autoHome Report #2

Group #1

Paul Kania

Elie Rosen

Calvin Chiu

Rohith Dronadula

Elvison Dominguez

Wayne Chang

http://autohome.mylifeiscomputers.net
https://github.com/autohomeproject

Contribution Breakdown

Responsibilities	Calvin Chiu	Elie Rosen	Elvison Dominguez	Paul Kania	Rohith Dronadula	Wayne Chang	Totals	Possible Points
UML DIAGRAMS	80%	7%	13%				100%	10
DESC OF DIAG		70%				30%	100%	10
TRACEABILITY MATRIX				100%			100%	10
CLASS DIAG & DESC	80%					20%	100%	6
SIGNATURES	100%						100%	5
STYLES				100%			100%	5
PKG DIAG		100%					100%	3
MAP HARDWARE		100%					100%	3
DATABASE						100%	100%	3
OTHER ($)	33%	67%					100%	5
ALG & DATA STRUCT		50%		50%			100%	6
APPEARANCE					100%		100%	7
PROSE DESC (+)					100%		100%	6
TESTING DESIGN			60%		20%	20%	100%	10
DOC MERGE				100%			100%	6
PROJ CORDINATION		50%		50%			100%	2
PLAN OF WORK				100%			100%	3
Totals:	16.67	16.67	16.67	16.67	16.67	16.67	100%	100

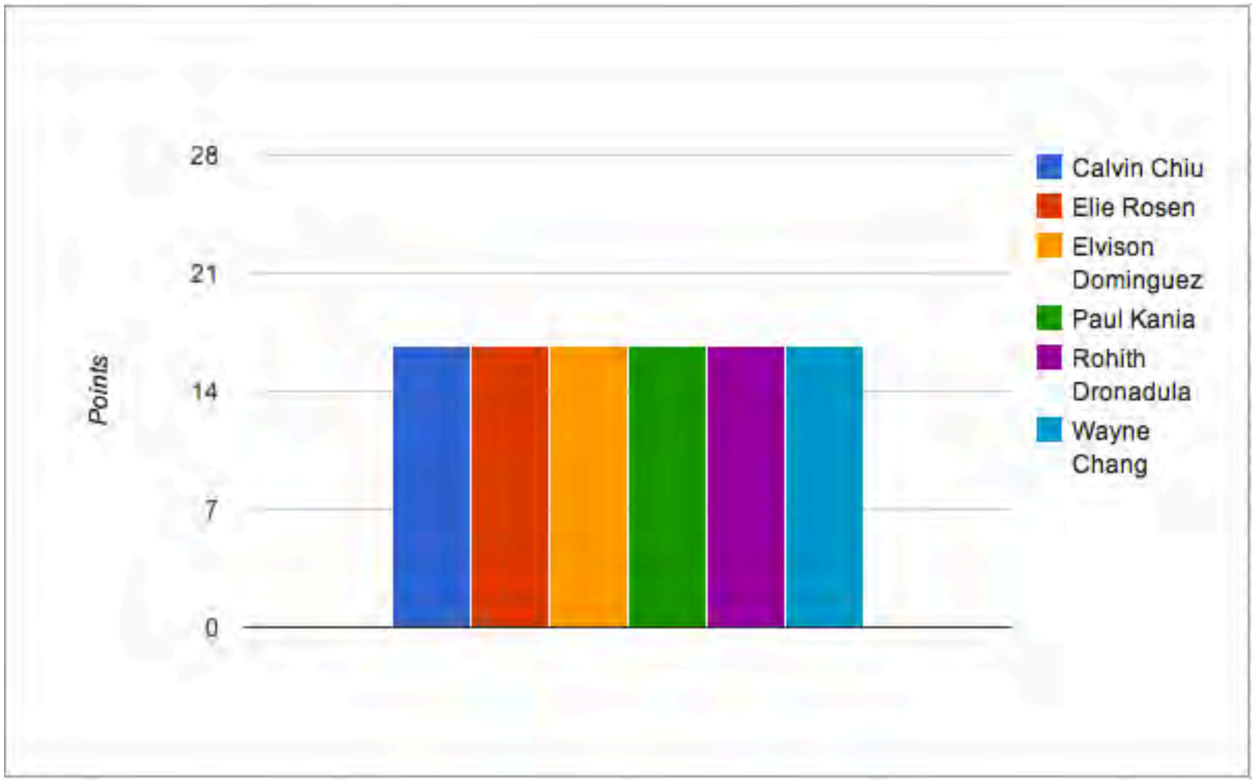


Table of Contents

1. Interaction Diagrams

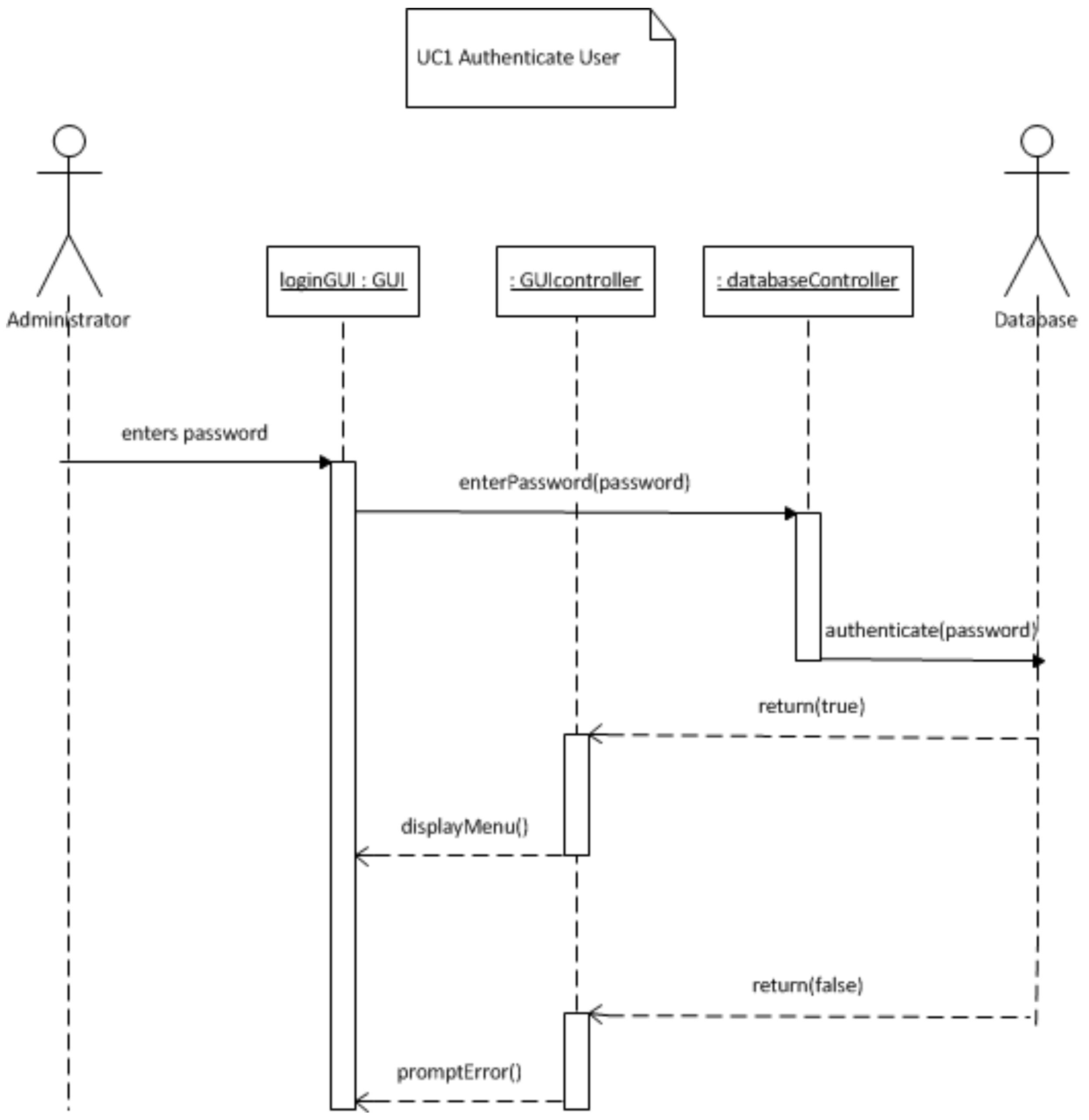


Figure 1-1 - UC-1 - Authenticate User

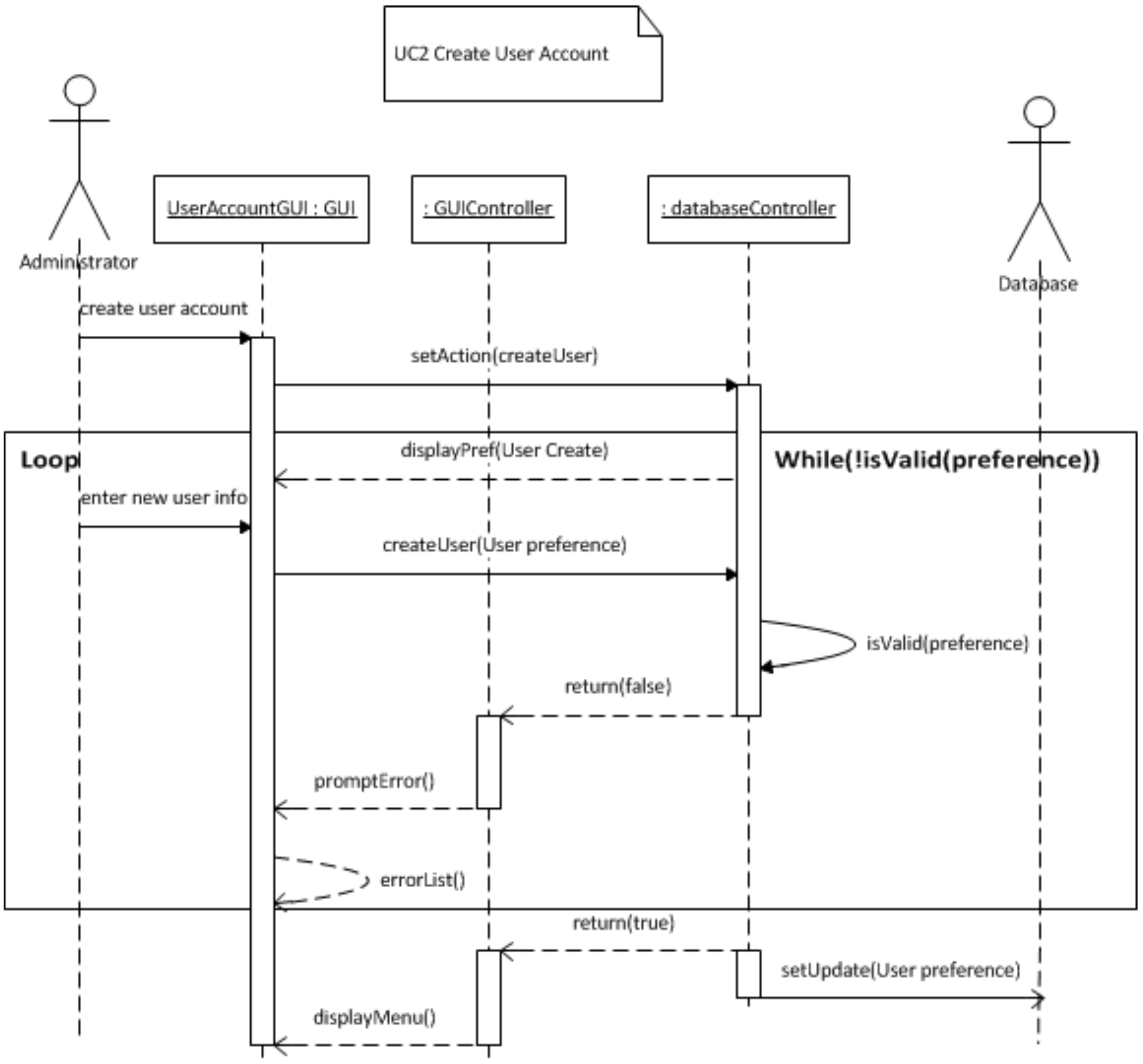


Figure 1-2 - UC-2 - Create User Account

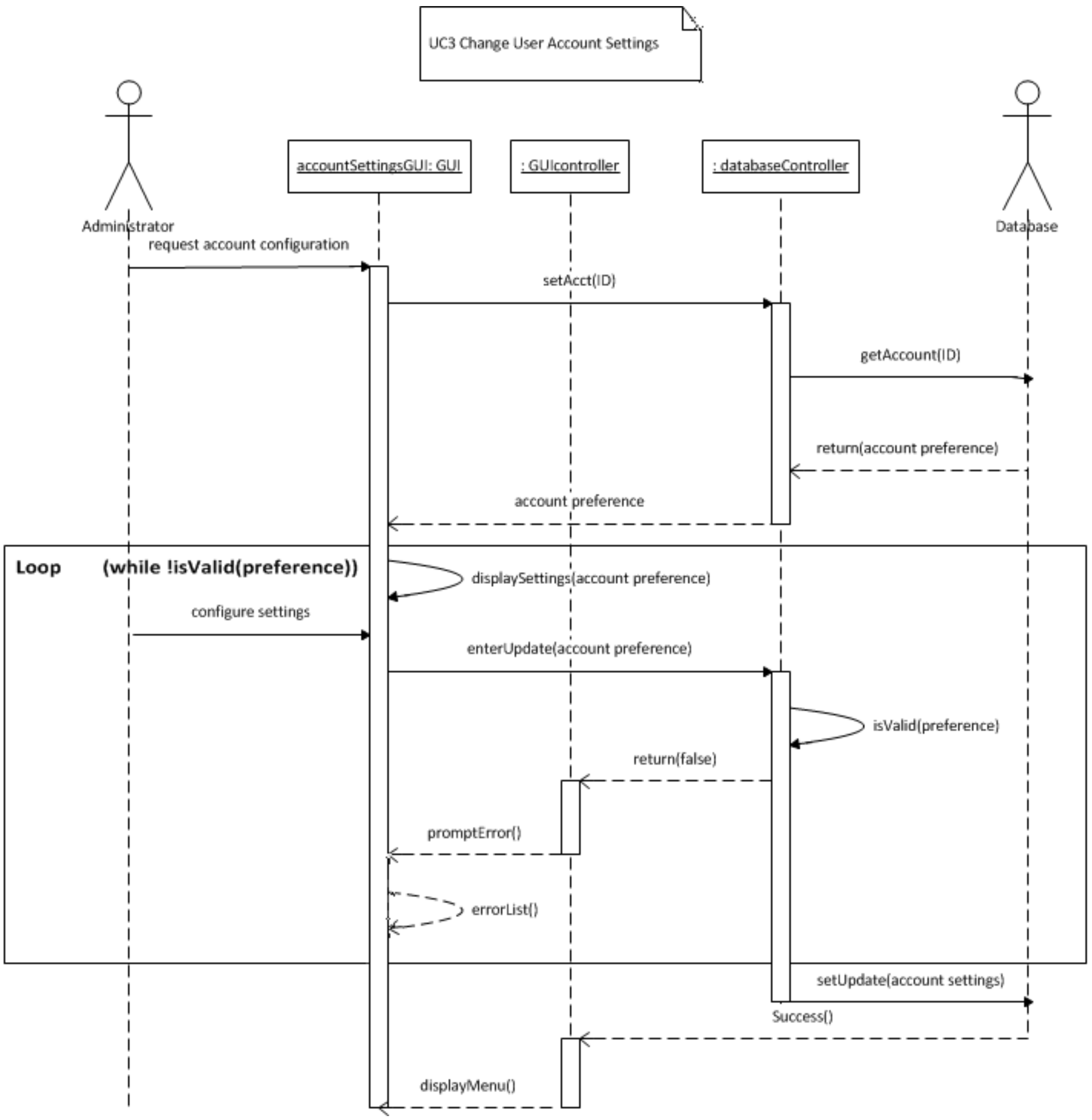


Figure 1-3 - UC-3 - Change User Account Settings

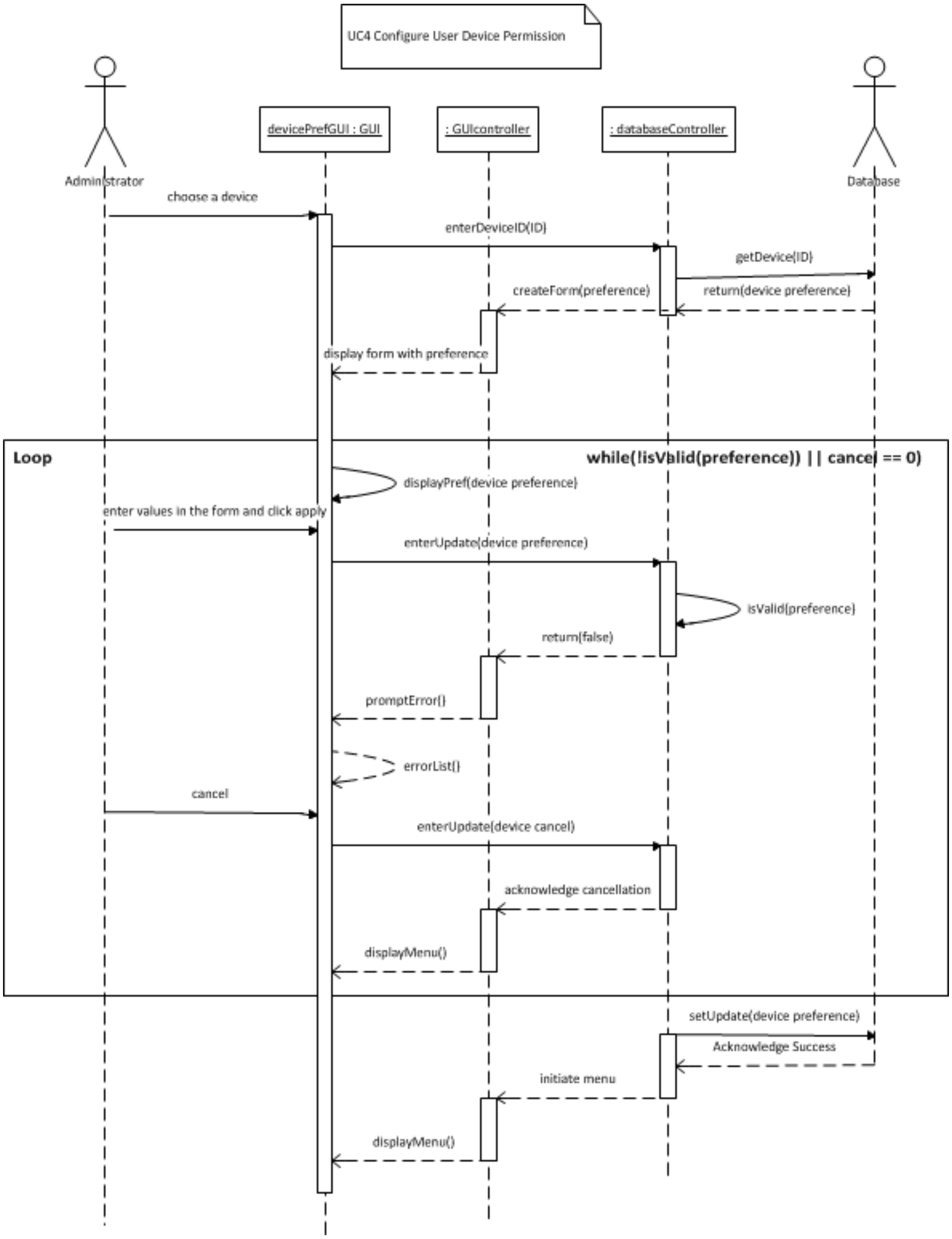


Figure 1-4 - UC-4 - Configure User Device Permission

Use Case 4: Configure User Device Permissions

Use Case 4 is designed with the intention of allowing the Administrator (in a typical setting, this would be a parent or owner of the system) to allow or deny access to devices connected to the system by users registered to the system (such as a child). This can be important for example if a parent wants to restrict children from turning on the oven while the parent is out of the house. The interaction diagram displays the general process that an administrator uses to make a change to available devices. It is assumed that the administrator is already on the device preference page. A module is loaded on the device information page that shows the current status and features of a particular device. The administrator is able through use of a checkbox list to select which user roles can and cannot access the device. The setting is then saved into the database when the form is submitted. Error controls are built into the system such as not allowing an administrator to get locked out from using a device if it has control over all devices. Additionally, any user that enters a device setting page with a role that has been declared by the administrator to not have access to the device, cannot change these permissions as well.

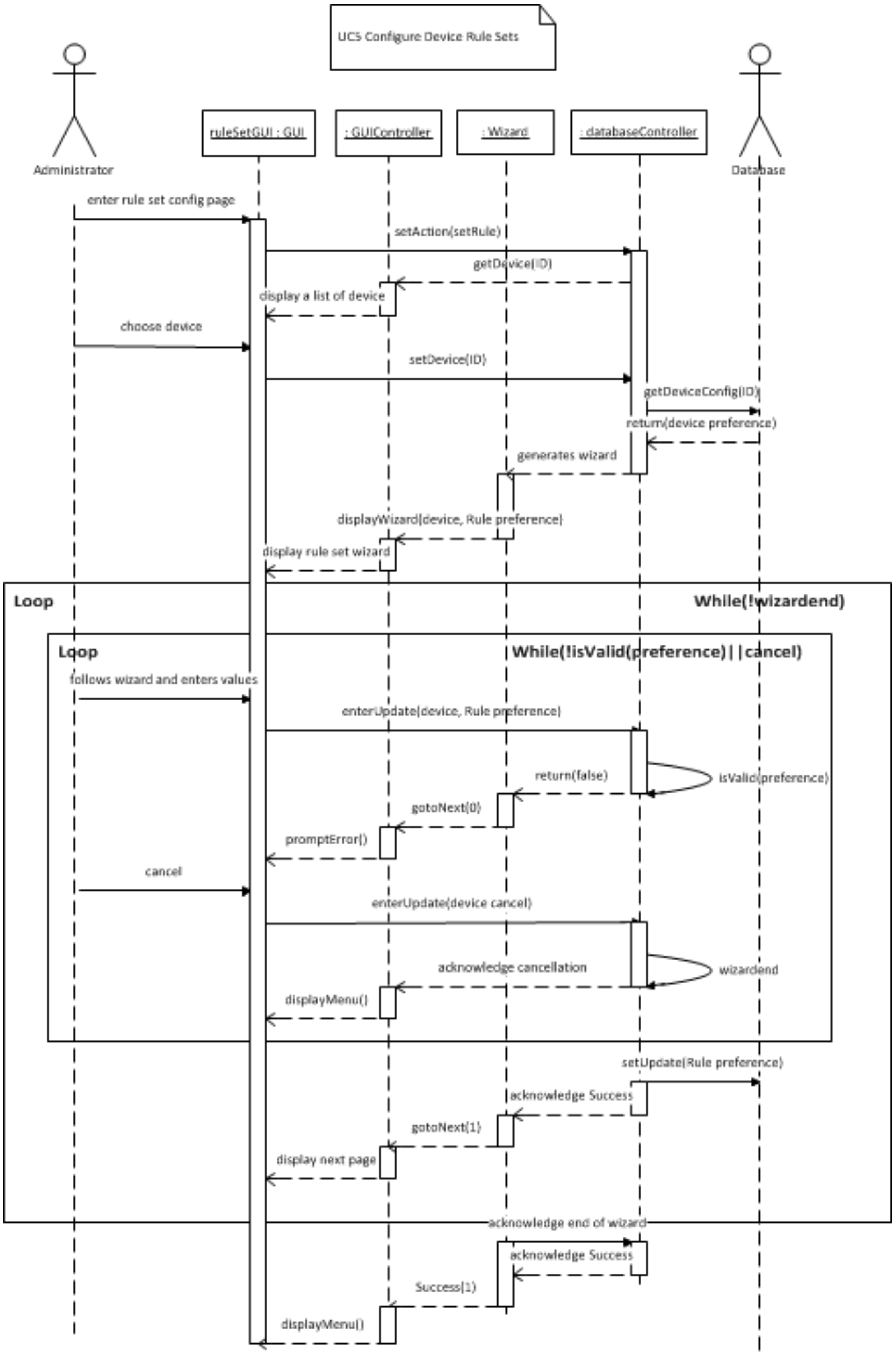
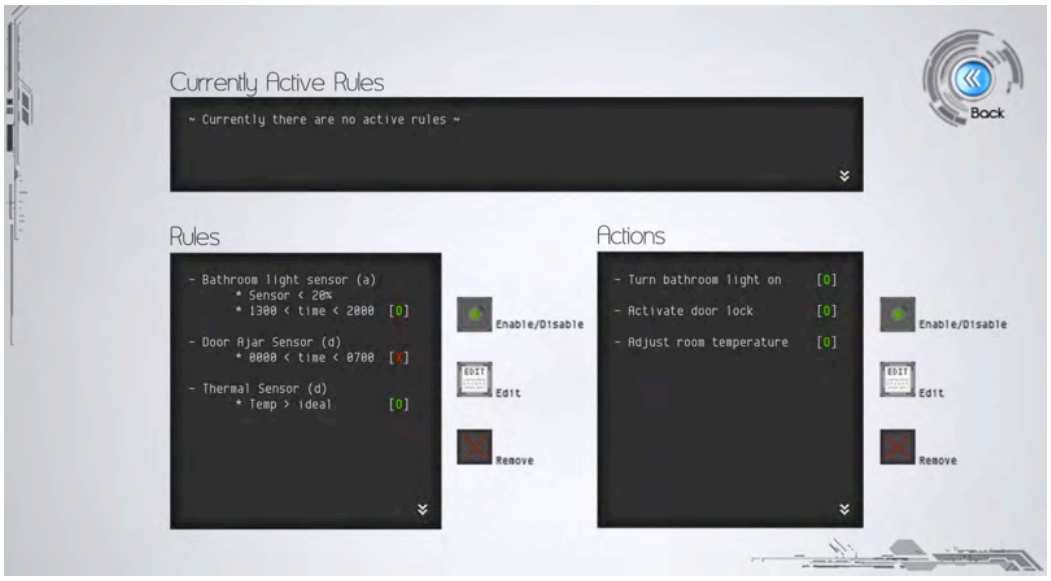


Figure 1-5 - UC-5 - Configure Device Rule Sets

Use Case 5: Configure Device Rule Sets

Use Case 5 allows users and administrators to configure RuleSets, which are the core-functionality of the autoHome system. The image below is an example of what the user will see when configuring a RuleSet. RuleSets consist of a set of Conditions (Condition Sets) and a set of corresponding Actions (Action Sets) to be executed when all conditions in the Condition Set are met. The list of available Conditions to be added into a ConditionSet can be seen on left, and the list of available Actions for an ActionSet can be seen on right. Both lists have their members selectable and editable.



```
~ Currently there are no active rules ~
```

```
- Bathroom light sensor (a)
      * Sensor < 20%
      * 1300 < time < 2000  [O]

- Door Ajar Sensor (d)
      * 0000 < time < 0700  [X]

- Thermal Sensor (d)
      * Temp > ideal        [O]
```

```
- Turn bathroom light on       [O]
- Activate door lock           [O]
- Adjust room temperature      [O]
```

As seen above, Conditions are simple comparisons of the status of sensors. A user may select a condition as “the bathroom light is off” or “the back door is open.” When these two Conditions are put into a ConditionSet together, both of the Conditions must be met for the ConditionSet to be true. The system provides internal Conditions that need not compare any physical devices. These are received as Virtual Device Signals instead of Device Signals and may include information such as Time, weather in another city, CPU Load of another computer, or even user triggers from mobile applications.

Actions are similar to Conditions, in that they usually use a value to command something. Examples of actions include “lock the back door,” “turn off the bathroom light,” “set the porch light to 80%.” Actions are chained together in Action Sets, in which all actions are executed together when called. The system may provide Actions that do not correspond to any physical device. These are sent through Virtual Device Signals, which actually run system code instead, sending text messages, making phone calls, or updating websites. Actions can occur from 100ms (locking a door) to infinite time (keeping a light on).

To edit either Actions or Conditions, the user selects “Edit” for either screen and they are then brought to another screen with an available device list displaying all the relevant devices that the user has permissions for.

On another screen, the Conditions may be formed into a ConditionSet and Actions may be formed into an ActionSet. A ConditionSet is then linked to a RuleSet to provide a list of actions to execute when a list of conditions are met. This provides extreme flexibility, allowing users a simple RuleSet as turning the lights off when it is after 6AM and before 8PM, or a complex RuleSet as sending a text message to the owner only when there is no motion detected in the kitchen and the oven has reached 400 degrees F. The possibilities are endless.

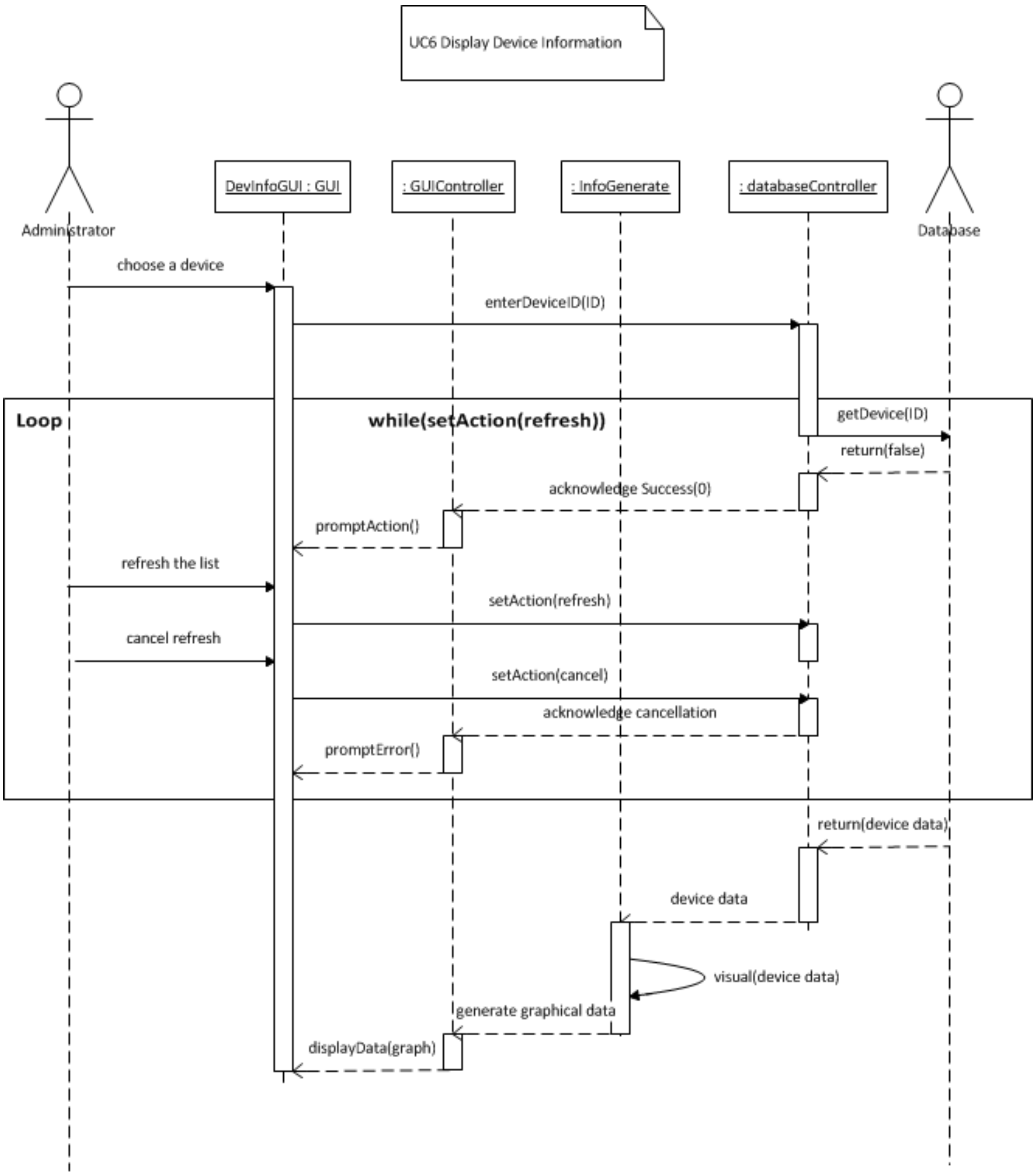


Figure 1-6 - UC-6 - Display Device Information

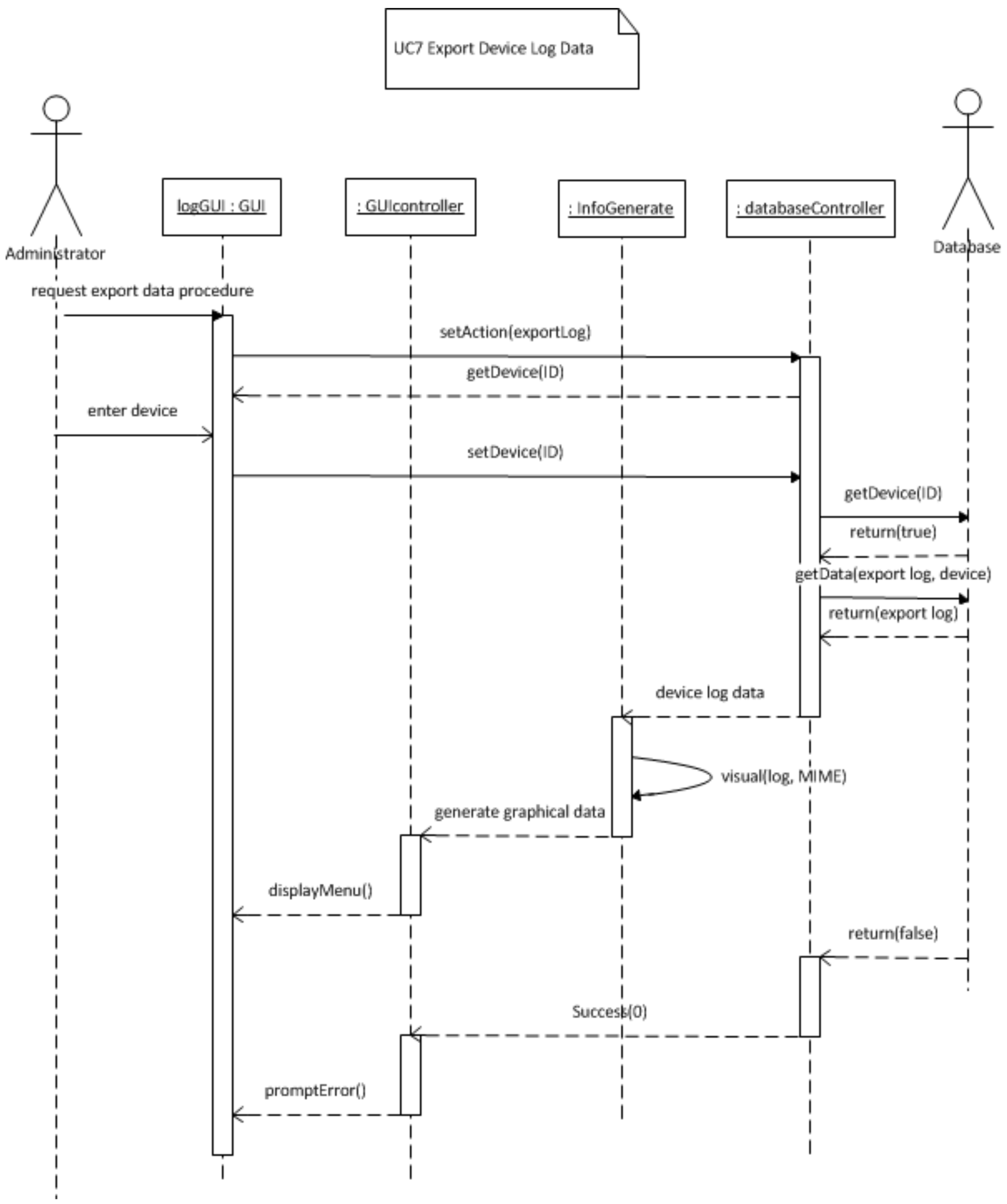


Figure 1-7 - UC-7 - Export Device Log Data

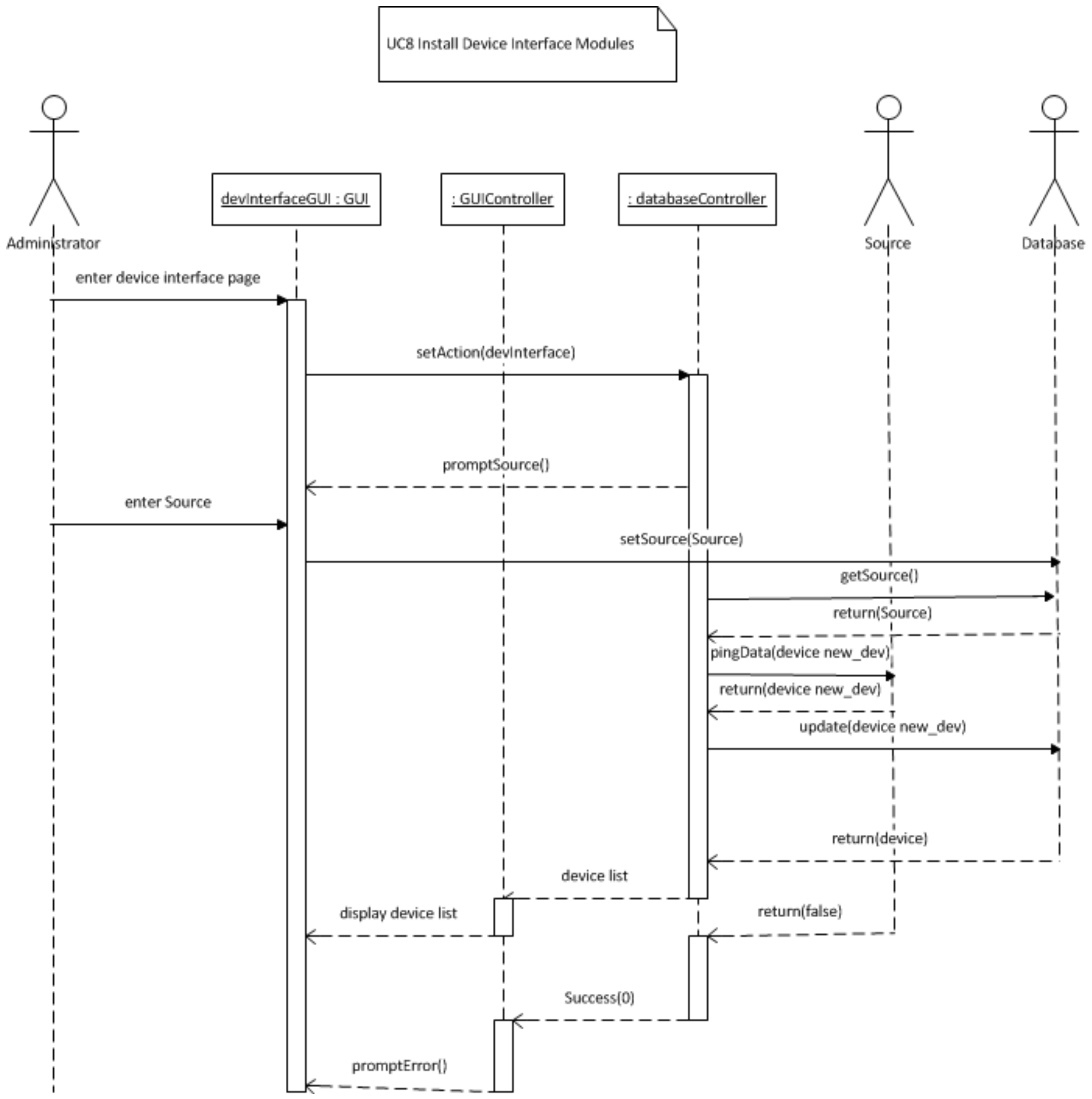


Figure 1-8 - UC-8 - Install Device Interface Module

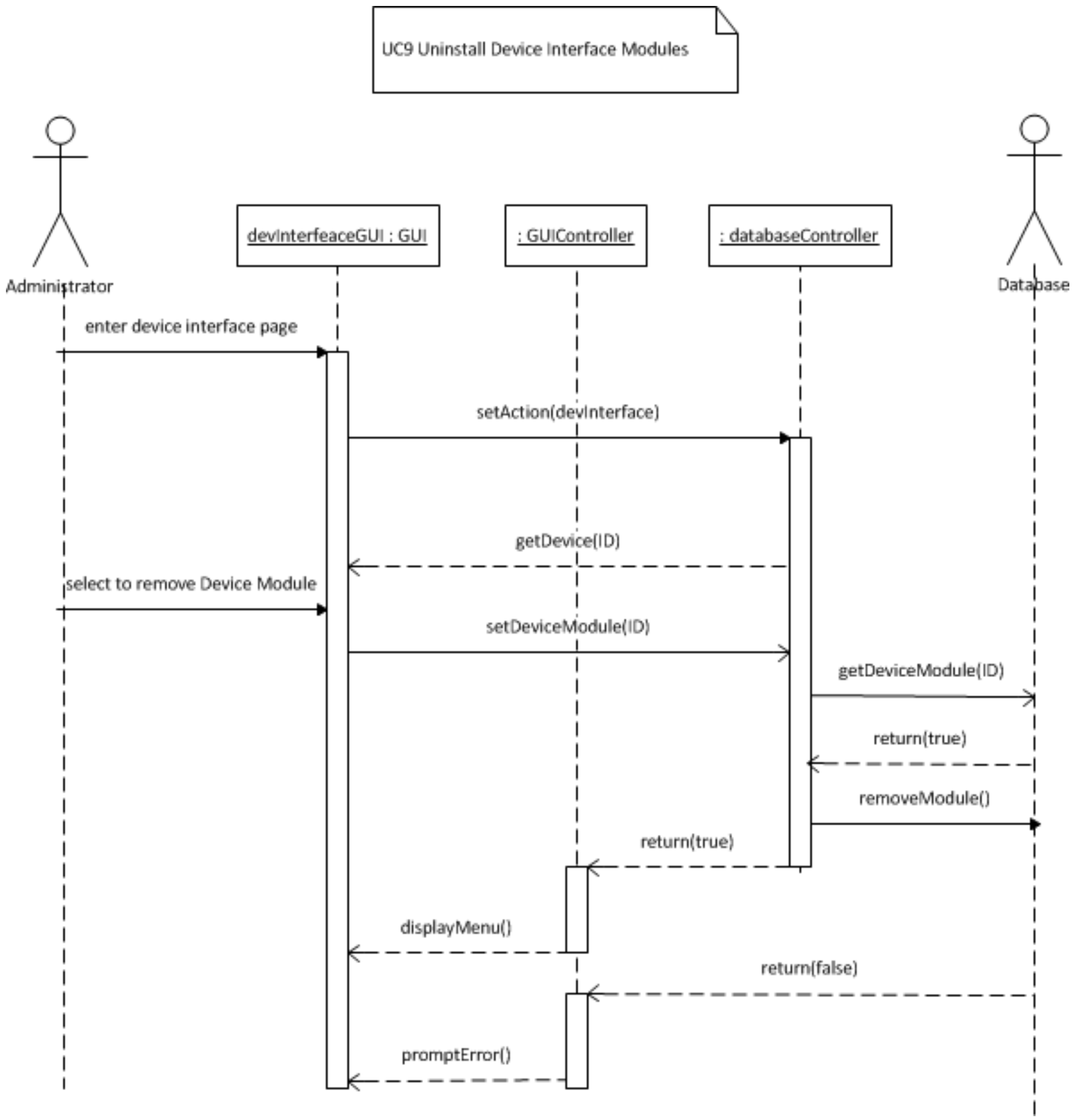


Figure 1-9 - UC-9 - Uninstall Device Interface Modules

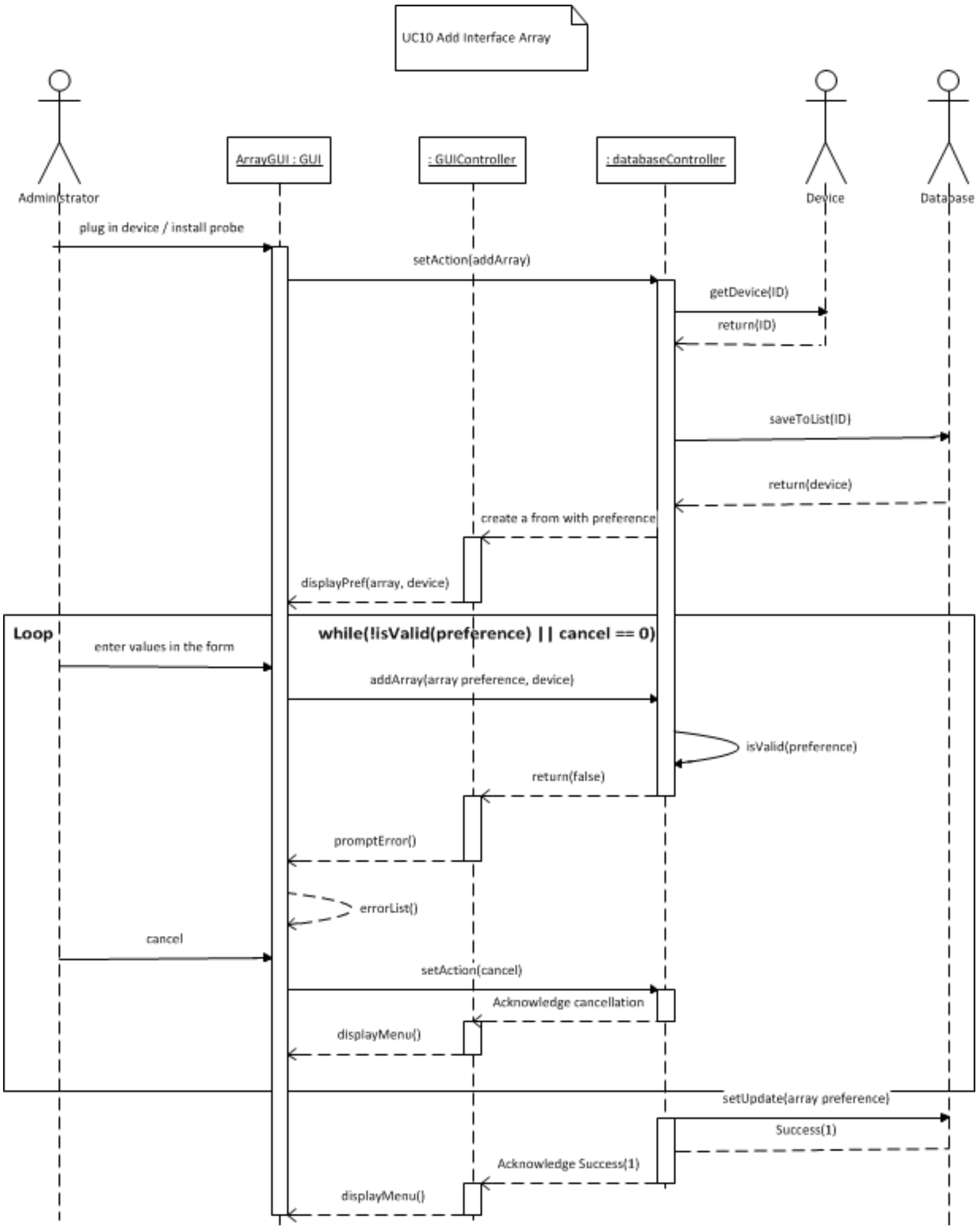


Figure 1-10 - UC-10 - Add Interface Array

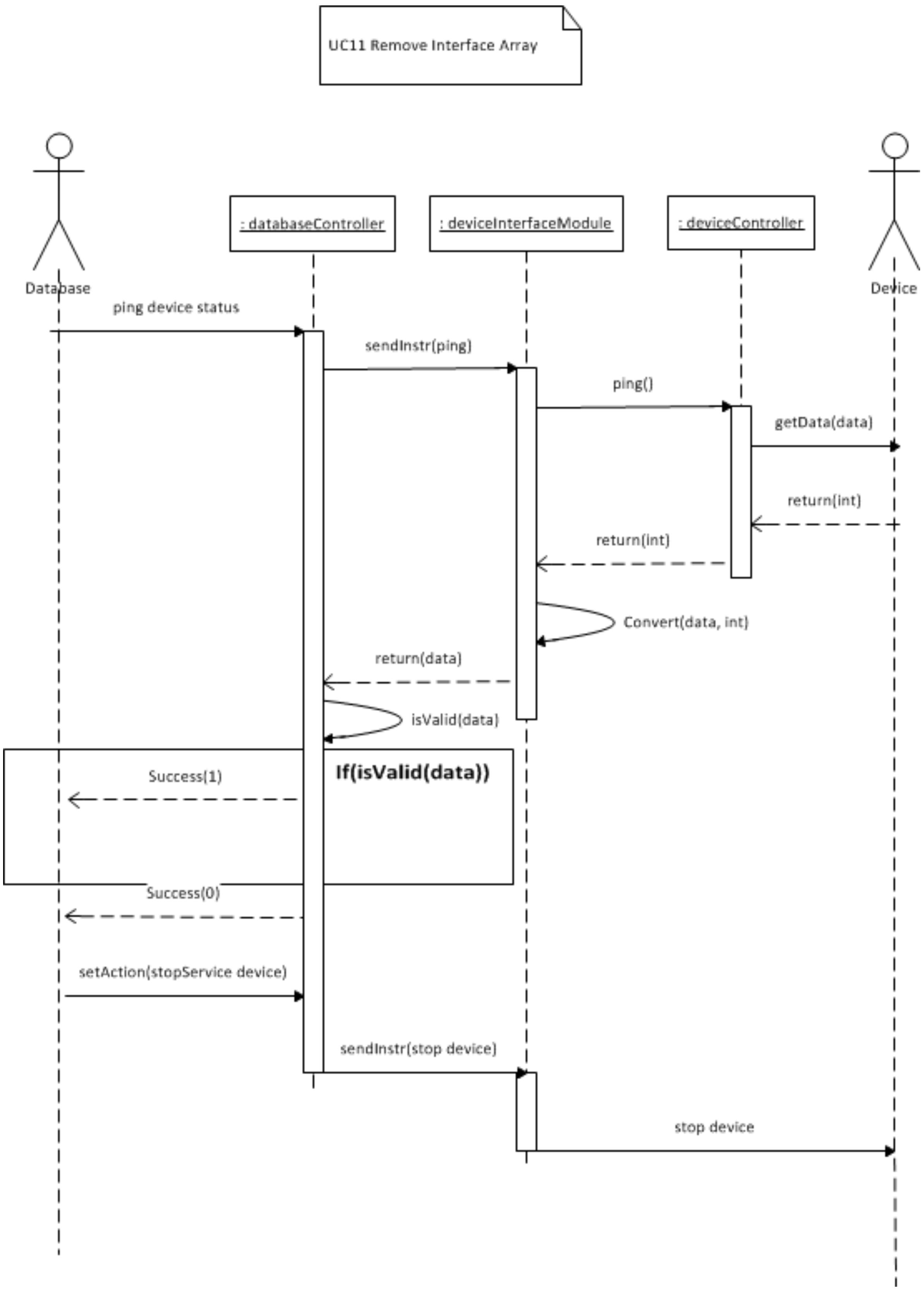


Figure 1-11 - UC-11 - Remove Interface Array

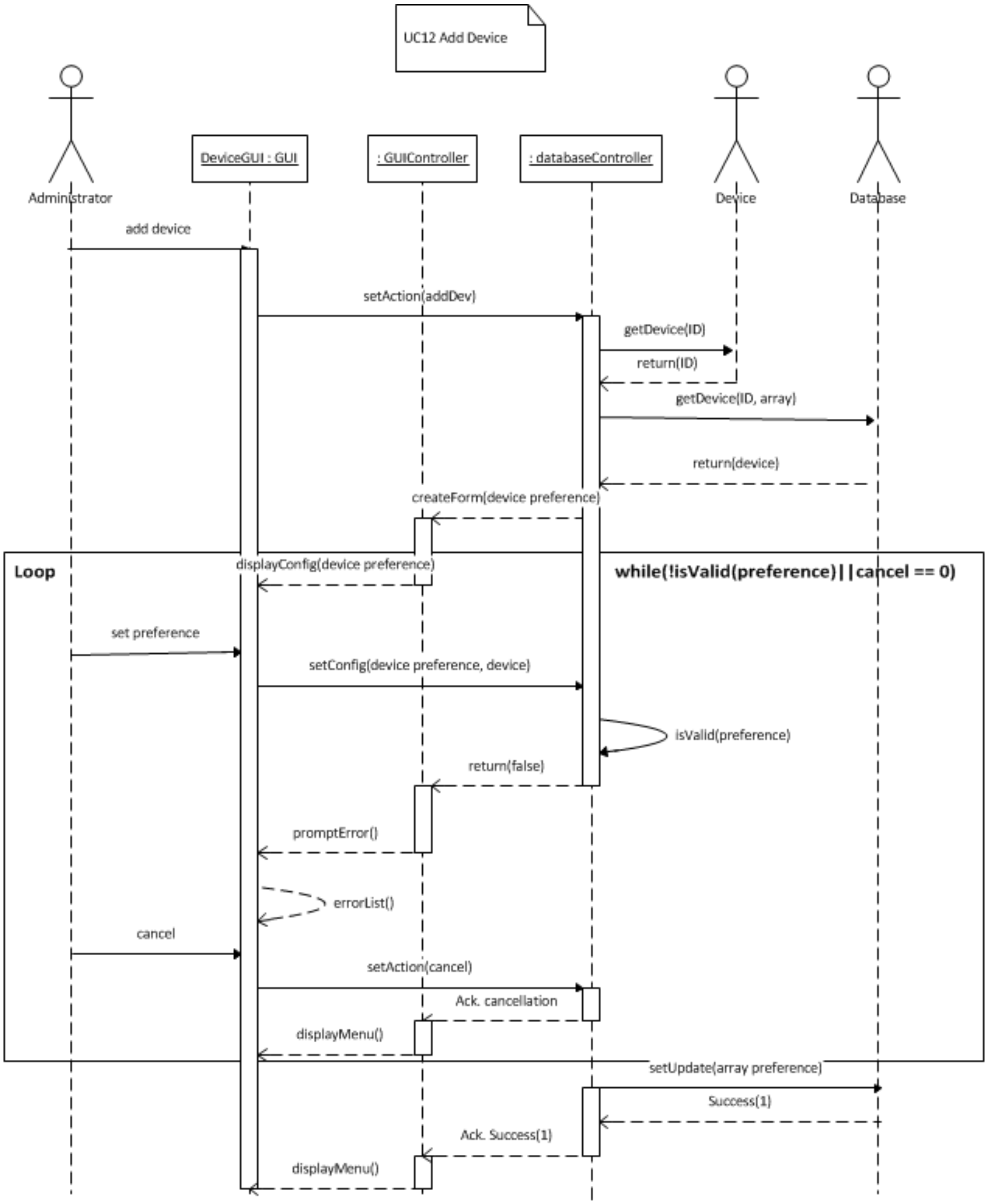


Figure 1-12 - UC-12 - Add Device

Use Case 12: Add Device
Use Case 12 is designed with the intention of allowing an administrator to add a new device into the autoHome system, thus allowing it to becoming accessible to other devices and the system as a whole. In order to facilitate this, the user must connect the device to the system and install the device physically in its appropriate position in the home such as “a lamp in the master bedroom.” Once this step is completed the administrator will proceed to go to the add device module on the web based control panel and tell the system what type of device it is (electrical, security, temperature, etc…) and the location of where the device is placed as well. Once completed and submitted, the information is saved to the database for the system to keep track of. Additionally, the add device panel will then become the device preference panel where regular settings for specific devices can be set such as on and off and more as well as device permissions as discussed in UC 4. If a preference selection is invalid the administrator will be prompted to correct the issue.

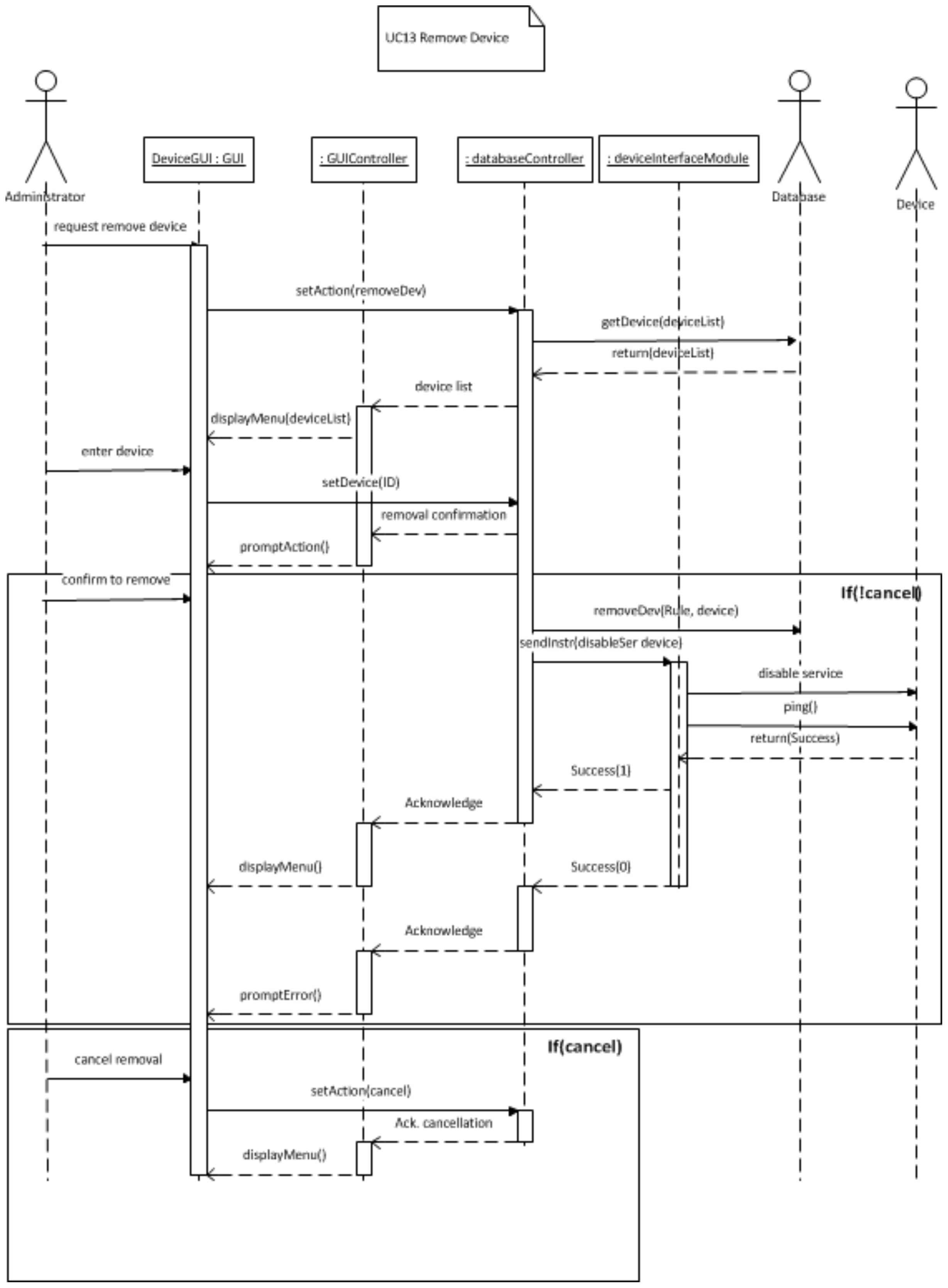


Figure 1-13 - UC-13 - Remove Device

Use Case: 13 Remove Device

Use Case 13 is designed with the intention of safely removing a device from the system without corrupting data and protecting the integrity of the device by assuming that all devices may be fragile as to the point that a sudden disconnect may damage a component within the device and or should also be shut down appropriately before removal to ensure that all data communication between the device and system has terminated. In order to remove a device, an administration will go to the device preferences page of the device the administrator wishes to remove, and then clicks on the button for permanently removing device located at the bottom. When this is done, the system will respond with a confirmation to ensure that the administrator is certain of the decision. After the final confirmation, the system will set the device to an off state or initiate a shutdown protocol that is handled by the individual device and is beyond the scope of autoHomes offering. The system will then ping the device to see if it is still alive and once the system can confirm that the device is indeed removed, it will remove all device presences from the database.

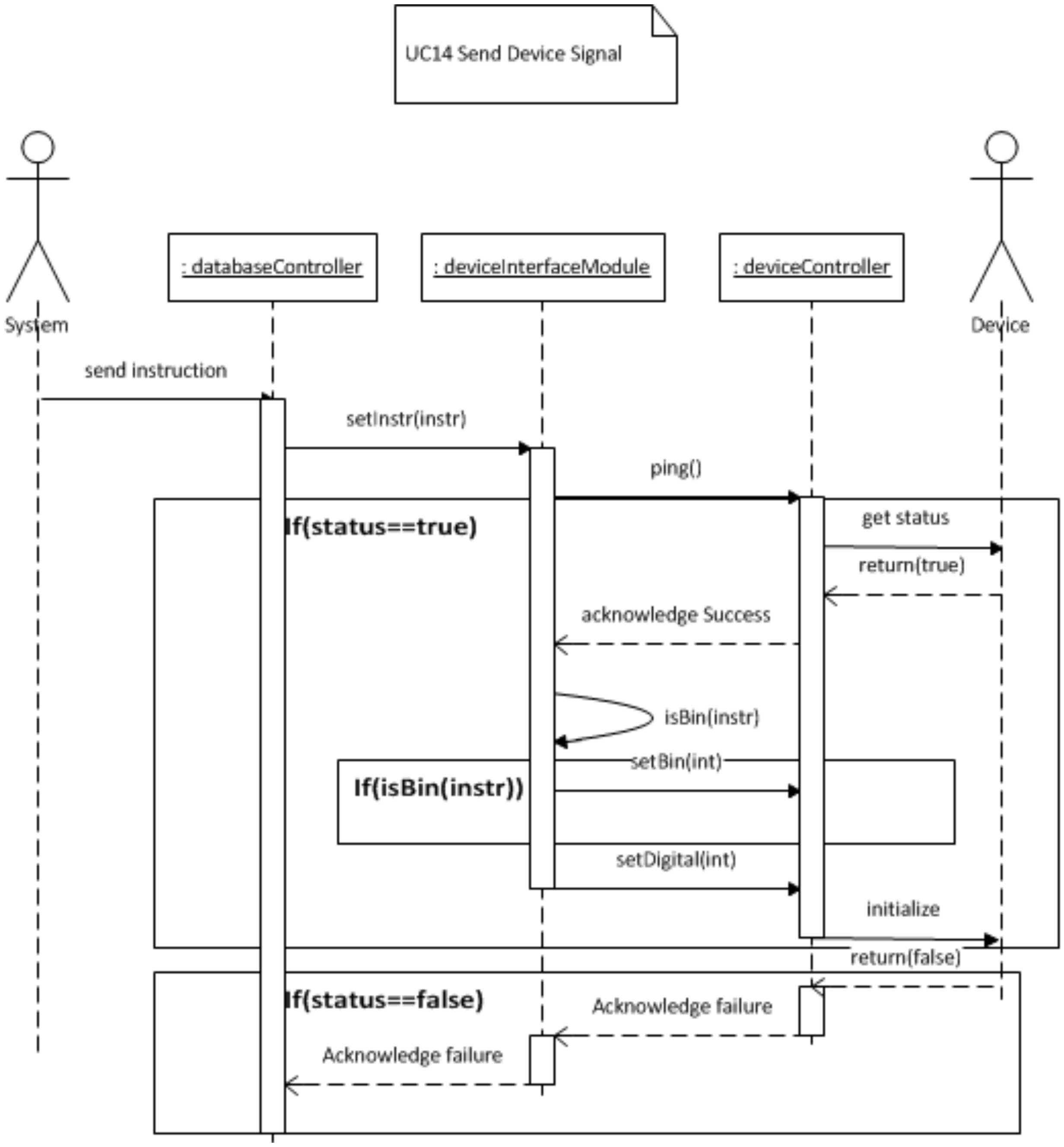


Figure 1-14 - UC-14 - Send Device Signal

Use Case 14: Send Device Signal
Use Case 14 is designed with the intentions to take the settings previously set in UC 4, 5, 12, and 13 and then implements them onto the actual physical selected device. This task requires no user interaction and is handled entirely by the system. When the system wants to send a signal to a device it first sends the instruction; once that has been handled the deviceInterfaceModule checks to see if the device is active and willing to accept a command. When ping responds by indicating that the device is on and ready, the deviceInterfaceModule checks to see what kind of data is being sent, if it is an analog value the data is encoded into a binary value which is then sent to the device. If the data is not analog the data must be digital and a Boolean value is used to indicate on or off command. As long as everything is smooth sailing the process completes.

When an error occurs such as the device disappearing by being pulled out or so while the process is happening, an error is sent to the log and the process completes in failure.

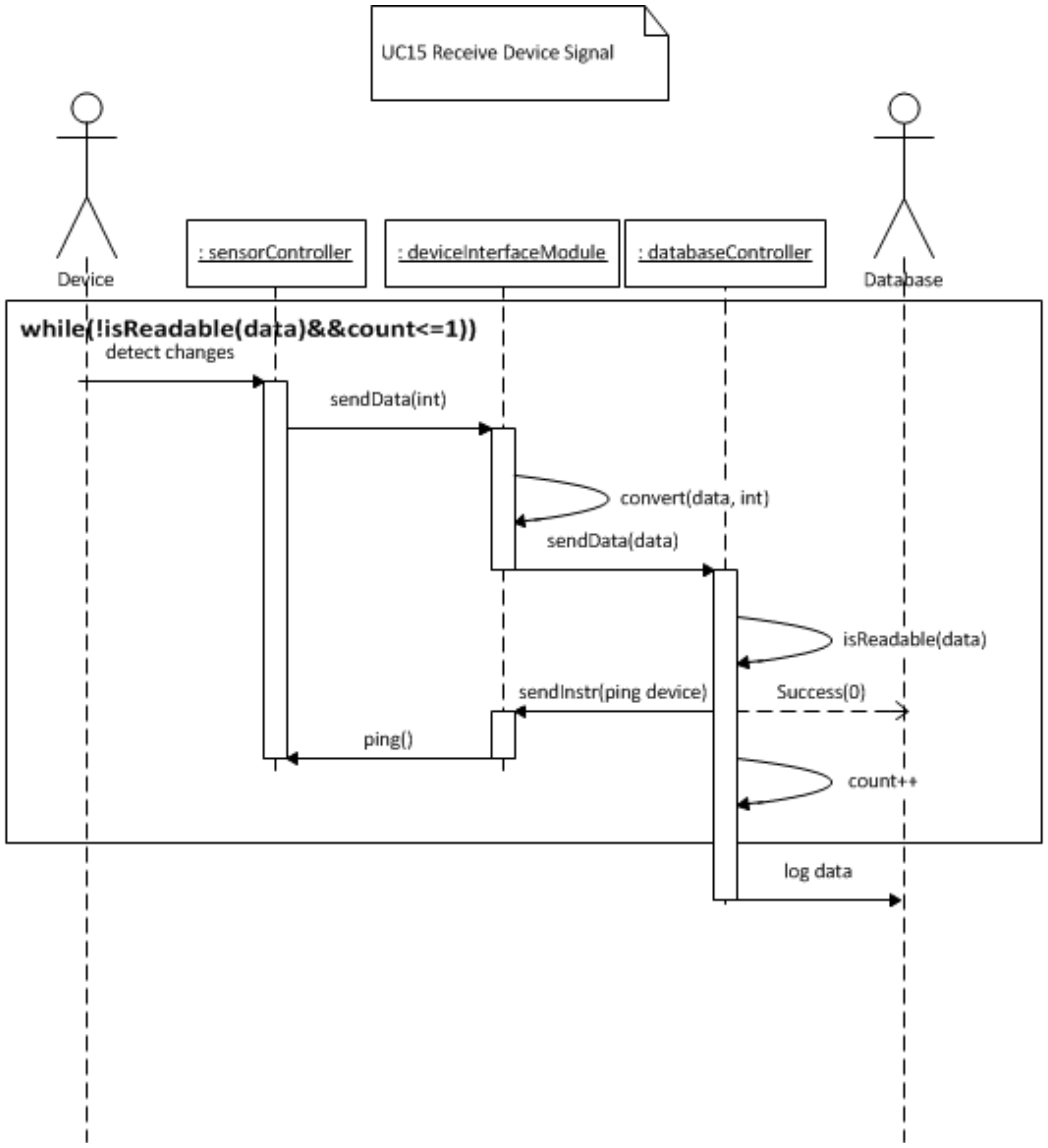


Figure 1-15 - UC-15 - Receive Device Signal

Use Case: 15 Receive Device Signal
Use Case 15 Is designed with the intention of receiving incoming data and responses from individual devices. An incoming result can come from two different methods; either by a trigger that is hosted on the device, such as a manual change to a device (ie turning off a light) or by responding to a system ping query where ping returns to the system with device information such as if the device is alive and running, status of it as

well. The basic principal to this use case is that ‘something’ in the system has changed as in a value that needs to be recorded and this use case takes that changed data, stores it in the database, and adds it to the log. This use case is performed by measuring to see if a change has been made, if this is the case, the deviceInterfaceModule converts the data acquired into formats better suited for logging such as translating an analog value to an equivalent temperature value. Once the data is converted and correct, the data is placed into the database, the log count for the device history is bumped up by one and then this allows a history of past changes to be kept. All of this happens as long as the device is connected.

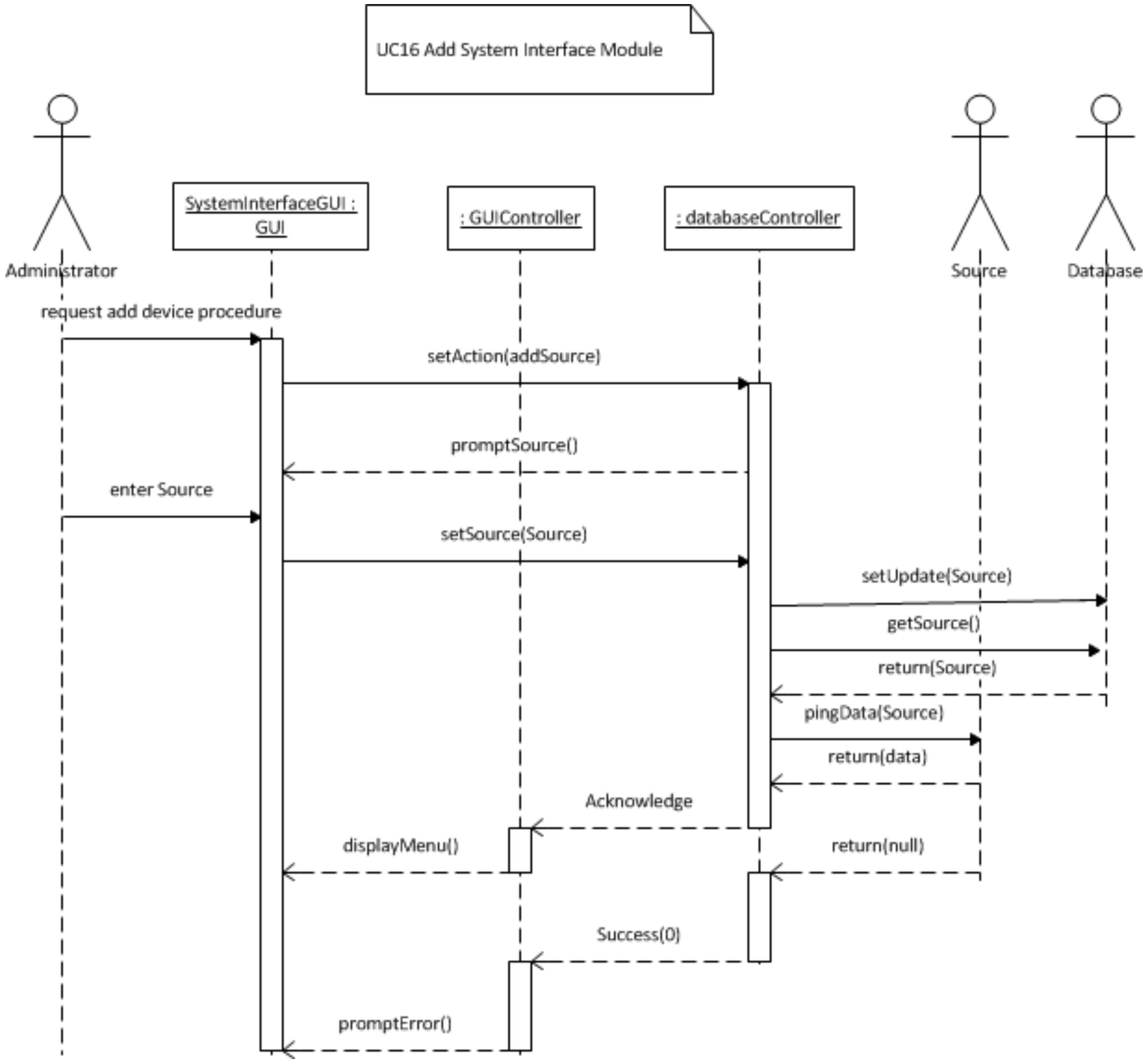


Figure 1-16 - UC-16 - Add System Interface Module

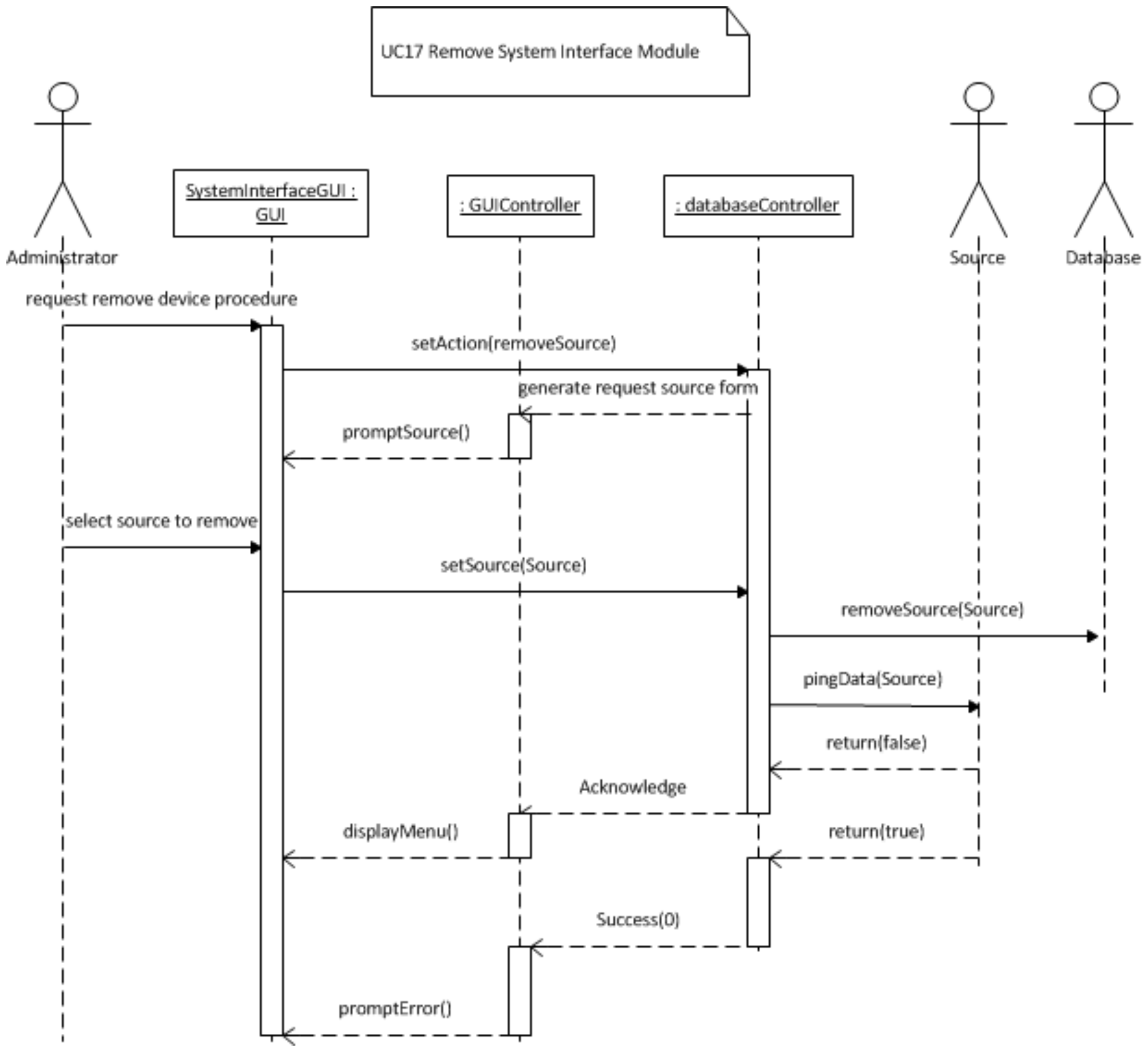


Figure 1-17 - UC-17 - Remove System Interface Module

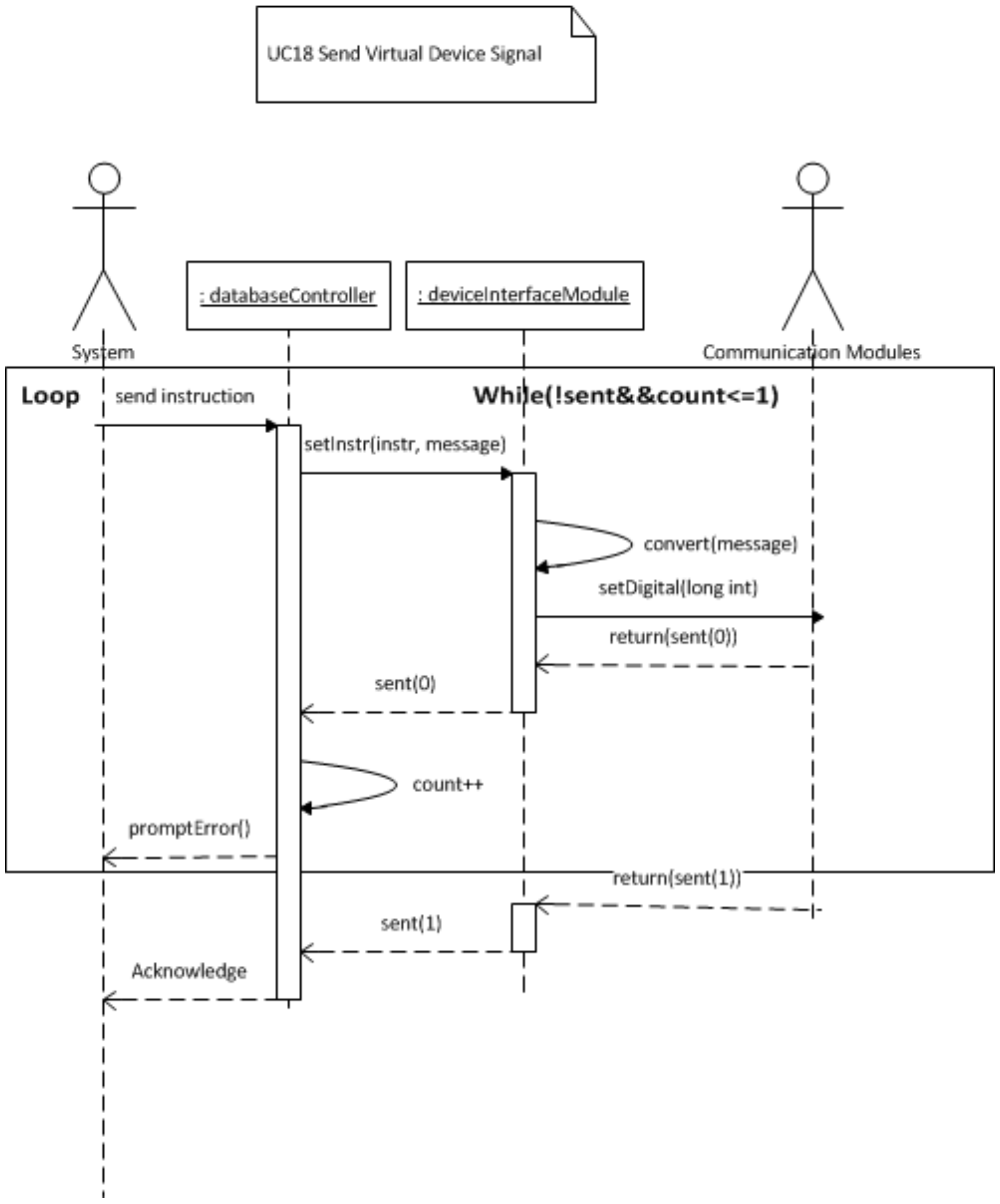


Figure 1-18 - UC-18 - Send Virtual Device Signal

Use Case 18: Send Virtual Device Signal
Use Case 18 was designed with the intention of sending a command to an external api such as the control for sending out text messages. A typical scenario involving this use case would include a fire alarm being

activated and alerting the administrator through means other than the web panel such as a text messaging service. Another example may be a notification to the administrator to warn about a garage door being left open over a specified threshold where it is possible that the administrator forgot to close the door and the presence of the open door exposes a security risk to the home which should be addressed in a more active manor. When the system sends a command such as ‘initiate a text message’ the deviceInterfaceModule checks to see that the command is valid, if so the command is forwarded to the appropriate communication modules which include the text messaging service and email client. Once this occurs to completion, the action is logged. If an error occurs throughout the process it will be logged, and an attempt will be made to still process the request since high priority tasks such as a fire may be occurring and action will need to be prompt.

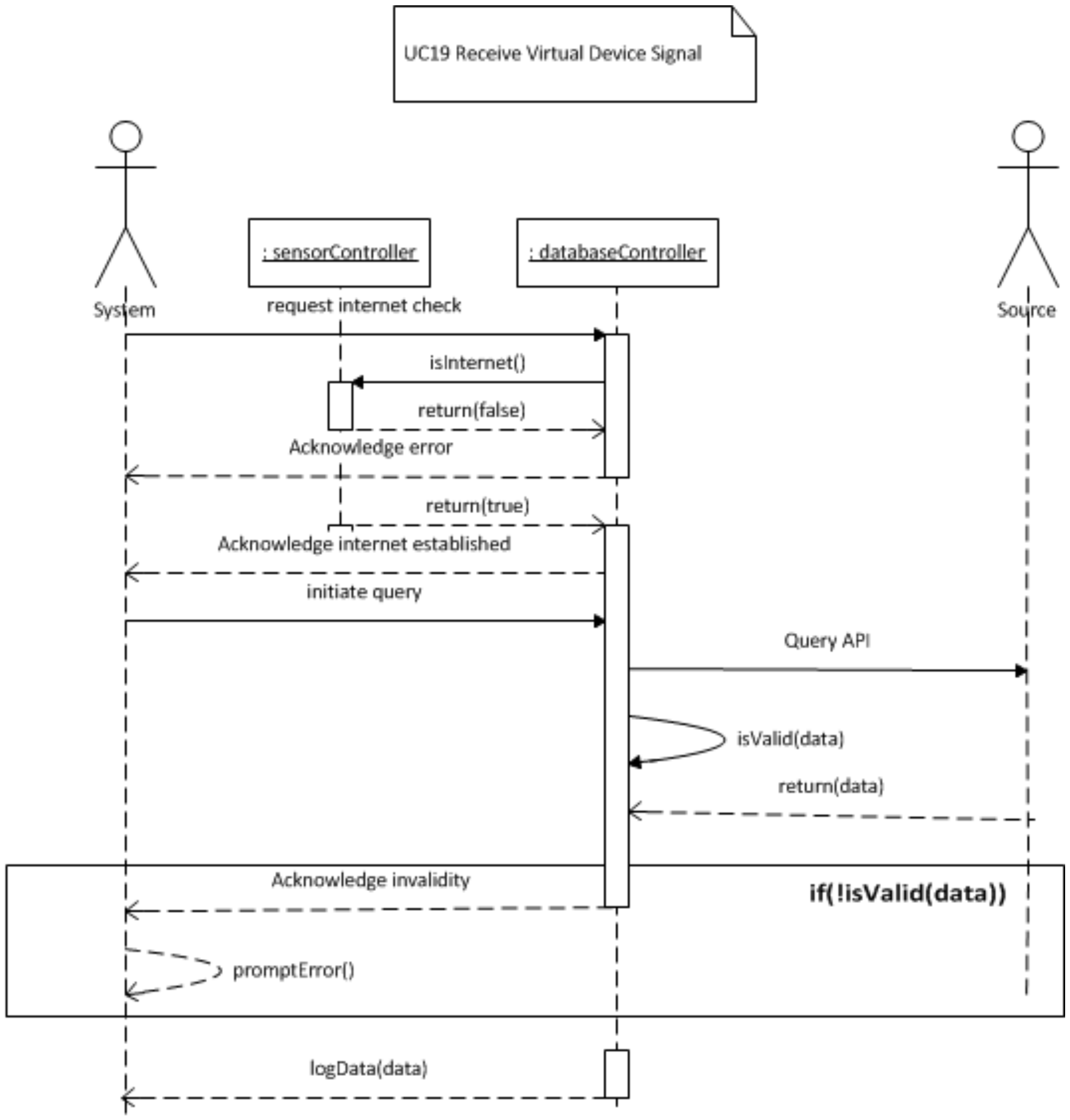


Figure 1-19 - UC-19 - Receive Virtual Device Signal

Use Case 19: Receive Virtual Device Signal
Use Case 19 was designed to perform two tasks; it will receive confirmation that UC 18 sent out for example a text message properly. But more importantly it will handle the result of querying external data sources such as local weather and almanac data, this includes sunrise/sunset times and also the clock synchronization that is set to occur about every week, this is discussed in the Network section. The general flow for this use case is that the system is already aware of UC18 being completed and as such it now will check connection is still available to these external communication API's. Then it will receive the data, verify that the data acquired is valid and then store this new data into the database. If the data is invalid the receive request will fail and and an error will be logged as well as a notification will be presented to the administrator.

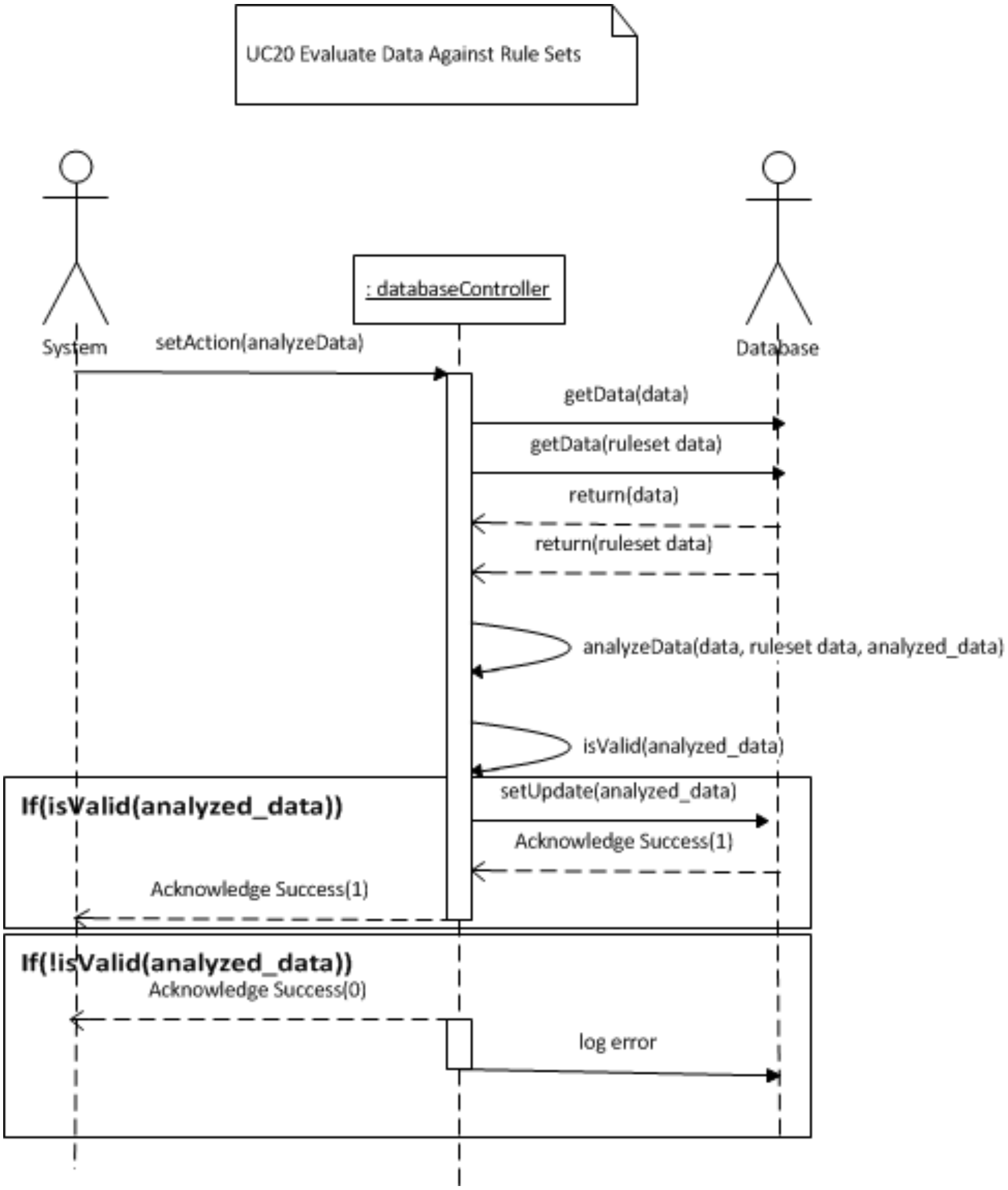
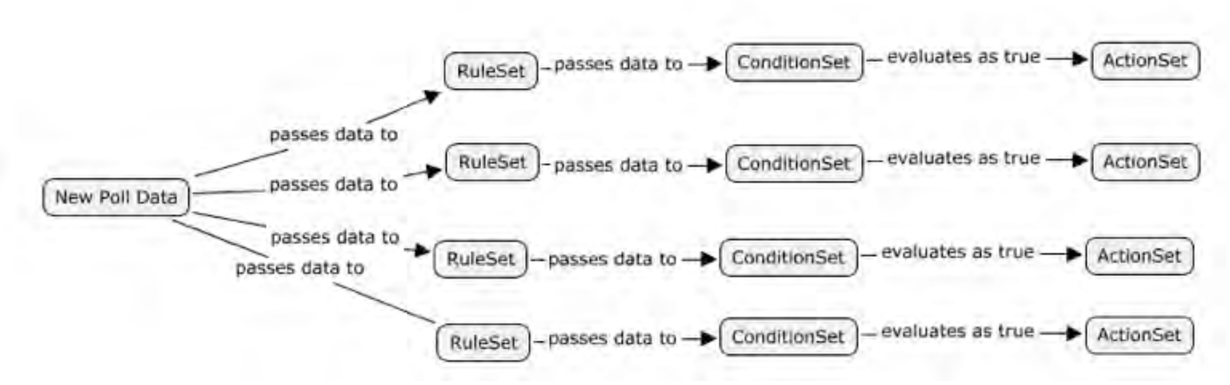


Figure 1-20 - UC-20 - Evaluate Data Against Rule Sets

Use Case 20: Evaluate Data Against Rule Sets
Use Case 20 is triggered when new data is fed into the system. The devices have pre-configured polling rates in their drivers, and when this period has elapsed, new data is read from the sensors. There is always data flowing through the reporting conductors, but data is only polled so often due to data logging and processing constraints of computers. This use-case is not directly-accessible by the user, but instead is activated

as soon as a user successfully adds a device to the system. The physical voltage values from the devices are converted to real units (Fahrenheit, Percentage, True/False, Lumens) by their respective drivers. This use case takes those converted values and evaluates them against all active RuleSets (and therefore, the ConditionSets).



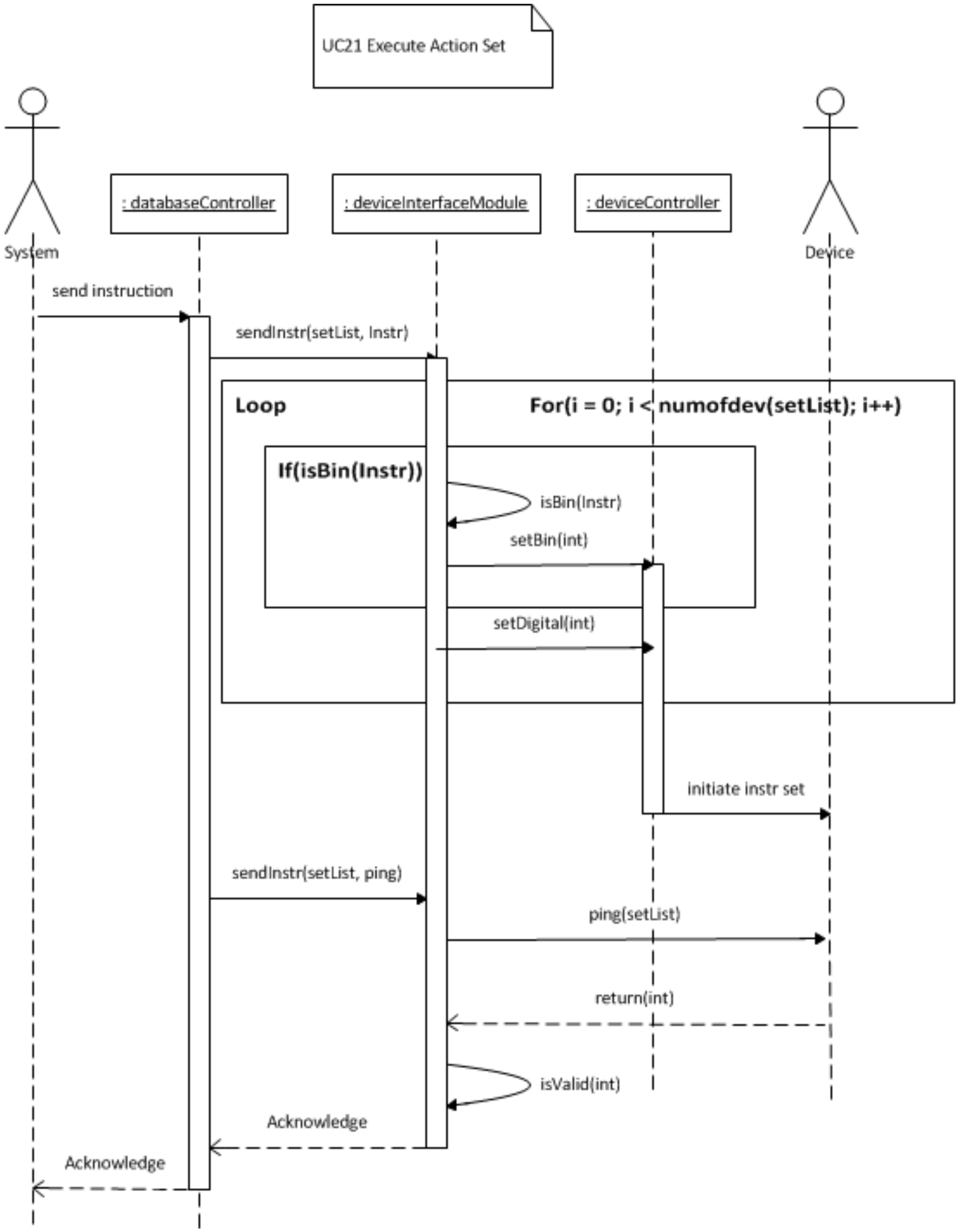


Figure 1-21 - UC-21 - Execute Action Set

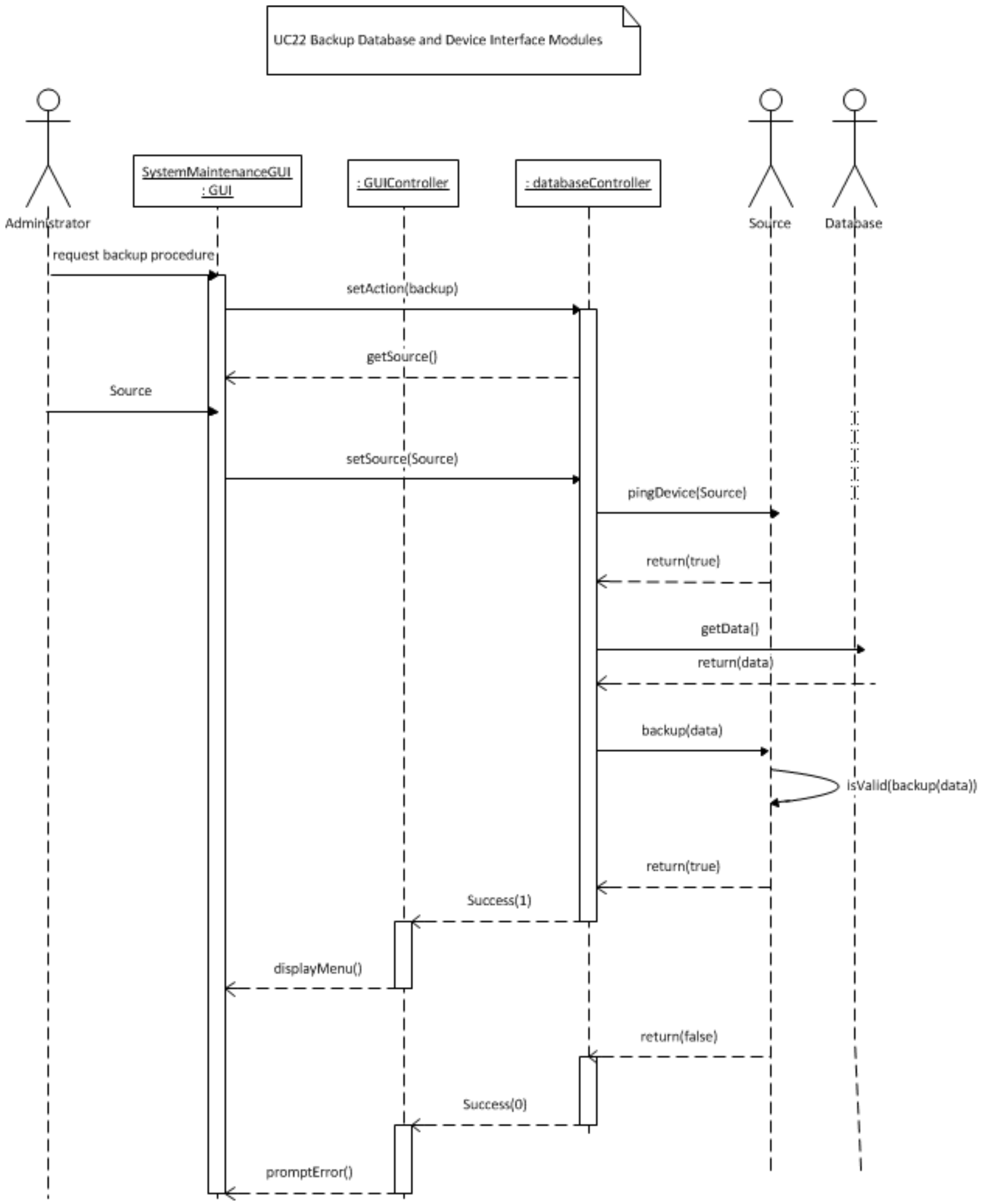


Figure 1-22 - UC-22 - Backup Database and Device Interface Modules

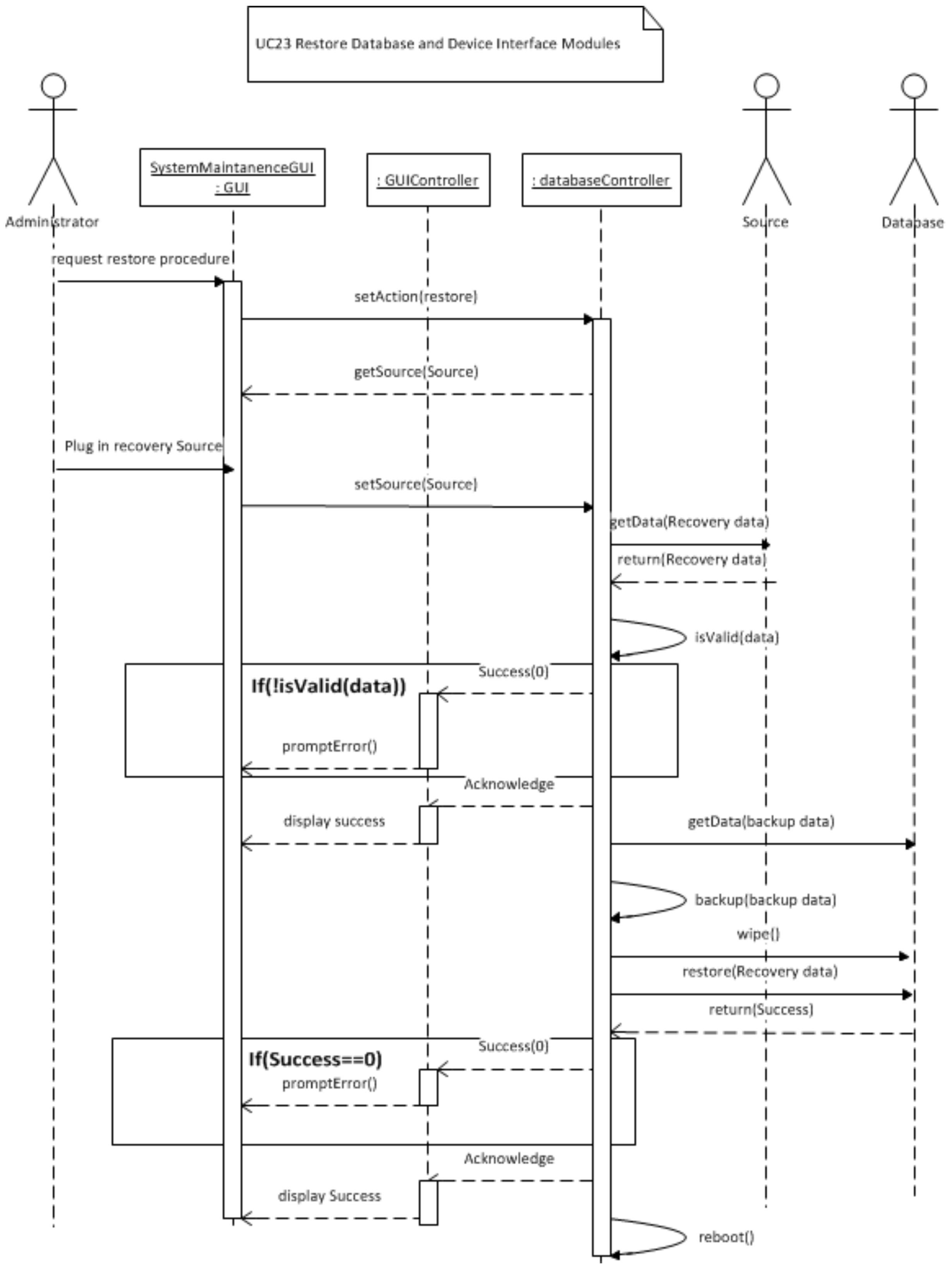


Figure 1-23 - UC-23 - Restore Database and Device Interface Modules

2. Class Diagram and Interface Specification

The following pages will show our class diagrams and traceability matrix.

a. Class Diagram

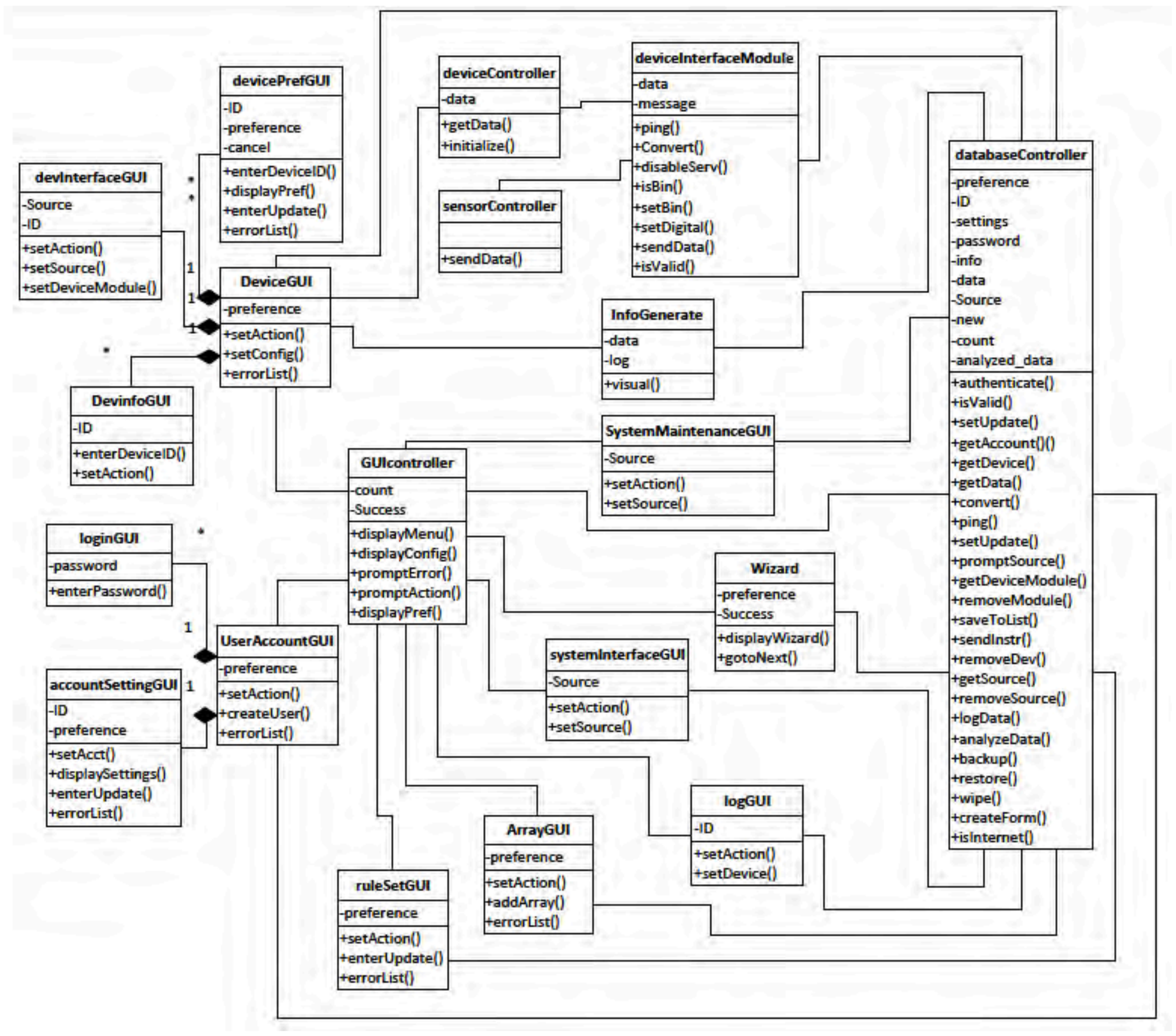


Figure 2-1 - Class Diagram

b. Data Types and Operation Signatures

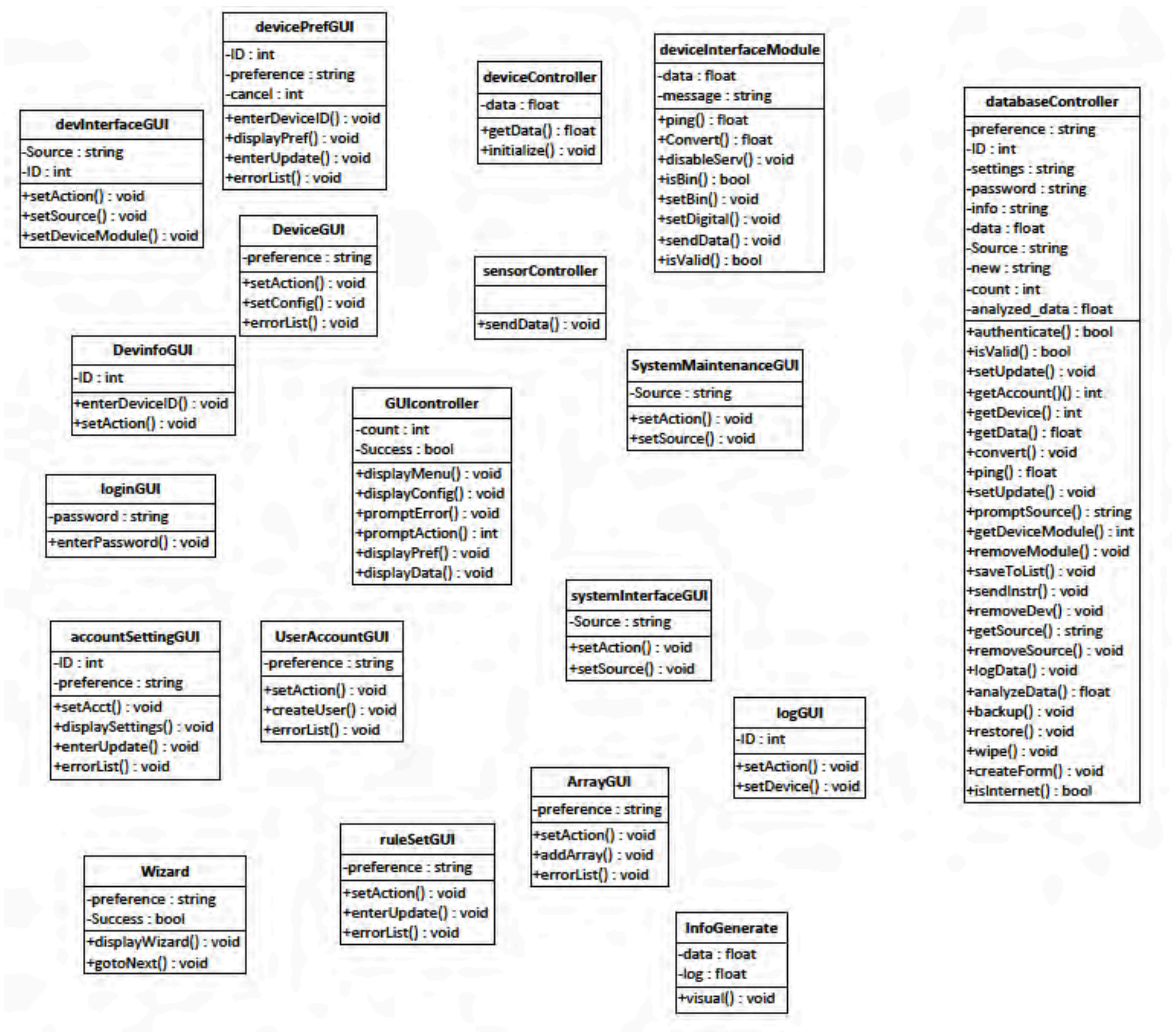


Figure 2-2 - Data Types and Operation Signatures

c. Traceability Matrix

DatabaseManager

ArrayGUI is derived from *DatabaseManager* because the array of devices is shown to the user. The array of devices is logged in the Database since the Database has information on what devices are in the system.

databaseController is derived from *DatabaseManager* based on the principle that *databaseController* has over 20 different functions that the database uses. It is crucial that the functions in *databaseController* are only used by the Database and that they are not tampered with. If a user were to use these functions, they could possibly mess up the internal functioning of autoHome and brick their system.

GUIcontroller is derived from *DatabaseManager* because the Database will have to push GUI's to the user and *GUIcontroller* is used to regulate the GUI's and when they are pushed out. This is important because the system could malfunction if the database didn't have a class to regulate the GUI's actions.

InfoGenerate is derived from *DatabaseManager* based on the principle that *InfoGenerate* displays information from within the Database. It is important to have information within the Database displayed with classes that are embedded in the Database as well.

logGUI is derived from *DatabaseManager* because it will display a log in the form of a GUI to the user based on the current status of the system and it will also provide a list of events for the system in a certain period of time.

RuleSetGUI is derived from *DatabaseManager* because ruleset preferences need to be shown and the GUI needs to allow the user to change preferences as desired.

SystemInterfaceGUI is derived from *DatabaseManager* based on the fact that *DatabaseManager* has a number of different pieces of information that *SystemInterfaceGUI* needs to display. This is important because the system should print out correct and accurate information that the Database holds. If the system were to print out incorrect information, autoHome would lose credibility on all counts which would lead to negative publicity, among other things.

SystemMaintenanceGUI is derived from *DatabaseManager* based on the principle that *SystemMaintenanceGUI* requires access to parts of the Database that shouldn't be accessible to the user. Since *SystemMaintenanceGUI* checks for updates, it needs to use the "getUpdate()" function found within the Database.

Wizard is derived from *DatabaseManager* because *Wizard* provides the user with a step-by-step guide on how to use autoHome. This is beneficial because it will eliminate any possible confusion on how to use the system. This will also make autoHome more user-friendly for people that are "afraid" of learning how to use newer pieces of technology.

ControlManager

deviceController is derived from *ControlManager* due to the principle that *deviceController* receives data for the system. This is important because if our system does not take in any input, it wouldn't function as it should.

sensorController is derived from *ControlManager* due to the principle that *sensorController* sends data through the system. This is very important since our system will be taking in data, and users will expect that since they provided data, data should also be sent back to them.

<u>DeviceManager</u>

DeviceGUI is derived from *DeviceManager* due to the fact that a GUI needs to be displayed, which allows the user to choose what they want to see and change. This means that this GUI will display the option to pick device info, current device information, and device preferences.

devicePrefGUI is derived from *DeviceManager* due to the principle that the preferences of a device are held within DeviceManager and need to be available for the user to change as desired.

DevinfoGUI is derived from *DeviceManager* due to the principle that it will display information about the device which is kept in *DeviceManager*. This is important because the device's information will almost constantly be changing and *DevinfoGUI* needs the most up-to-date status reports on the device.

devInterfaceGUI is derived from *DeviceManager* due to the principle that it will give the interface of the desired device which will provide different preferences for the device that the user can change.

deviceInterfaceModule is derived from *DeviceManager* due to the principle that *deviceInterfaceModule* sends and receives data depending on what the user inputs for the preference they want to change.

<u>GUIManager</u>

accountSettingGUI is derived from *GUIManager* because if a user wants to change the settings on their account, such as changing their password, there needs to be a GUI that has options for the user to change. The user would need these options for a number of reasons like changing their password, a security question, or even their name on the account. There are a number of settings that should be provided for the user.

ArrayGUI is derived from *GUIManager* because the system has to produce a GUI that shows the array of devices to the user. Without this, the user would not be able to know what devices are in the system. Without knowing that at the very least, the system would crash entirely.

DeviceGUI is derived from *GUIManager* due to the fact that a GUI needs to be displayed, which allows the user to choose what they want to see and change. This means that this GUI will display the option to pick device info, current device information, and device preferences.

devicePrefGUI is derived from *GUIManager* because a GUI must be displayed that allows the user to change the preferences pertaining to the device as desired. This is beneficial because the user is allowed to have complete control over the device.

DevinfoGUI is derived from *GUIManager* due to the principle that it will display information about the specific device. This is important because the device's information will almost constantly be changing and *DevinfoGUI* needs the most up-to-date status reports on the device to provide to the user..

devInterfaceGUI is derived from *GUIManager* because a GUI needs to be shown that provides the user with preferences.

GUIcontroller is derived from *GUIManager* because there needs to be a class to regulate when a GUI is sent to the user and when it is necessary or not. This is beneficial because, without it, an unwanted GUI could pop up on the user's screen and lead to a number of confusions. For instance, a prompt could come up that wants the user's input on changing the temperature in the house. The user could put in a new number that overrides a previously inputted value, without realizing what they have done.

logGUI is derived from *GUIManager* because it will display a log in the form of a GUI to the user based on the current status of the system and it will also provide a list of events for the system in a certain period of time.

loginGUI is derived from *GUIManager* because when a user logs into autoHome, they must be prompted to enter their password. If they were to not be prompted, their account would be open for anyone to access and someone could change their settings without the user realizing it.

ruleSetGUI is derived from *GUIManager* because ruleset preferences need to be displayed for the user so that the user can choose what preferences they can change. Once again, if the GUI is not displayed, the user wouldn't be able to change them as they should.

systemInterfaceGUI is derived from *GUIManager* because a GUI needs to be displayed to show the overall system and provide options pertaining to the system such as backup, restore, wipe, check for updates, and a number of other options solely pertaining to the system as a whole.

SystemMaintenanceGUI is derived from *GUIManager* because when the system needs to check for updates, or is possibly being updated, the user needs to be prompted to update a certain part of the system, and the system needs to receive either an "update now" or "save for later" option. If the user was not prompted on when updates happen, their interface may become sluggish due to the unknown update in the background, and the system might even restart to apply the new updates to the system, without the user being aware.

UserAccountGUI is derived from *GUIManager* because when the user goes to login, upon successful login, they will be prompted with a screen showing their account in brief. They will have a brief rundown of their account and an option to change settings pertaining to their account. Without this GUI, the user would not know if they entered their password correctly, since this GUI would appear after a successful login is made, but they also would not know if their account exists since there would not be a prompt for them to see.

UserAccountManager

accountSettingGUI is derived from *UserAccountManager* due to the principle that the user should be able to change their account settings if they want, and the *UserAccountManager* should have all pertinent information stored, available for the user to change upon pulling up their "account settings."

loginGUI is derived from *UserAccountManager* due to the principle that the system should allow different users to log in to the system. *UserAccountManager* will be able to retrieve the respective account upon being authenticated.

UserAccountGUI is derived from *UserAccountManager* due to the principle that the system should be able to retrieve a certain user account and display that certain account in a GUI for the user to see. This is important because the user may want to verify any important information on their account, i.e. changing their password.

Authenticator

loginGUI is derived from *Authenticator* based on the principle that when a user enters in their user name and password, the *Authenticator* will ensure that the user name and password match. This is essential because users shouldn't have access to accounts and information that doesn't belong to them.

	Database Manger	Control Manager	Device Manager	GUI Manager	User Account Manager	Authenticator
AccountSettingsGUI				x	x	
ArrayGUI	x			x		
DatabaseController	x					
DatabaseManager	x					
DeviceController		x				
DeviceGUI			x	x		
DeviceInterfaceModule			x			
DevicePrefGUI			x	x		
DevInfoGUI			x	x		
DevInterfaceGUI				x		
GUIController	x			x		
InfoGenerate	x					
LogGUI	x			x		
LoginGUI				x	x	x
RuleSetGUI	x			x		
SensorController		x				
SystemInterfaceGUI	x			x		
SystemMaintenanceGUI	x			x		
UserAccountGUI				x	x	
Wizard	x					

Figure 2-3 - Traceability Matrix

3. System Architecture and System Design

a. Architectural Styles

Software architecture refers directly to the composition of our system. The architecture can be shown through a conceptual model, or models, that show the system's behavior and other views of our system. Our system uses a number of different software architectures, which are defined as follows:

- The client-server model allows for a client to use their computer to access a server through the Internet. [1]
- Plug-in computing is the idea that smaller software can be used and added to a system to provide additional functionality to a larger piece of software. [1]
- An event-driven architecture is where the system focuses primarily on events. An event is defined as the action that changes the state of something in the system. [1]

It may seem peculiar that our system uses a few different software architectures, but it also makes a lot of sense. Since autoHome is a considered to be a complete system, there are a lot of different parts that need to work together in order to accomplish their assigned tasks through software.

autoHome works by allowing a consumer to access a web-based interface, allowing them to access their entire home and the home's respective utilities/components. At this level, it sounds like the software architecture would fall under the client-server model, since the consumer can access the system from any computer with an Internet connection. However, on the interface, there will also be a list of the current states of all devices connected to autoHome. These states can be changed at any given time by the consumer in just a few clicks. This would now make our system seem to fall under an event-driven software architecture. Despite our system being very large, it will have number of different functionalities when it is put in consumers' hands. Nonetheless, we plan to have additional modules that will provide added functionality to the physical system as well as the software interface. These additional modules will continue to be developed for autoHome for added functionality. This makes our architecture now seem to fall under a plug-in architecture.

As you can see, once you understand how our system works and how the interface reacts to the consumer, we have to consider it under three different software architectures.

b. Identifying Subsystems

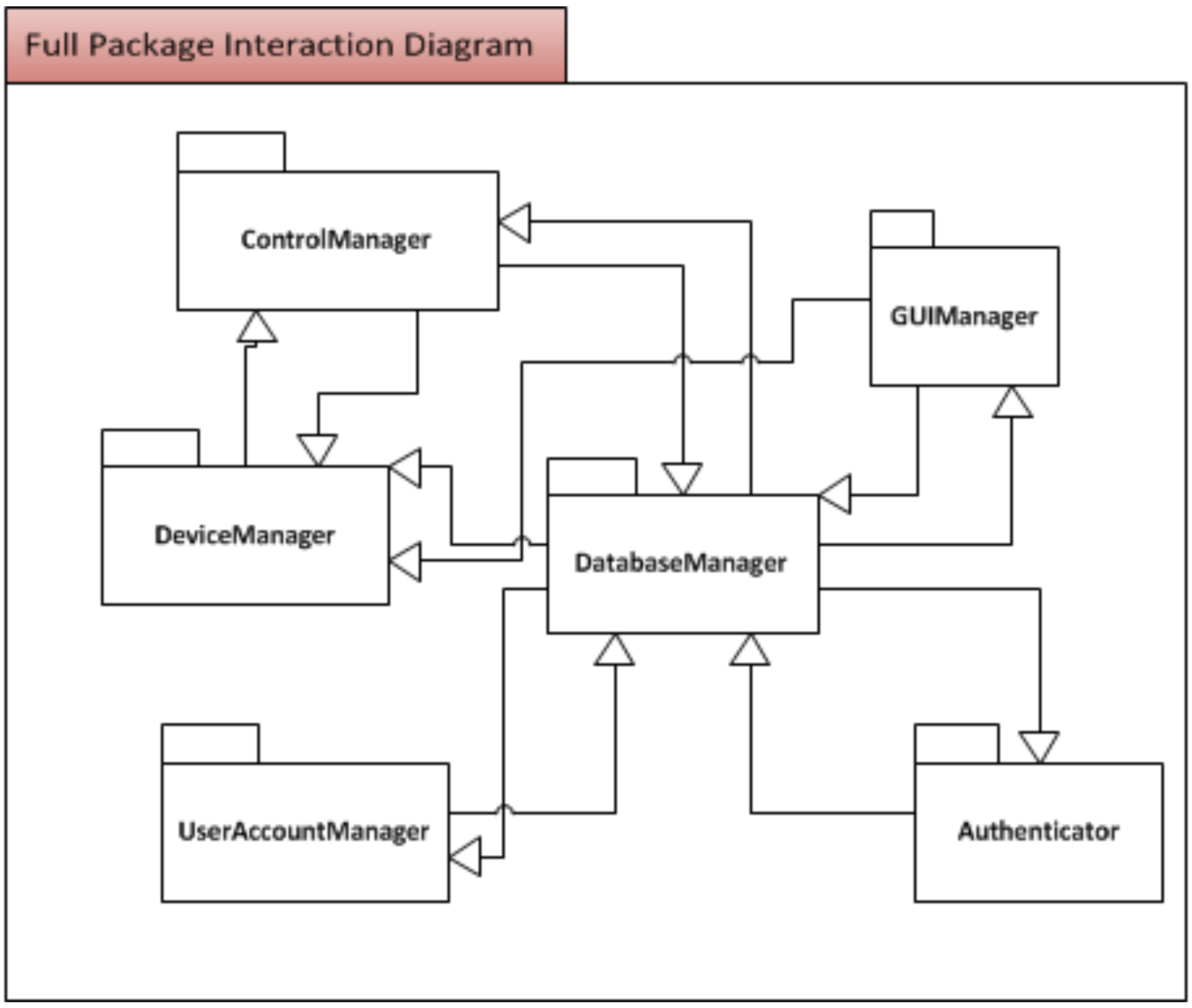


Figure 3-1 - Packaging Digram

From the traceability matrix above, it can be seen that 6 distinct packages exist in the autoHome system. These subsystems are as follows; DatabaseManager, DeviceManager, GUIManager, UserAccountManager, ControlManager, and Authenticator. As a result of these subsystems and contained classes, a relationship can be seen between the subsystems which match the classes.

c. Mapping Subsystems to Hardware

autoHome exists on a physical in home server as well as embedded hardware, tailored to work with devices in the home.

The server in the users home will be used to connect all of the devices and record system status and properties such as whether or not a light is on or if an instruction is sent to close the garage door. The server will store all of this data into a database as well as a web server to display the content stored in the database. The server will also run a daemon to query all of the devices plugged in.

Embedded systems will be used to run hardware based code, the embedded devices will transmit data between the devices and the system which in a sense give the devices the ability to talk.

d. Persistent Data Storage

Below are select table properties along with completed database schematics.

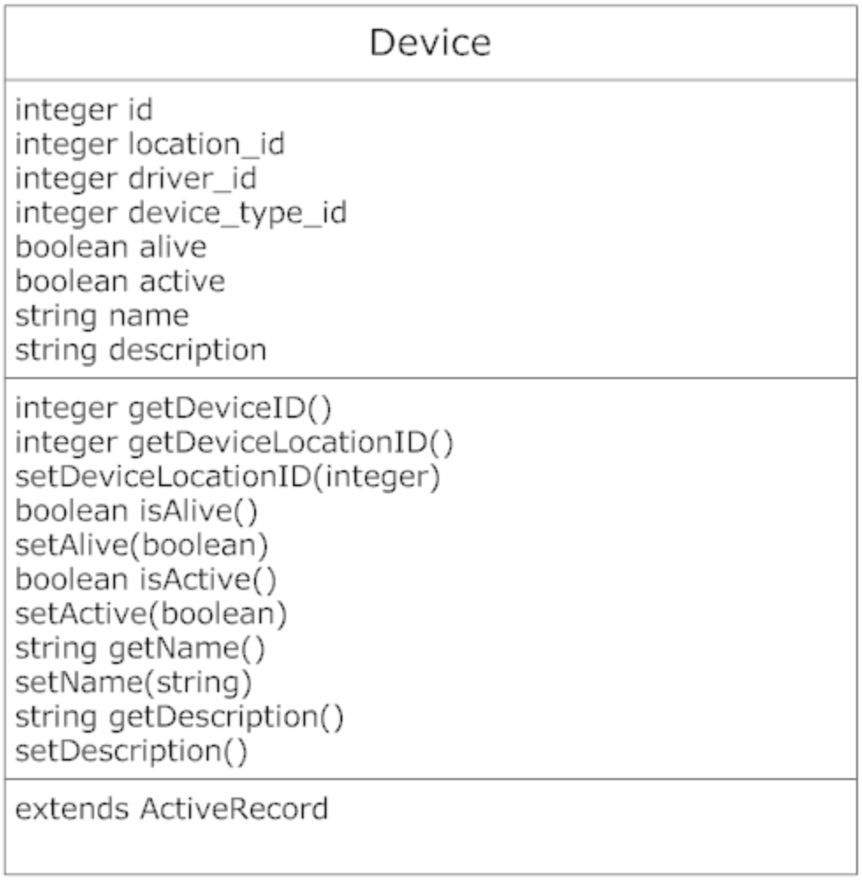


Figure 3-2 - Device Database Properties

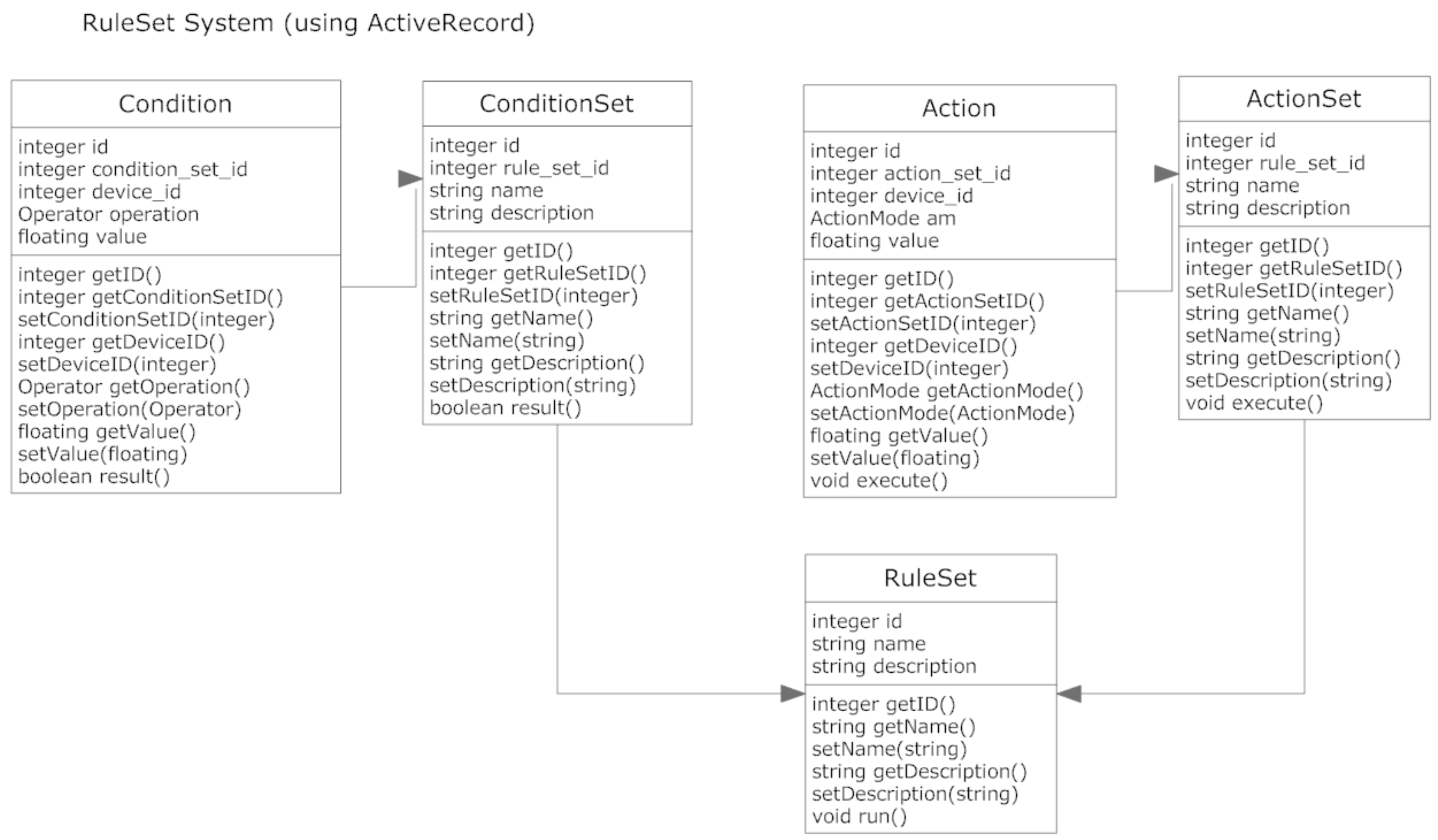


Figure 3-3 - RuleSet Database Interaction Process

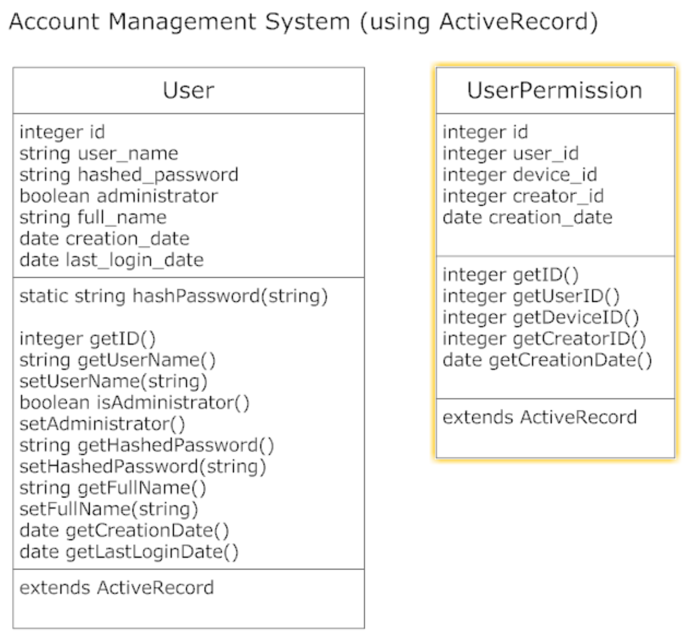


Figure 3-4 - User and User Permissions Database Properties

actions

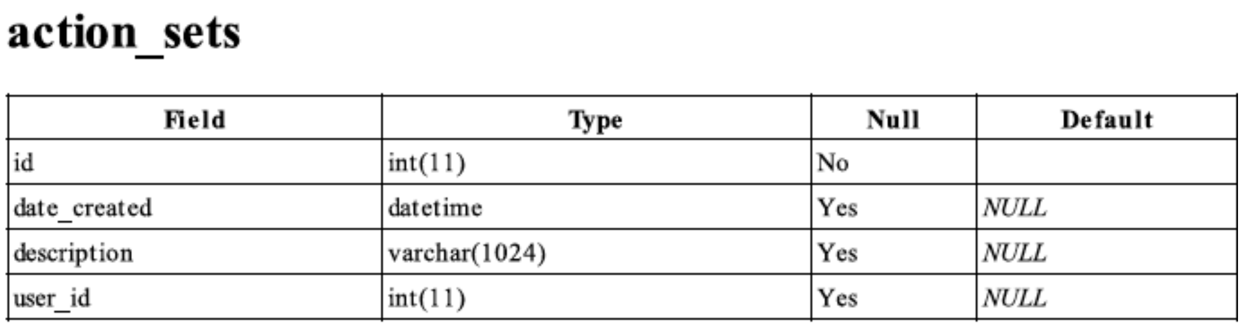
Field	Type	Null	Default
id	int(11)	No	
date_created	datetime	Yes	*NULL*
device_id	int(11)	Yes	*NULL*
value	decimal(5,3)	Yes	*NULL*
duration	int(11)	Yes	*NULL*
user_id	int(11)	Yes	*NULL*

Figure 3-5 - “actions” Database Table

action_relation

Field	Type	Null	Default
id	int(11)	No	
action_id	int(11)	Yes	*NULL*
action_set_id	int(11)	Yes	*NULL*

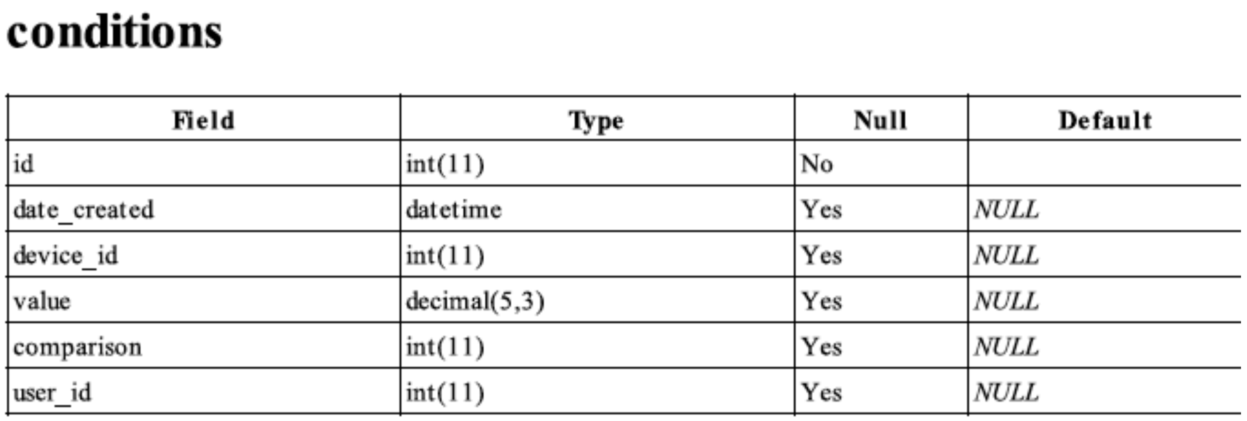
Figure 3-6 - “action_relation” Database Table



action_sets

Field	Type	Null	Default
id	int(11)	No	
date_created	datetime	Yes	*NULL*
description	varchar(1024)	Yes	*NULL*
user_id	int(11)	Yes	*NULL*

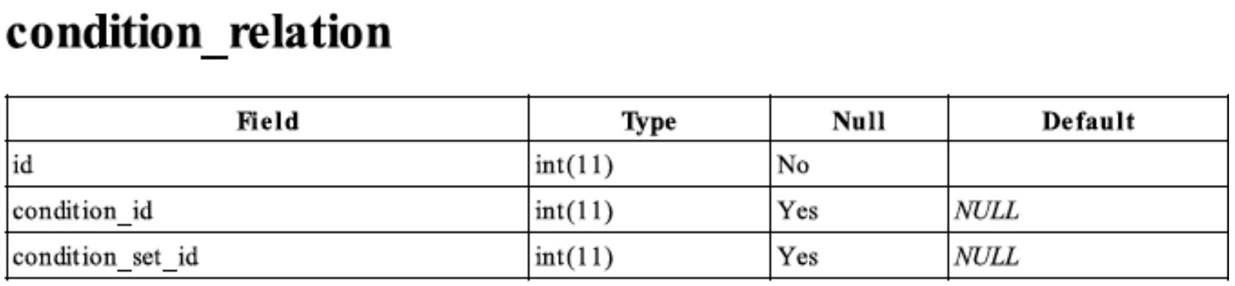
Figure 3-7 - “action_sets” Database Table



conditions

Field	Type	Null	Default
id	int(11)	No	
date_created	datetime	Yes	*NULL*
device_id	int(11)	Yes	*NULL*
value	decimal(5,3)	Yes	*NULL*
comparison	int(11)	Yes	*NULL*
user_id	int(11)	Yes	*NULL*

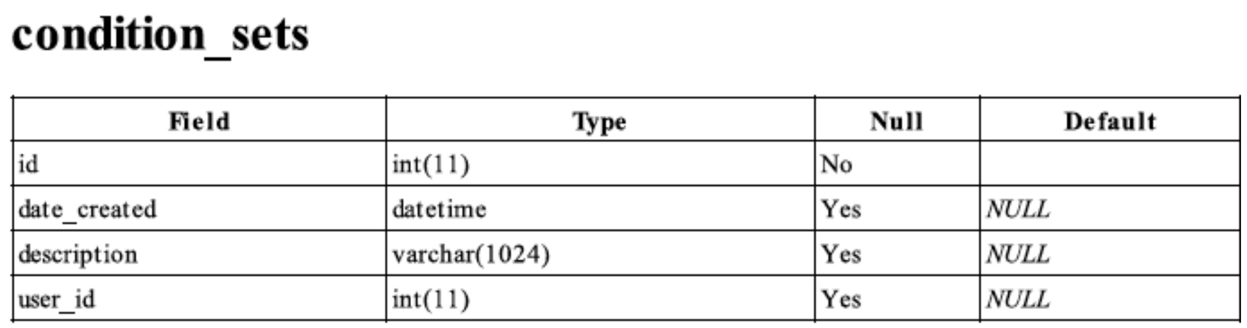
Figure 3-8 - “conditions” Database Table



condition_relation

Field	Type	Null	Default
id	int(11)	No	
condition_id	int(11)	Yes	*NULL*
condition_set_id	int(11)	Yes	*NULL*

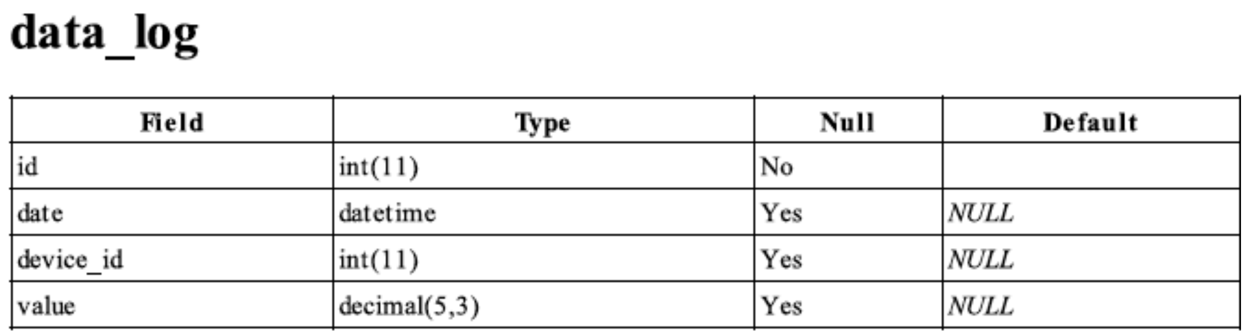
Figure 3-9 - “condition_relation” Database Table



condition_sets

Field	Type	Null	Default
id	int(11)	No	
date_created	datetime	Yes	*NULL*
description	varchar(1024)	Yes	*NULL*
user_id	int(11)	Yes	*NULL*

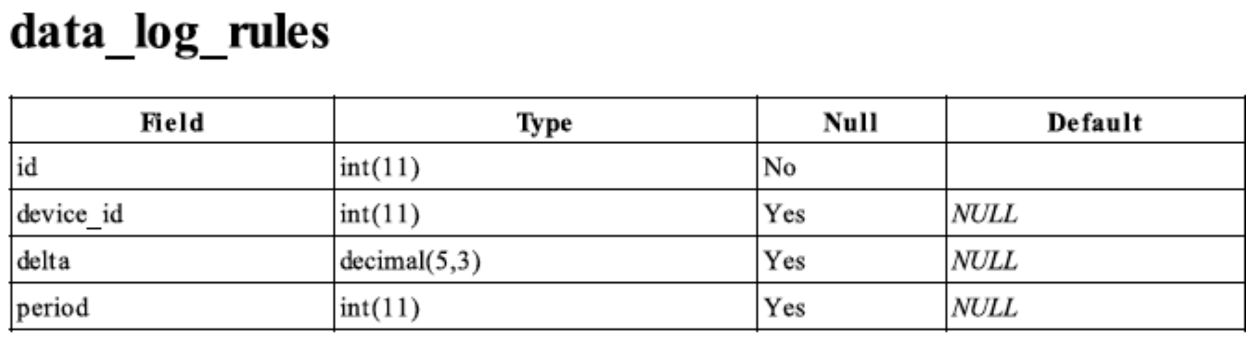
Figure 3-10 - “condition_sets” Database Table



data_log

Field	Type	Null	Default
id	int(11)	No	
date	datetime	Yes	*NULL*
device_id	int(11)	Yes	*NULL*
value	decimal(5,3)	Yes	*NULL*

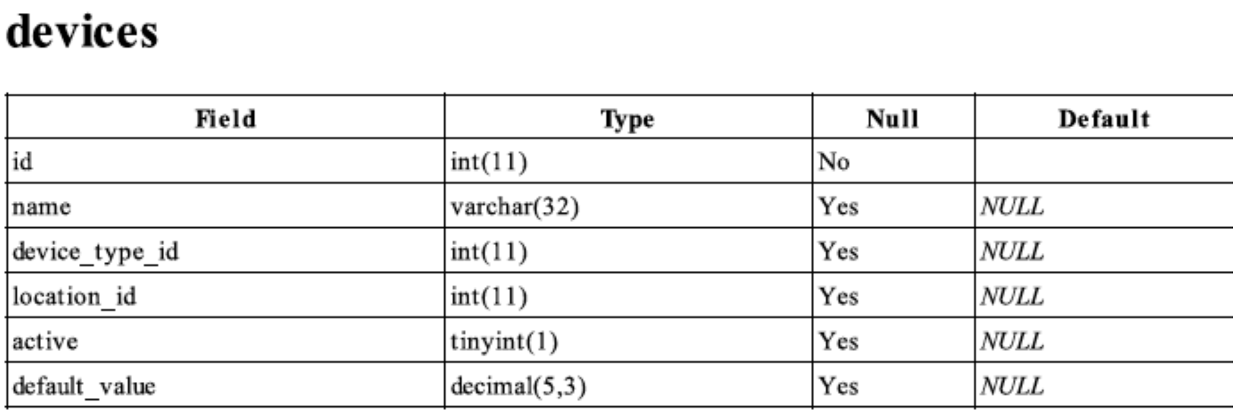
Figure 3-11 - “data_log” Database Table



data_log_rules

Field	Type	Null	Default
id	int(11)	No	
device_id	int(11)	Yes	*NULL*
delta	decimal(5,3)	Yes	*NULL*
period	int(11)	Yes	*NULL*

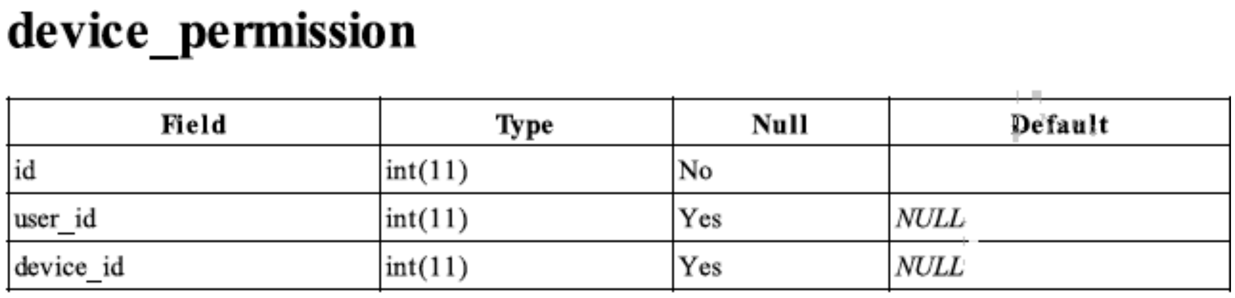
Figure 3-12 - “data_log_rules” Database Table



devices

Field	Type	Null	Default
id	int(11)	No	
name	varchar(32)	Yes	*NULL*
device_type_id	int(11)	Yes	*NULL*
location_id	int(11)	Yes	*NULL*
active	tinyint(1)	Yes	*NULL*
default_value	decimal(5,3)	Yes	*NULL*

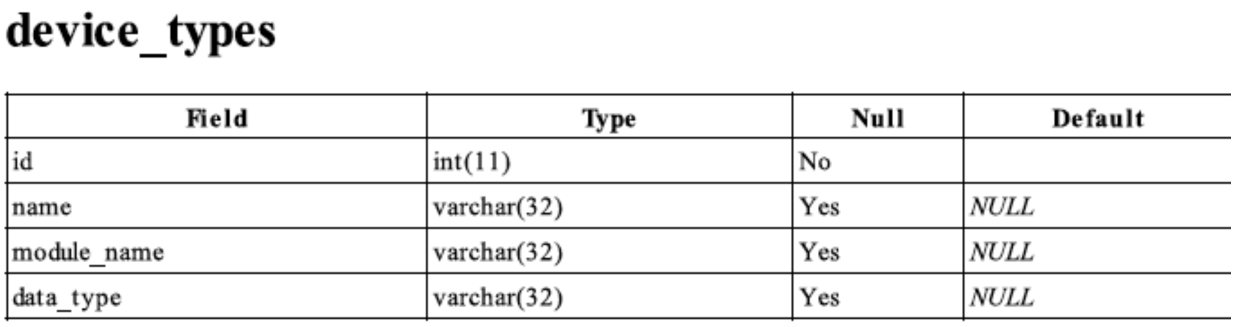
Figure 3-13 - “devices” Database Table



device_permission

Field	Type	Null	Default
id	int(11)	No	
user_id	int(11)	Yes	*NULL*
device_id	int(11)	Yes	*NULL*

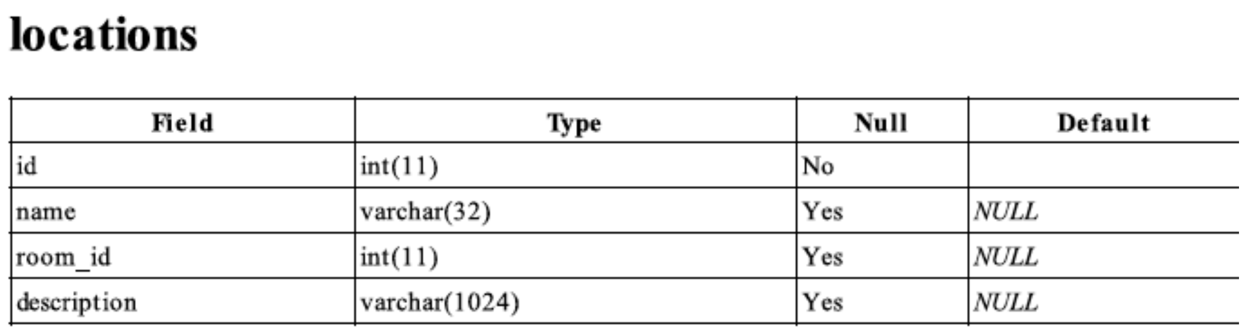
Figure 3-14 - “device_permission” Database Table



device_types

Field	Type	Null	Default
id	int(11)	No	
name	varchar(32)	Yes	*NULL*
module_name	varchar(32)	Yes	*NULL*
data_type	varchar(32)	Yes	*NULL*

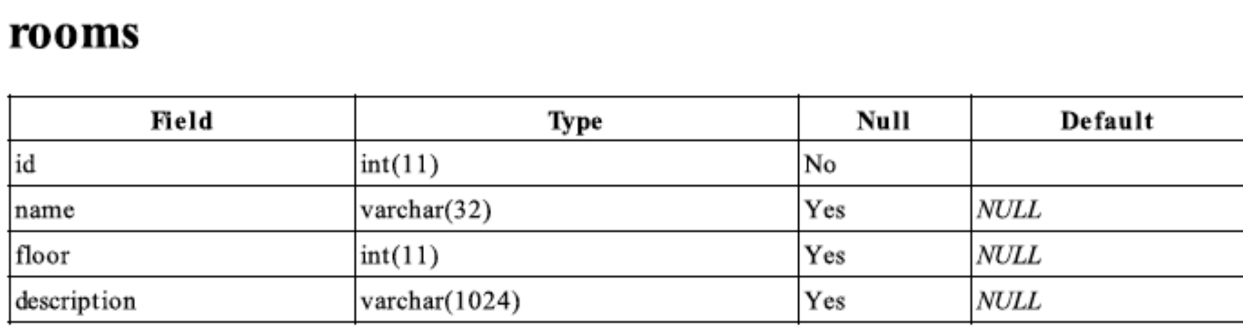
Figure 3-15 - “device_types” Database Table



locations

Field	Type	Null	Default
id	int(11)	No	
name	varchar(32)	Yes	*NULL*
room_id	int(11)	Yes	*NULL*
description	varchar(1024)	Yes	*NULL*

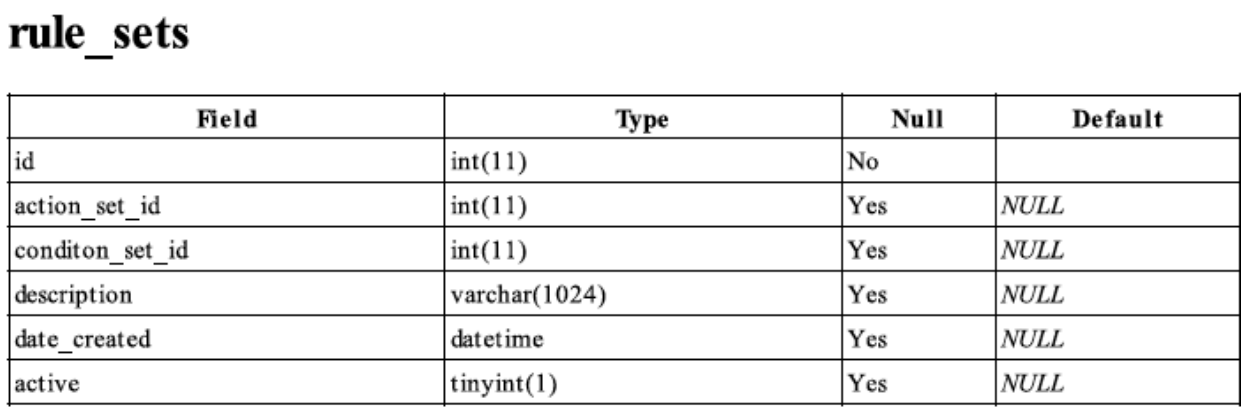
Figure 3-16 - “locations” Database Table



rooms

Field	Type	Null	Default
id	int(11)	No	
name	varchar(32)	Yes	*NULL*
floor	int(11)	Yes	*NULL*
description	varchar(1024)	Yes	*NULL*

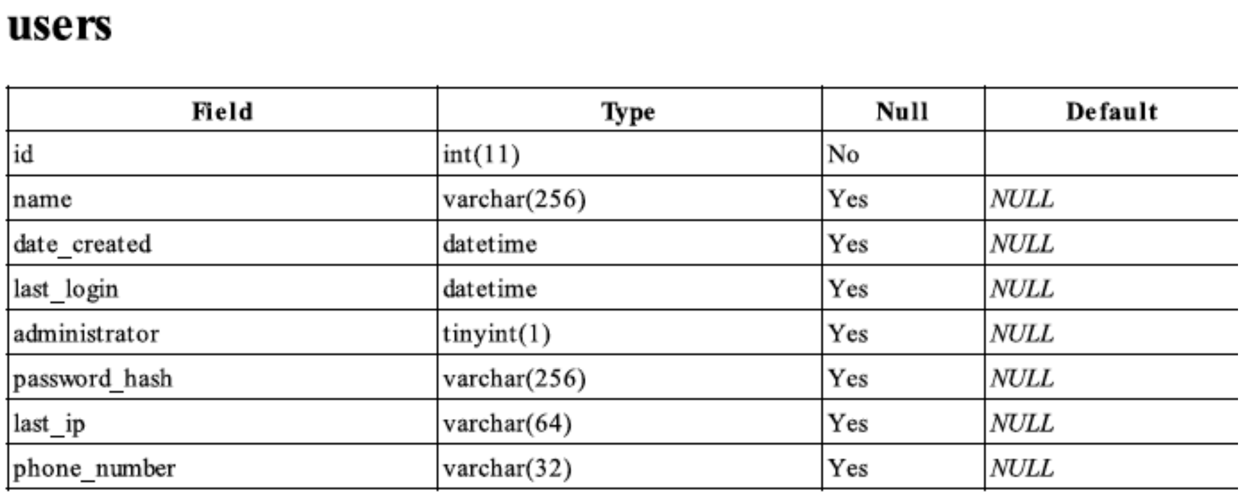
Figure 3-17 - “rooms” Database Table



rule_sets

Field	Type	Null	Default
id	int(11)	No	
action_set_id	int(11)	Yes	*NULL*
conditon_set_id	int(11)	Yes	*NULL*
description	varchar(1024)	Yes	*NULL*
date_created	datetime	Yes	*NULL*
active	tinyint(1)	Yes	*NULL*

Figure 3-18 - “rule_sets” Database Table



users

Field	Type	Null	Default
id	int(11)	No	
name	varchar(256)	Yes	*NULL*
date_created	datetime	Yes	*NULL*
last_login	datetime	Yes	*NULL*
administrator	tinyint(1)	Yes	*NULL*
password_hash	varchar(256)	Yes	*NULL*
last_ip	varchar(64)	Yes	*NULL*
phone_number	varchar(32)	Yes	*NULL*

Figure 3-19 - "users" Database Table

e. Network Protocol

autoHome uses serial communication to connect the embedded hardware to the server. Serial or RS-232 communication is an older platform and as such a FTDI chip [4] will be used to translate the serial communication to Universal Serial Bus or USB. USB will serve two purposes for autoHome; It will power low level devices in the range of 0 - 5v as supplied from the USB standard, while the up and down streams will be used to transmit messages that can be read in to the embedded hardware. Messages can easily be read on the embedded devices through built in serial libraries, while the server can read incoming messages with some simple C code.

f. Global Control Flow

The autoHome system is an event driven system where actions are based from cause and effect type actions such as "turn on a light in this room" then the light in the room turns on. Also an event doesn't just have to come as a result of a user based action, an event can come from a timer such as an alarm to wake up or a sensor being triggered such as a fire alarm.

System time will be maintained by the computer clock which is also synchronized with the national atomic government time [2] once a week. Accuracy of time is not crucial because the timing of events that rely on time such as turning on a sprinkler really do not need to be more accurate than to the nearest minute. Time is also backed up through use of a simple clock battery that can last around fifteen years.

autoHome is a real-time system however multiple actions can occur at any given time and the system will be able to prioritize by importance such as a safety control like a fire alarm over tuning on a light but it can also process tasks in a queue like system.

g. Hardware Requirements

Our server hardware does not need to be too sophisticated given that it will only be processing PHP, MySQL, AJAX, jQuery, and C. To enhance system performance, the following specification is recommended:

Processor: Intel Pentium 4 2.0 GHz or better
Memory: 512MB or more
Storage: 50GB or larger
Broadband Internet Connection ≥ DSL 1.2 MBits/s or equivalent

Client (Administrator) platform requires the following hardware specification to run our software:

Processor: Intel Pentium III 2.0GHz or better
Memory: 256MB or more
Storage: 2GB or more
Display: 1024 x 768 16-bit color or better

Client can also access the system interface via Android devices [3]. Our Android software requires the following hardware specification:

OS: Google Android 2.1 or later
Processor: Qualcomm MSM8250 768MHz or better
Memory: 512MB or more
Storage 1GB or more
Display: HVGA 480 X 320 16-bit color or better

4. Algorithms and Data Structures

The autoHome system has been designed in such a way that it requires no algorithms to be implemented in order for decisions to occur from within the system. This is due to the fact that many of the processes occurring in the system occur on a schedule such as “query the weather, every hour” this requires no algorithm to make this happen.

Because of the structure of the database, all information is stored within the database tables. As a result when the user wants to for example see a list of all devices on the system, all the system has to do is spit out the table that contains all of the devices. It is with this simplicity and convenience in using MySQL with PHP that this can be done all without implementing any data structures.

5. User Interface Design and Implementation

a. Design and Implementations

The following pages display our interface design with brief explanations.

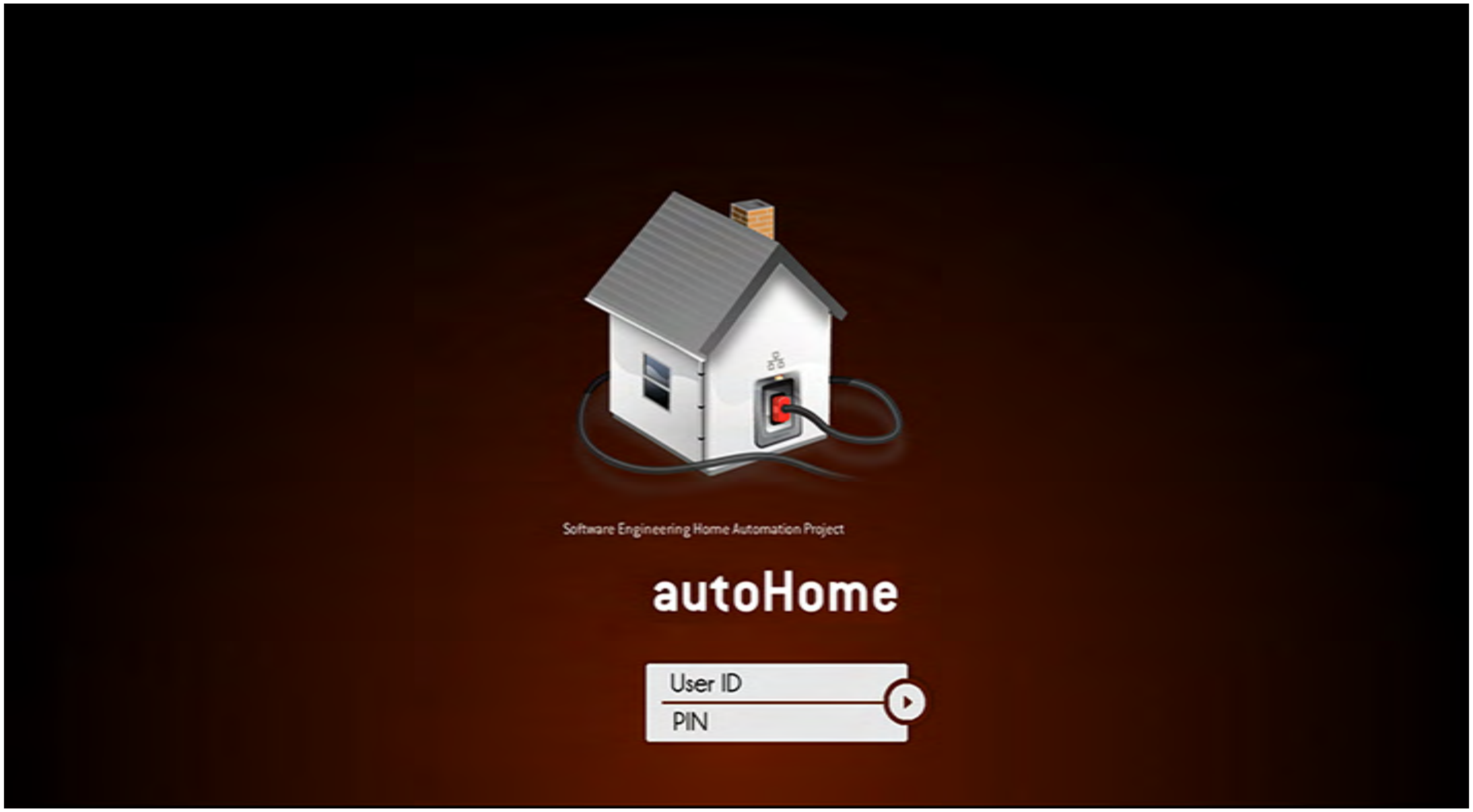


Figure 5-1 - Login

The user is initially greeted with the login page for the web interface of autoHome. Here, they will login with their given ID and PIN to access the interactive settings section of the web interface.

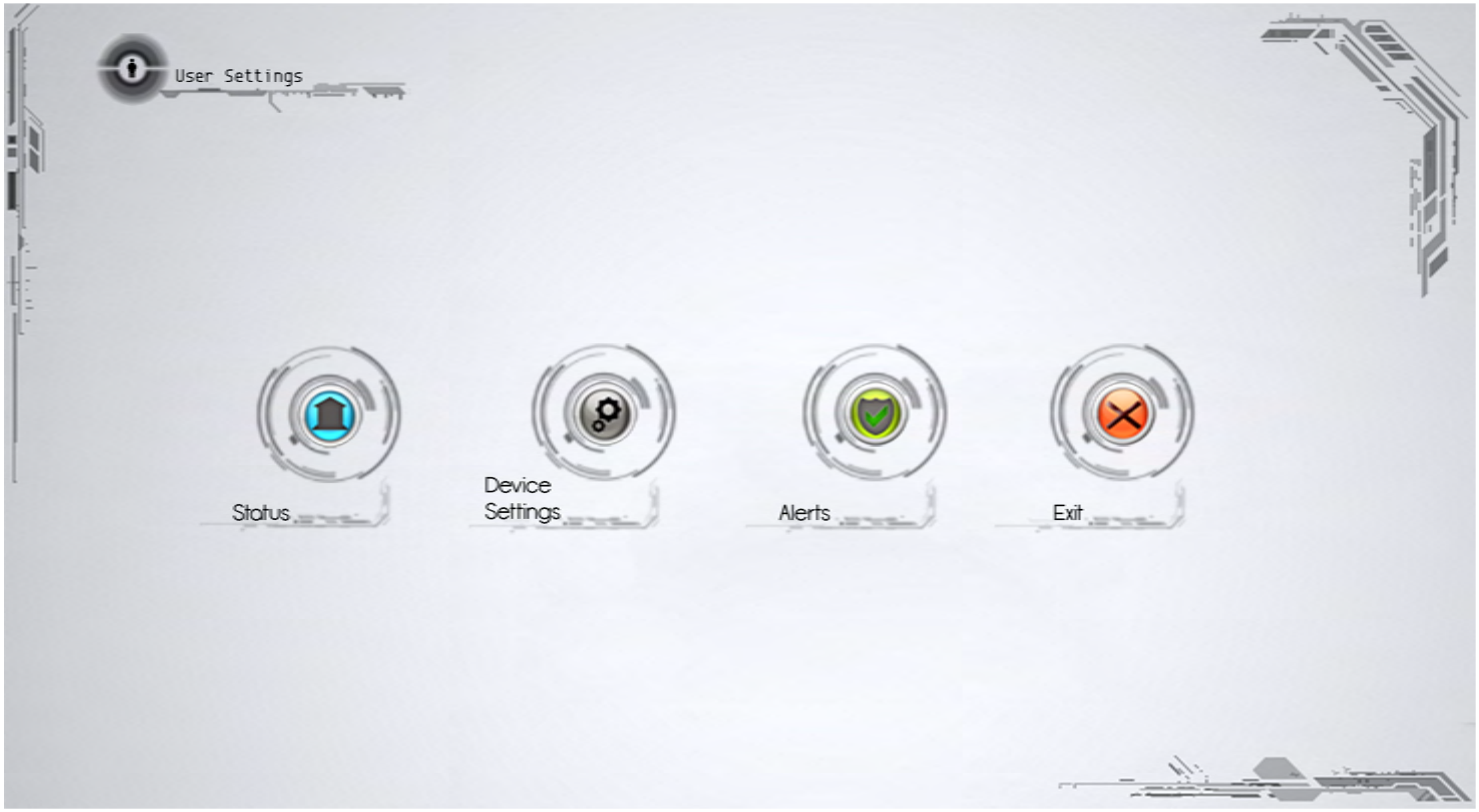
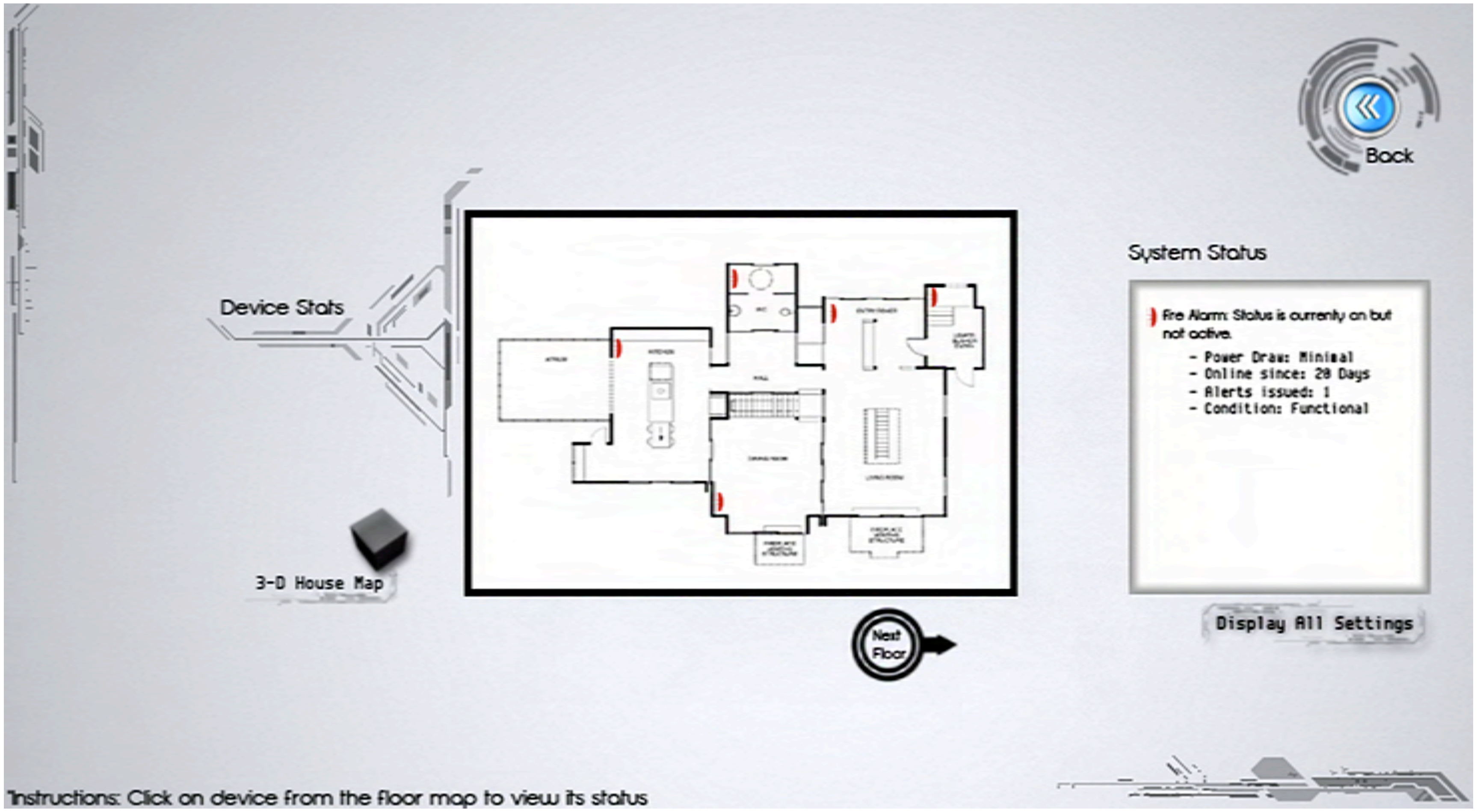


Figure 5-2 - Selection

After logging in successfully, the user is presented with a very simple selection of options. This is an effort to make the system as user friendly as possible. The selections range from Status, Settings, Alerts, and Exit. Status button redirects the user to the status page, which displays the status of the devices connected to the system. Settings redirects the user to the settings page where the user can configure the devices. Alerts will redirect to the alerts page where all of the alerts are displayed in a simple format for the user to check. The Exit button simply exits from the web interface. This screen is made larger to augment with the entire minimalist and user friendly theme.



System Status

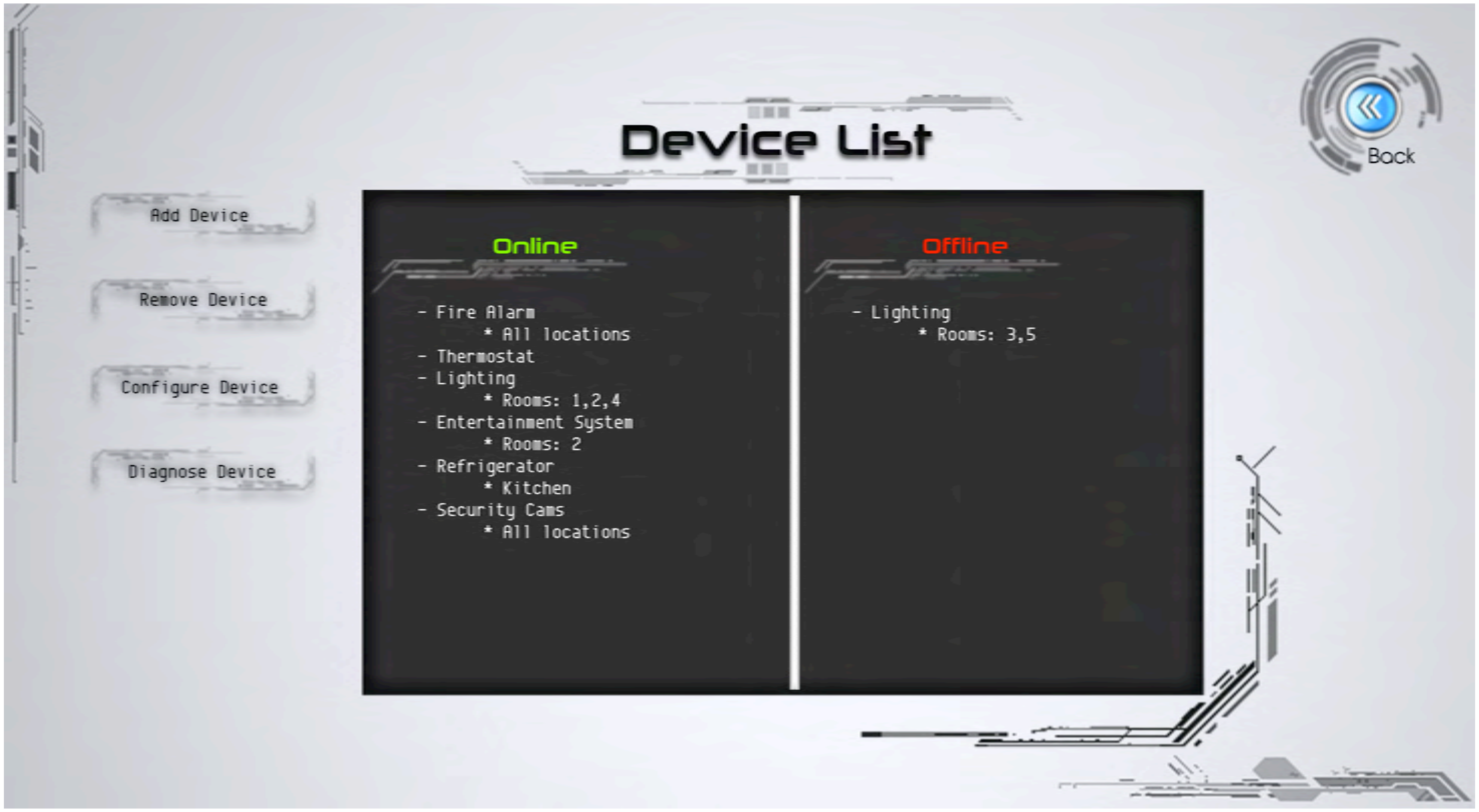
Fire Alarm: Status is currently on but not active.

- Power Draw: Minimal
- Online since: 20 Days
- Alerts issued: 1
- Condition: Functional

*Instructions: Click on device from the floor map to view its status

Figure 5-3 - Status

Selecting the Status option will bring the user to this page where the user can check the status of the devices connected to the system. To eliminate any complexity a graphical interface is used where all the connected devices are shown in their respecitve location on the floor map, clicking the “next floor” button will change the floor, there is also a “previous floor” button as well but it only appears with the user is on a higher floor. The 3-D house map will show the entire house in a three dimensional view for asthetic purposes. The “Display All Settings” will show the basic working status of all the devices connected to the system. This section’s requirements stem from requirements 5 and 6, it would be trivial to include a completely different requirement since the status page basically utilizes the information from these requirements and display’s them in a much more presentable way. The 3D view is basically an asthetic option, which is there to show a more grandoise prespective of the house.

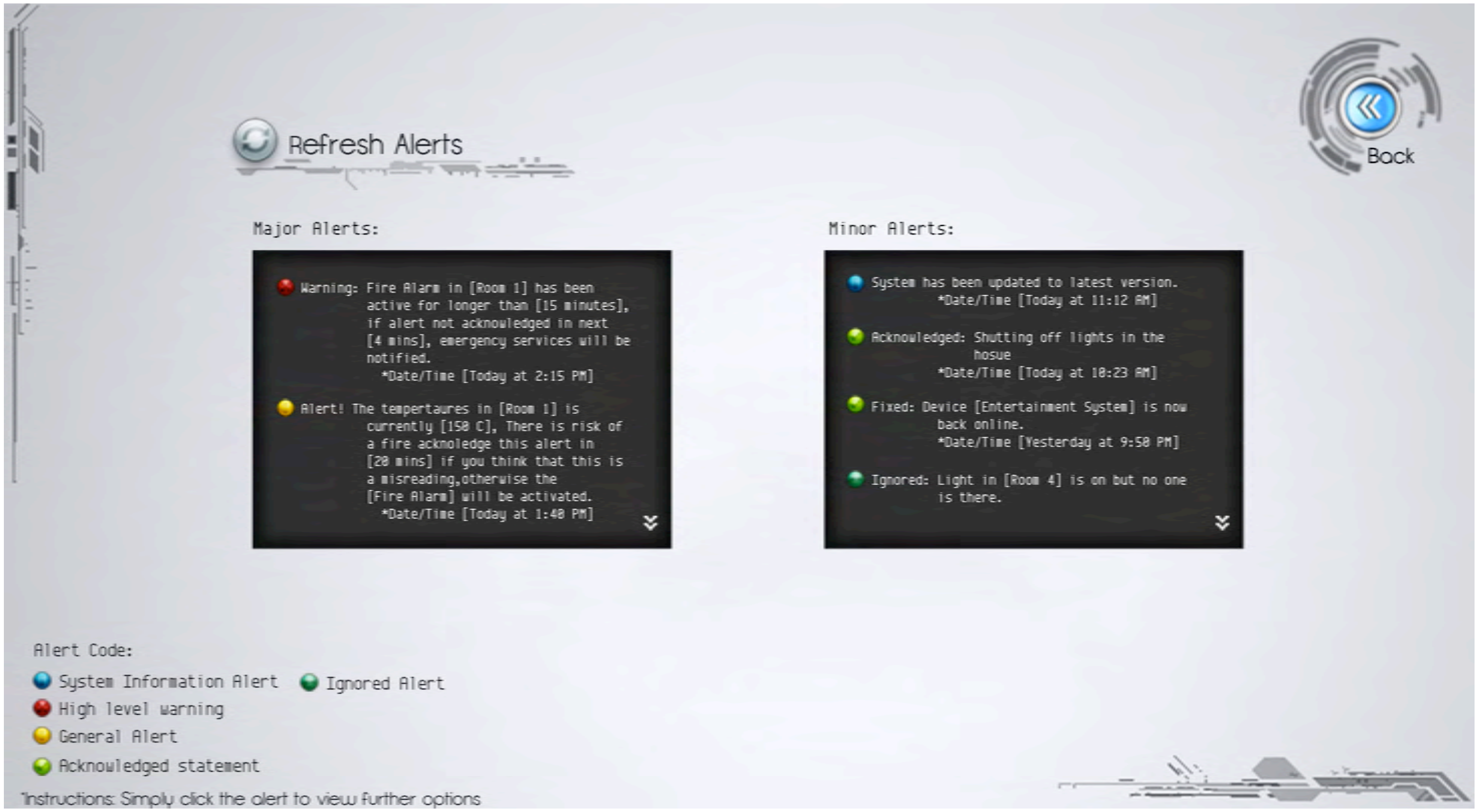


```
- Fire Alarm
        * All locations
- Thermostat
- Lighting
        * Rooms: 1,2,4
- Entertainment System
        * Rooms: 2
- Refrigerator
        * Kitchen
- Security Cams
        * All locations
```

```
- Lighting
        * Rooms: 3,5
```

Figure 5-4 - Settings

The Settings page gives the user the options to add, remove, configure or diagnose a device. The add and remove functions open a dialogue box where the user can simply select from a preconfigured list of common devices or manually set up a device. Remove device is very simple, clicking remove device will prompt the user to select a device from the center panel device list and the selected deivce disconnects from the system and no longer appears on the list. Diagnose device prompts the user to select a device from the list and automatically begins to check if there are any connection issues with the device, and reports the issue to the user. The device list has an online and offline section, this basically shows which devices are being used and which devices are not being used along with the locations of those devices.



Major Alerts:

● Warning: Fire Alarm in [Room 1] has been active for longer than [15 minutes], if alert not acknowledged in next [4 mins], emergency services will be notified.
*Date/Time [Today at 2:15 PM]

● Alert! The tempertaures in [Room 1] is currently [150 C], There is risk of a fire acknoledge this alert in [20 mins] if you think that this is a misreading,otherwise the [Fire Alarm] will be activated.
*Date/Time [Today at 1:40 PM]

Minor Alerts:

● System has been updated to latest version.
*Date/Time [Today at 11:12 AM]

● Acknowledged: Shutting off lights in the hosue
*Date/Time [Today at 10:23 AM]

● Fixed: Device [Entertainment System] is now back online.
*Date/Time [Yesterday at 9:50 PM]

● Ignored: Light in [Room 4] is on but no one is there.

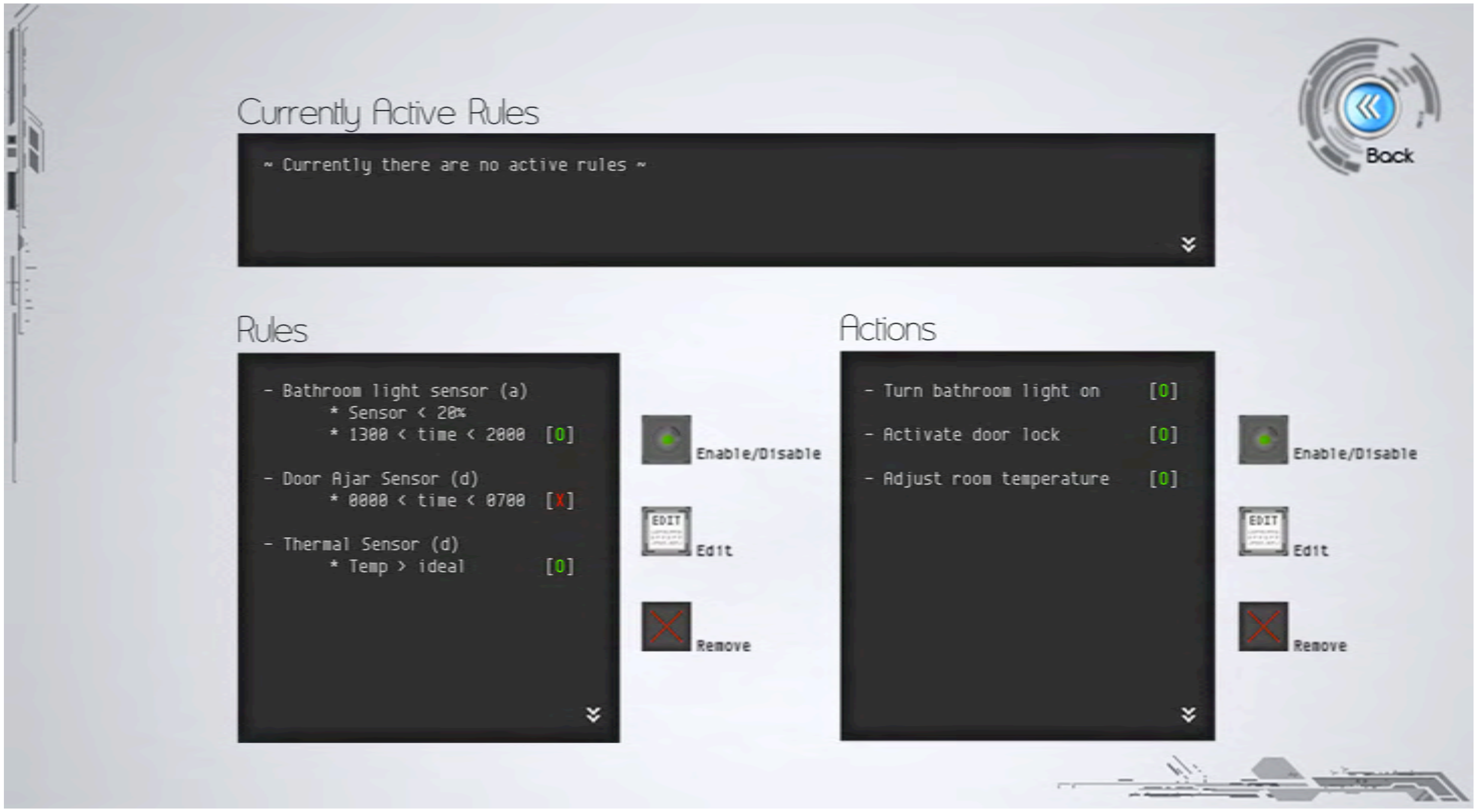
Alert Code:

● System Information Alert
● Ignored Alert
● High level warning
● General Alert
● Acknowledged statement

*Instructions: Simply click the alert to view further options

Figure 5-5 - Alerts

The Alerts page displayed is seperated into two columns major alerts and minor alerts, this allowes the user quick access to the any messages generated by the system. The system itself can distinguish major and minor alerts quite easily, major alerts are generated if the alert is based on a data anomoly such as a major inconsistancy in the data pool ie. If there is large change in room temperature that is nor consistent with the previous data, constituting to a fire there; Minor alerts however, are basically just alerts that the user has acknowledged or any system based message such as updates and the daily activity from users. The major alerts secion is seperated so that the severe alerts are not buried within alerts that do not require as much attention. To act on the alert itself the user has to simple select the alert by clicking on it, this brings up a window with the following options: “Details”, “Ignore”, and “Close”, the user would then have to click one of those options to proceed. Clicking on Details gives a much more elaborate description of the alert, such as where the alert is originating from, what caused the alert and a log of previous alerts of the same kind and the choice to acknowledge the alert, which would then be maked as acknowledged. Clicking details from any alert in the major alert’s section gives the user the option to contact emergency services (which the system would automatically do if the user does not acknowledge the alert in a given time-span) if need be. Clicking ignore will simply ignore the alert, if this alert was in the major alert’s section then it would just be moved into the minor alerts section; however, if the alert is in the the minor alert’s section then the alert is just deleted. Close is quite self-explanitory it just closes the window without doing anything. The Alerts system is based on requirement 6, since it’s a means of communicating with the user by reporting any issues or status-updates to them.



Currently Active Rules

```
~ Currently there are no active rules ~
```

Rules

```
- Bathroom light sensor (a)
      * Sensor < 20%
      * 1300 < time < 2000  [O]

- Door Ajar Sensor (d)
      * 0000 < time < 0700  [X]

- Thermal Sensor (d)
      * Temp > ideal        [O]
```

Actions

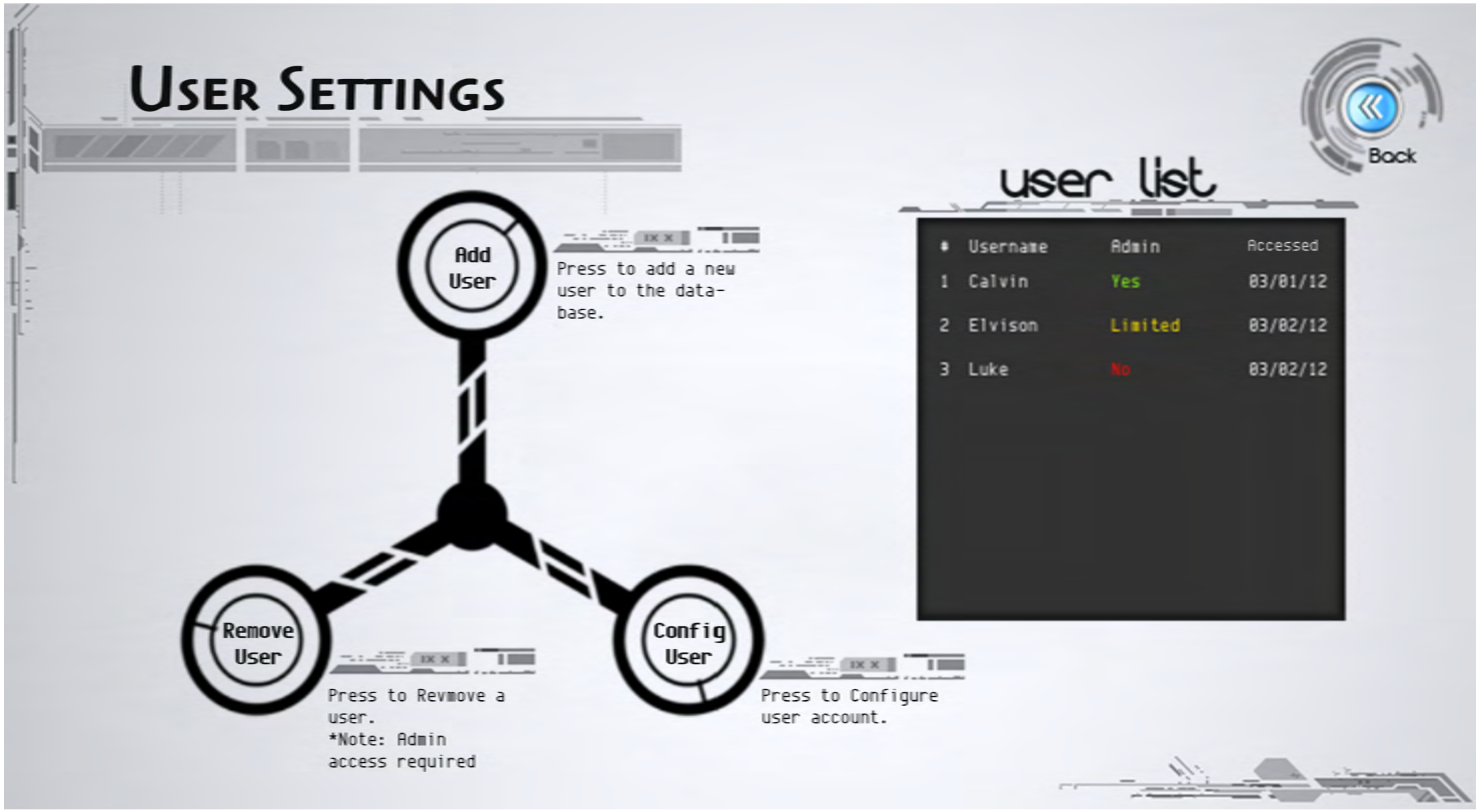
```
- Turn bathroom light on      [O]

- Activate door lock          [O]

- Adjust room temperature     [O]
```

Figure 5-6 - DeviceRules

The Device rules page allows the user to set rules to certain devices. This includes setting the lighting to turn on at a certain time. This allows flexibility to completely customize rules to how the user wishes to “run” the house. Rules are set and actions can be enabled to augment the rules.



Press to add a new user to the data-base.

Press to Revmove a user.
*Note: Admin access required

Press to Configure user account.

user list

#	Username	Admin	Accessed
1	Calvin	Yes	03/01/12
2	Elvison	Limited	03/02/12
3	Luke	No	03/02/12

Figure 5-7 - User Settings

The User settings page is basically a backend user management system. It allows users to configure their information, add new users and/or remove users. Adding a new user is quite basic, a system admin can add a user and give access permissions; these access permissions are ment to restrict non-administrative users from attempting to configure devices that are restriced to them, ie. Modules that control the fire-alarms are not accessible to non-administrative users unless the admin decides to give permission for that control and something as simple as removing a user from the database. The admin can give control to specific devices that are restriced either when adding a new user or by configuring the user profile at a later time, this would mark the user as limited in admin rights since they are unable to perform all administrative actions and only a select few that have been given to them. Removing a user as described before is also only possible by an administrator, this ensures a level of safety so normal users can't go about removing any user they wished, which might lead to issues down the road. Configure User, the selection of this brings up a dialogue which is slightly different for admin's and normal users. Normal user config window allows them to make profiles for device settings, change their information, and start/stop messaging services. The admin config window has all of the normal user's configuration options and another set of options which allow the admin to configure permissions for other users in the system. There is a user list window displayed for information purposes to show all the users in the database and the dates which they accessed the system.

b. User Interface Changelog

User Interface: Report #2 Changes

Changes were made with respect to the feedback that was given; only the portions that underwent changes are going to be displayed here. Descriptions have been adjusted in order to describe further details.

Change log:

- Picture resolution has been increased for better detail

- New section: User Settings has been added, this was not included in report 1 and has now been added with new descriptions.

- Settings choice in the Selection Page is now called Device Settings for ease of understanding

- Requirements for the proposed sections: Status and Alerts has been stated in further detail.

- Remaining parts have been left untouched.

In order to maintain picture quality, only the descriptions have been pasted here the pictures will be in a separate zip file.

6. Design of Tests

a. Casual Test Cases

Use Case 1: Authenticate User

A user logs in to his/her account in order to configure the alarm system. The tests confirm that the correct passwords and usernames allow the logins to the correct account.

Use Case 4: Configure User Device Permissions

An administrator grants users the ability to access a specific device using the web-interface. Access to a device allows the user to create Rules polling from the device if it is a sensor or Actions commanding the device if it is a controller. The tests ensure that the device data is updated and log data is accessible.

Use Case 6: Display Device Information

A user displays information about a device that the user has access to. The web interface will bring up information including the device location, the device name, device ID, and data log information about the device's history in tabular or graph format.

Use Case 12: Add Device

An administrator will log in and configure a new device. The administrator shall add ID, driver type, name, and location to the database through the web interface. A successful test will result in a device usable to the autoHome system.

Use Case 14: Send Device Signal

A value is sent to a device driver to be converted to a correct voltage, and then sent through controlling conductors to set device values. Tests will assure that the software-set values will accurately reflect the values actually set in the physical device.

User Case 15: Receive Device Signal

A voltage is read by the device driver and converted to a valid value usable by humans. The tests assure that the values in the software reflect the values of sensors.

Use Case 18: Send Virtual Device Signal

A programmed routine is executed by the system with specified parameters. The routine can be anything from text messaging, to phone calls, to website updates. A successful test will result in these actions being successfully taken.

Use Case 19: Receive Virtual Device Signal

The autoHome system may receive signals from virtual devices such as system time, internet weather, or external information. The test will assure that the information pulled from external sources is correct.

b. Descriptive Test Cases

Use Case 1: Authenticate User

Success

- User inputs username and password successfully. The system has to verify if the username and password exists and matches in the system. The match is made and the user is logged into his/her account.

Failure

- User inputs username and password, but when the system goes to verify, the username is incorrect and it doesn't match any username in the database and it displays a "wrong user-name/ password" error message.
- User inputs username and password, but when the system goes to verify, the password does not match with the one saved in the database and it displays a "wrong username/ password" error message.
- User inputs username and password, but when the system goes to verify, neither the user-name nor the password match with the one saved in the database and it displays a "wrong username/ password" error message.

Use Case 4: Configure User Device Permissions

Success

- The authenticated administrator signs into his/her account successfully. From this point, he/ she goes to the settings of the security device. Within this menu, he/she enters the menu of the individual sensors or controllers to the security device and enables the sensors and con-trollers to their given security device.
- The administrator is able to add users to command to a particular device. These users will now have access to use the device in their rule sets.

Failure

- The authenticated administrator signs into his/her account successfully. From this point, he/ she goes to the settings of the security device. Within this menu, he/she enters the menu of the individual sensors or controllers to the security device and tries to enable the sensors and controllers of the given security device for another user to use, but one or more permissions did not follow through and it denies access to some or all of the security devices.

- The authenticated administrator signs into his/her account successfully. From this point, he/she goes to the settings of the security device. Within this menu, he/she tries to enter to the menu of the individual sensors or controllers to the security device but the autoHome interface displays a different menu from that isn't related to security devices or security device permissions.
- A user tries to change the permissions to the security device, but an error message pops up displaying that only an authenticated administrator has access to change the device permissions.
- A user with permissions to a device is denied access to use in their rule sets.

Use Case 5: Configure Device Rule Sets

Success

- An authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will be able to select a security device and set the different schedules and conditions for the given security device and/or security sensors successfully. By doing this, the user will be able to turn on the light switches on the sensors and the motion sensors for the alarms successfully.
- When the condition sets for an enabled rule set are matched, the corresponding action set will be executed.

Failure

- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu, but the autoHome interface displays a different menu from that isn't related to the security device list device rule sets.
- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will want to select a security device and set the different schedules and conditions for the given security device and/or security sensors but the system configures a different schedule or condition instead of the one intended.
- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will want to select a security device and set the different schedules and conditions for the given security device and/or security sensors but the system doesn't turn on the lights for the sensors at the given time, or it manages to set the time for a different device.
- The condition set does not trigger its related action set.

Use Case 6: Display Device Information

Success

- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will be able to select any security device and be able to view where the security device is located on a floor map and view the data values the security device is producing, in text and/or graphs. The autoHome database will keep all of the data logs stored in memory in case of any error occurs within the system.
- Device settings in the database are displayed clearly and correctly in the user interface.

Failure

- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu, but the autoHome interface displays a different menu from that isn't related to the security device list or its information.
- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will be able to select any security device and be able to view where the security device is located on a floor map and view the data values the security device is producing, in text and/or graphs, but the interface only shows the floor maps to the security device, but not the values nor the text or graph data.
- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will be able to select any security device and be able to view where the security device is located on a floor map and view the data values the security device is producing, in text and/or graphs, but the interface doesn't show the floor maps, and only shows the texts and/or graphs.
- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will be able to select any security device and be able to view where the security device is located on a floor map and view the data values the security device is producing, in text and/or graphs, but the interface doesn't any type of data
- The authenticated user logs into his/her account successfully. He/she then manages his/her way down to the security device list menu. In this menu, he/she will be able to select any security device and be able to view where the security device is located on a floor map and view the data values the security device is producing, in text and/or graphs, but the interface shows data of other devices in the house and not the security devices.

<u>Use Case 12: Add Device</u>

<u>Success</u>

- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "add new device" option and then the user will install the device manually and the system will be able to read it automatically. Then the administrator will enable the device accordingly.

<u>Failure</u>

- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "add new device" and it fails to work.
- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "add new device" and it works, but then when the administrator attaches the security device manually, the system won't read it.
- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "add new device" and it works, then the administrator attaches the security device and when he/she tries to enable it, it does not enable.

<u>Use Case 13: Remove Device</u>

<u>Success</u>

- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "remove device" option and then Then the administrator will disable the device accordingly. The user will remove the device manually.

<u>Failure</u>

- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "remove device" and it fails to work.
- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "remove device" and it works, but then when the administrator detaches the security device manually, the system still believes the security device is still attached.
- An authenticated administrator logs into his/her account successfully. He/she enters the device list menu and clicks on the "remove device" and it works, but when the administrator tries to disable the device, the system still leaves it enabled and it refuses to disable the security device, resulting the device to still is presentable within the autoHome user interface. This will result in false and inaccurate data when monitoring the entire house's other devices.

<u>Use Case 14: Send Device Signal</u>

<u>Success</u>

- Without flaw, the system specifies values to the set security device that are currently connected. Then the device interfacing modules convert the values into meaningful voltages to set the controlling conductors. When no device is present, the interfacing modules provide a default value to be held for the controllers.

<u>Failure</u>

- The system specifies the wrong values to the set security device that is currently connected.
- The interfacing modules don't convert the values to meaningful voltages or they convert the values into a higher or lower than needed voltage for the specified security device value.
- When the device or devices are not present, the interfacing modules stay with the previously given voltage value for the device and it does not recognize that the device has been removed.
- The interfacing module configures the voltage and specified value from the security device to another device in the home, and/ or vice versa.

<u>Use Case 15: Receive Device Signal</u>

<u>Success</u>

- Without flaw, the system receives values from the set security device that are currently connected. Then the device interfacing modules convert the values into meaningful voltages to set the controlling conductors. When no device is present, the interfacing modules provide a default value to be held for the controllers.

<u>Failure</u>

- The system receives the wrong values from the set security device that is currently connected.
- The interfacing modules don't convert the values to meaningful voltages or they convert the values into a higher or lower than needed voltage for the received security device value.
- When the device or devices are not present, the interfacing modules stay with the previously given voltage value for the device and it does not recognize that the device has been removed.
- The interfacing module configures the voltage and received value from the security device to another device in the home, and/ or vice versa.

<u>Use Case 18: Send Virtual Device Signal</u>

<u>Success</u>

- The system successfully sets an active value to the system to use as a controller value in the rule set. The user can check the values of the text message profiles, email profile, and shell-script profile IDs in the interfacing module.

<u>Failure</u>

- The system unsuccessfully sets the active values to the system to use as a controller value in the rule set, and inputs different values or possibly none.
- The user is unable to enter the interfacing module to view the ID profiles for the security device that is wanted to be checked.

<u>Use Case 19: Receive Virtual Device Signal</u>

<u>Success</u>

- The system successfully receives an active value to the system to use as a sensor value in the rule set. The user can check the values of the system time, system date, and information such as when the sensor detected a movement in the interfacing module.

<u>Failure</u>

- The system unsuccessfully receives the active values to the system to use as a sensor value in the rule set.
- The system receives the active values to the system but configures invalid values or possibly none.
- The user is unable to enter the interfacing module to view the ID profiles for the security device that is wanted to be checked

<u>Use Case 20: Evaluate Data Against Rule Sets</u>

<u>Success</u>

- New data is received and it is checked against the relevant active rule sets that contain the data value as a condition in its condition set. Then when the condition is met, the associated action set will be executed. The user will try to break into his own home or try to move in front of the motion sensors and see if he/she would receive a text message or email, and to see if the alarm would go off.

<u>Failure</u>

- No data is received

- New data is received but when checked against the relevant active rules sets; it does not match towards its condition set.
- When the user tries to break into his/her own home or try to move in front of the motion sensors, he/she does not receive any alert messages
- When the user tries to break into his/her own home or try to move in front of the motion sensors, the alarm does not go off.

7. Project Management and Plan of Work

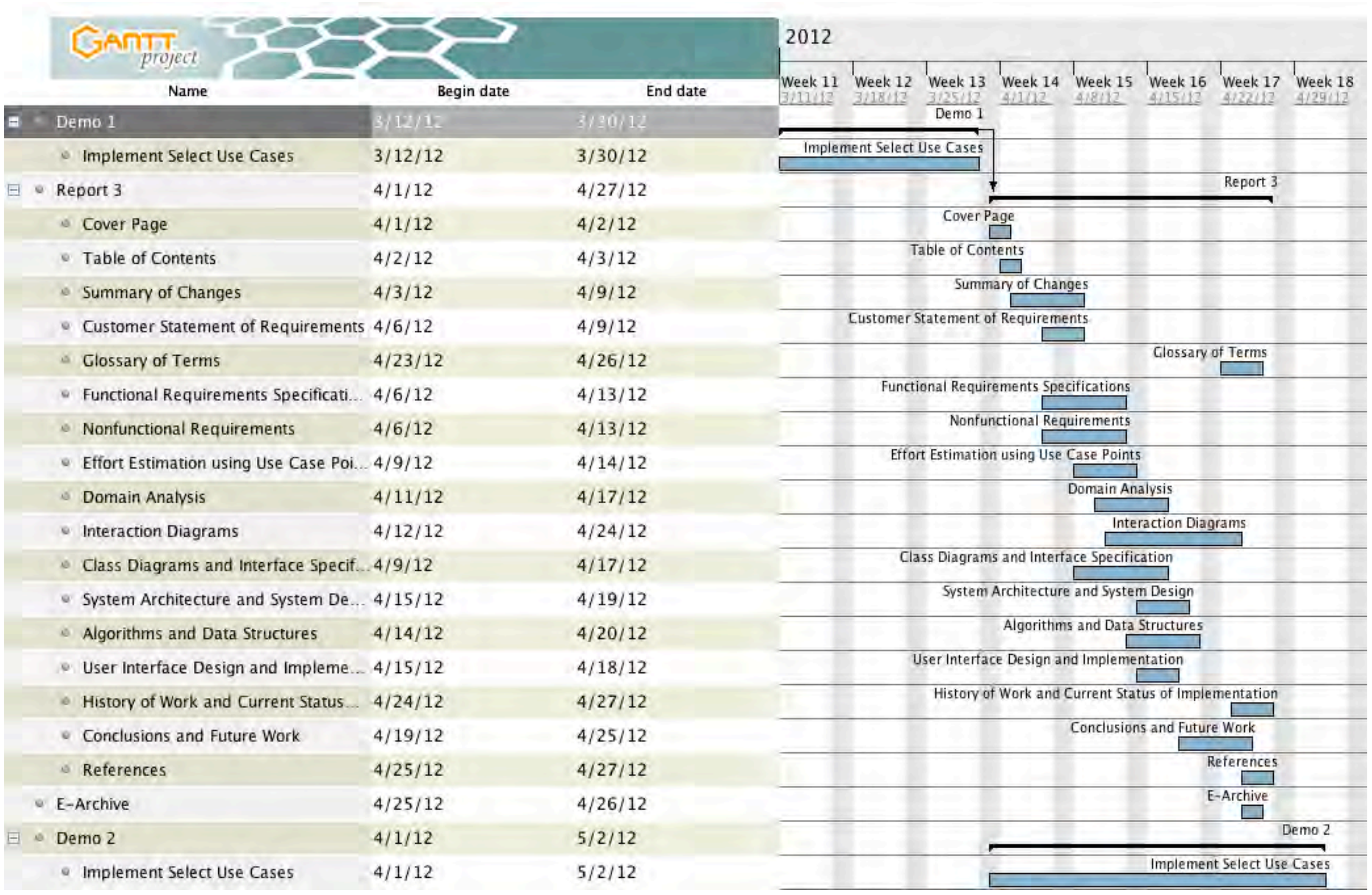
a. Merging the Contributions from Individual Team Members

Merging everyone's contributions is not as easy as it looks. This is mainly due to the fact that everyone used a different file format to do their portion of the project. Anything from Microsoft Word, to Adobe Acrobat, to even Notepad. In order to merge all of these contributions into one document, Apple Pages was used for the project. From there, contributions were copy and pasted to the new document in Pages, and correct font type as size was used to make sure everything stayed in a consistent manner. This is the only issue that was encountered and was very easily tackled. The only other "issue" was trying to fit the pictures on each page. So, each large picture constituted an entire page. However, I believe this is rightfully just.

b. Project Coordination and Progress Report

Currently the database has been implemented and will work when it is populated with simulated data. Once the website is created, the database will be linked to it. When this occurs, actions committed on the site will be reflected into the system and everything necessary for the first demo will in effect be working. To make it to the second demo, The PHP server will be set to read incoming data from devices. By creating a serial reader in C, the server will be able to read changed data as well as send data to the devices.

C. Plan of Work



Name	Begin date	End date
Demo 1	3/12/12	3/30/12
Implement Select Use Cases	3/12/12	3/30/12
Report 3	4/1/12	4/27/12
Cover Page	4/1/12	4/2/12
Table of Contents	4/2/12	4/3/12
Summary of Changes	4/3/12	4/9/12
Customer Statement of Requirements	4/6/12	4/9/12
Glossary of Terms	4/23/12	4/26/12
Functional Requirements Specificati...	4/6/12	4/13/12
Nonfunctional Requirements	4/6/12	4/13/12
Effort Estimation using Use Case Poi...	4/9/12	4/14/12
Domain Analysis	4/11/12	4/17/12
Interaction Diagrams	4/12/12	4/24/12
Class Diagrams and Interface Specif...	4/9/12	4/17/12
System Architecture and System De...	4/15/12	4/19/12
Algorithms and Data Structures	4/14/12	4/20/12
User Interface Design and Impleme...	4/15/12	4/18/12
History of Work and Current Status ...	4/24/12	4/27/12
Conclusions and Future Work	4/19/12	4/25/12
References	4/25/12	4/27/12
E-Archive	4/25/12	4/26/12
Demo 2	4/1/12	5/2/12
Implement Select Use Cases	4/1/12	5/2/12

d. Considerations for Grading

In order to maintain focus of the overall project, it has been determined to focus on nine core use cases. These are the use cases that the most time was spent divulging into:

1. Use Case 4: Configure User Device Permissions
2. Use Case 5: Configure Device Rule Sets
3. Use Case 12: Add Device
4. Use Case 3 Remove Device
5. Use Case 14: Send Device Signal
6. Use Case 15 Receive Device Signal
7. Use Case 18: Send System Interface Module Signal
8. Use Case 19: Receive System Interface Module Signal
9. Use Case 20 Evaluate Data Against Rule Sets

By keeping focus on these use cases, the core functionality of the system will be able to be created. The other use cases are not as important to the system as a whole and can be added easily at a later date for additional vanity to the final product.

8. References

[1] Example as stated on the report format page -
http://en.wikipedia.org/wiki/Software_architecture#Examples_of_architectural_styles_and_patterns

[2] - United States Government Time - http://time.gov/

[3] Android Operating System Documents - http://www.android.com/about/

[4] FTDI communication for embedded devices - http://www.ftdichip.com/