|  |
| --- |
| Carleton University |
| House Alarm System |
| SYSC 4805 Project Report |
|  |
| Alexander Craig Alexander Dinardo  Michail Greshishchev  Michael Wright |
| **April 5th, 2011** |

|  |
| --- |
|  |

Contents

[1 Introduction 4](#_Toc289681746)

[1.1 Problem Statement 4](#_Toc289681747)

[1.2 Contribution of Team Members 4](#_Toc289681748)

[2 System Requirements and Use Case Models 5](#_Toc289681749)

[2.1 System Requirements 5](#_Toc289681750)

[2.1.1 Functional Requirements 5](#_Toc289681751)

[2.1.2 Non-functional Requirements 5](#_Toc289681752)

[2.1.3 Assumed non-functional Requirements 5](#_Toc289681753)

[2.2 Use Case Model 5](#_Toc289681754)

[2.2.1 Use Case Diagram 6](#_Toc289681755)

[2.3 Use Case Descriptions 6](#_Toc289681756)

[2.3.1 Arm System 7](#_Toc289681757)

[2.3.2 Disarm System 7](#_Toc289681758)

[2.3.3 Enable System 8](#_Toc289681759)

[2.3.4 Disable System 8](#_Toc289681760)

[2.3.5 Run Self Test 9](#_Toc289681761)

[2.3.6 Enter Password 10](#_Toc289681762)

[2.3.7 Notify Break-In 10](#_Toc289681763)

[2.3.8 Invalidate Break-In 11](#_Toc289681764)

[2.3.9 Handle Break-In 11](#_Toc289681765)

[2.4 Changes from Milestone 1 12](#_Toc289681766)

[3 Analysis and Design Documentation 13](#_Toc289681767)

[3.1 System Architecture 13](#_Toc289681768)

[3.2 Roles and Responsibilities 13](#_Toc289681769)

[3.2.1 SystemHandler 13](#_Toc289681770)

[3.2.2 Keypad 15](#_Toc289681771)

[3.2.3 Display 16](#_Toc289681772)

[3.2.4 PhoneLine 16](#_Toc289681773)

[3.2.5 CellHandler 17](#_Toc289681774)

[3.2.6 Sensor 19](#_Toc289681775)

[3.2.7 Alarm 20](#_Toc289681776)

[3.3 Concurrency Issues 20](#_Toc289681777)

[3.4 Advanced Features 21](#_Toc289681778)

[3.4.1 Hierarchical State Machine – System Handler 21](#_Toc289681779)

[3.4.2 Passive Class – PeripheralIdentifier Class 21](#_Toc289681780)

[3.4.3 Dynamic Configuration – CellConfiguration Class 22](#_Toc289681781)

[3.5 Changes from Milestone 2 22](#_Toc289681782)

[4 Testing 24](#_Toc289681783)

[4.1 Testing Strategy 24](#_Toc289681784)

[4.1.1 UnitTestSystem 24](#_Toc289681785)

[4.1.2 UnitTestCellHandler 24](#_Toc289681786)

[4.1.3 IntegrationTestSystem 24](#_Toc289681787)

[4.1.4 ConcurrencyTestSystem 25](#_Toc289681788)

[4.1.5 TestProtocol 25](#_Toc289681789)

[4.1.6 TestPeripheral 25](#_Toc289681790)

[5 Unit Tests and Harnesses 25](#_Toc289681791)

[5.1 UnitTestSystem 25](#_Toc289681792)

[5.2 UnitTestCellHandler 25](#_Toc289681793)

[6 Integration Tests 26](#_Toc289681794)

[6.1.1 Use Case: Arm System 26](#_Toc289681795)

[6.1.2 Use Case: Disarm System 26](#_Toc289681796)

[6.1.3 Use Case: Enable System 26](#_Toc289681797)

[6.1.4 Use Case: Disable System 26](#_Toc289681798)

[6.1.5 Use Case: Run Self-Test 26](#_Toc289681799)

[6.1.6 Use Case: Enter Password 26](#_Toc289681800)

[6.1.7 Use Case: Notify Break-In 26](#_Toc289681801)

[6.1.8 Use Case: Invalidate Break-In 26](#_Toc289681802)

[6.1.9 Use Case: Handle Break-In 27](#_Toc289681803)

# Introduction

## Problem Statement

The objective of our project was to design, implement and test the UML-RT model of a House Alarm System.

Given a problem statement, we were to derive a set of functional and non function requirements, define the use cases and problem domain classes, and, through use case realization, define the signal protocols. After designing and implementing our House Alarm System solution in Rational Rose, we performed unit, integration, and concurrency testing.

## Contribution of Team Members

All team members contributed to domain class diagram, requirements elicitation, and overall system design.

* Alexander Craig:
  + Run Self-Test and Notify Break-In use cases specification and realizations
  + State machine implementations
  + Automated integration tests
  + Report section 3 (Analysis and Design Documentation)
* Alexander Dinardo:
* Michail Greshishchev:
* Michael Wright:

# System Requirements and Use Case Models

Given a House Alarm System problem statement, we’ve identified a set of functional and non-functional requirements. Using these requirements, we’ve generated various scenarios to isolate a minimal list of system use cases. The following section of the report details our identified system requirements and the resulting set of use cases.

## System Requirements

The following system requirements were identified through analysis of the problem statement.

### Functional Requirements

* The system must allow the user to enable and disable the House Alarm System.
* The system must allow the user to arm and disarm the House Alarm System.
* The system must sound alarm upon detection of intrusion.
* The system must place a phone call upon failure of break-in invalidation.
* The system must provide an interface to facilitate multi digit password entry.
* The system must prompt for password to perform Disable, Arm and Disarm actions.
* The system must allow the user to invalidate a break.
* The system must perform self tests on sensors and alarms.

### Non-functional Requirements

* The system shall be verbose.
* The system’s configuration shall be stored in persistent memory.
  + The system shall be configurable for variable number of sensors and alarms.
  + The system’s emergency contact number shall be configurable.
* The system shall contain an internal alarm.
* The system’s events shall be recorded to a log.
* The system shall be compatible with various types of sensors.

The following requirements have been assumed during system implementation:

### Assumed non-functional Requirements

* The system password is 4 numeric characters, 0-9.
* The system break-in invalidation timeout is 5 seconds.
* The system will sound the alarm indefinitely, until disarmed.
* The system may not be disabled until it is disarmed.
* The system will respond to the failure of a self test as it would to detection of intrusion.

## Use Case Model

We’ve identified the following set of use cases to satisfy the requirements defined in section 1.1 System Requirements: Arm System, Disarm System, Enable System, Disable System, Run Self Test, Enter Password, Notify Break-In, Invalidate Break-In and Handle Break-In.

### Use Case Diagram

The House Alarm System has 6 system actors: The Keypad, Phone Line, Timer, Alarm and Sensor, and the Display, which outputs messages to the user. Figure 1 is a Use Case Diagram which representations the relationships between the 6 actors and our system use cases.

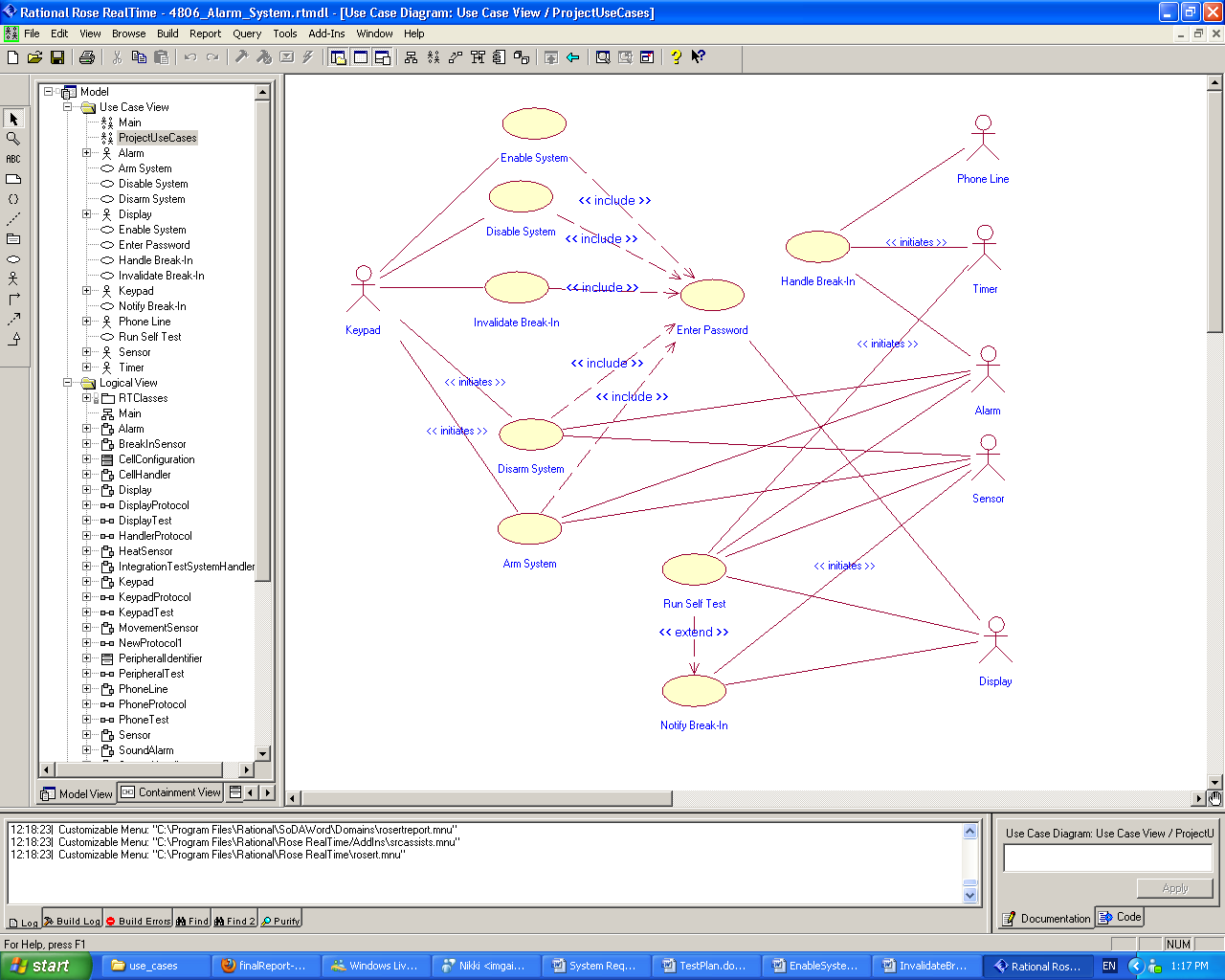


Figure .1: Home Alarm System Use Case Diagram

## Use Case Descriptions

The following section is a collection of summaries of each identified use case. A formal Use Case description for each use case is available in Appendix A.

### Arm System

The user uses the keypad interface to send the arm signal to the system. Password verification is needed for this function. The system must be enabled and must be disarmed.

The following is the expected flow of events:

1. The arm system button on the keypad is pressed.
2. The system requests a password from the user.
3. The system validates that the password is correct.
4. The system sends an arming signal to all connected sensors.
5. The system sends an arming signal to all connected alarms.

Given these steps, the system should be armed. In the event that the entered password is invalid,

1. System handler sends ‘invalid password’ message to Display.

As a result of incorrect password entry, the user is notified of invalid password and the system is not armed.

### Disarm System

The user uses the keypad interface to send the disarm signal to the system. Password verification is needed for this function. The system must be enabled and armed.

The following is the expected flow of events:

1. The disarm system button on the keypad is pressed.
2. The system requests a password from the user.
3. The system validates that the password is correct.
4. The system sends disarming signal to all connected cells.
5. The system sends a disarming signal to all connected alarms.

Given these steps, the system should be disarmed. In the event that the entered password is invalid,

1. System handler sends ‘invalid password’ message to Display.

As a result of incorrect password entry, the user is notified of invalid password and the system remains armed.

### Enable System

The user uses the keypad interface to send the enable signal to the system. Password verification is needed for this function. The system must be disabled.

The following is the expected flow of events:

1. Keypad sends ‘enable alarm system’ signal to the system.
2. The system requests a password from the user.
3. The system validates the entered password is correct.
4. The system is set to ‘alarm system enabled’ state.
5. The system sends an ‘alarm system enabled’ message to Display.

Given these steps, the system should be enabled. In the event that the entered password is invalid,

1. System handler sends ‘invalid password’ message to Display.

As a result of incorrect password entry, the user is notified of invalid password and the system remains disabled.

### Disable System

The user uses the keypad interface to send the disable signal to the system. Password verification is needed for this function. The system must be enabled and disarmed.

The following is the expected flow of events:

1. Keypad sends ‘disable alarm system’ signal to the system.
2. The system requests a password from the user.
3. The system validates the entered password is correct.
4. The system is set to ‘alarm system disabled state.
5. The system sends an ‘alarm system disabled message to Display.

Given these steps, the system should be disabled. In the event that the entered password is invalid,

1. System handler sends ‘invalid password’ message to Display.

As a result of incorrect password entry, the user is notified of invalid password and the system remains enabled.

### Run Self Test

A periodic timer triggers a complete self-test of the system. The system sends test signals to all connected alarms and sensors, and triggers a break-in alert if any alarm or sensor fails to respond with a heartbeat signal.

The following is the expected flow of events:

1. The timer sends a signal to the system to initiate the self-test procedure.
2. Until all connected sensors have been tested...
   1. The system sends a self-test signal to a connected sensor.
   2. The system signals the timer to start a new timeout of predefined length.
   3. The sensor performs a self-test routine.
   4. The sensor sends a heartbeat signal to the system to indicate test success.
   5. The system validates that a heartbeat signal is received from the sensor before the timeout expires.
3. Until all connected alarms have been tested...
   1. The system sends a self-test signal to a connected alarm.
   2. The system signals the timer to start a new timeout of predefined length.
   3. The alarm performs a self-test routine.
   4. The alarm sends a heartbeat signal to the system to indicate test success.
   5. The system VALIDATES THAT a heartbeat signal is received from the alarm before the timeout expires.

Given these steps, a heartbeat signal has been received from all connected alarms and sensors and all connected alarms and sensors have performed a self-test routine.

In the event that a test does not receive an alarm or sensor heartbeat within the timeout period,

1. The system logs the current time and identifying information of the sensor that failed to self-test.
2. If the system is armed then activate the break-in process, otherwise display the failure on the display.

As a result of test failure, at least one alarm is identified as faulty or has lost connection to the system.

### Enter Password

The Display prompts the user to enter password on the Keypad. An action requiring a password must have first been received by the system.

The following is the expected flow of events:

1. The system sends an ‘enter password’ message to Display.
2. The User enters a 4 digit numeric password into Keypad.
3. Keypad sends password message to system.
4. The system sends a signal to clear the Display.

As a result, the password has been entered.

### Notify Break-In

A break-in signal (or a failed self-test) from a connected sensor causes the system to trigger a break-in notification. The system logs the event, and starts a timeout during which a password can be entered to dismiss the break-in alert. The system must be enabled and armed.

The following is the expected flow of events:

1. A sensor sends a break-in signal to the system (or fails to respond to a self-test signal before the self-test timeout expires)
2. The systems logs the current time and identifying information of the sensor the break-in signal (or test failure) originated from.
3. The system sends a signal to the display to output an indication that a break-in alert has been triggered.
4. The system sends a signal to the display to output the identifying information of the sensor that triggered the alarm.
5. The system sends a signal to the timer to initiate a timeout of predefined length.

As a result, a break-in alert timeout is triggered by the system. In addition, a message is displayed on the system display indicating that a break-in alert has been triggered.

### Invalidate Break-In

The system waits for a pre-determined interval, during which the owner can enter the password to invalidate the alarm process started by the “break-in” signal. If password received the break-in will be nullified. The system must be enabled and a break-in time must be pending.

The following is the expected flow of events:

1. A sensor sends a break-in signal to the system (or fails to respond to a self-test signal before the self-test timeout expires)
2. The user is prompted for a password.
3. The system validates that password is correct.
4. The system cancels the pending break-in timeout.

Given these steps, the break-in is invalidated and the break-in timeout is no longer pending.

In the event the entered password is invalid, the break-in process resumes.

### Handle Break-In

The system triggers its own internal sound alarm, broadcasts trigger signals to all cells, causing all alarms in the entire house to be triggered, places a phone call to a pre-defined phone number and finally writes the event to the log. The system must be armed and enabled with a break-in in progress.

The following is the expected flow of events:

1. The timer sends a signal to the system to initiate the handle break-in procedure.
2. The system triggers internal sound alarm.
3. Until all connected alarms are triggered,
   1. The system sends a trigger system to a cell handler.
   2. The cell handler triggers its connected alarms.
4. The system places a phone call through the phone line.
5. The system logs the break in with all the relevant information.

Given these steps, the internal sound alarm and all connected alarms have been triggered, and a phone call has been placed.

In the event that a phone call fails, a phone line failure event is logged.

## Changes from Milestone 1

All non-functional requirements of the system were not yet identified at the time of completing Milestone 1. During implementation, we’ve encountered multiple undefined non-functional requirements, such as password representation or break-in invalidation timeout length.

Our original use case for enter password had an alternative flow triggered upon timeout of password entry. During implementation, we removed the timeout restriction on password entry and the user is now given an unlimited amount of time. This change was made to simplify our design as password entry timeout was not identified as a requirement.

Except for the above changes, our Milestone 1 has not been modified.

# Analysis and Design Documentation

## System Architecture

The alarm system uses a distributed architecture where every element of the system is represented by a separate active class which executes independently and communicates with the system at large purely through signal passing. These elements include the sensors, alarms, the keypad, the LCD display and the phone line, each of which is represented by an active boundary class. Additional cell and system handler active classes are used to coordinate and control signal passing between the boundary classes. This architecture closely mirrors the likely deployment architecture of the system, where processing units would be distributed throughout a large structure (i.e. each cell handler would execute on a separate microcontroller).

All system functionality is encapsulated within an instance of the top level SystemHandler. This component contains all other system components within its structure, and coordinates all input from the keypad and all output to the display and phone line. The SystemHandler would likely be deployed on the user accessible control panel in an actual deployment of the system. The SystemHandler contains a configurable number of instances of CellHandler, each of which coordinates signal passing for a single “cell,” which would represent all the sensors and alarms in a given physical region (for example, a single room). Each instance of CellHandler contains a configurable number of alarms and sensors, and is responsible for multiplexing signals from the SystemHandler to each encapsulated peripheral.

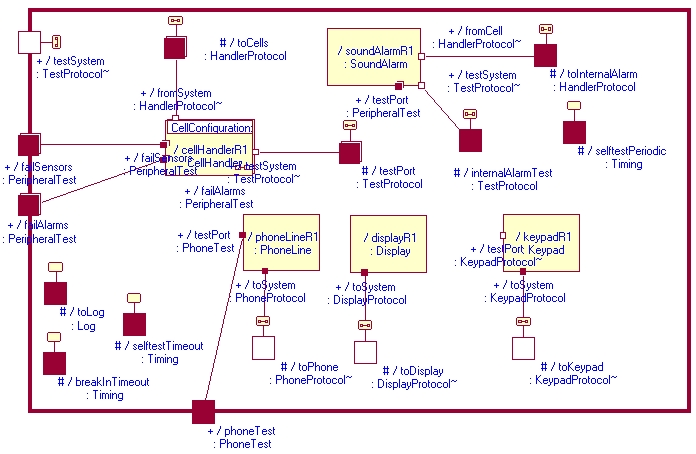
Please see the Rational Rose model for a complete class diagram of the implementation.

## Roles and Responsibilities

### SystemHandler

The SystemHandler is the top level component of the system, and is responsible for coordinating signals from all user accessible input / output devices as well as storing the overall state of the system (i.e. enabled/disabled and armed/disarmed states).

Figure .1: Structure diagram of the SystemHandler



The internal structure of SystemHandler can be seen in Figure 3.1. The SystemHandler contains a configurable number of CellHandlers which are used to relay messages to connected alarms and sensors. The multiplicity of CellHandler is configurable by setting the constant NUMBER\_OF\_CELLS in the CellConfiguration utility class. The SystemHandler communicates with instances of CellHandler using the HandlerProtocol, whose specification can be seen in Figure 3.1. This allows the SystemHandler to arm and disarm cells, as well as initiate self-tests and receive heartbeat or failure signals in response. The SystemHandler can also issue an alarm trigger signal, and receive break-in signals from CellHandlers (which are propagated from the CellHandler’s contained sensors). The SystemHandler uses a periodic timer to trigger the self-test procedure, and the period can be configured by setting the SYSTEM\_SELFTEST\_INTERVAL constant in CellConfiguration. However, it must be asserted that the SYSTEM\_SELFTEST\_TIMEOUT and CELL\_SELTEST\_TIMEOUT constants are less than the self-test period, as these represent the amount of time the SystemHandler and CellHandler respectively will wait for a heartbeat signal before assuming a component has failed. The SystemHandler also propagates most events directed toward CellHandlers to its own internal sound alarm.

Figure .2: HandlerProtocol specification

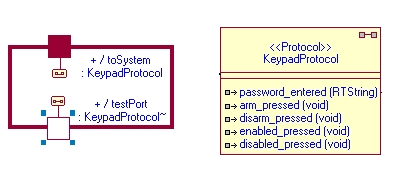


The SystemHandler also contains an instance of Keypad, Display, and PhoneLine. Each I/O device is connected to a protected port of SystemHandler using a device dependent protocol (KeypadProtocol, DisplayProtocol, and PhoneProtocol respectively). All user interaction occurs through the keypad, and signals from the keypad are processed depending on the current state of the SystemHandler. The SystemHandler stores the state of the system using a hierarchical state machine described in section 3.4.1. No separate state is stored for the “triggered” state of system (in which all alarms are ringing) because it was determined that incoming signals to the SystemHandler should be handled identically regardless of whether the system is simply armed or actively triggering (i.e. a disarm command still disarms all connected alarms and sensors, and all other signals from the keypad should be ignored).

### Keypad

The Keypad class represents the control panel that the end user interacts with, and is the only source of user input to the system.

Figure .3: Structure diagram of the Keypad capsule, and the specification of the KeypadProtocol

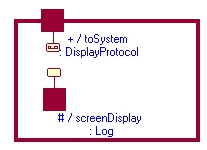


The Keypad is implemented as an internal component of SystemHandler, and has a wired port which connects to a protected port in the SystemHandler. This port is connected using the KeypadProtocol, which supports five operations: Pressing the enable / disable buttons, pressing the armed / disarmed button, and entering a password (which is passed as a string). A second wired port using the conjugated KeypadProtocol is used to inject signals into the system for testing purposes. The structure of the keypad capsule, as well as the specification for KeypadProtocol can be seen in Figure 3.3. The keypad has only a single state, and serves purely to propagate user input from the user to the SystemHandler.

### Display

The Display class represents a user viewable LCD screen in the alarm system control panel. The Display is responsible for displaying status messages and prompts to the user.

Figure .4: Structure diagram of the Display capsule

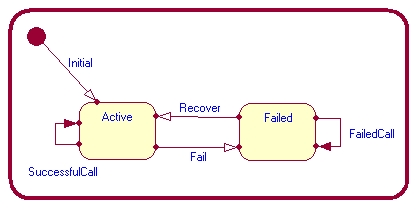


The Display class is a very simple component, with only a single state. The Display is deployed as an internal component of the SystemHandler, and communicates over a wired port using the DisplayProtocol. The structure of the Display capsule can be seen in Figure 3.4. The DisplayProtocol only supports a single signal (display\_string) which takes a string as its data type. In the current implementation, the Display class simulates display output by outputting to a standard logging port, and by prefixing all logged messages with “DISPLAY: “.

### PhoneLine

The PhoneLine class represents a connection to a phone network, and is used to place calls to the police when the system’s alarms are triggered.

Figure .5: State machine of the PhoneLine capsule

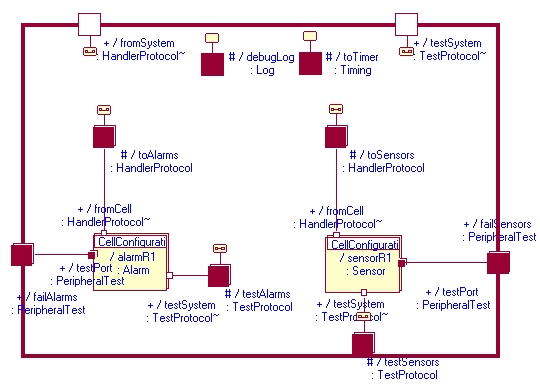


The PhoneLine class is deployed as in internal component of SystemHandler, and communicates over a wired port using the PhoneProtocol. The PhoneProtocol supports one incoming signal (place\_call) which triggers a phone call to the number specified in the string argument. The PhoneLine class has two states, “Active” and “Failed”, and a testing port which allows signals to be injected to down or recover the phone line. The state machine for the PhoneLine capsule can be seen in Figure 3.5. When a call is placed, the phone line will generate an outgoing call\_placed or call\_failed signal on the port connecting to SystemHandler depending on the current state of the phone line.

### CellHandler

The CellHandler is responsible for replicating signals from the SystemHandler to all its internal sensors and alarms, and performing a self-test procedure whenever the SystemHandler triggers a self-test. In addition, the CellHandler also propagates trigger signals from its contained sensors to the SystemHandler. In effect, it acts as a simple relay between the SystemHandler and connected peripherals (alarms and sensors).

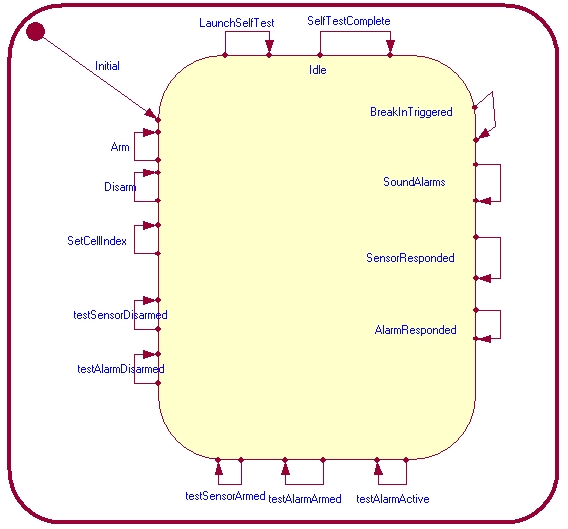
Figure .6: Structure diagram of the CellHandler capsule



The CellHandler contains a number of instances of Alarm and Sensor, whose multiplicity can be adjusted by changing the ALARMS\_PER\_CELL and SENSORS\_PER\_CELL constants respectively in the CellConfiguration utility class. The structure diagram of the CellHandler can be seen in Figure 3.6. The CellHandler can uniquely identify these peripherals by using a combination of its own cell index (which is set at initialization by the SystemHandler), and the port number on which it communicates with the given peripheral. This allows the CellHandler to generate instances of PeripheralIdentifier which are sent with break-in or failure signals to the SystemHandler to identify the specific peripheral which has detected a break-in or failed.

The CellHandler communicates with these peripherals using the HandlerProtocol (the same protocol which it uses to communicate with SystemHandler). All outgoing signals (arm, disarm, self-test, and trigger) are simply replicated to all connected peripherals whenever they are received from the SystemHandler. When a self-test signal is received, a timer is started (with a duration defined by the CELL\_SELFTEST\_TIMEOUT constant in the CellConfiguration utility class) which determines the amount of time peripherals have to respond with a heartbeat signal. If a peripheral does not respond, it is assumed to have failed, and a failure signal carrying a unique PeripheralIdentifier is sent to the SystemHandler.

Figure .7: State machine for the CellHandler capsule

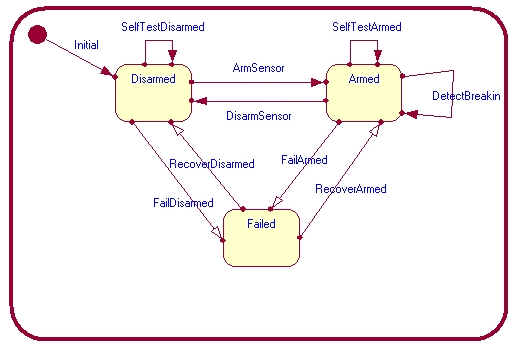


Since it acts as a simple relay between the SystemHandler and peripherals, the CellHandler only has a single state, and behaves identically regardless of the overall state of the system. The state machine of the CellHandler capsule can be seen in Figure 3.7. The CellHandler is not logically required for the functioning of the system, and references to CellHandlers could easily be replaced with direct references to peripherals at the SystemHandler level. This is reflected in the simplicity of the CellHandler state machine. CellHandler therefore exists primarily because its usage was specified in the project requirements, and possibly because an actual deployment of the system would run CellHandlers on separate distributed processors.

### Sensor

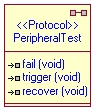
The Sensor class represents a sensor connected to the system, and is primarily responsible for propagating break-in signals to the CellHandler when a break-in is detected. Inheritance is used to allow the representation of different sensor types (through the Sensor subclasses MovementSensor, HeatSensor, and BreakInSensor).

Figure .8: State machine of the Sensor capsule



The Sensor class has three states: “Disarmed”, “Armed”, and “Failed”. The state machine for the capsule can be seen in Figure 3.8. A testing port using the PeripheralTest protocol is used to simulate the physical triggering of the sensor, as well as failing or recovering the component for testing purposes. The specification of the PeripheralTest protocol can be seen in Figure 3.9. Sensors connect to a CellHandler using the HandlerProtocol. When a sensor is disarmed or failed it will not propagate break-in signals to the CellHandler, and when it is failed it will also not respond to self-test signals with an appropriate heartbeat signal.

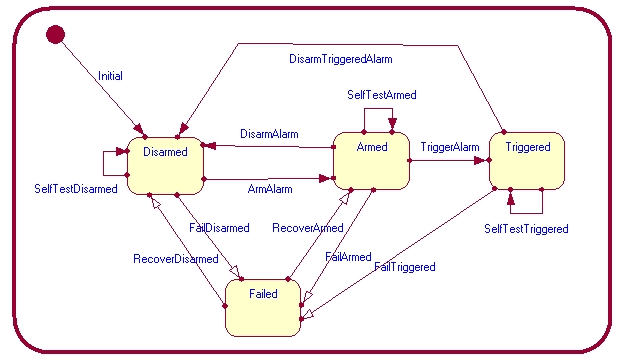
Figure .9: Specification of the PeripheralTest protocol



### Alarm

The Alarm class represents an alarm connected to the system, and is primarily responsible for propagating break-in () signals to the CellHandler when a break-in is detected. Inheritance is used to allow the representation of different alarm types, but only a single subclass (SoundAlarm) exists in the current implementation.

Figure .10: State machine of the Alarm capsule



The Alarm class has four states: “Disarmed”, “Armed”, “Triggered”, and “Failed”. The state machine of the Alarm capsule can be seen in Figure 3.10. A testing port using the PeripheralTest protocol is used to fail or recover the component for testing purposes. Incoming trigger signals from the CellHandler will be ignored if the Alarm is not in the “Armed” state. Alarms connect to a CellHandler using the HandlerProtocol. The only outgoing signals generated from Alarms are heartbeat signals sent to the CellHandler to indicate that a self-test procedure was performed successfully.

## Concurrency Issues

Concurrency was primarily a concern when ensuring that the system self-test procedure did not interfere with the usual functioning of the system. This was solved primarily by using a hierarchical state machine for the SystemHandler. Transitions relating to the self-test procedure were primarily implemented on the “Enabled” super-state, ensuring that the self-test procedure is carried out regardless of what particular sub-state of “Enabled” is in use. This is further supported by the single state design of the CellHandler. Rather than using separate states to store results of the self-test process, an array of boolean flags is used to flag responses from peripherals. This allows the CellHandler to receive and process heartbeat signals without affecting any other state.

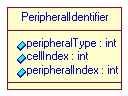
## Advanced Features

### Hierarchical State Machine – System Handler

The SystemHandler uses a hierarchical state machine to represent the current system state. After an initializing phase, the SystemHandler transitions to the “Initialized” state. This state contains the “Enabled” and “Disabled” states, as well as additional states to handle password entry. The “Enabled” state is also a compound state, and contains the “Armed” and “Disarmed” states, as well as additional states to handle password entry. System self-test functionality is implemented as self-transitions on the “Enabled” state, which allows self-testing of the system to continue whenever the system is enabled, regardless of whether the system is armed or not. Please see Rational Rose for the complete state diagram of SystemHandler.

### Passive Class – PeripheralIdentifier Class

Figure .11: Specification of the PeripheralIdentifier



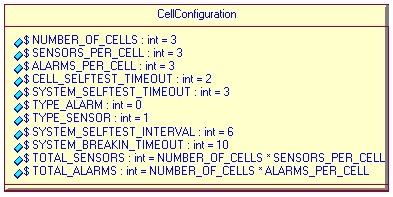
All logging and output messages use the PeripheralIdentifier class to uniquely identify individual alarms and sensors. The specification of the PeripheralIdentifier class can be seen in Figure 3.11. PeripheralIdentifier has three fields:

* cellIndex: The index of the CellHandler which generated the identifier.
* peripheralType: One of TYPE\_ALARM or TYPE\_SENSOR (defined in CellConfiguration)
* peripheralIndex: The index of the peripheral on its associated CellHandler (corresponds to the port index used to communicate with the peripheral).

**Note:** A PeripheralIdentifier with a peripheralType of TYPE\_ALARM, and a cellIndex and peripheralIndex of 999 represents the internal sound alarm of the SystemHandler.

### Dynamic Configuration – CellConfiguration Class

Figure .12: Specification of the CellConfiguration utility class



All system configuration (other than setting the system password and emergency phone number) is performed by modifying the constants in the CellConfiguration utility class. The specification of CellConfiguration can be seen in Figure 3.12. The following configuration options are available:

* **ALARMS\_PER\_CELL:** Sets the number of Alarm instances contained in each CellHandler.
* **SENSORS\_PER\_CELL:** Sets the number of Sensor instances contained in each CellHandler.
* **NUMBER\_OF\_CELLS:** Sets the number of CellHandler instances contained in the SystemHandler.
* **SYSTEM\_BREAKIN\_TIMEOUT:** The number of seconds which should elapse between a break-in being detected and alarms being triggered (effectively the amount of time the user has to disarm the system after triggering a sensor).
* **SYSTEM\_SELFTEST\_INTERVAL:** The number of seconds which should elapse between self-test cycles being triggered by the SystemHandler.
* **CELL\_SELFTEST\_TIMEOUT:** The number of seconds which should elapse after the CellHandler issues a self-test signal to a peripheral before the CellHandler should assume the peripheral has failed if no heartbeat signal is received.
* **SYSTEM\_SELFTEST\_TIMEOUT:** The number of seconds which should elapse after the SystemHandler issues a self-test signal to a CellHandler before the SystemHandler should assume the CellHandler has failed if no heartbeat signal is received.

## Changes from Milestone 2

All functional requirements of the system were complete and functioning for milestone 2, and as such no changes have been made to the model which affect end-to-end functionality. However, many additions were made to support testing of the model. Since all components of the system are deployed as internal components of SystemHandler, an encapsulating test framework does not have access to the objects and signals it requires to ensure the system is functioning correctly. To remedy this situation, a new TestProtocol was created which supports all signals that can be generated in the system, as well as additional output signals to better track system state. A port supporting this protocol was added to most active classes, and additional relay ports were setup to propagate these test signals to the external interface of SystemHandler.

Although this method was effective in testing the functionality of the system, it is the opinion of the project team that this testing method was less than ideal. Propagating signals to the top level component required modifications to the state machines of CellHandler and SystemHandler which have no clear effect on the user visible functionality of the system, and these modifications reduce the clarity of the resulting model. A more effective test framework would require a means to access the internal components of a capsule without resorting to chaining relay ports.

# Testing

## Testing Strategy

This system for testing purposes has been split into multiple levels of testing as well as types of testing and a development of distinct testing models which will focus on specific functionality. The eventual goal of the project’s testing was to provide an automated system which could be run to cover all of our use case realisations, as well as provide complete code coverage, while testing for overall system integration and providing basic concurrency tests for the system.

The system has been split into 4 separate test capsules. These capsules provide the following: unit testing for the system provided by UnitTestSystem, unit tests for the cell handler and peripherals found in UnitTestCellHandler, integration tests for the system found in IntegrationTestSystem, and for the concurrency tests found in ConcurrencyTestSystem. These all communicate with the necessary system capsules that are under test through a general purpose protocol called TestProtocol. Furthermore to enable specific changes to the system that are needed to be controlled for testing purposes, a protocol to perform these changes on peripherals called PerhiperalTest was created. All these protocols and capsules will be discussed in the following.

### UnitTestSystem

This capsule provides black box unit testing on the high level functions of the system. The purpose of this is to provide complete transition coverage for the SystemHandler capsule while at the same time provide 100% coverage of the unit cases which are related to SystemHandler. Examples of the unit test coverings are: arm/disarm the system, enable/disable the system, handling break-ins, phone lines etc.

### UnitTestCellHandler

This capsule provides white box testing to the functionality needed to provide the functionality stated for the cell handler. This test suite provides tests for arming/disarming each alarm and sensor. After those test have been completed it tests for sensors being tripped and alarms being sounded. Finally it tests for whether or not the cells are able to pass their self tests.

### IntegrationTestSystem

IntegrationTestSystem provides white box testing for the actual operation of the system. It consists of 6 testing states with their own internal test suites. They are as follows:

* EnableSystemTests: This tests that the system is able to enable itself while also ensuring it does not enable if there is a faulty password to be entered.
* DisableSystemTests: This tests that the system is able to disable itself while also ensuring it does not disable if there is a faulty password to be entered.
* ArmSystemTests: This tests that the system is able to arm itself while also arming all cells while also ensuring it does not arm if there is a faulty password to be entered.
* DisarmSystemTests: This tests that the system is able to disarm itself and its cells while also ensuring it does not disarm if there is a faulty password to be entered.
* SelfTestTests: This tests that the system is able to start and complete its system tests.
* NotifyAndHandleBreakinTests: This tests the white box tests needed to cover all possible cases of alarm use, sounding and handling.

### ConcurrencyTestSystem

This suite provides the testing for concurrent events such as when an alarm gets tripped during a transition or if the system receives a break-in signal while passwords are being entered. The goal is to ensure that the system is able to transition properly even if there is a flood of un expected signals into the system

### TestProtocol

TestProtocol is used to act as the communication protocol for our test cases. This protocol has all the signals used in the actual system operation. This allows all test suites to use the same protocol and allow the system to have the same signals be injected as would be expected over the course of standard operation.

### TestPeripheral

This protocol is used as a tool to control perhiperal states. It can arm, disarm, trigger and fail any perhiperal or cell it is used in conjunction with other tests to get the functionality that we require while maintaining system modularity.

# Unit Tests and Harnesses

## UnitTestSystem

* Arm/Disarm System with both correct and incorrect passwords
* Enable/Disable System with both correct and incorrect passwords
* Break in with password entered correctly and on time
* Break in triggered no password entered during time out
* Break in handled, phone call placed
* Break in handled phone call failed
* Failure with password entered correctly and on time
* Failure triggered no password entered during time out
* Failure handled, phone call placed
* Failure handled phone call failed

## UnitTestCellHandler

* Arm/Disarm an individual cell (loop through all cells)
* Trigger an individual cell (loop through all cells)
* Check that cells are able to respond to a timeout signals
* Tests that systems are able to respond to self test signals

# Integration Tests

### Use Case: Arm System

* Ensure post condition on the basic flow is asserted (all cells are armed) after valid password entry.
* Ensure system returns to previous state when incorrect password is entered.

### Use Case: Disarm System

* Ensure post condition on the basic flow is asserted (all cells are disarmed) after valid password entry
* Ensure system returns to previous state when incorrect password is entered

### Use Case: Enable System

* Assert happy path satisfies basic flow post condition.
* Assert invalid password entry satisfies "The entered password is invalid" global alternative flow post condition (the system remains disabled).

### Use Case: Disable System

* Assert happy path satisfies basic flow post condition.
* Ensure that use case is not executed when system is not in ready state (system must be enabled).

### Use Case: Run Self-Test

* Assert happy path satisfies basic flow post condition.
* Assert that the use case “Notify Break-In” is executed if the system is armed and a sensor fails to respond to a heartbeat.
* Assert that the use case “Notify Break-In” is executed if the system is armed and an alarm fails to respond to a heartbeat.
* Assert that failure details are displayed on the Display if the system is disarmed and a sensor fails to respond to a heartbeat.
* Assert that failure details are displayed on the Display if the system is disarmed and an alarm fails to respond to a heartbeat.

### Use Case: Enter Password

* Assert happy path satisfies basic flow post condition.

### Use Case: Notify Break-In

* Assert happy path satisfies basic flow post condition (timeout is started, message is displayed on Display, and the event is logged).

### Use Case: Invalidate Break-In

* Assert happy path satisfies basic flow post condition.
* Assert if password is not valid, “Password is not valid” alternative flow is satisfied and the USE CASE Handle Break-in is executed

### Use Case: Handle Break-In

* Assert happy path satisfies basic flow post condition.
* Assert if Phone Line has failed “Phone line has failed” specific alternative flow has been satisfied, and execution was able to resume.