Design Document for ALS Home Automation System

Who's Doing What

- UI and Messaging
  - Jordan (jrhoeber@buffalo.edu)
  - Vincent (vcheng3@buffalo.edu)
  - Dominique (dnhickso@buffalo.edu)
- Server and Client
  - George (vanhoose@buffalo.edu)
  - Bin (bli5@buffalo.edu)
- TV Control
  - Jordan (jrhoeber@buffalo.edu)
  - Robert (rmslick@buffalo.edu)
- Fan and Light Control
  - George (vanhoose@buffalo.edu)
  - Bin (bli5@buffalo.edu)

Design

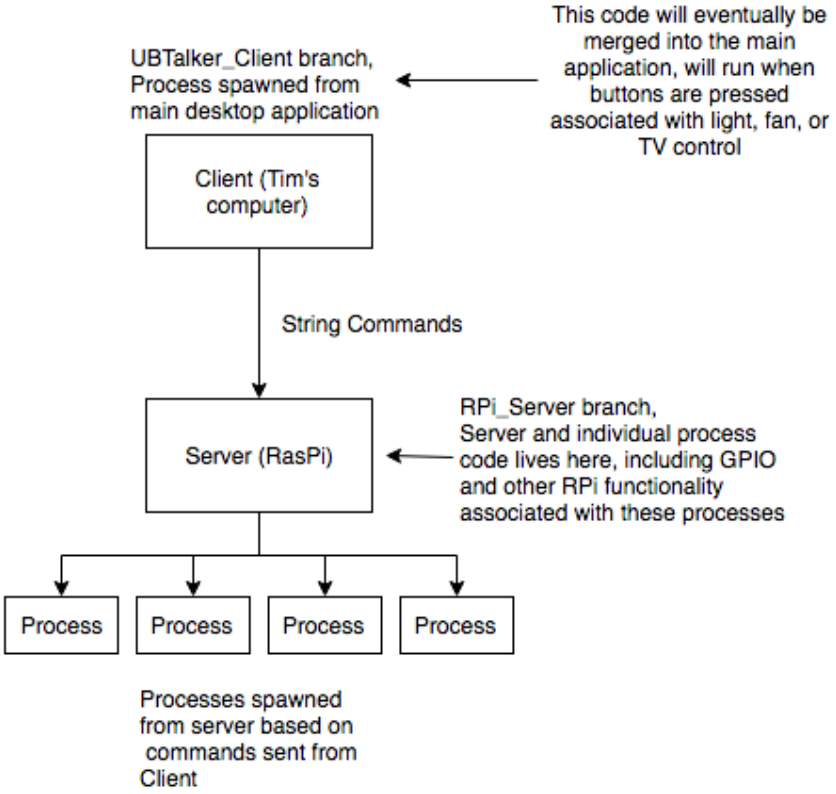
- **UI Changes to previous project for new functionality**

- **Messaging**
  - All messaging will be handled through email
    - Text messages from phone numbers can be sent and received through email.
    - This will consolidate the messaging process and make the code cleaner and more non-repetitive
  - Messages sent using C#'s built in StmpClient class
  - Contact list
    - App pulls data from Google Drive using Rest API.
    - Stored on Google Drive in file called 'contacts.csv'
      - Must be CSV file in format:
        - <Contact Name>, <Number or Email>
      - Phone numbers must be 10 digit
      - Phone numbers must have carrier domain appended

Carrier	Domain
AT&T	@txt.att.net (SMS) @mms.att.net (MMS)
T-Mobile	@tmomail.net
Verizon	@vtext.com (SMS) @vzwpix.com (MMS)
Sprint	@messaging.sprintpcs.com (SMS) @pm.sprint.com (MMS)
US Cellular	@email.uscc.net (SMS) @mms.uscc.net (MMS)
Boost Mobile	@myboostmobile.com
Virgin USA	@vmobl.com

      - Example: for Verizon number "7165551234@vtext.com"

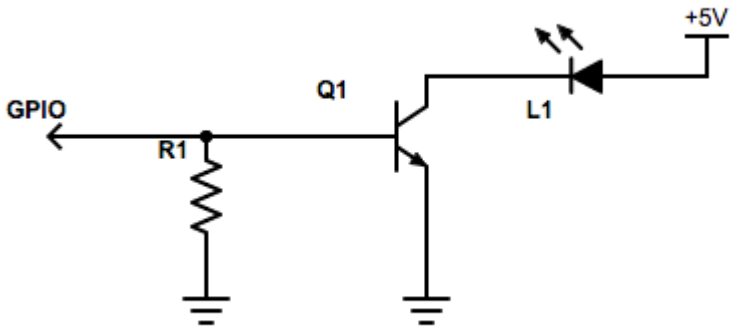
- **Raspberry Pi Server and Tim's Laptop Client**
    - Abstract breakdown of code



- Client Code
    - Client code will be called from the main application on Tim's computer
    - Main application will send strings to client based on buttons what have been pressed
    - Strings will be send to Raspberry Pi server for processing
        - [Insert explanation for client sending commands]
- Server Code
    - Server will be hosted on Raspberry Pi
    - Server will listen for commands to come in from the Client
        - Server runs in background on startup
        - Server waits on information from a specific port
        - When such information is received, Server translates this information into an array of arguments (of type `char* argv []`)
        - This array of arguments is passed to a `fork-execv` function which runs concurrent to the parent Server
        - Server logs all access, including errors and the actual information thus received.
        - The Server program is being piggybacked by TV Control as well, since there is much overlap in the requirements of these efforts.

        - The Server program has no need to send information back to the Client, but this functionality can be added with ease if so desired
    - Upon successful execution of a Server-Client interaction:
        - Client will have transmitted data to Server
        - Server will have logged receipt of this transmission
        - Server will have interpreted this data and spawned a child process which runs concurrent with the Server.
        - Server will have logged the creation of this child process.
        - Server will have waited for child process to exit.
        - Server will have returned to a listening state, which is also logged.
    - Upon error
        - Errors are logged in a separate file
        - Each entry in this file will be timestamped
        - If the error is non-fatal, the Server returns to a listening state, which is logged
        - If the error is fatal and the Raspberry Pi crashes and cannot reboot, it will not interfere with any functionality on Tim's tablet provided the Client program is wrapped in adequate exception handling.
    - Updates to the Server can be accomplished remotely, so as to minimize disruptions to Tim's quality of life and expedite reconciliation of any errors.
    - Individual process expanded upon in *TV Control* and *Fan and Light Control* sections

- **TV Control using Raspberry Pi and Infrared Emitter**
    - Parts Needed
        - Raspberry Pi 3
        - IR Emitter and Receiver
        - Circuit with Transistor and resistor for signal amplification and current/voltage regulation



    -
    - Raspberry Pi is programmed with software called LIRC (built into Raspbian repositories) used to generate Infrared signals to be transmitted to the TV

        - LIRC needs to be programmed using an Infrared receiver and the remote used to control the TV
            - To program it, use LIRC's built in **'irrecord'** process, that allows you to program the GPIO to listen to the IR receiver using LIRC's config files. Then send signals from the remote to the IR receiver to map signals to buttons
        - Then configure another GPIO pin to the IR Transmitter using LIRC's config files
        - Once all the configuration was ready, the command line call **'irsend'** can be called from the TV control C code
            - Example: To increase volume, call "`irsend SEND_ONCE samsung KEY_VOLUMEUP`" using the C system() function
        - The code can then be organized to take string commands from the Raspberry Pi Server, send them to the TV process code, then called the correct **'irsend'** function based on the string
    - Once the Raspberry Pi is configured to send IR signals to the TV, a system can be designed to make sure the IR Emitter was in the line-of-sight
        - TBD, based on layout of room

- **Fan and Lamp Control**
    - Because of the layout of Tim's room, GPIO can be used to control relays which open or close the AC circuitry which powers the Lamp and Fan.
        - If, in the future, more devices are desired to be controlled through GPIO than the pins will allow, and if such control is of a binary nature (i.e.: "on" or "off") a multiplexer may be added to accommodate the extra demand.
        - Pins which have serial data capabilities will not be used for simple binary GPIO.
    - All AC circuitry will go through surge protection.
    - Wireless control of devices can be implemented in code and migrated remotely to the Raspberry Pi, but hardware to receive commands from the Raspberry Pi will need to be ordered and installed as required.
        - Wireless control of devices will allow for a dynamic number of devices
        - Will minimize number of cords and wires running across the room
    - A casing will be fabricated which will protect this circuit and its components, both from physical damage, but also from overheating, and will minimize the effect of loose wires.
        - This casing will include separate compartments for the Raspberry Pi controller and the relay controls.
            - The relay control compartment will be subdivided to prevent any potential loose wires from contacting one another.
            - The design of this relay control compartment will also prevent wires carrying AC electricity from contacting wires carrying DC electricity.

        - Safety will be a top priority.
- This casing will also include a cooling fan, since the Raspberry Pi will be running in perpetuity, and a small air filter to prevent dust and other particulates from breaching the casing and causing problems.
- This casing can be painted as desired, but is highly recommended in order to protect the material of which the casing is made