# 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, the phone line, and the more abstract cell and system handler components. 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).

The SystemHandler contains a configurable number of CellHandlers. 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. 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 of 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.

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 below. 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 armed or actively triggering.

### 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.

The Keypad is implemented as an internal component of SystemHandler, and has a single 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). 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.

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 DisplayProtocol only supports a single signal (display\_string) which takes a string as its data type. In the current implementation, the Display class simply prints any incoming string message to console with a "DISPLAY: " prefix to differentiate the messages from logging messages.

### 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.

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. 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).

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 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.

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 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. CellHandler therefore exists primarily because its existence 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.

The Sensor class has three states: “Disarmed”, “Armed”, and “Failed”. 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. 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.

### 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.

The Alarm class has three states: “Disarmed”, “Armed”, “Triggered”, and “Failed”. 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.

## 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.

### Passive Class – PeripheralIdentifier Class

All logging and output messages use the PeripheralIdentifier class to uniquely identify individual alarms and sensors. 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

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 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.