the system through a web browser

- **streetAddress**: Street address of the home. Used when communicating with the monitoring company to identify the property
- **activationState**: The state of the system as a whole. Discussed later.
- **masterPassword**: A password that must be entered

Configuration: A setup of the system with various Devices (in zones), FloorPlans, and ActivationCodes. There will always be at least one configuration – a configuration named 'default' is created when the system starts. This will contain an activation code 9999 and all devices initially added to the system. Additional configurations can be created (Duplicate Configuration use case) to allow the homeowner to experiment with the system, set up temporary configurations (e.g. when guests will be present and need their own activation codes) etc. There is an association from SafeHomeSystem that identifies the current configuration; this is changed during the Set Current Configuration use case.

- **configurationName**: A name given to the configuration by the person setting up the configuration. Can be changed at any time.

ActivationCode: Contains a simple integer identifying a code that is typed to arm or disarm the system (Specify Activation Code use case). Different people can be given different codes, e.g. to allow a cleaner to enter temporarily (Add Activation Code use case). A future extension of the system would be to identify the time period during which certain codes are active.

DeviceType: Maintains information about each type of device that may be installed in the system. Each Device has a DeviceType – information that is common to all devices of the same type is kept here.

Device: A representation of particular piece of hardware installed in the system. A device is installed by telling the user interface that it exists (specifying its serial number) and then powering it up (the Add Device use case). The software should then be able to detect its presence wirelessly.

- **serialNumber**: The number printed on the back of the device, and permanently recorded in the hardware of the device. Cannot be changed. Unique for each device manufactured by SafeHome.
- **label**: A name assigned to the device by the user when the device is being installed. May be left blank (although this would make the system less informative) and may be changed at any time. The label might, for example, describe symbolically the location of a sensor or camera.
- **isOn**: True if the device is currently being detected wirelessly.

Sensor: Represents a device that should trigger a reaction by the system if some condition becomes true (a door is open, CO is detected, water is detected, motion is detected, fire is detected). There can be several subclasses.

- **detectingAnomaly**: True if the sensor is detecting the undesirable state it is designed to detect.

AlarmSignaler: Represents a device that will sound an alarm. There can be several subclasses.

Camera: Represents a camera that can send images to the system, and can be panned and zoomed.

DeviceInConfiguration: An association class between Configuration and Device. Represents certain parameters set for that device in a particular configuration. These parameters are changed in the Change Device In Configuration use case.

- **zone**: A number from 0 to 5 that can be defined to help the user understand where an emergency is occurring. This can be used to divide the house into up to five zones (e.g. outside, basement, living room, eating areas, upstairs). The default is zero, meaning undefined zone. The sound of an the alarm depends on the zone of the sensor that triggers the system (zone 0 = continuous sound, zone 1 = evenly spaced beeps, zone 2 = pairs of beeps, etc.) An AlarmSignaler in zone 0 will always sound. An AlarmSignaler given a numbered zone will only sound if a sensor in that zone triggers the alarm.
- **activeOnStay**: Determines whether the device will be used when the system is in 'Stay' state. This is discussed more later.
- **activeOnAway**: Determines whether the device will be used when the system is in 'Away' state. This is discussed more later.
- **alarmIfOff**: Determines whether the system should consider it an anomaly if the wireless signal from the device cannot be detected (e.g. if a burglar has disconnected it). The default is true. The homeowner may want to set this to false for some devices – e.g. they may want to allow certain cameras to turned off from time to time.

FloorPlan: Part of an optional feature of the system. The homeowner may set up several floor plans that can be looked at visually in the web interface – there would be one for each floor of the house. These can be used, for example, to help him or her understand where a camera is pointing, etc. The floor plan has a set of devices (each with its own FloorCoordinates) and a set of Segments (representing doors, walls, etc.).

- **floorName**: The name given by the user to the floor (e.g. 'ground floor', 'upstairs')

Segment: A Wall, Door or Window in a floor plan. Each appears differently visually when drawn in the user interface.

FloorCoordinates: Specify the location (in meters) from the top-left corner of the FloorPlan (although the homeowner does not have to draw the unit to scale if he or she does not want to).

5. Activity and state diagrams

In this section we will describe aspects the SafeHome system's behavior.

Figure 6 is a simple activity diagram showing the top level behavior of the SafeHomeSystem class. When the SafeHome central processor is running, it must be able to do two things at once: Perform its main security monitoring functions and respond to configuration changes. Figure 6 shows that these 'Monitoring' and 'Configuration' activities are conceptually concurrent – any part of either may overlap the other. For example, the system could be armed and detecting burglars at the same time as the homeowner is logging in from some external location via the web to change the active configuration (e.g. to add an ActivationCode so a friend can enter the house).

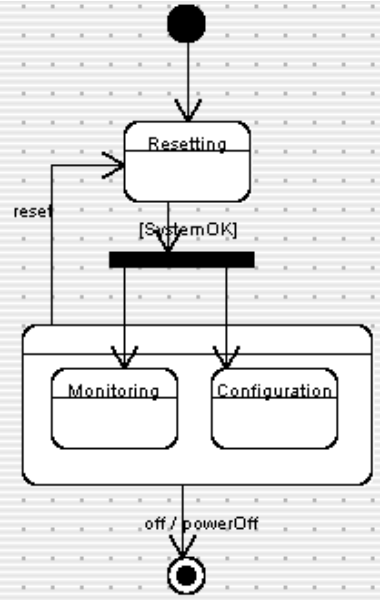

Figure 6: Top level activity diagram

Figure 7 describes the behavior of the SafeHomeSystem class during the Monitoring activity of Figure 6. There are three possible values for its activationState attribute: CheckingSystem, Disarmed and Armed; the latter two are substates of NormalOperation – in which the system spends most of its time. The system toggles backwards and forwards between Disarmed and Armed in response to user actions. Note that Figure 7 does not model the user interface – this is kept quite separate, as is good practice in software engineering and is discussed in the context of Figure 9. The successfulActivation and successfulDeactivation events are triggered by the user interface.

Note that in Figure 7, after CheckingSystem is complete, the system transitions to a symbol marked H*. This is the *deep history* symbol; it means that after resetting, the system will go back to doing what it was doing before (i.e. it goes back to being armed or disarmed). This is necessary to prevent the reset process from circumventing security.

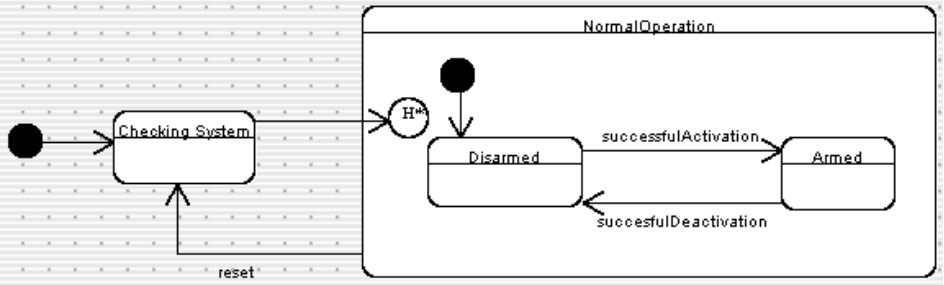

Figure 7: Behavior of the SafeHomeSystem class during its Monitoring activity

Figure 8 shows details of the Armed state. In this state the system has to respond to sensors by triggering alarms. It starts off in the No Sensors Triggered substate of the Nothing Unusual substate. Everything is normal while it is in this substate.

If a motion detector detects motion, no alarm is immediately sounded: The system requires more than a short period of motion to sound an alarm; the motion could be caused by the homeowner coming home and going through the process of deactivating the system at a control panel. Or the motion could be caused by a minor earth tremor or a gust of wind. However, the system does go into Motion Detector Triggered state since it needs to behave differently if the motion persists. After 45 seconds the system goes into Heightened Motion Sensitivity state. In this state, the system will respond immediately to any further motion; it will stay in this state for 5 minutes before dropping back to No Sensors Triggered state.

If any other sensor is triggered (or if a motion sensor is triggered in Heightened Motion Sensitivity) state, then the system goes into Acting On Alarm state. It starts the alarms sounding and calls the monitoring company. A timer is started on entry into this state, if no further sensors are triggered, this timer will time out after a user-configurable amount of time and the alarm will go off. This prevents a persistent false alarm from ringing indefinitely. On the other hand, if sensors continue to be triggered, the timer is reset, so the alarm will keep ringing.

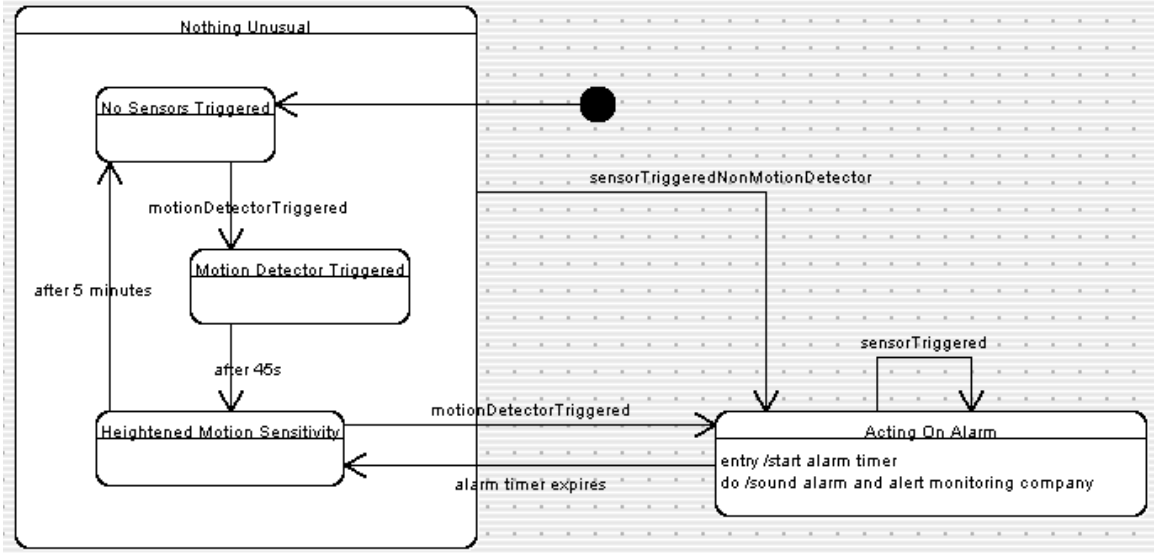

Figure 8: State diagram of the Armed state

Figure 9 shows the user interface of a model of control panel that is used to arm or disarm the system. Note that other user interfaces could be developed – in particular, there may be a web based interface that would allow remote arming of the system, or an interface that could be controlled by the monitoring company. No matter what user interface is used, it must at some point trigger the successfulActivation and successfulDeactivation transitions shown in Figure 7, so the system can be armed or disarmed respectively

The control panel has a display capable of displaying a message, along with a series of buttons: Ten digit buttons (0-9), and keys labeled ‘Arm - Stay’, ‘Arm - Away’, ‘Test’, and ‘Cancel’. In the following we will only model the events and state transitions, not what is displayed on the screen.

The control panel starts up displaying a welcome message. Then it goes into Ready For Use state if the system is currently disarmed, or Security Delay 2 state if the system is currently armed. We will explain the purpose of Security Delay 2 a bit later.

Ready For Use state means that the system is not armed; in order to perform some function, however, the user must enter a valid ActivationCode. As soon as the first number key is pressed, the control panel goes into Entering Activation Code state. The panel stays in this state as long as the user keeps pressing number keys (activation codes can be very long if the configuration manager wants).

When the user has finished entering his or her code he or she presses a function key: There are three function keys in the control panel modeled here: Arm – Stay, Arm – Away, and Test. If the user presses Test, and the activation code was correct, then the system goes into Test Alarm state for five seconds before returning to Ready For Use state.

If the user presses either Arm key after entering a valid code, the system goes into Delay To Leave state. This gives the user time to leave the house, lock the doors, etc. before the system becomes armed. The difference between the Arm keys is that the

system will be sensitive to different sets of sensors – however from the perspective of the current diagram, both keys cause the same effect.

If the user presses a function key without entering a valid activation code, the system goes into Security Delay 1 for a few seconds. The security delay prevents somebody with ill intent from trying out codes over and over again rapidly until they randomly stumble on a valid one; they always have to put up with a delay, which should make the random guessing process infeasible.

If a user presses the Cancel key while entering the activation code, the system goes back to Ready For Use state. This ensures that if a legitimate user starts typing the wrong code, they can try again immediately.

Sixty seconds after entering Delay To Leave state, the system triggers the successfulActivation event and goes into Awaiting Disarm Activation Code state. Triggering this event forces the SafeHomeSystem to become armed, as indicated in Figure 7.

Note that it is possible for there to be more than one user interface in use (another control panel, e.g. at a different door, or a web-based UI). At any time if a different UI arms the system, a transition will always be made instantly to Awaiting Disarm Activation Code state – in other words, this would happen, even if the user of this control panel was in the middle of activating the system.

Once the system is armed, the only way to disarm it is to enter a valid ActivationCode again. As soon as the user presses the first number key, the system goes into Accepting Disarm Code state. The system keeps accepting keys until a valid code has been entered, at which time the UI triggers the successfulDeactivation event and transitions to Ready For Use state.

If the user types an invalid code, he or she can press ‘Cancel’ to try again. However it will not be possible to try again for 20 seconds due to the presence of the Security Delay 2 state. This prevents rapid retries by someone trying to guess the code. Starting up the control panel while the system is armed also incorporates this security delay to account for the situation where the user unplugs and plugs in the control panel to reset it.

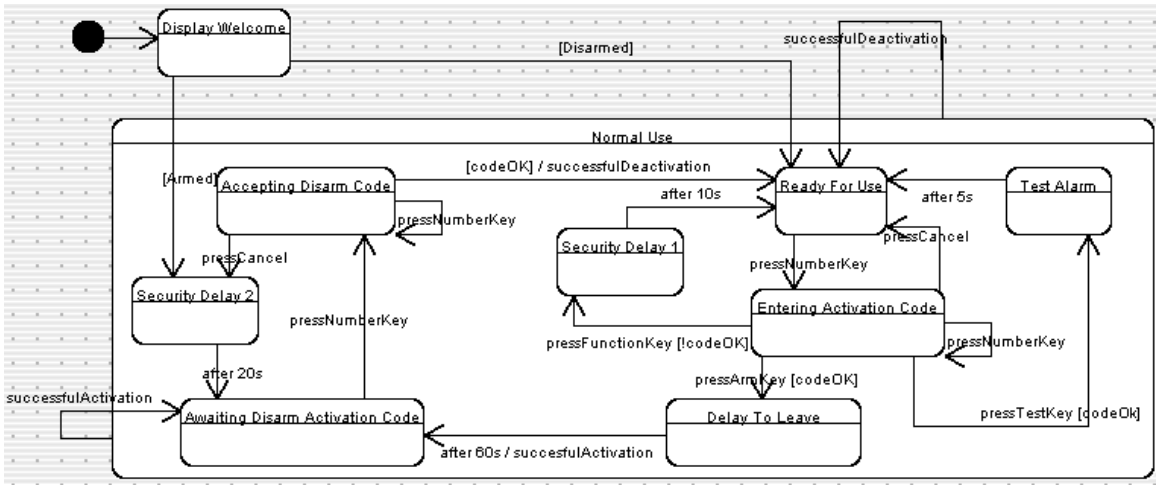

Figure 9: Behavior of the Control Panel user interface

6. Thoughts about extensions to this model

The model presented above could be extended and improved indefinitely. We leave it as an exercise for the reader to make changes. The purpose of this document has been to present the top level architecture of the system, and to help you learn about modeling, rather than exhaustively model every detail of SafeHome. However, as pointed out at the beginning, we may update this document at some future time.

Some of the issues you might consider when improving and extending the model are:

- Adding full details to all the use cases
- Modelling the configuration user interface, including the UI for building floor plans
- Fine-tuning the details of the diagrams presented above. For example, in Figures 8 and 9 there are several fixed delays built into the system. It is not really good software engineering practice to hard-code values like this into the system. These delays should really be parameters that can be set in a configuration, and therefore adjusted by the user. This would mean changing the class diagram (Figure 5) as well.
- Modelling the aspects of the system that have to do with controlling the cameras and obtaining images from them (some of this is discussed in the SEPA book).
- Adding operations to all the classes – and in particular, building an API (set of public operations) that can be accessed by the user interfaces.

Appendix 1: Version of UML used in this Model

UML 1.3 was used in this model. More specifically, the subset of UML 1.3 supported in ArgoUML 0.16.

As of mid-2004 UML has evolved somewhat: The most recent version is 2.0. However, there are no widely available tools that support UML 2.0 properly as of the time of writing.

For the purposes of learning UML there is relatively little in the SafeHome model that would need to be changed for UML 2.0. The minor differences we are aware of are as follows:

- All the multiplicities in the class diagrams that are left blank would have to be shown explicitly as '1'. This change was in fact present in UML 1.4, but is not implemented in many tools, such as ArgoUML 0.16.
- The symbol for a UML component has changed somewhat in UML 2.0