One Device to Rule Them All:  
Controlling Household Devices with a Mobile Phone

William Shato

Introduction

This project was undertaken as part of a seminar course in Mobile Computing. While searching for a semester project, my primary concern was to do something unique, and to use the mobile phone in a way that people would not normally think of using it. To that end, this project investigates the potential for using the mobile phone to remotely control other devices. Throughout this paper we will see the mobile device used in a manner common within the field of Mobile Computing, that is as an interface to a system in which it is connected through as gateway, and we will explore how this familiar paradigm can be used to facilitate very unfamiliar uses for the device. We will also consider the various connection protocols that can be used to implement the system, and discuss the relative advantages and disadvantages of each.

Motivation

We live in a world filled with electronic devices. These devices commonly include televisions, DVD players, stereos, MP3 players, etc. While these devices enrich our lives by providing us with entertainment, education, and various utilities, they bring with them the frustration of having to learn numerous different interfaces and having to adjust to a different interface each time we switch devices. This is nowhere more apparent than in the average person's home entertainment system. Such a system often requires a different remote control for each separate device, and these remotes tend to occupy a great deal of space and create an unsightly pile in a user's living space. Furthermore, each remote is usually filled with numerous buttons that control functions that will never be used, and naming conventions regarding a particular button's function are practically nonexistent.

To some extent, the mobile phone is just another device in the plethora of gadgets, yet it is unique in several ways. First of all, the mobile phone is nearly ubiquitous. It is, more than any other device, extremely uncommon to meet someone who does not own a mobile phone. This is especially true if you consider the consumer electronics cultures of other countries, where it is not uncommon for significantly more people to own a mobile phone than a desktop or laptop computer. Secondly, the mobile phone is somewhat omnipresent, in that the owners of these devices tend to have them on hand at all times. Granted, the mobile nature of the devices facilitate this to a large degree, but we still find phones to be constantly present more than other mobile devices such as MP3 players. Finally, and perhaps most importantly, the mobile phone can communicate with other devices and is programmable. While most other devices serve a particular purpose, the mobile phone is in its essence just a small computer. Thus, within the limits imposed by the phones hardware, it can be programmed to do nearly anything any other computer can do.

Taking all this into account, it was hypothesized that the mobile phone, being owned by nearly everyone, being present with the owner nearly all of the time, and being programmable to do nearly anything, could be used as a central interface to remotely control the other devices. The following is an examination of this hypothesis and a summarization the results of this inquiry.
Implementation

A first observation upon setting out to design an implementation was obvious, but never the less important. That is, most every consumer device has a clearly defined user interface, and can only be interacted with through that interface. For example, a television that can be controlled either by the buttons on the television itself or by an infrared remote control can only be controlled in those ways. Such a device can not be controlled directly over the Internet because it simply lacks the means to communicate in such a fashion. There is, however, a way to get around this inconvenience, and that is through a gateway. It was decided that the project would be implemented by using computer peripherals that could control various devices, and then controlling the computer that manages the peripherals using the mobile phone. Given the multitude of potential device interfaces, it was realized that this project needed to be limited to a manageable subset of available devices. After a period of consideration, it was decided that the project would deal with infrared and X10 devices. The primary reason for this decision was a pragmatic one; there were commercially available and relatively cheap computer peripheral capable of controlling these devices.

The first feature of the project to be implemented was the ability to control X10 devices. X10 is an open industry standard protocol that allows electronic devices to communicate over power line wiring. It is particularly popular among devices used for home automation. In its typical use, household devices are plugged into X10 control modules such as the one pictured in figure 1 below.

![Figure 1: X10 Control Module](http://en.wikipedia.org/wiki/File:X10_3.jpg)

Messages are then sent between the modules that turn them on or off, effectively acting as a switch for the outlet. Some more sophisticated devices also permit dimming. The devices are addressed using a “house code” along with a “unit code”. There are 16 possible settings for each code, allowing for 256 different devices in one house.

The device used to control the X10 modules in this project was the X10 CM17A, more commonly known as the FireCracker. The device plugs into a serial port on a computer and sends commands wirelessly to an X10 module with a built-in receiver. From there the commands are forwarded to their desired recipient. The FireCracker was controlled using a software library written in the Python programming language by Jimmy Retzlaff. The library functions were accessible through the command-line or through the a Python module import.
The next feature of the program to be implemented was the ability to control infrared devices. Infrared is a communications protocol that relies on encoding messages in pulses of electromagnetic radiation. It requires line-of-sight to successfully transmit and is a popular protocol used in remote controls. Generally, infrared communication requires a transmitter to send a signal and a receiver to accept the signal. The situation with this project was slightly more complicated. Because the goal of the project was to be able to control any infrared device, there needed to be a way to discover the infrared codes used by a particular device. To do so requires having a receiver/transmitter which can accept an infrared signal and then store it for future use. This process of saving a particular remote's infrared codes to be retransmitted in known as “training” the device.

The device used to accomplish this, seen in figure 2 below, is called the Tira-2.1. It is an infrared receiver/transmitter which connects to a personal computer via a USB port. The device is usually controlled by a piece of software known as Girder, but for this project was controlled using a third-party, open-source Python library from Lincor Solutions. The library provides command-line utilities for training the device and retransmitting saved infrared codes. The codes themselves are stored in configuration files for future reuse. One potential limiting factor concerning this library is that it is only designed to work on the Linux operating system.

The implementation of the project was designed using a client/server architecture. The peripherals used to control devices were hooked up a central computer which ran a server. The server communicated using TCP sockets and was written in the Python programming language. The mobile phone, having connected to the server, would send commands which would be parsed by the server and acted on appropriately. Having both of the libraries used to control the peripheral devices written in Python was convenient, as it was quite simple at that point to write a server that important both libraries and called their functions as necessary.

The client program, which ran on the mobile phone, was written using J2ME. As stated earlier, the client communicated with the server using TCP sockets. The client program presented the user with an interface, pictured below in figure 3, and sent commands to the server based upon the users selection.
Figure 3: Client Interface

The client program interface is built dynamically upon connection to the server. The communication protocol is as follows:

1. The client connects to the server
2. The server sends the client the number of commands it accepts
3. For each command the server accepts it sends:
   1. The name of the command to display on the screen to the user
   2. The message to send to the server to invoke that command

By building the interface dynamically, it is possible to use the same client program for any combination of devices and commands desired by the user. This layer of abstraction is particularly helpful since it is much easier to alter and restart the server than to alter and reinstall the client. Also, it prevents the situation where the user must have multiple remote control application on his/her phone to handle different situations.

Alternate Implementation

A second version of the system was created which used SMS to communicate rather than TCP. To accomplish this, an SMS to TCP gateway was written. In order to use this system, the following steps are necessary:

- A central gateway server must be started on machine with a publicly accessible IP address
- A slightly modified version of the original server must connect to the gateway server
- A mobile gateway running on a mobile phone must also connect to the gateway server

Once these steps are accomplished, a the client can interact with the server by sending SMS messages to the mobile gateway, which upon receiving the messages passes the information along to the gateway server over a TCP connection. The gateway server then passes the message to the server that controls
the peripheral devices. Likewise, the server controlling the central devices can send a message to the client program by first sending the message to the gateway server via TCP, which then passes it along to the mobile gateway also using TCP, where it is then sent to the client program in an SMS message. The gateway server would have not been strictly necessary for this project, but was included to make the gateway system more general. Other than the way the information is passed from device to device, the system functions the same as in the previously described version. The one area that is the most different is the building of the interface, as in this implementation the entire list of commands is sent to the client program in one SMS message rather than one command at a time over the TCP connection. This requires some slightly more complicated string parsing on the client program.

There are advantages and disadvantages to both implementations, which are summarized in the following table:

**TCP**
- **Advantages**
  - Direct connection with the server
  - Quick and reliable connection
- **Disadvantages**
  - User must have a data plan

**SMS**
- **Advantages**
  - Most mobile phones support SMS
  - No connections for program to manage
- **Disadvantages**
  - Requires gateway (additional phone dedicated to gateway)
  - Slower

**Issues Faced**

The first issue faced while working on the project was with using third-party software. The program used to control the Tira-2.1 contained an error which prevented it from operating correctly. Quality is always a concern when using third-party software, and fortunately in this case the error was discovered and corrected. Another less important issue with the Tira-2.1 software was, as mentioned before, it is written to only run on Linux. This was not a major problem as the author is proficient with the Linux operating system, but was inconvenient as there were no publicly available computers which could run the application.

Another issue regarded querying for device state. It was originally intended for users to be able to use the client application to request information from the server about the current state of each device. With this feature the user would be able to tell at any time whether or not a device had been unintentionally left turned on or off and then alter the state accordingly. While it was relatively easy to keep track of the changes made to a device by the application itself, it was quickly realized that any implementation would be naïve in that there is no way to know whether the device was altered by a force outside of the system. For example, if a user were to use the application to turn a television on, and then the television was to be turned off manually by someone else, the system would have no record of this and would still believe the television was on. A subsequent request to turn the television off would then retransmit the power signal and turn the television back on again. Unfortunately, until
two-way communication between infrared and X10 devices is possible, this situation is unavoidable.

An issue which was outside the scope of this project but would have to be considered if future work was to be done is security. Obviously, if a user is able to connect to a server, then a different user could potentially connect to the same server. Should this application ever be released to the public, there would have to be some mechanism by which unwanted parties were prohibited from connecting to a user's server. This would most likely be implemented using some type of encrypted password scheme.

Future Work

The following is a brief list of possible work to be done on the project in the future:

- Support for more devices
  - Ovens, coffee makers, other appliances
  - Garage door openers
  - Security cameras / alarm systems
- Continued work on device state queries
- Support for additional communication protocols
  - UDP
  - Bluetooth
- Security implementation

Earlier Work

While researching the project a similar application was discovered. In 2005, Robert Stone wrote an X10 remote control for Symbian Series 60 using Python for S60. This project also used the Firecracker device to control the X10 modules and used the same Python library to interface with the FireCracker. Figure 4 is a screen shot of his client interface.

![Figure 4: Robert Stone's X10 Remote Interface](http://www.obviousobscurity.org/?page_id=2)

This application communicated with the server using Bluetooth rather than TCP, and the server was ran on a Windows computer. The author describes the work as a “crude prototype” and offers only a brief
description of the project and instructions on installing it. Unfortunately, the link to code on his site is no longer active and emails requesting more information were not returned.

Application to Mobile Computing

This project utilizes and highlights the unique characteristics and abilities of Mobile Computing in a number of ways including:

- The ability to access and interact with a service from any physical location
- The use of gateways to extend the functionality of limited devices
- Extensive networking of devices
- The possibility of using multiple communications protocols to transmit data

Most importantly, this project demonstrates the possibilities for using mobile phones in interesting and unintended ways. As mobile phone ownership increases and the capabilities of mobile phones and more traditional computers continues to converge, it is evident that these devices, like the Internet before them, will change the way people think about computing.
References


