Raspberry Pi Smart Mirror Project Architecture Design

Scott English
Thomas Sanchez
Ryan Faulkenberry
Teegan Duong

TABLE OF CONTENTS

Preface

The expected readership of this document shall be the development team, as well as the product owner. This document shall be designed to describe the systems architecture for use during product development. This is the first version of this document. This version shall lay out the overall architecture and is intended to map out a general blueprint for use in system implementation. Here is a presentation of our systems architecture: https://docs.google.com/presentation/d/1p9kI18FUXuMyFByUJzedtSa9qlc7tiSYbb5A6P-stsI/edit

1.Glossary

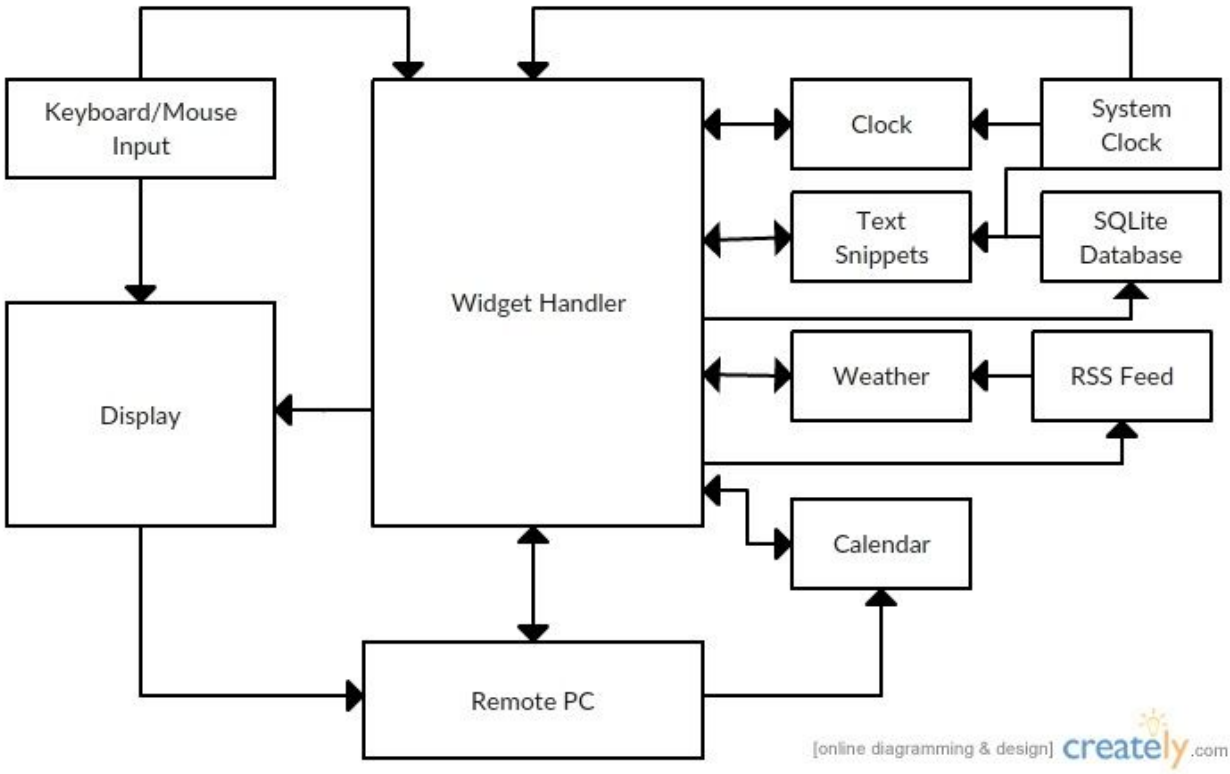
- **Raspberry Pi**: is a tiny affordable computer. (https://www.raspberrypi.org)
- **DIY**: Do it yourself, common term when describing projects that involve doing all of the work yourself
- **One-way mirror**: A mirror that is partially reflective and partially transparent. Allows the user to see through while still seeing their reflection as if they were staring into a mirror
- **Smart Mirror Project**: a DIY project where you take a one-way mirror and put a monitor behind it and display images through the mirror as you look at yourself in the mirror.
- **SDL2**: Simple DirectMedia Library, we are using to enable our GUI and keyboard/mouse input
- **SQLite3**: Structured Query Language, we are using the SQLite3 version of the SQL software to implement a self-contained, serverless, zero-configuration,transactional database that is local to the Raspberry Pi. (https://www.sqlite.org/about.html)
- **Raspbian**: The operating system most commonly used on Raspberry Pi's
- **Widget**: an application, or a component of an interface, that enables a user to perform a function or access a service.
- **Step**: A single iteration of the program loop.
- **State**: A configuration of variables at a certain time in a program's execution.
- **GUI**: Graphical User Interface that we are implementing using SDL.
- **RAM**: Random Access Memory that we will be accessing when communicating with different widgets.
- **I/O:** An input or output device.
- **RSS feed:** a web channel that contains frequently updated data or information.

2. Overall Architecture

MVC Architecture:

The architectural design pattern we are basing our system off of is roughly equivalent to a **model-view-controller** architectural pattern.

- The **controller** for this system is the Widget Handler, which shall act as a buffer between the widgets and system data, the input from the user, and the output to the view.
- The **model** within this system shall represent the widget objects and all the supplemental system data the widgets require as input (SQL database, system clock, RSS feed, etc).
- The **view** for this system shall represent the actual display screen that the program generates. Each widget object within the system represents a visible and modifiable object to be drawn to the screen.



Outputs from each module, from left to right, top to bottom:

Keyboard/Mouse Input interactions:

- Display: Mouse interactions shall be telegraphed on the display.
- Widget Handler: The widget handler shall take keyboard and mouse input for use in modifying the widgets.

Display interactions:

- Remote PC: The display shall be echoed on the remote PC.

RSS Feed interactions:

- Weather: The RSS Feed shall, when prompted, provide the current local weather to the weather widget.

Widget Handler interactions:

- Display: The widget handler shall send the information about the current state of every widget in the system each step.
- Remote PC: Every change made to the widget shall be updated in the state of the remote PC.
- Calendar: The widget handler shall modify the position and state of the calendar based on the input from the user.
- Clock: The widget handler shall modify the position and state of the clock based on the input from the user.
- Text Snippets: The widget handler shall modify the position and state of the text snippets based on the input from the user.
- Weather: The widget handler shall modify the position and state of the weather based on the input from the user.
- RSS Feed: The widget handler shall change the user location for the RSS Feed based on user input.
- SQLite Database: The widget handler shall change the stored text snippets within the SQLite database based on user input.

Remote PC interactions:

- Widget Handler: The remote PC should act as a secondary source of keyboard and mouse input for the widget handler.
- Calendar: The remote PC should provide the data to populate the calendar widget.

Clock interactions:

- Widget Handler: The clock shall, when prompted, send data about the current time to the widget handler.

Text Snippets interactions:

- Widget Handler: The text snippets shall, when prompted, send data about the current quote to the widget handler.

Weather interactions:

- Widget Handler: The weather shall, when prompted, send data about the current local weather to the widget handler.

Calendar interactions:

- Widget Handler: The weather should, when prompted, send data about the current user appointments, and the current date to the widget handler.

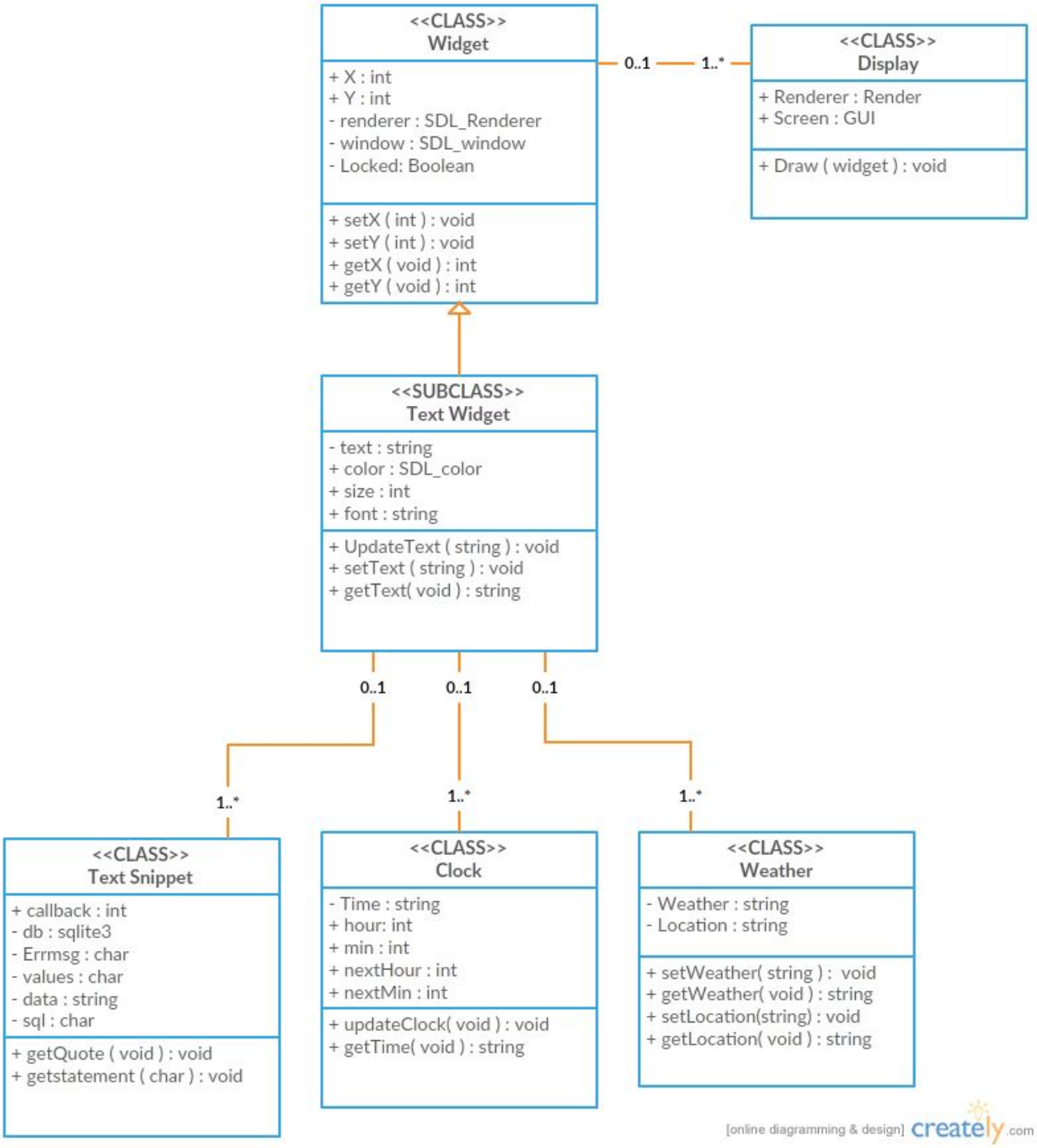
System Clock interactions:

- Widget Handler: The System Clock shall continuously send information to the widget handler for use in any timers used in widget optimization.
- Clock: The System Clock shall, when asked, update the current time on the clock widget.
- Text Snippets: The System Clock update the text snippet timer, so that after a set period the text is changed to a new randomized text snippet.

SQLite Database interactions:

- Text Snippets: The SQLite Database shall, when prompted, provide a random quote.

3. Class Architecture and Hierarchy

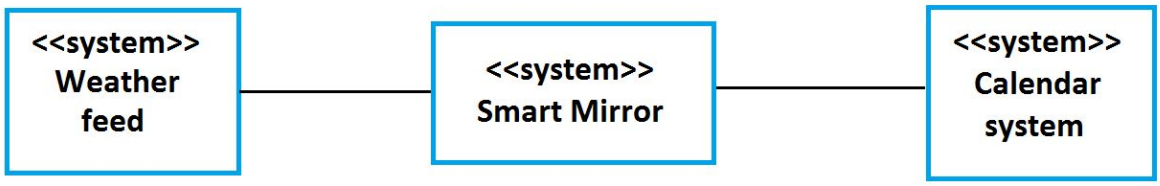


- For our design we have a Widget class, which shall extend to a sub-class TextWidget and will gather strings from the three other classes within our project: Clock, Text Snippet and Weather.This shall cause all interactions between those three classes to go through the TextWidget sub-class since they do not interact with each other directly. The TextWidget and Widget class shall pass data to the Display class in order for the objects to be drawn on the screen. The Widget class shall be directly responsible for handling the location and all movement, locking/unlocking, as well as resizing the display class.The TextWidget shall be responsible for managing and updating strings which are displayed to the screen by calling the draw method from its' associated display class. Upon receiving input from the I/O device the TextWidget class shall make changes to the inherited variables for location (x,y), lock status, as well as its own local variables for font, color, size, and text.

- The three classes Clock, Text Snippet, and Weather shall provide data string for the TextWidget class to use. This data will shall then be used in the creation of a new TextWidget object and then passed to the display to be drawn on the render. The variables that will contain each object's individual location shall also be within the object as inherited variables from its superclass Widget. The TextWidget class shall then handle manipulating the variables to change location, which will be passed to the display to be drawn in the new location.The TextWidget class will maintain the boolean value for whether or not the TextWidget object is locked or unlocked, this will determine whether or not to invoke the move method when input is received from the keyboard. The TextWidget shall also store variables which will contain the data to be to displayed to the screen, this variable will be broken up into parts containing a string for font, size and colors which will be set at the creation of the object and will be unchangeable.

- Drawing shall be handled by the display class which will gather new locations every-time the inherited move method is called from the widget class. This class shall also be resized within the widget. If the update text method is called the Display class will gather the updated TextWidget and redraw it to the screen, reflecting any changes made to the TextWidget's text data..

4. System Boundaries

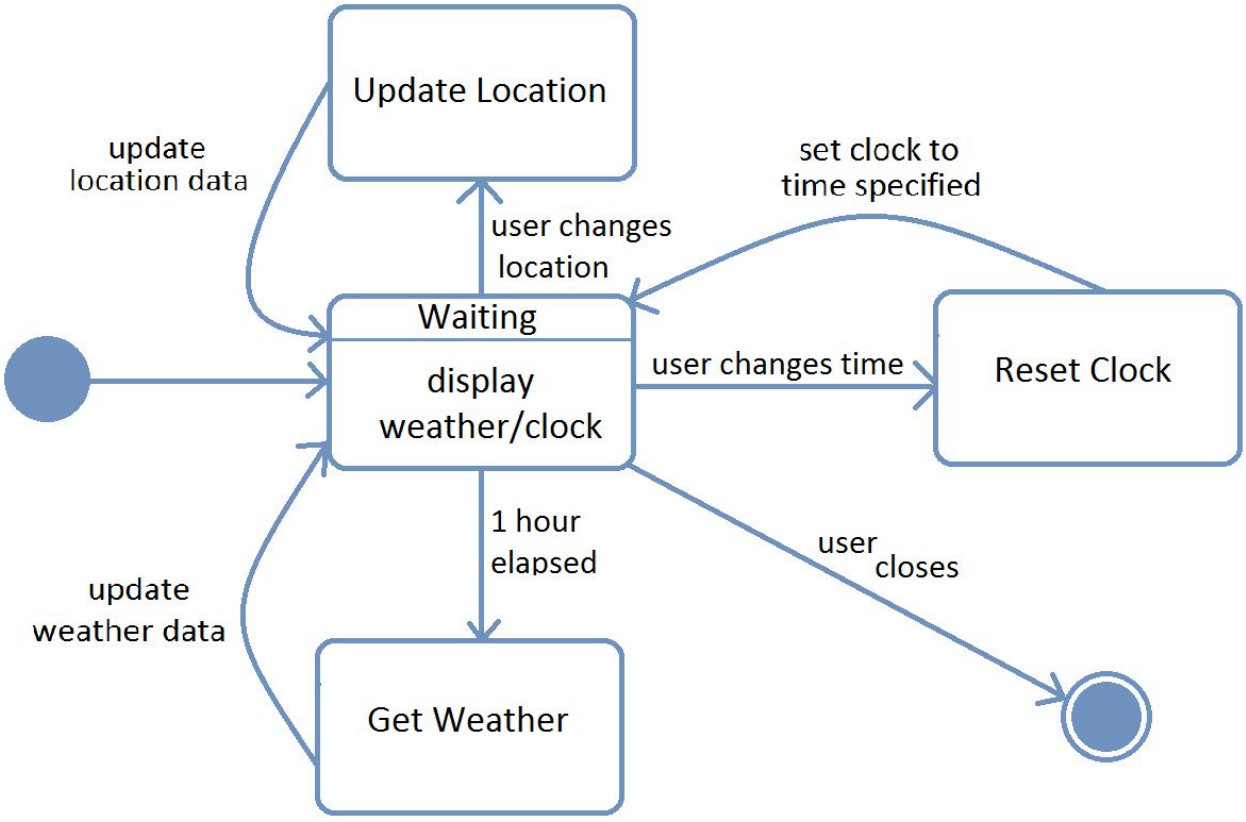


The system boundaries of this architectural design shall be reliant on a functional RSS feed, a SQLite database, and should have potential support for a outside calendar.

- **Weather feed**
    - We consider the weather feed to be an outside boundary because we can not access local weather information from within our system because our system does not generate weather information. We must reach outside of our system and access the information from a database that does generate this information.
- **Database**
    - The database is within the system boundaries because our system generates the database within itself and can access the information stored in it without any outside help. All information concerning the database is contained within the system.
- **Calendar System**
    - This calendar feature, if implemented, will be outside the boundaries because the information it displays to the screen will be accessed from outside of the system due to the nature of the type of information that is needed. The info will be completely dependent upon the system manager and how he decides to feed the information.

5. Event Handling Architecture

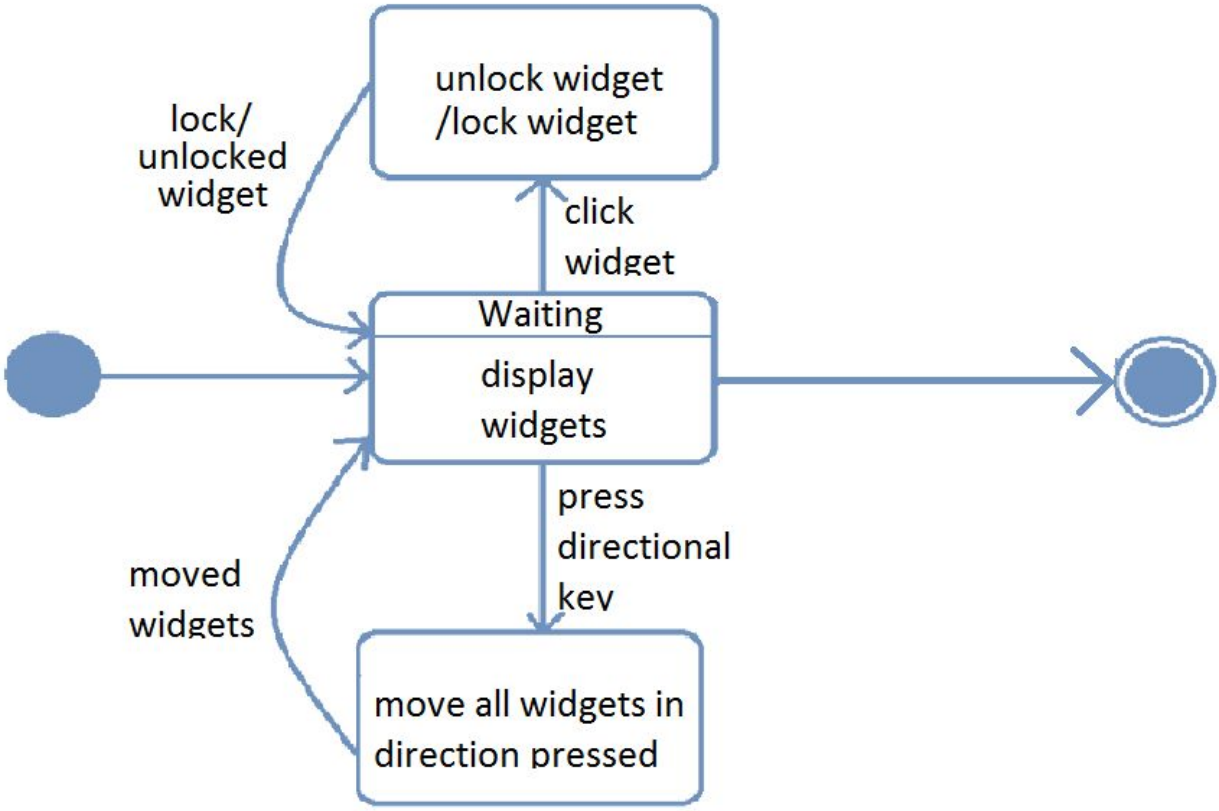
Back-end State Diagram:



The event handling in the back-end of this system shall be mainly controlled through input and from a keyboard/mouse through the system admin. The primary state the system shall boot into will be a waiting state, which shall wait for one of several simple events to occur in order to change the state of the machine. The changes to the state of the back-end of the machine fall into three categories:

- A system admin updates the location, changing the state of the RSS feed to pull weather data from.
- A system admin changes the local time-zone or adjusts for daylight savings time.
- The system detects either through system admin input or after a set duration that it shall check and return the weather data, in which case it provides the weather data to the weather widget to be updated.

Front-end State Diagram:



The state of the front-end of the software (the part of the software that the user will interact with) shall be encapsulated in a waiting state with two branching sub-states that shall loop back to the waiting state. These branching states shall occur when one of two events occur:

- The system admin left-clicks on a widget on the screen, toggling the lock on the widget clicked.
- The user presses one or more directional keys, moving every unlocked widget a set number of pixels in the direction pressed.