Electronics & Computer Science
Faculty of Physical and Applied Sciences
University of Southampton

Miles Hampton-Armstrong

April 26, 2014

Resurrecting the Rabbit: A pervasive computing device 8 years on

First Examiner: Dr. Kirk Martinez

Second Examiner: Dr. Klaus-Peter Zauner

A Project Report
Submitted For The Award Of:
Masters in Computer Science

Abstract

The Nabaztag, a Wi-Fi enabled smart rabbit, was a once popular foray into the world of Pervasive Computing. Part toy, part personal assistant, it was designed to be a friendly electronic companion that could convey a variety of information to the user using lights, sound, and motion. Whilst it was a reasonably successful product, technical issues plagued the cloud architecture supporting it, until eventually its creator Violet filed for bankruptcy. This report details the process of resurrecting a single Nabaztag:tag device through a combination of replacement hardware and software. By using the emerging WebSocket protocol as a medium for publish-subscribe communication, and by developing a RESTful API for the device itself, the Nabaztag:tag is brought up to date as a modern, internet connected 'thing'.

Acknowledgements

I would like to thank Dr. Kirk Martinez, my project supervisor, for his help and guidance, many interesting discussions, and the loan of his Nabaztag.

I would also like to thank Tyler Ward and Philip Basford for their invaluable electronics advice and access to their lab whilst I created Nabaztag's new hardware, and my second examiner, Dr. Klaus-Peter Zauner for sharing his insights on the project.

Contents

List of Figures

List of Tables

Nomenclature

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BBB	BeagleBone Black
DC	Direct Current
DNS-SD	Domain Name System Service Discovery
DOM	Document Object Model
DWR	django-websocket-redis
GPS	Global Positioning System
HCI	Human Computer Interaction
HTTPS	HTTP Secure
HTTP	HyperText Transmission Protocol
I/O	Input/Output
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IM	Instant Messaging
IoT	‘Internet of Things’
IR	Infrared
ISR	Interrupt Service Routine
JSON	JavaScript Object Notation
LED	Light Emitting Diode
MAC	Media Access Control
MVC	Model-View-Controller
PCB	Printed Circuit Board
REST	Representational State Transfer
RFC	Request for Comments
RFID	Radio Frequency Identification
RGB	Red, Green & Blue
SMD	Surface Mount Device
TTS	Text-to-Speech
URL	Uniform Resource Locator
USB	Universal Serial Bus
VM	Virtual Machine
WoT	‘Web of Things’
WSGI	Web Server Gateway Interface
XHR	XMLHttpRequest
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1 The Nabaztag & Pervasive Computing

Ubiquitous or Pervasive Computing is a concept first introduced by Weiser in his seminal paper ‘The Computer for the 21st Century’ [75]. He describes a future where computers will cease to be seen as distinct devices, and will instead become part of the environment, fading into the background and becoming “so ubiquitous that no one will notice their presence” [75]. He argues that this will occur as a natural result of human psychology, stating that “whenever people learn something sufficiently well, they cease to be aware of it” [75].

In 2005, nearly 15 years later, the Nabaztag, shown in Figure 1, was first released. Developed by a French company, Violet [74], it was different from the concepts of inch-scale ‘tabs’, foot-scale ‘pads’ and yard-scale ‘boards’, the three categories of Ubiquitous Computing devices proposed by Weiser [75]. A Wi-Fi enabled smart device designed to look like a rabbit, it could convey information to the user through a combination of light, sound and movement. The intention was that, as a fun and friendly companion, the technology behind the Nabaztag would be irrelevant to its users and they would use it to complete day to day tasks, such as checking the weather, without considering it as a computer at all.

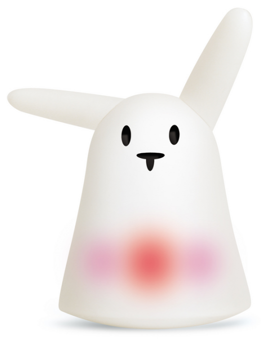


Figure 1: A Nabaztag rabbit (from [76])

During its lifetime, the Nabaztag has been through three iterations. The original Nabaztag was the simplest device, offering five Red, Green & Blue (RGB) Light Emitting Diodes (LEDs), motorised ears, a speaker with a volume control, and a Wi-Fi connection. Released a year later in 2006, the Nabaztag:tag [72] added a microphone and a Radio Frequency Identification (RFID) reader, as well as an improved Wi-Fi card. The final iteration, available in 2012, was renamed to ‘Karotz’ [73], and added a webcam, and the ability to run from battery or Universal Serial Bus (USB) power.

All versions of the Nabaztag relied on communication with servers operated by Violet, which stored the configuration information for each device, providing the only way for a Nabaztag owner to alter the behaviour of their device. In December 2006, when large numbers of purchased Nabaztag devices were activated simultaneously, Violet’s infrastructure was unable to cope with the demand [14], and there were service disruptions for both new and existing Nabaztag owners.

In October 2009, after a long period of technical troubles, Violet declared bankruptcy and was purchased by a Dutch software house, Mindscape [23]. In July 2011, Mindscape ended support for the Nabaztag [1], and renamed the product to 'Karotz', releasing the previously closed-source code to the public [77]. Efforts to understand the Nabaztag code have been hindered by poor documentation, much of which is in French.

Several open source projects have attempted to create new cloud architectures for the Nabaztag [54, 56], but they suffer from the requirement that they must interface with the original firmware running on the microcontroller at the heart of the device. Whilst compatibility with the client software on the Nabaztag is necessary to allow existing Nabaztag owners to return functionality to their devices without any complex modifications, it limits the potential to experiment with new and improved technologies which could be used to develop new use cases for the Nabaztag.

2 Project Goals

1. Examine the existing hardware of a Nabaztag:tag, to determine how it could be replaced with an open hardware platform, whilst maintaining as much original functionality as possible.
2. Develop a software infrastructure to restore core features of the device, such as control from a web application.
3. Implement a novel feature for the Nabaztag:tag that was not previously possible due to software constraints.

3 Background Reading

3.1 Literature Review

It is evident that the history of the Nabaztag was not particularly easy to follow. Many of the original websites pertaining to the product no longer exist and as a result, much of the information was sourced from third party news sites and internet archive services. Only a handful of papers have been written on the Nabaztag, and these discuss the device from a Human Computer Interaction (HCI) perspective.

In 'Can Your Pet Rabbit Read Your Email?' [42], Huang, Bardzell, and Terrell report on their experiences gained from a years worth of use of the Nabaztag:tag. They conclude that "Nabaztag seems to have too many functions for its own good" [42], suggesting that there is a problematic conflict between its intended use as an "ambient information display" and the strong "pet metaphor" conveyed by the design and marketing of the device [42]. They found that the undefined user experience did eventually settle, but leaned heavily towards use of the Nabaztag:tag as an information display, a task which they felt it performed intrusively, suffering from a lack of contextual awareness and often interrupting the user with unexpected auditory alerts [42].

In 'Ambient Conversations Using a Physical Avatar' [52], Lund, Coulton, and Edwards propose a mobile phone application allowing direct interaction with a Nabaztag:tag device through Violet's original Application Programming Interface (API). An initial review of their application was undertaken by a family, in which one Nabaztag:tag device was left in the family home, and one in the office of the Father. The study showed that the children in the family preferred using the the Text-to-Speech (TTS) functionality, where they could send a message to a Nabaztag:tag and it would read it out, over typical text based Instant Messaging (IM) applications to contact the Father. From this, they suggest that "The Nabaztag and similar ambient devices present a new method of disseminating information in an integrated, ubiquitous manner" [52], and discuss the sense of community around the device, suggesting that this kind of technology may enable "more accessible and dynamic social interactions" [52]. They conclude that 'widgets' similar to their application will, in the future, provide an important way of interacting with ubiquitous devices [52].

Today, looking at some emerging ubiquitous devices such as the LIFX Smart Bulb [15] and the Nest Thermostat [26], we can see that, although they differ from the Nabaztag due to their lack of personification or emotional connection to the user, similar areas of importance can be identified. Nest is a replacement thermostat for the home, fulfilling the need for 'contextual awareness' recognised by Huang, Bardzell, and Terrell [42] through its ability to monitor a user's temperature adjustments and create a schedule by learning their habits, and by its ability to determine if a home is empty based on movement, and reduce the temperature accordingly. The LIFX Smart Bulb is a device that really deserves the 'ubiquitous' moniker, there are few things more commonplace than lightbulbs. It demonstrates the importance of 'widgets', identified by Lund, Coulton, and Edwards [52], by providing a fun and engaging smartphone application enabling the user to choose from 16 million possible light colours, enable a 'sunrise' alarm clock feature, or set the LIFX bulbs to react to music.

In order to introduce some new terminology to this discussion, it is useful to consider the progress made in the area of Ubiquitous Computing since Weiser's initial paper. When he proposed the concept the Internet was in its infancy and although he mentions the need for "a network that ties them [the ubiquitous devices] all together" [75], there is no mention of that network being the Internet. Instead he speaks of networks that are "capable of supporting hundreds of devices in a single room" [75].

Twenty years later, the concept of Ubiquitous Computing in the age of the Internet has become known as the 'Internet of Things' (IoT), a term coined by Kevin Ashton. Explaining what he meant by the term, Ashton, whose main work at the time was with RFID technology, offered “Ideas and information are important, but things matter much more. [...] If we had computers that knew everything there was to know about things — using data they gathered without any help from us — we would be able to track and count everything, and greatly reduce waste, loss and cost.” [11]. Many other definitions have been given, with Butgereit, Coetzee, and Smith offering “The phenomenon of an increasing number of physical objects (things) having the ability to connect to the Internet” [17], and Tan and Wang suggesting that “things have identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environment, and user contexts” [66].

Under these definitions, the three ubiquitous devices considered previously — Nabaztag:tag, LIFX and Nest — could also been seen as IoT devices, as everyday objects that have been connected to the internet in order to offer ‘smart’ functionality and contextual awareness.

3.2 Original Use Cases

In researching the Nabaztag:tag, it is useful to look at some of the original use cases for the device, shown in Table 1. These provide a better understanding of the required functionality of the device than hardware specifications alone.

No.	Use Case	Details
1	Email & social media notifications	Different combinations of LED colours and ear movements could be performed when a monitored email account or social network was updated.
2	Weather information	By configuring the Nabaztag:tag with location information, it was possible for it to deliver weather updates at a glance using different combinations of LED colours.
3	RFID Nano:ztags & Zstamps	These were accessories for the Nabaztag:tag, Nano:ztags were mini Nabaztag rabbits, and Zstamps were stickers, both containing RFID chips. The user could configure actions to be performed when a particular RFID tag was registered by the device.
4	TTS functionality	Violet hosted a TTS service, allowing Nabaztag:tag devices to download and play MP3 files of TTS audio.
5	Pairing between Nabaztag:tags	Actions performed on one Nabaztag:tag, such as manually moving the ears, could be mirrored on a paired Nabaztag:tag, allowing simple communication between Nabaztag:tag owners.

Table 1: Original Nabaztag:tag use cases

4 Hardware

The first stage of the project was to remove the original hardware from the Nabaztag:tag, determine its characteristics, and design a replacement hardware system that would restore as much of the original functionality as possible.

4.1 Original Hardware Analysis

The Nabaztag:tag was the most popular iteration of the Nabaztag, and as such there are several detailed hardware investigations available online [69, 53]. Unfortunately, these mostly focus on the components making up the main Printed Circuit Board (PCB). A third investigation [55] was more useful, focusing mainly on the Input/Output (I/O) hardware. With the exception of the RGB LEDs, the I/O hardware is attached to the main PCB via leads terminating in female 2.54mm headers, so it is easy to remove the PCB whilst leaving other components intact. The following information is a combination of the author's own investigations, and information from [55].

Ears

- Each ear is driven independently, through a belt and gearbox system, using a Direct Current (DC) motor.
- An ear's position is determined using an Infrared (IR) encoder. A toothed cog is attached to the ear shaft, and the teeth pass between an IR LED and a phototransistor, alternately blocking and unblocking the IR beam as the ear turns.
- The cogs in the Nabaztag:tag ears have a tooth missing to enable the absolute position of the ears to be determined.

LEDs

- The 5 RGB LEDs are Surface Mount Device (SMD) components, soldered directly to the main PCB. Because of this, the LEDs will need to be replaced when replacing the main PCB.
- Black plastic cones act as light guides for the LEDs, so the light forms circles on the case of the Nabaztag.

Audio

- The volume is controlled by a user-adjustable potentiometer.
- There is a simple rear facing speaker.
- There is an 3.5mm audio output jack on the rear of the Nabaztag:tag.
- There is a front-facing microphone.

Head Button

- There is a momentary push switch on top of the Nabaztag:tag, between the ears.

RFID Reader

- The RFID antenna and supporting circuitry is mounted on a separate PCB, raised so it sits just beneath the front surface of the plastic shell of the Nabaztag:tag.

4.2 Requirements of Replacement Hardware

In order to determine a suitable replacement electrical architecture, the requirements of the existing I/O components were considered. It was decided to leave the RFID reader as an extension task to be performed if time permitted, and to limit the number of RGB LEDs to reduce the current requirements.

4.2.1 Electrical Requirements

After determining the hardware in use within the Nabaztag:tag, the components requiring power were monitored whilst in operation to determine their voltage and current requirements. The measurements taken can be seen in Table 2. Also in the table are the characteristics of the RGB LEDs [45] chosen to replace the SMD LEDs soldered to the main PCB.

Component	Quantity	Voltage (V)	Current (mA)
Ear Motor	2	3.2	15
IR LED	2	1.2	15
RGB LED	2	2.5(R), 4(G, B)	30 (R, G, B)

Table 2: Requirements of existing & replacement components

Although the DC motors have a low current draw, they should not be driven directly by a microcontroller. A motor driver circuit should be used to protect the controlling hardware from large current draws and from voltage spikes generated by the user of the Nabaztag:tag moving the ears manually.

4.2.2 Input/Output Requirements

A total of ten digital outputs, three interrupt inputs, and one analog input are required. The breakdown is given below:

Ears

- One digital output per ear to control starting and stopping of the motor.

- One digital output per ear to control IR LED.
- One interrupt input per ear to receive signals from the phototransistor as the ear turns.

LEDs

- Three digital outputs per LED to control RGB channels.

Head Button

- One interrupt input to monitor button presses.

Volume Control

- One analog input to read the value of the potentiometer.

4.3 Design of Replacement Hardware

4.3.1 Arduino Duemilanove

"Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software" [7]. There are a variety of different Arduino boards based on different AVR microcontrollers, each providing easy access to analog and digital I/O, as well as advanced features such as external interrupts. There is an Integrated Development Environment (IDE) provided [8], and the many tutorials and libraries make it simple to quickly prototype circuits and logic to control a wide variety of components.

The Arduino Duemilanove [5] runs at 5V, and features an ATMega328-P microcontroller [9], running at 16MHz . It provides fourteen digital I/O pins and six analog inputs. Two of the digital pins can be used to receive external interrupts, and a further two pins can be used for a 5V serial connection. Additionally, all pins can receive pin change interrupts, a less versatile form of interrupt which is sufficient for a button press event.

4.3.2 BeagleBone Black

"BeagleBone Black is a [...] community-supported development platform for developers and hobbyists" [31]. It is capable of operating from a 5V power supply, and can run Linux based operating systems from its onboard flash storage. Access to the Linux kernel provides networking functionality and USB support, so the outdated 802.11b/g Wi-Fi capabilities of the original PCB can be replaced using an 802.11n USB adapter. It also provides access to the Python language, which is well suited for communicating with the Arduino over a serial connection.

4.3.3 Using the Arduino Duemilanove

An initial architecture is proposed in Figure 2, using the Arduino Duemilanove to control the Nabaztag:tag hardware, and the BeagleBone Black (BBB) to enable connection to the internet and audio capabilities. Whilst attaching components directly to the Arduino is useful at the prototyping stage, the size constraints of the Nabaztag:tag make it infeasible to fit both the BBB and the full size Duemilanove board inside. A further complication arises from the Duemilanove board's use of 5V serial logic. As the BBB can only accept 3.3V serial signals [32], a logic level shifter would be required to transform the signal. Lastly, as controlling the Nabaztag:tag's hardware requires additional components, including male headers for the female I/O headers, a third board would be needed to hold these.

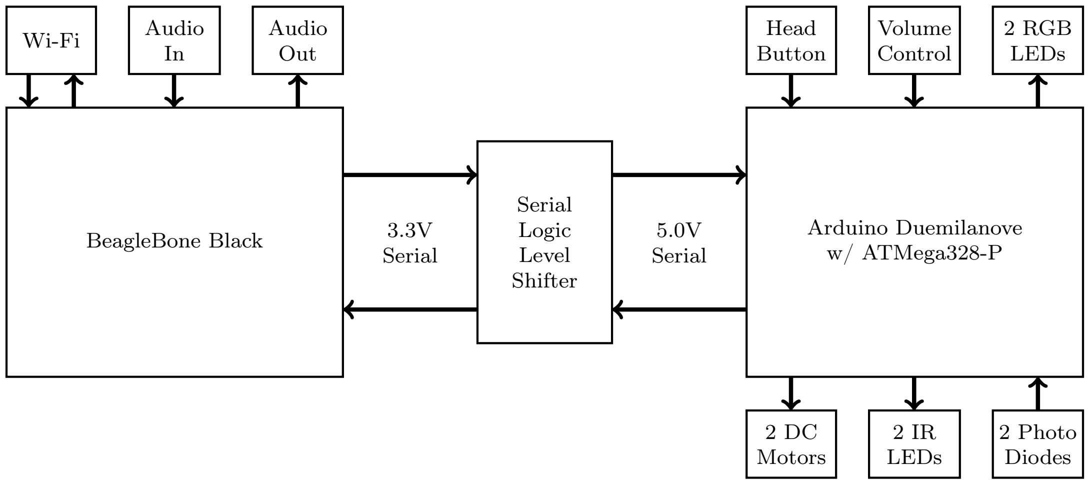


Figure 2: Initial high level overview of the replacement hardware for the Nabaztag:tag

4.3.4 Custom AVR Board

To overcome the issues with using the full size Duemilanove board, it was decided to create a custom AVR board using the ATMega328-P microcontroller from the Duemilanove. The ATMega328-P can operate at lower voltages, and at 3.3V can sustain an 8MHz operating frequency using its internal oscillator, which will be sufficient for this project. This also means that the serial logic level will be 3.3V, so no logic level shifter is required. The revised architecture can be seen in Figure 3.

The microcontroller is removable, so can be programmed whilst on the Duemilanove board, using the USB serial connection, before being removed and placed in the custom circuit for testing. Designing a custom board will enable all additional required components to be placed in a way that minimises the size of the board, meaning both the board and the BBB will fit in the Nabaztag:tag.

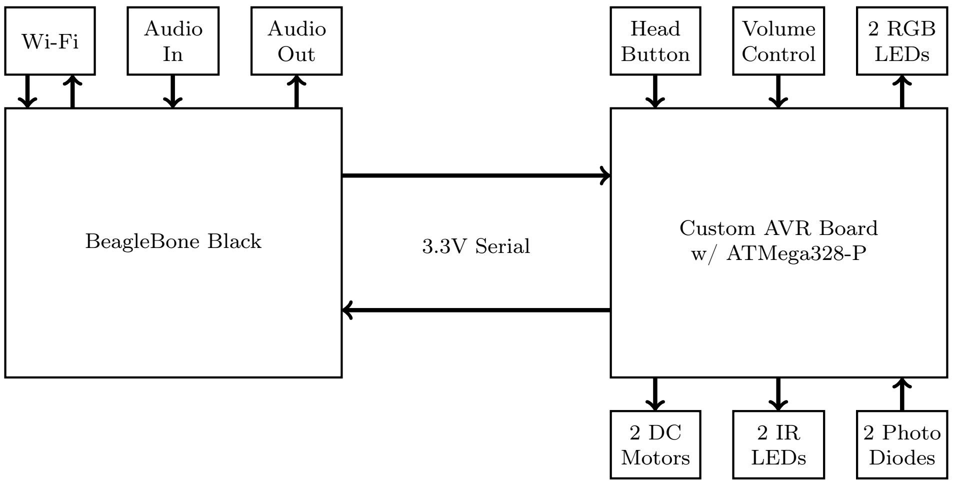


Figure 3: Revised high level overview of the replacement hardware for the Nabaztag:tag

From this outline design, a schematic was created for the custom AVR board using Cadsoft's Eagle PCB Software [18]. The original Arduino Duemilanove schematic [6] and the ATMega328-P pinout (Figure 16 in Appendix A) were used for guidance. The schematic (Figure 17 in Appendix B) integrates the additional circuitry required to control the ears, LEDs, head button and volume control. A list of parts used in the schematic is available in Table 6 in Appendix D.

4.4 Prototyping

Using the schematic as a guide, the various parts of the circuit were laid out on a breadboard (Figure 18 in Appendix C), and tested by connecting to the relevant female headers on the Nabaztag:tag hardware.

In order to test the prototype, code was written for the ATMega328-P in the Arduino IDE, and flashed to the chip through the Duemilanove. Each section of the code was first tested using the relevant hardware from the Nabaztag:tag connected directly to the Arduino Duemilanove, and then retested with the ATMega328-P in the prototype board, and the relevant Nabaztag:tag hardware connected to the board.

Figure 19 in Appendix E shows a code example for serial communication with the ATMega328-P. It uses the SerialCommand library [20] to parse commands sent over the serial connection and call the relevant method, which then extracts arguments and performs the action. In the example given, sending `EARPOS L` would return the position of the left ear over the serial connection.

Figure 20 in Appendix F shows a code example for an Interrupt Service Routine (ISR) to handle moving the left ear. In this example, `EARMOV` is called using a serial command such as `EARMOV L 10`, which will move the left ear to the tenth tooth on the encoder cog. When this command is issued, `moveLeftEar` is set as the ISR for interrupt input one, and the ear is set in motion. Each time a tooth of the encoder cog breaks the IR beam, an interrupt is triggered and `moveLeftEar` is called. It increments the variable storing the encoder tooth position, and stops the ear motor when the correct tooth, in this case 10, is reached. To enable a more logical control of ear position, the function translates a position of 0 to the actual upright position of the ear, which is at encoder tooth 2.

4.5 Final Circuit

With prototyping showing the design was correct, the circuit was transferred to perfboard for installation in the Nabaztag:tag. The completed board is shown in Figure 4.

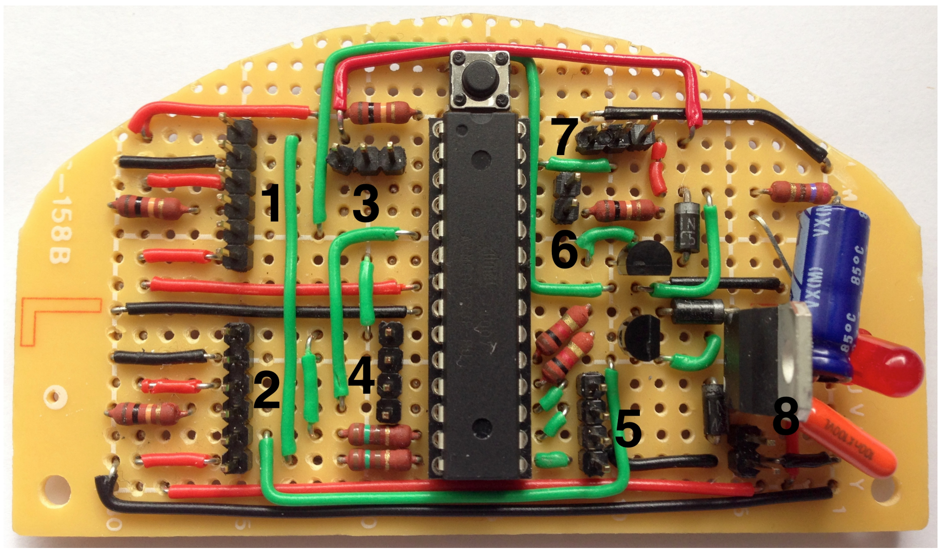


Figure 4: Completed custom AVR board. The headers are used as follows: 1) Right ear, 2) Left ear, 3) Serial connection to BBB, 4) Bottom LED, 5) Top LED, 6) Head button, 7) Volume control - unused, 8) DC input.

4.6 Assembly

The final task after completion of the custom AVR circuit was to connect it with the BBB and install them in the Nabaztag:tag shell. This also required finding a way to add audio functionality back to the Nabaztag:tag, and it was decided the simplest way to achieve this was to attach a USB soundcard to the BBB. This added an additional requirement of a small USB hub as the BBB has only a single USB port, in use by the Wi-Fi adapter. Two different soundcards and two different small USB hubs were ordered using the project budget to determine which would best fit in the Nabaztag:tag. Once these items were obtained, the assembly of the device was completed, as shown in Appendix G in Figure 21 and Figure 22. Although the circuit design included a header for the volume control, volume control functionality was not re-implemented.

5 Software

With the Nabaztag:tag re-assembled, attention moved to the software side of the project. A replacement software architecture was required to support both basic and more advanced functionality of the device.

5.1 Original Software Analysis

Alongside the Nabaztag:tag hardware investigations used for reference earlier in the report, several projects have attempted to understand the software architecture which supported the Nabaztag ecosystem [13, 47, 48, 49, 58]. Beausset and Soumoy captured the traffic passing between a Nabaztag:tag and Violet's servers, identifying the protocol used to carry the messages as the Extensible Messaging and Presence Protocol (XMPP) [13]. XMPP provides push, request-response, and publish-subscribe paradigms for message delivery, using Extensible Markup Language (XML) streams [62]. The technology was originally developed for Jabber, an open IM service, whose core protocols were later standardised as XMPP by the Internet Engineering Task Force (IETF) [61].

```xml
<iq from="net.violet.platform@xmpp.nabaztag.com/sources"
to="0019db9ed017@xmpp.nabaztag.com/boot" id="3" type="result">
  <query xmlns="violet:iq:sources">
    <packet xmlns="violet:packet" format="1.0" ttl="604800">
      fwQAAAx////+BAAFAA7/CAALAAABAP8=
    </packet>
  </query>
</iq>
```

Figure 5: A captured XMPP message, sent from Violet's servers to a specific Nabaztag device (from [13]). The `<iq />` stanza within the XMPP specification is a "request-response mechanism, similar in many ways to HTTP, that lets entities make requests of and receive responses from each other" [62]

The XMPP client software on the Nabaztag:tag ran entirely inside a purpose-made Virtual Machine (VM), whose bootcode was retrieved from Violet's servers with an initial HyperText Transmission Protocol (HTTP) request each time the Nabaztag:tag was turned on [58]. Once the VM had booted, an authenticated XMPP stream was established with Violet's servers, using the Media Access Control (MAC) address of the Nabaztag:tag device as an identifier, and instructions for the device were sent over the stream as `<iq />` or `<message />` stanzas [58].

It is no longer possible to access the original Violet servers and there is no information available as to the specific reasons for the failing of the original architecture. A close approximation to the original architecture involved using an unaltered

Nabaztag:tag with an open source server, Nabaztaglives [56], but the experience was a frustrating one. Messages sent from the server often took a long period of time to reach the device, and were sometimes lost.

It was decided that rather than attempting to recreate the XMPP architecture, other technologies should be explored in the hope of achieving improved performance.

5.2 Requirements

Based on the information from Section 1 and Section 5.1, a list of requirements for the software side of the project, shown in Table 3, was drawn up. The choices made to meet the requirements are addressed in Sections 5.2.1 to Section 5.2.5

No.	Requirement	Details
1	A new communication protocol to replace XMPP.	Due to the shortcomings of the previous infrastructure, where communication was performed over XMPP, it was decided to explore a different technology for communication.
2	A simple, consistent message format.	To help with development, and provide opportunities for expansion of functionality, a simple and human readable message format with low overhead should be used.
3	A web interface able to send control messages to the Nabaztag:tag.	For basic functionality, a simple web application should be created to control features of the Nabaztag:tag, e.g. control of the ears, LEDs and TTS functionality.
4	Push communication of messages from server to Nabaztag:tag.	A major issue with Violet's infrastructure was the slow delivery of messages from the server to the device, a replacement system should attempt to solve this issue.
5	Pairings should be possible between Nabaztag:tag devices.	To show more advanced functionality can be supported using the web application, pairings should be possible, where actions such as ear movements or button presses performed on one device are mirrored on any devices that are paired with it. This feature was advertised for the original Nabaztag:tag, but was very temperamental.
6	The Nabaztag:tag should be location aware	The original Nabaztag:tag had no location awareness, so information such as weather was not available without the user configuring the location of the device.

Continued on next page

Table 3 — *Continued from previous page*

No.	Requirement	Details
7	An advanced feature not seen before on the Nabaztag:tag should be implemented.	With a new open hardware and software system in place, a new feature should be implemented that would improve user interaction with the Nabaztag:tag and bring it more in line with current internet connected devices.

Table 3: Requirements of replacement software architecture

5.2.1 Requirement 1

The initial requirement is to find a suitable alternative to XMPP. We require full-duplex communication over a long-lived channel, allowing messages to be sent from the server to the Nabaztag:tag, and vice versa, with minimal overhead. We look to HTTP and its various mechanisms for achieving such communication.

WebSocket is a relatively young protocol, standardised by the IETF in 2011 in Request for Comments (RFC) 6455 [28]. It defines a bi-directional communication protocol over a single TCP connection. Whilst it was primarily designed for sending real-time data to web browsers, WebSocket clients can be implemented outside the browser in standalone applications.

Other methods do exist for creating long-lived connections over HTTP, such as long-polling and XMLHttpRequest (XHR)-polling (often collectively referred to as Comet methods), but a number of papers have discussed the performance improvements of the newer WebSocket protocol over such methods.

Gutwin, Lippold, and Graham compared WebSockets to Comet methods in the context of browser-based collaborative applications, stating that “real-time interaction has much stricter network requirements (in terms of update rate, message throughput, and latency) than semi-synchronous applications” [38]. They found that round-trip latencies were between two and six times lower using WebSockets than with any of the Comet methods, with WebSockets showing the highest message per second throughput of any of the methods tested [38].

Agarwal showed that whilst XHR-polling was limited to 44kbits/s, WebSockets were able to achieve a throughput of 66 kbits/s [3]. He found that “XHR-polling imposes a large overhead of up to 5x” because “each interaction between a client and server is expressed in the form of HTTP requests and responses [...] for example, browser cookies and HTTP headers are included in each request and response” [3]. By contrast he found only “a 1.16x overhead in the case of HTML5 WebSockets” (the baseline is the performance of a raw TCP socket) [3].

A third comparison was performed by Puranik, Feiock, and Hill in the context of a real-time monitoring system. They compared Asynchronous JavaScript and XML (AJAX), a technology making use of XHR-polling, and WebSockets, finding that "WebSockets can send up to 215.44% more data samples when consuming the same amount of network bandwidth as AJAX" [57].

Given the results presented in these papers, it seems clear that the WebSocket protocol provides better performance than other long-lived HTTP connection methods, and would be a suitable replacement for XMPP in the Nabaztag:tag system.

5.2.2 Requirement 2

As highlighted in Section 5.1, the original Nabaztag:tag used XMPP, meaning that messages were sent to and from the device in XML. Whilst XML is human-readable, parsing is expensive, requiring use of the Document Object Model (DOM) [51]. JavaScript Object Notation (JSON) [44] is a lightweight alternative to XML introduced in 2001 [64]. Figure 6 shows an example message in both XML and JSON.

```xml
<message>
  <ear>L</ear>
  <pos>10></pos>
</message>
```

```json
{
  "ear": "L",
  "pos": 10
}
```

Figure 6: A comparison between XML (left) and JSON (right)

In a comparison betwen XML and JSON, Lin et al. found that for JSON and XML objects representing the same data, the JSON object could be transmitted $30 - 35\%$ faster, due to reduced redundancy in describing the data [51]. They also showed that, as the object size increases, the time taken to de-serialise JSON remains reasonably constant, whilst the time taken to de-serialise the equivalent object in XML increases significantly [51].

JSON will be used as a replacement for XML in sending messages over the WebSocket connection between the server and the Nabaztag:tag.

5.2.3 Requirements 3, 4 & 5

A web application meeting Requirement 3 will provide an easy to use interface for interacting with and managing Nabaztag:tag devices. The decision was made to use the Django Web Framework [33] as the author is familiar with the framework, having used it for a previous project. Django is written in Python, meaning the same programming language will be used for both client and server applications, aiding consistency in programming style.

In order to improve the quality of interactions with the Nabaztag:tag, it is important that messages sent to the device through the web application are seen to arrive instantaneously. Combined with Requirements 1 and 2, Requirement 4 identifies the need for a technology that is able to push messages in JSON format from the server to a Nabaztag:tag connected via a WebSocket connection. Brustel and Preuss describe push communication as being "based on the general publish-and-subscribe model" [16], in which a Subscriber subscribes to Topics offered by a Broker, and a Publisher publishes messages to the same topics. The diagram used in their explanation is reproduced in Figure 7. In this example, a message for Topic B is published to the Broker, and any Subscribers subscribed to Topic B receive the message.

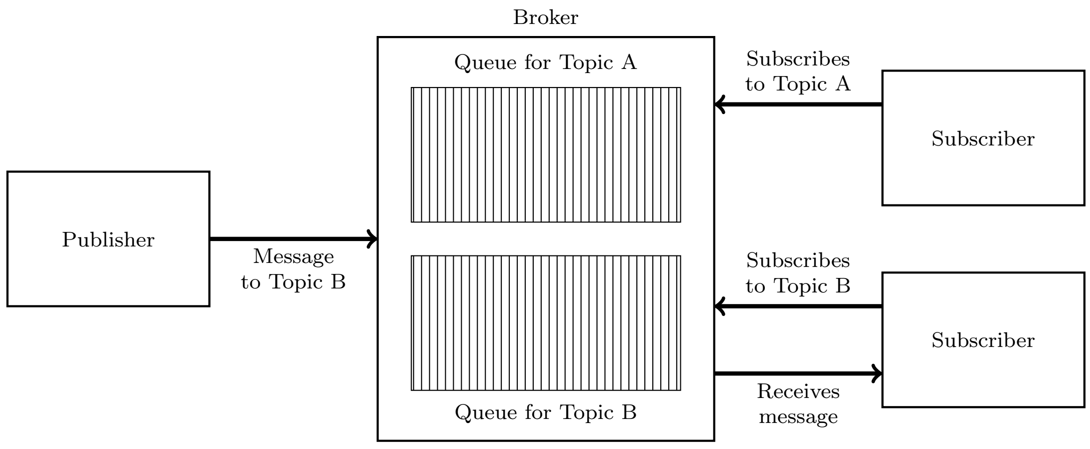


Figure 7: The publish-subscribe model (reproduced from [16])

With the decision to use Django to create the web application, a Django plugin, django-websocket-redis (DWR) [59], was found that enables publish-subscribe behaviour over WebSocket connections. It uses Redis [63], a key-value store, to provide the message queues. A client connects to its own WebSocket using a unique identifier, and any messages placed by Django into a Redis queue with the same identifier are sent across the WebSocket to the client.

Requirement 5 can also be fulfilled as a consequence of using Django for the server application. Its use of the Model-View-Controller (MVC) paradigm means that a class can be created to model a Nabaztag:tag, and separate instances of the class can be stored as pairs in a relation in Django's database.

5.2.4 Requirement 6

A general shortcoming of the Nabaztag:tag, identified by Huang, Bardzell, and Terrell [42], was a lack of contextual awareness. An example of this was the device's weather information feature. As stated in Table 1 in Section 3.2, although the Nabaztag:tag did provide weather information, the user had to manually configure their location for this feature to work. Abowd et al. define context as "any information that can be used to characterise the situation of [...] a computational object" [2], with the contextual information used to "make its [the object's] behaviour more relevant to the situation in which it is being used" [2].

With the advent of smartphones offering both Wi-Fi and Global Positioning System (GPS) technologies, companies such as Apple and Google have been able to collect large amounts of anonymised Wi-Fi base station addresses with associated GPS co-ordinates. Google offers access to their version of this data using their GeoLocation API [35]. Using the Wi-Fi capabilities of the new Nabaztag:tag hardware, we can automatically obtain a reasonably accurate estimation of the Nabaztag:tag's current location, removing a user configuration step and improving the contextual awareness of the device.

5.2.5 Requirement 7

The final requirement is open-ended, we must add a feature to the Nabaztag:tag that was previously not possible, but that adds some utility or function. The major shortcoming of the Nabaztag:tag was its complete reliance on Violet's supporting infrastructure. The eventual failure of this infrastructure resulted in thousands of owners with useless Nabztag:tag devices, and drove the more technically minded to the projects mentioned in Section 1 in attempts to bring their devices back to life.

A truly useful feature, then, would be to enable direct communication with the Nabaztag:tag, removing the central point of failure, and allowing infinitely more customisable interactions with the device. The concept of the IoT was introduced in Section 3.1, and to meet Requirement 7 we turn to a specific embodiment of this idea termed the 'Web of Things' (WoT), which is simply the IoT concept applied specifically to the Web and Web technologies. Gupta, Goldman, and Udupi define the WoT as "everyday objects [...] seamlessly integrated into the World Wide Web (WWW) using its well-known standards and blueprints, e.g. URIs, HTTP and REST." [37].

Representational State Transfer (REST) is an architectural style for the Web introduced by Roy Fielding in his Doctoral Thesis [29]. Guinard et al. offered that "the essence of REST is to focus on creating loosely coupled services on the Web, so that they can be easily reused." [36]. Following the principles of REST, the aim is to allow the resources of the Nabaztag:tag to be easily accessed and used by its owners.

To fulfil this requirement, the Nabaztag:tag's resources, e.g. ears, LEDs, location awareness, and TTS will be made available via an API residing on the device. It will make use of two of HTTP's four main methods to interact with resources — `GET` and `PUT`. This will enable easy use of the resources, and enable third party applications to be created with full access to the Nabaztag:tag's features. Although there are Django plugins available for creating RESTful APIs, Django is unnecessarily heavyweight for a simple API such as this, especially as it will be running on the BBB. Flask [60] is a lightweight web framework, enabling quick creation of RESTful APIs, and the Flask-RESTful plugin [68] adds support for class-based creation of APIs. Again the framework is written in Python, which benefits development and code consistency.

5.3 Design

With the requirements of the system outlined, and major choices of frameworks and libraries made, we move on to the design of the system.

5.3.1 System Architecture

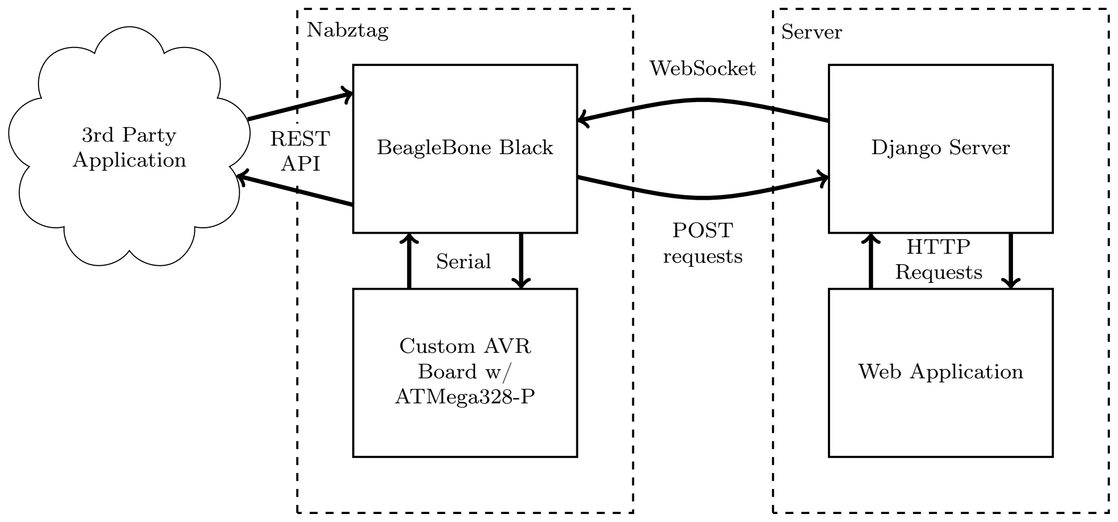


Figure 8: High level overview of the proposed system architecture for the Nabaztag:tag

Using the choices identified in Sections 5.2.1 to 5.2.5 a high level architecture is proposed for the system in Figure 8. In this design, a bi-directional WebSocket connection is established between a client running on the BBB in the Nabaztag:tag, and a Django server instance running the DWR plugin. The user of the Nabaztag:tag can interact with the device through a web application, their actions in the application are sent to the Django server as HTTP requests, then transformed to corresponding JSON messages for transmission over the WebSocket. The BBB converts the JSON message to a serial string suitable for the SerialCommand library running on the ATMega328-P AVR.

For events occurring on the Nabaztag:tag, such as the user moving the ears or pressing the head button, a JSON message is created on the AVR, sent to the BBB via the serial connection, and then forwarded on to the Django server using a HTTP `POST` request for processing. In this case the Django application will determine if any Nabaztag:tags are paired with the sender of the message, and update them accordingly. Although ideally the design would fully utilise the bi-directional nature of the WebSocket in order to send updates to the server, the DWR library chosen doesn't currently support this behaviour, only allowing messages placed into the WebSocket by the client to be stored in Redis, and not allowing them to trigger actions in the Django event loop [59]. This library is under active development, so this feature may be available in the future.

Finally, a RESTful API also runs on the BBB. HTTP requests to the API endpoints are transformed to serial strings suitable for the SerialCommand library running on the ATMega328-P AVR.

5.3.2 Django Application

The Django application is relatively simple, the database stores information for each registered Nabaztag:tag device, and its structure is created from the model classes shown in Figure 9 using Django's `syncdb` tool. Each Nabaztag:tag device is represented within the application by a `Nabaztag` class instance, and a pairing between registered Nabaztag:tags by a `PairedNabaztags` class instance.

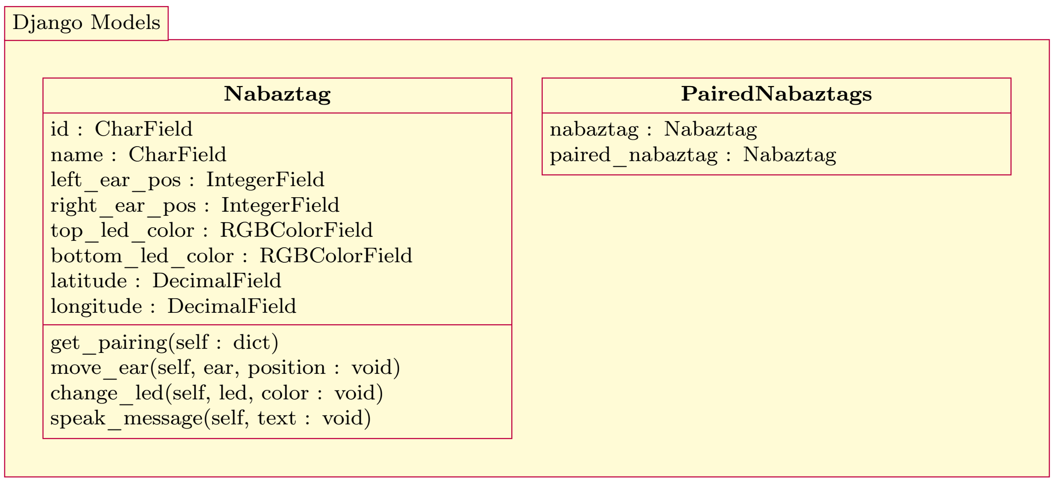

Django Models
Nabaztag
id : CharField
name : CharField
left_ear_pos : IntegerField
right_ear_pos : IntegerField
top_led_color : RGBColorField
bottom_led_color : RGBColorField
latitude : DecimalField
longitude : DecimalField
get_pairing(self : dict)
move_ear(self, ear, position : void)
change_led(self, led, color : void)
speak_message(self, text : void)
PairedNabaztags
nabaztag : Nabaztag
paired_nabaztag : Nabaztag

Figure 9: Class diagram for Django model classes, showing the properties and methods of a `Nabaztag` object, and the pairing relationship between two Nabaztags.

It was decided to continue to use the MAC address of a Nabaztag:tag as its identifier when registering it with the Django application.

Webpages in the application are handled by defined view classes with associated templates that are instantiated when a particular Uniform Resource Locator (URL) pattern is matched. Visiting the root of the site displays an index page where a list of registered Nabaztag:tags is displayed.

Selecting a Nabaztag:tag takes the user to a page, shown in Appendix H in Figure 23, where they can view information about the device and control it. There are a variety of forms controlling different functions of the device. When a form is submitted, the appropriate notification method in the `Nabaztag` class instance is called, e.g. `move_ear`, `change_led` or `speak_message`, which converts the command to a JSON message and places it in the Redis queue identified by the ID of the Nabaztag:tag. If a Nabaztag:tag client is connected via a WebSocket with a matching ID, it will receive the message.

In order to handle updates via `POST` requests from Nabaztag:tag devices, an API was created using the Django REST Framework plugin [19]. This plugin provides class-based API creation, where the developer creates methods, such as `post` and `get` that they wish to provide functionality for. Three `POST` methods are provided by the API:

- `/update/<id>/ear` - If an ear was moved on the device.
- `/update/<id>/button` - If the head button was pressed on the device
- `/update/<id>/location` - To allow the device to update its current location.

5.3.3 Nabaztag:tag Client

Creating the web application was fairly simple as Django handles much of the heavy lifting, however there was no similar framework available to help develop the client application, the classes of which are shown in Figure 10.

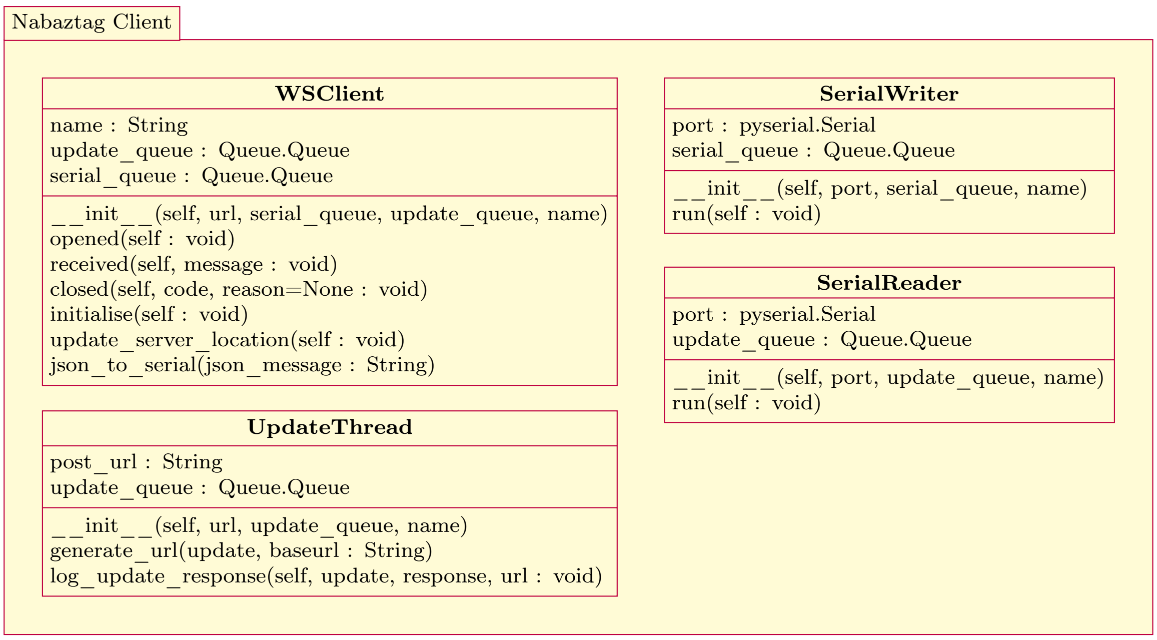

Nabaztag Client
WSClient
name : String
update_queue : Queue.Queue
serial_queue : Queue.Queue
__init__(self, url, serial_queue, update_queue, name)
opened(self : void)
received(self, message : void)
closed(self, code, reason=None : void)
initialise(self : void)
update_server_location(self : void)
json_to_serial(json_message : String)
UpdateThread
post_url : String
update_queue : Queue.Queue
__init__(self, url, update_queue, name)
generate_url(update, baseurl : String)
log_update_response(self, update, response, url : void)
SerialWriter
port : pyserial.Serial
serial_queue : Queue.Queue
__init__(self, port, serial_queue, name)
run(self : void)
SerialReader
port : pyserial.Serial
update_queue : Queue.Queue
__init__(self, port, update_queue, name)
run(self : void)

Figure 10: Class diagram for the Nabaztag client application. Shows the use of callback methods in the `WSClient` class, and shows use of threads and queues in other parts of the application

Two key communication protocols had to be incorporated into the application. Serial communication with the BBB is provided by the `pyserial` library [50], whilst WebSocket communication with the Django application is provided by the `ws4py` library [41]. This was found to be the simplest WebSocket library to work with, allowing inheritance from a threaded `WebSocketClient` class and simply requiring the developer to override the necessary callback methods such as `opened()`, `received_message()` and `closed()`. These methods are then performed upon successful opening of the WebSocket connection, upon receipt of each message, and upon an expected (or unexpected) termination of the connection respectively. Other WebSocket libraries available either had more dependencies or required that callbacks be registered when initialising the class, making for more complex code.

The methods provided by `pyserial` for reading from and writing to the serial port are blocking, so it was decided to perform serial reads and writes in separate threads, both distinct from the WebSocket thread. Using Python's core `Queue` library, two queues were created to achieve communication between the separate threads. One queue handles messages arriving from the server over the WebSocket connection, and the other handles messages arriving from the AVR over the serial connection. The library provides the necessary locking to ensure the queues are thread safe.

A `WSClient` instance receives JSON messages, e.g. `{"ear": "L", "pos": 10}`, from the server and converts them to serial messages, e.g. `EARMOV L 10`, before placing them in the `serial_queue` queue. This is monitored by a `SerialWriter` instance, and messages are taken from the queue and written to the serial port, to be received by the AVR.

A `SerialReader` instance receives JSON messages from the AVR and places them in the `update_queue` queue. This is monitored by an `UpdateThread` instance, which takes messages from the queue and sends `POST` requests containing the message to the correct API method in the Django application.

5.3.4 RESTful API

The use of Flask with the Flask-Restful plug-in allowed the API to be created quickly using knowledge learnt from developing the two previous parts of the system. Here the `pyserial` library is still required to send messages to the AVR, but a WebSocket connection is no longer required, so `ws4py` is not used.

Similar to the approach used in the Django REST Framework, an API resource is created by inheriting from a `Resource` class and overriding the methods which should be made available as API endpoints. Figure 11 shows the five endpoints provided by the Nabaztag:tag API and the HTTP methods available for each. The parameters of the methods, e.g. `ear` in `NabaztagEar`'s `put(self, ear)` method are defined in the URL for the resource, e.g `/nabaztag/api/ear/<string:ear>`, so a PUT request made to `/nabaztag/api/ear/left` would call the `put` method of the `NabaztagEar` class, with `ear="left"`. All requests to and responses from the API are in JSON format.

By specifying an instance of the `RequestParser` class for each method to parse the body of a received request, Flask-Restful can handle many common errors automatically. It can detect if parameters are of the wrong type or are not present, and return an error to the user of the API. Other checks were manually implemented, such as ensuring ear position values and colour values were within allowed ranges.

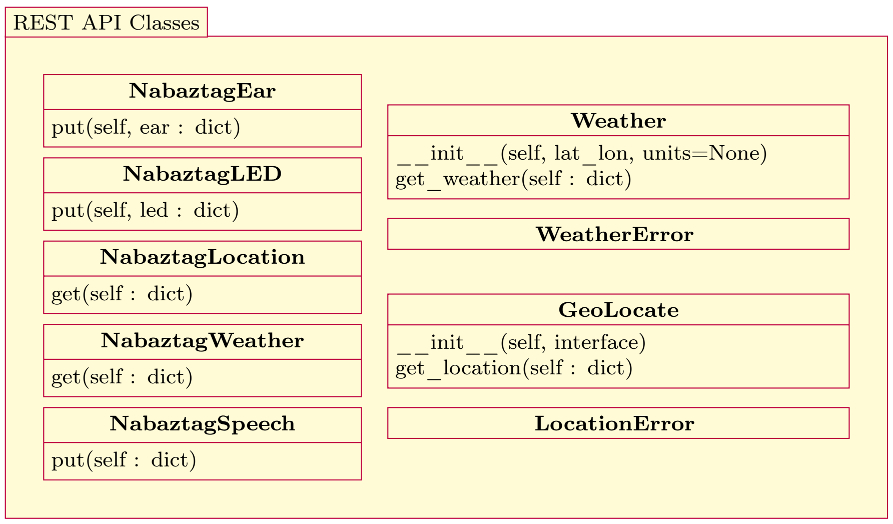


Figure 11: Class diagram for the Nabaztag:tag RESTful API

The `Weather` and `GeoLocate` classes interact with the OpenWeatherMap API [24] and Google's GeoLocate API [35] respectively, parsing the results and making them available through the Nabaztag:tag's API. In case either third party service returns an error, custom Python errors `WeatherError` or `LocationError` are raised so that the API can act accordingly.

During development of the API, a markdown-based language called API Blueprint [4] was used to describe its structure. This enabled full documentation, in the form of a static webpage, to be created easily using a tool called Aglio [67]. This documentation was made available directly from the Nabaztag:tag device, from the root of the API, so Nabaztag:tag users can access the documentation from a URL following a similar pattern to the API itself. An example section of this documentation for the NabaztagEar API method is shown in Appendix I in Figure 24.

5.4 Deployment

Deployment of the system is treated separately for each application, as each requires a different configuration.

5.4.1 Django Application

Following the deployment instructions given in the documentation for the DWR plugin [59], the most scalable deployment solution was chosen, with the structure shown in Figure 12. An open-source HTTP server, Nginx [21], is used as a front-end proxy server. Nginx version 1.4 introduced support for proxying WebSocket connections. Web Server Gateway Interface (WSGI) is a specification for interfacing between Python applications and web servers, and uWSGI [70] is a popular implementation of this, encapsulating applications like Django and allowing HTTP servers to talk to them using WSGI.

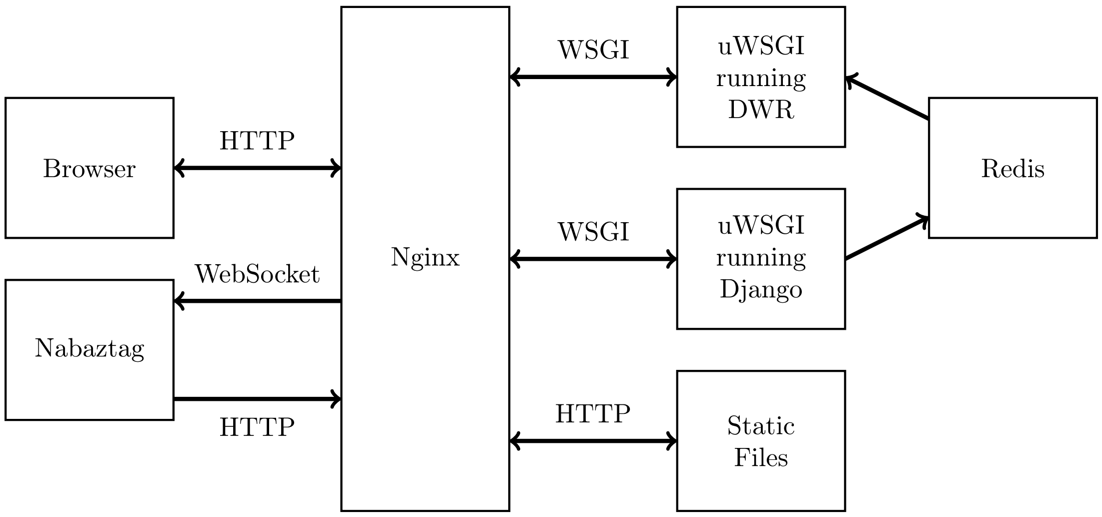


Figure 12: Deployment diagram for Django application (adapted from [59])

The configuration works as follows:

- Static files needed for the Django application, such as stylesheet and image files, are served directly by Nginx.
- A URL path of `/ws/` is defined in the Nginx configuration, and requests to URLs matching this are proxied by Nginx to a uWSGI instance running the DWR plugin.
- All other requests, which are for pages generated by Django, are proxied by Nginx to a uWSGI instance running the Django application.
- The Django application communicates with the DWR plugin solely through Redis message queues.

uWSGI provides a tool called `emperor`, which can monitor a directory for configuration files, and start a separate uWSGI instance for each one. The `emperor` tool is configured to run on startup using Ubuntu's `upstart` utility, and configuration files for the Django and DWR uWSGI instances are placed in the directory it monitors, ensuring they are run on startup. All configuration files and a readme for this deployment are available in the submitted archive, in the `nabaztagserver/deployment` directory.

5.4.2 Nabaztag:tag Client

The client application is also deployed using the `upstart` utility, which registers it as a system service, allowing it to be started and stopped by the user. It is automatically started on boot, reading its configuration information from a file.

TTS is provided by Festival [65], which provides reasonable speech quality, without the slowdown experienced from fetching a TTS audio file from a third party API.

5.4.3 RESTful API

The API is deployed in a similar way to the Django application, although the configuration is considerably less complicated. Tornado [25], another WSGI container application, serves the API to localhost only. Nginx then serves the API documentation as a static webpage at the root of the API, and proxies all other requests to the localhost port for processing as API requests.

To enable easy access to the API, the Ubuntu `avahi-daemon` [46] is installed. This provides zero configuration networking services, enabling the Nabaztag:tag to broadcast a Domain Name System Service Discovery (DNS-SD) address of `nabaztag.local` to its subnet. `http://nabaztag.local` serves the API documentation, and the API itself is at `http://nabaztag.local/nabaztag/api`.

5.5 Testing

During the project, several techniques were employed to test functionality of code and help assist debugging.

5.5.1 Logging & Debugging

For the Django application, the django-debug-toolbar plugin [43] was used, which provides a browser-based utility displaying useful debugging information when visiting the webpages served by the application. Additionally, when the `DEBUG=True` flag is set in the Django application settings, verbose debugging messages are displayed instead of the usual HTTP error pages.

For the Nabaztag:tag client and REST API applications, the Python core logging library was used to record actions taken, messages received, and requests received by the applications. These logfiles, stored in `/var/log/nabaztag/`, could be monitored interactively using the Unix `tail -f` command, and this was invaluable in determining how messages were sent and received by different parts of the system.

5.5.2 Unit Tests

The Python core `unittest` library was used to perform unit testing on the Nabaztag:tag client application. This is a testing method where small sections of the code, responsible for a single function, are tested for success, failure and edge cases using automated test scripts. Also used were the `httpretty` library [27], which allows mocking of HTTP responses for testing, and `mock` [30], which allows mocking of methods and classes for testing. Mocking ensures that the tests are only dependent on the code they are testing, and not on API responses or the behaviour of other methods or classes.

15 tests were created in total, with 89% of the code of the client application covered by the tests, as measured by the `coverage` tool [12]. Results of these tests are shown in Appendix J in Figure 25. Some issues were encountered when testing the serial classes `SerialReader` and `SerialWriter`, due to use of threads and infinite while-loops, so results for these classes are not included.

5.5.3 Integration Testing

To ensure that all parts of the system functioned correctly once communicating with one another, various integration level tests were carried out. Two main techniques are detailed.

In order to test and be able to demonstrate the pairing functionality, a Raspberry Pi [71] with attached Berryclip [39] was used to represent another Nabaztag:tag in the system. The client software was modified slightly in order to control the LEDs of the Berryclip, using them to represent ears, and to treat the button as if it were the head button of a Nabaztag:tag. The two devices can then be registered as Nabaztags and paired in the Django application, and it can be demonstrated both visually and through monitoring of logs that actions performed on one device are relayed to the other.

Another useful testing utility was the Unix `cURL` tool [40]. This enables HTTP requests to be made from the command-line, useful when sending and receiving JSON requests and responses to and from the RESTful API. Correct, incorrect and malformed requests can be made, and the responses examined.

6 Project Management

Project management is essential for any medium or longer length projects with multiple deadlines and tasks. To ensure that the project remained on track and that any incidents were dealt with, various project management techniques and exercises were undertaken.

6.1 Version Control

A private Git repository, hosted on GitHub, was used to hold and track changes to all project files. This allowed easy access to the project from any computer, as well as providing a backup of the project, and a detailed version history so that any erroneous modifications or deletions could be reverted.

During the course of the project, there were 255 commits to the repository. Figure 13 shows the commits for the software implementation stage of the project which occurred during the months of February and March. There were 173 commits in this period.

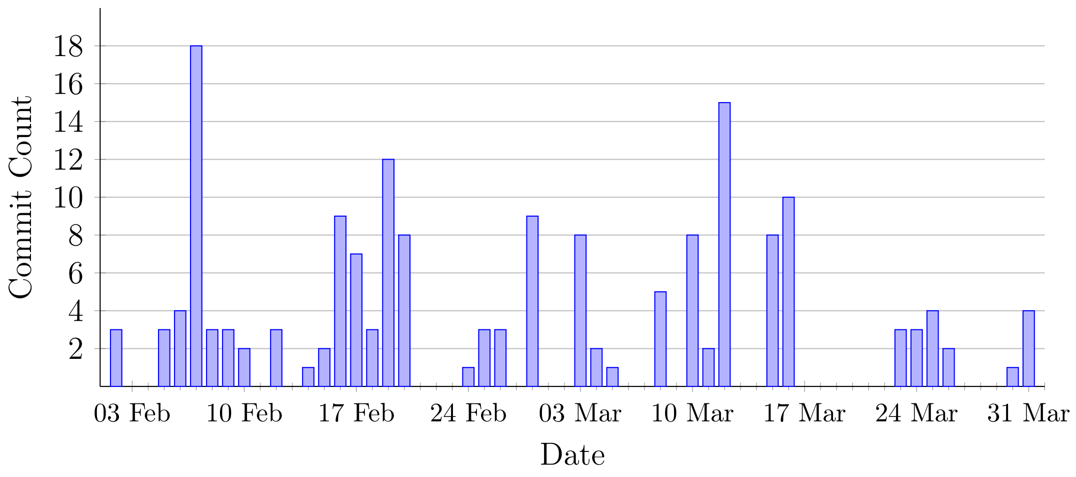


Figure 13: A graph showing frequency of commits during the implementation stage

The code breakdown for the project, calculated using `cloc` [22], is given in Table 4.

Language	Files	Comments	Lines of Code
Python	20	370	1034
HTML	3	6	317
Arduino Sketch	1	75	183

Table 4: Count of line of code in the project.

6.2 Risk Assessment

In order to identify and prepare for issues arising during the course of the project, a risk assessment was carried out. Major risks were first identified, then given an impact score and a rough likelihood of occurrence. The findings of the assessment and the steps taken to mitigate the risks are found in Table 5.

6.2.1 Risks

- My main development machine will be my personal iMac, which could suffer data corruption or hardware failure.
- I have only basic knowledge of electronics, so may find the hardware aspects of the project difficult.
- Software development will be largely in Python, however I have only a little experience developing in Python.

- Whilst disassembling and testing the hardware of the Nabaztag:tag, I may inadvertently damage it.
- Given the length of the project, it is likely that I will be ill at least once during the project, which may affect my project progress.
- During the course of the project, I will also be expected to complete coursework for other modules, as well as sitting my January exams.

6.2.2 Assessment

Event	**Loss (1-5)**	**Probability (1-10)**	**Risk (L*P)**	**Steps to Mitigate Risk**
Illness and Other Deadlines	5	8	40	Plan my project time carefully, ensuring awareness of other commitments such as examinations and coursework deadlines. If ill, attempt to complete work at home.
Difficulty with electronics	3	9	27	I have access to a lab, led by Dr. Martinez, with two postgraduate students who will be able to assist me in identifying and interfacing with components of the Nabaztag.
Development in a new language	4	5	20	Within ECS, there are students who have experience in Python from whom I will be able to seek advice.
Damage to Nabaztag hardware	3	4	12	Dr. Martinez has a second Nabaztag device from which spare parts could be sourced if necessary.

Continued on next page

Table 5 — *Continued from previous page*

Event	**Loss (1-5)**	**Probability (1-10)**	**Risk (L*P)**	**Steps to Mitigate Risk**
Computer Failure or Data Corruption	5	2	8	If my main personal development machine fails, I can use any ECS lab machine. All source code for the project will be version controlled in Git, and hosted on GitHub. Other project work will be stored in Dropbox, so accessible on any machine. Additionally, I have an external hard drive attached to my development machine performing daily backups.

Table 5: Risk assesment calculations & mitigation techniques

6.3 Final Project Gantt Charts

In October, Gantt charts covering the entire duration of the project, shown in Appendix K in Figure 26 and Figure 27, were created to monitor progress and avoid clashes with other deadlines. The schedule set out in the charts was followed largely successfully throughout the project, with several exceptions. Due to the pressure of other deadlines at the start of December, the software design was not carried out, and was instead completed at the start of Semester 2. Prototyping of hardware was performed earlier than expected, but took longer than anticipated, and again the deadline pressure in the first week of December meant that there was a gap of several weeks where no work was performed for the project.

Due to deliberation around whether to re-implement the audio output functionality of the device, the Onecall order for a USB soundcard and USB hub was not placed until the 05/02/14. An error was made when ordering which resulted in a lengthy delay, and the goods were not received until the 28/02/14. This meant several days had to be found at the start of March to find a way to fit these devices into the Nabaztag:tag and finish the assembly of the hardware. This was successfully completed alongside the software implementation process due to relatively little other work.

Finally, software testing was not as thorough as desired, and only took place at the end of March.

These changes are reflected in revised Gantt charts in Figure 14 and Figure 15, with any activities that changed highlighted by a diagonal pattern.

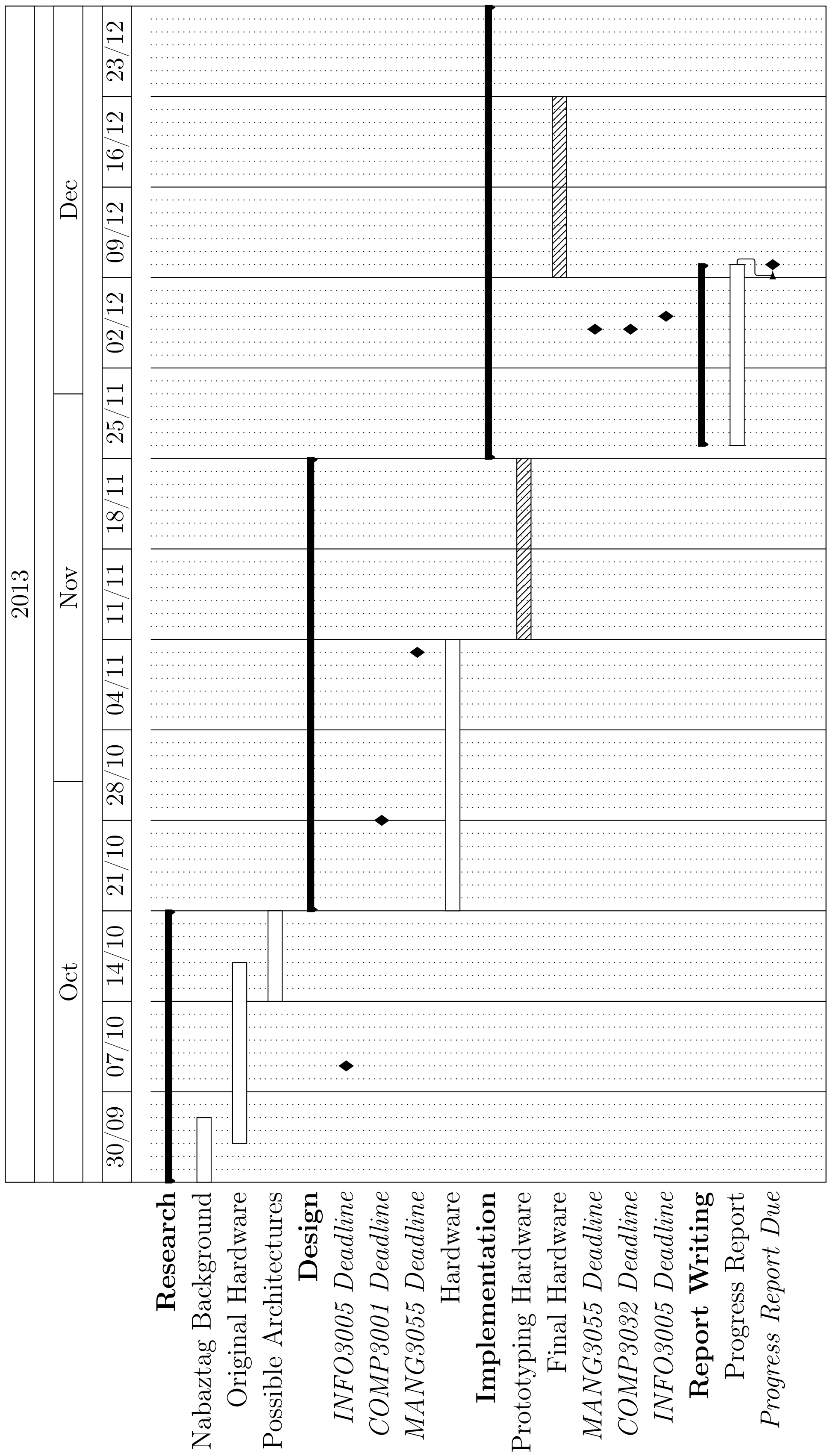


Figure 14: Revised Semester 1 Gantt chart

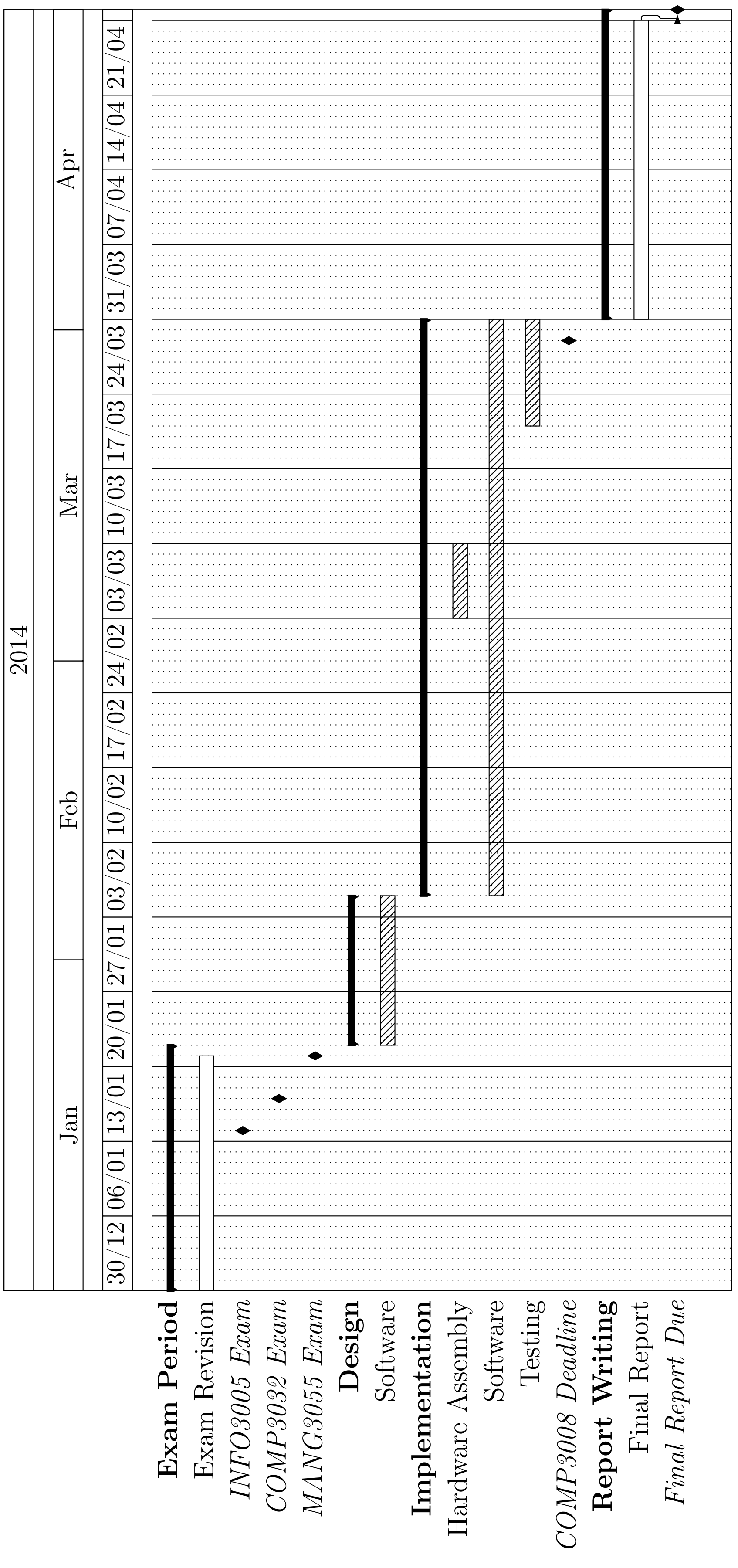


Figure 15: Revised Semester 2 Gantt chart

7 Critical Evaluation

In order to evaluate the success of the project, we refer back to the project goals in Section 2, and the project brief in Appendix M.

7.1 Goal One

The first goal deals with examination and replacement of the Nabaztag:tag's hardware. Using a variety of investigations found online, plus the author's own examinations, the components making up the hardware of the device were identified and their characteristics, with the exception of the RFID reader, discovered. The ear movement and LEDs were reimplemented using a custom designed AVR-based circuit, with the internet connectivity and TTS capabilities provided using a BBB with attached USB Wi-Fi adapter and soundcard.

A number of compromises were made, due to time constraints and minimal electronics knowledge, to allow the hardware section of the project to be completed in the first semester. The number of available LEDs was reduced from five to two, and the 3.5mm audio output, volume control and RFID reader were not reimplemented. Because of this, the initial goal is only partially met, but it was felt that the functionality that was restored would be sufficient to showcase the software system that was the main focus of the project. An additional issue, identified too late in the project to rectify, was the very low volume audio output from the USB soundcard. Although this was considered the simplest way to reimplement audio output via the speaker in the device, a more advanced solution would have used the BBB directly for audio output and incorporated an amplifier circuit.

7.2 Goal Two

The second goal focuses on creation of a general software infrastructure to support the device. The original brief suggested a client application running on the Nabaztag:tag, along with a server application connected to the Nabaztag:tag using a push architecture. To achieve this goal, a web application was developed using the Django framework, and a client application was developed in Python. The two were linked by a combination of a WebSocket connection and an HTTP API. This enabled messages to be delivered almost instantaneously over the WebSocket from the web application to the Nabaztag:tag client, and allowed updates from the Nabaztag:tag device to be sent to the server using the API. Requirement 4 in Table 3 specified that the replacement system should attempt to rectify the speed issues of the previous infrastructure. Whilst no formal tests were performed, it seems that the replacement system is responsive, with sub-second message delivery times routinely observed. The goal also explicitly mentions control from a web application, and this has been reimplemented in a simple manner, allowing control of one function at a time.

As this goal neared completion, several possibilities for improvement were identified with the architecture. A major concern is the lack of security consideration whilst developing the system. All messages sent from the server to a Nabaztag:tag are in plaintext, and thus readable with packet sniffing tools such as Wireshark [34]. Although the WebSocket protocol does provide a secure connection with `wss://`, this could not be implemented as it was not supported by the DWR plugin. Additionally, no authentication mechanism has been implemented for the API running on the server, so anyone discovering the identifier of a Nabaztag:tag device could pose as that device, sending spurious updates to the server and potentially abusing the pairing functionality. Finally, it is clear that more functionality could be added to the web application itself, including features such as user accounts, ability for users to register Nabaztag:tags and ability to script actions or set timed events for the Nabaztag:tag.

7.3 Goal Three

The final goal, not suggested in the project brief but added as a goal during the first semester, was to implement a new feature for the Nabaztag:tag enabled by the new hardware and software. This feature was chosen to be a RESTful API, running directly on the Nabaztag:tag itself. This was not previously possible due to the closed nature of the original hardware, but successfully removes the reliance on the server application for control of the device. The API exposes all restored functionality through five API endpoints, with detailed documentation also available from the device itself. Although the goal was deliberately vague, it is felt to be adequately addressed by the functionality implemented.

7.4 Time Management

Throughout the project, time was well managed with minimal deviations from the initial schedule, and no major issues or incidents occurred. Due to user error, there was a delay in the delivery of ordered components, but the use of the schedule meant that this delay was managed and did not affect other parts of the project.

8 Conclusions and Future Work

8.1 Conclusions

In this project, a single Nabaztag:tag device was revived through a combination of replacement hardware and software. The hardware part of the project was successfully completed in the first semester, with the software part completed in good time during the second semester. The end result is a functional Nabaztag:tag with many of the major features re-implemented, demonstrating the use of several current technologies for IoT devices. Areas for improvement have been identified, and the project has raised some possible areas for continuing development, detailed in Section 8.2.

The project was an enjoyable and educational experience, greatly furthering the author's electronics knowledge, and improving their project management skills and Python programming ability.

8.2 Future Work

During this project, several potential areas for further work were identified:

BeagleBone Black Cape — The custom AVR board plus USB hub, soundcard, and Wi-Fi dongle were challenging to fit into the case of the Nabaztag:tag, and the nature of the modifications mean they would be difficult to repeat. The process could be greatly simplified by the creation of a cape for the BBB, which would attach to its headers and provide the sound, Wi-Fi and connections for the Nabaztag:tag's hardware in one package. The total modifications to the Nabaztag:tag would then be limited to removing the old PCB, then fitting the BBB and cape and attaching the Nabaztag:tag's hardware.

Framework for WebSocket-based IoT devices — Despite the issues mentioned in the evaluation, the overall architecture developed for push communication to an IoT device is successful. To provide a reusable way to apply this architecture, a general framework could be created, solving issues such as:

- Secure communications.
- WebSockets used for communication in **both** directions, as intended.
- Support for applications other than Redis to provide the message queue.
- Support for multiple message formats.

Discovering APIs on WoT devices — As the WoT grows, our home and office networks will contain more and more devices offering APIs. Whilst REST provides a standard for interacting with these APIs over HTTP, automatic discovery of the APIs will also become important, e.g. when adding a new device to the network or when a new client joins the network and wishes to see available APIs. This could be combined with a standard way for devices to provide documentation for their API using the same service.

References

Articles, Books & Proceedings

[2] Gregory D. Abowd et al. “Towards a better understanding of context and context-awareness”. In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. 1999, pp. 304–307.

[3] Sachin Agarwal. “Real-Time Web Application Roadblock: Performance Penalty of HTML Sockets”. In: *2012 IEEE International Conference on Communications*. 2012, pp. 1225–1229.

[11] Kevin Ashton. *That ‘Internet of Things’ Thing*. [Accessed Oct. 15, 2014]. June 2009. URL: `http://www.rfidjournal.com/articles/view?4986`.

[13] Clément Beausset and Pierre Soumoy. *Construction d’un Proxy XMPP pour un Nabaztag*. Tech. rep. [Accessed Oct. 10, 2013]. Telecom SudParis, 2009. URL: `http://www-public.it-sudparis.eu/~leriche/projetud/XMPP-Nabaztag/RapportPFE_Nabaztag.pdf`.

[16] Jonas Brustel and Thomas Preuss. “A Universal Push Service for Mobile Devices”. In: *Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*. 2012, pp. 40–45.

[17] Laurie Butgereit, Louis Coetzee, and Andrew C. Smith. “Turn Me On! Using the "Internet of Things" to Turn Things On and Off”. In: *Proceedings of the 6th International Conference on Pervasive Computing and Applications*. 2011, pp. 4–10.

[29] Roy T. Fielding. “Architectural styles and the Design of Network-Based Software Architectures”. PhD thesis. University of California, Irvine, 2000.

[36] Dominique Guinard et al. “From the Internet of Things to the Web of Things: Resource-Oriented Architecture and Best Practices”. In: *Architecting the Internet of Things*. Ed. by Dieter Uckelmann, Mark Harrison, and Florian Michahelles. 2011. Chap. 5, pp. 97–129.

[37] Vipul Gupta, Ron Goldman, and Poornaprajna Udupi. “A Network Architecture for the Web of Things”. In: *Proceedings of the Second International Workshop on Web of Things*. 2011, 3:1–3:6.

[38] Carl Gutwin, Michael Lippold, and TC Graham. “Real-Time Groupware in the Browser: Testing the Performance of Web-Based Networking”. In: *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. 2011, pp. 167–176.

[42] Chung-Ching Huang, Jeffrey Bardzell, and Jennifer Terrell. “Can Your Pet Rabbit Read Your Email?: A Critical Analysis of the Nabaztag Rabbit”. In: *Proceedings of the 2011 Conference on Designing Pleasurable Products and Interfaces*. 2011, 25:1–25:8.

[51] Boci Lin et al. “Comparison between JSON and XML in Applications Based on AJAX”. In: *2012 International Conference on Computer Science and Service System*. 2012, pp. 1174–1177.

[52] Kate Lund, Paul Coulton, and Reuben Edwards. “Ambient Conversations Using a Physical Avatar”. In: *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*. 2009, pp. 11–14.

[57] Darshan G. Puranik, Dennis C. Feiock, and James H. Hill. “Real-Time Monitoring using AJAX and WebSockets”. In: *20th IEEE International Conference and Workshops on Engineering of Computer Based Systems*. 2013, pp. 110–118.

[62] Peter Saint-Andre. “Streaming XML with Jabber/XMPP”. In: *Internet Computing, IEEE* 9.5 (2005), pp. 82–89.

[64] Charles Severance. “Discovering JavaScript Object Notation”. In: *Computer, IEEE* 45.4 (2012), pp. 6–8.

[66] Lu Tan and Neng Wang. “Future internet: The Internet of Things”. In: *3rd International Conference on Advanced Computer Theory and Engineering*. Vol. 5. 2010, pages.

[75] Mark Weiser. “The Computer for the 21st Century”. In: *Scientific American* 265.3 (1991), pp. 66–75.

RFCs

[28] Ian Fette and Alexey Melnikov. *The WebSocket Protocol*. RFC 6455 (Proposed Standard). Internet Engineering Task Force, Dec. 2011. URL: `http://www.ietf.org/rfc/rfc6455.txt`.

[61] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120 (Proposed Standard). Internet Engineering Task Force, Mar. 2011. URL: `http://www.ietf.org/rfc/rfc6120.txt`.

Datasheets & Schematics

[6] Arduino. *Arduino Duemilanove Schematic*. [Accessed Oct. 29, 2013]. 2009. URL: `http://arduino.cc/en/uploads/Main/arduino-duemilanove-schematic.pdf`.

[9] Arduino. *ATMega 328 Datasheet*. [Accessed Oct. 28, 2013]. 2009. URL: `http://www.atmel.com/Images/doc8161.pdf`.

[10] Arduino. *ATmega168/328 - Arduino Pin Mapping*. [Accessed Oct. 28, 2013]. 2013. URL: `http://arduino.cc/en/Hacking/PinMapping168`.

[32] BeagleBoard.org Foundation. *BeagleBone Black Reference Manual*. [Accessed Oct. 15, 2013]. 2013. URL: `http://www.farnell.com/datasheets/1701090.pdf`.

[45] Kingbright. *T-1 3/4 (5mm) FULL COLOR LED LAMP*. [Accessed Oct. 16, 2013]. 2012. URL: `http://www.farnell.com/datasheets/1683536.pdf`.

Websites

[1] 01net. *Nabaztag, le Lapin est Mort ce Soir*. [Accessed Oct. 3, 2013]. 2011. URL: `http://www.01net.com/editorial/536566/nabaztag-le-lapin-est-mort-ce-soir/`.

[4] Apiary.io. *API Blueprint.* [Accessed Mar. 25, 2014]. 2014. URL: http://apiblueprint.org/.

[5] Arduino. *Arduino Duemilanove.* [Accessed Oct. 15, 2013]. 2009. URL: http://arduino.cc/en/Main/ArduinoBoardDuemilanove.

[7] Arduino. *Arduino - HomePage.* [Accessed Oct. 15, 2013]. 2013. URL: http://arduino.cc/.

[8] Arduino. *Arduino - Software.* [Accessed Oct. 15, 2013]. 2013. URL: http://arduino.cc/en/Main/Software.

[12] Ned Batchelder. *Coverage.* [Accessed Apr. 6, 2014]. 2014. URL: http://nedbatchelder.com/code/coverage/.

[14] Nabaztag Blog. *Paris Tuesday, December 26, 2006. Press release from Violet.* [Accessed Oct. 5, 2013]. 2006. URL: http://web.archive.org/web/20070104232121/http://blog.nabaztag.com/2006/12/paris_tuesday_d.html.

[15] Philip Boshua. *LIFX - The Lightbulb Reinvented.* [Accessed Nov. 18, 2013]. 2013. URL: http://lifx.co.

[18] CadSoft. *EAGLE PCB Software.* [Accessed Nov. 01, 2013]. 2013. URL: http://www.cadsoftusa.com/eagle-pcb-design-software/?language=en.

[19] Tom Christie. *Django REST Framework.* [Accessed Mar. 25, 2014]. 2014. URL: http://www.django-rest-framework.org/.

[20] Sean Cogswell. *ArduinoSerialCommand.* [Accessed Oct. 15, 2013]. 2012. URL: https://github.com/scogswell/ArduinoSerialCommand/.

[21] Nginx Community. *Nginx.* [Accessed Mar. 25, 2014]. 2014. URL: http://wiki.nginx.org/Main.

[22] Al Danial. *Cloc.* [Accessed Apr. 6, 2014]. 2014. URL: http://cloc.sourceforge.net/.

[23] DMEUROPE. *Mindscape buys Violet out of receivership.* [Accessed Oct. 4, 2013]. 2009. URL: http://www.siptrunkingreport.com/news/2009/10/21/4435681.htm.

[24] Extreme Electronics. *Open Weather Map API.* [Accessed Mar. 25, 2014]. 2014. URL: http://openweathermap.org/API.

[25] Facebook. *Tornado.* [Accessed Mar. 25, 2014]. 2014. URL: http://www.tornadoweb.org/en/stable/.

[26] Tony Fadell. *Nest Thermostat.* [Accessed Nov. 18, 2013]. 2013. URL: https://nest.com/thermostat/life-with-nest-thermostat/.

[27] Gabriel Falcao. *HTTPretty.* [Accessed Apr. 6, 2014]. 2014. URL: https://github.com/gabrielfalcao/HTTPretty.

[30] Michael Foord. *Mock.* [Accessed Apr. 6, 2014]. 2014. URL: http://www.voidspace.org.uk/python/mock/.

[31] BeagleBoard.org Foundation. *BeagleBone Black.* [Accessed Oct. 15, 2013]. 2013. URL: http://beagleboard.org/products/beaglebone%20black.

[33] Django Software Foundation. *Django.* [Accessed Mar. 25, 2014]. 2014. URL: https://www.djangoproject.com/.

[34] Wireshark Foundation. *Wireshark.* [Accessed Apr. 6, 2014]. 2014. URL: http://www.wireshark.org/download.html.

[35] Google. *Google GeoLocation API.* [Accessed Mar. 25, 2014]. 2014. URL: https://developers.google.com/maps/documentation/business/geolocation/.

[39] Matt Hawkins. *Berryclip.* [Accessed Apr. 6, 2014]. 2014. URL: http://www.raspberrypi-spy.co.uk/berryclip-6-led-add-on-board/.

[40] Haxx. *cURL.* [Accessed Apr. 6, 2014]. 2014. URL: http://curl.haxx.se/.

[41] Sylvain Hellegouarch. *Websocket for Python.* [Accessed Mar. 25, 2014]. 2014. URL: https://github.com/Lawouach/WebSocket-for-Python.

[43] Rob Hudson. *Django Debug Toolbar.* [Accessed Apr. 6, 2014]. 2014. URL: https://github.com/django-debug-toolbar/django-debug-toolbar.

[44] json.org. *JSON.* [Accessed Mar. 25, 2014]. 2014. URL: http://www.json.org/.

[46] Oliver Kurth. *Avahi-Daemon.* [Accessed Mar. 25, 2014]. 2014. URL: http://manpages.ubuntu.com/manpages/raring/man8/avahi-daemon.8.html.

[47] Mathieu Lesniak. *Le Nabaztag, comment ça marche? Partie 1: Le boot.* [Accessed Mar. 8, 2014]. 2011. URL: http://www.eskuel.net/le-nabaztag-comment-ca-marche-partie-1-le-boot-1484.

[48] Mathieu Lesniak. *Le Nabaztag, comment ça marche? Partie 2: L'authentification et la fin du boot.* [Accessed Mar. 8, 2014]. 2011. URL: http://www.eskuel.net/le-nabaztag-comment-ca-marche-partie-2-lauthentification-et-la-fin-du-boot-1485.

[49] Mathieu Lesniak. *Le Nabaztag, comment ça marche? Partie 3: Communiquer avec un lapin.* [Accessed Mar. 8, 2014]. 2011. URL: http://www.eskuel.net/le-nabaztag-comment-ca-marche-partie-3-communiquer-avec-un-lapin-1486.

[50] Chris Liechti. *Pyserial.* [Accessed Mar. 25, 2014]. 2014. URL: http://pyserial.sourceforge.net/.

[53] IHS Electronics & Media. *Violet Nabaztagtag Wireless Device Teardown.* [Accessed Oct. 10, 2013]. 2007. URL: http://electronics360.globalspec.com/article/3643/violet-nabaztagtag-wireless-device-teardown.

[54] OpenJabNab. *OpenJabNab.* [Accessed Oct. 5, 2013]. 2012. URL: https://github.com/OpenJabNab/OpenJabNab.

[55] Oripy. *Nabaztag Hardware Investigation.* [Accessed Oct. 6, 2013]. 2012. URL: https://github.com/Oripy/Rabbity-Pi/wiki/Nabaztag-hardware-investigation.

[56] Pokey. *NabaztagLives.* [Accessed Oct. 5, 2013]. 2013. URL: http://sourceforge.net/projects/nabaztaglives/.

[58] Queaker. *Nabaztag.* [Accessed Mar. 8, 2014]. 2008. URL: https://www.c3pb.de/wiki/nabaztag.

[59] Jacob Rief. *Django-Websocket-Redis.* [Accessed Mar. 25, 2014]. 2014. URL: http://django-websocket-redis.readthedocs.org/en/latest/.

[60] Armin Ronacher. *Flask Web Framework.* [Accessed Mar. 25, 2014]. 2014. URL: https://github.com/mitsuhiko/flask.

[63] Salvatore Sanfilippo. *Redis.* [Accessed Mar. 25, 2014]. 2014. URL: http://redis.io/.

[65] The Centre for Speech Technology Research. *Festival.* [Accessed Mar. 25, 2014]. 2014. URL: http://www.cstr.ed.ac.uk/projects/festival/.

[67] Daniel Taylor. *Aglio*. [Accessed Mar. 25, 2014]. 2014. URL: https://github.com/danielgtaylor/aglio.

[68] Twilio. *Flask-RESTful*. [Accessed Mar. 25, 2014]. 2014. URL: https://github.com/twilio/flask-restful.

[69] Peter Tyser. *Nabaztagtag (Nabaztag v2) Dissection*. [Accessed Oct. 6, 2013]. 2007. URL: http://www.petertyser.com/2007/03/11/nabaztag-nabaztagtag-dissection/.

[70] Unbit. *uWSGI*. [Accessed Mar. 25, 2014]. 2014. URL: http://projects.unbit.it/uwsgi/.

[71] Eben Upton. *Raspberry Pi*. [Accessed Apr. 6, 2014]. 2014. URL: http://www.raspberrypi.org/.

[72] Violet. *And Now, He Has a Belly Button*. [Accessed Oct. 3, 2013]. 2006. URL: http://www.regaletes.com/images/mascotaselectronicas/nabaztag_EN.pdf.

[73] Violet. *Karotz, Smart Wi-Fi Rabbit, New Version of Nabaztag*. [Accessed Oct. 3, 2013]. 2013. URL: http://store.karotz.com/en_GB/.

[74] Violet. *Violet - Let All Things Be Connected*. [Accessed Oct. 5, 2013]. 2009. URL: http://web.archive.org/web/20090206020240/http://www.violet.net/index_en.html.

[76] Wikipedia. *Nabaztag*. [Accessed Mar. 25, 2014]. 2014. URL: http://nl.wikipedia.org/wiki/Nabaztag.

[77] Business Wire. *Karotz, the Intelligent Rabbit from Mindscape, to Make North American Debut at Consumer Electronics Show*. [Accessed Oct. 4, 2013]. 2011. URL: http://www.businesswire.com/news/home/20101209006448/en/Karotz-Intelligent-Rabbit-Mindscape-North-American-Debut.

Appendices

A ATMega328-P Pinout

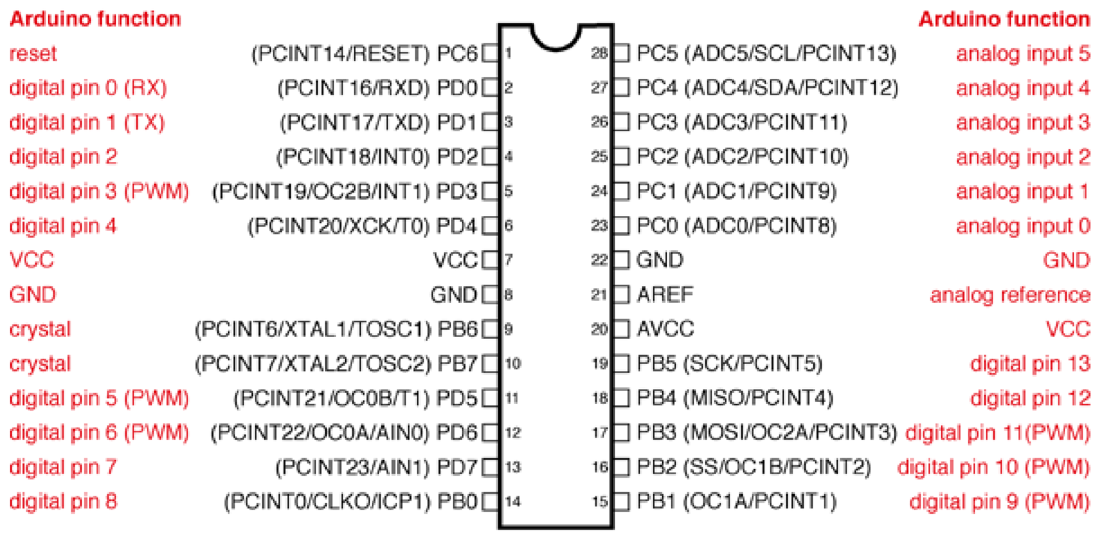


Figure 16: ATMega168/328 pinout (from [10])

B Custom AVR Board Schematic

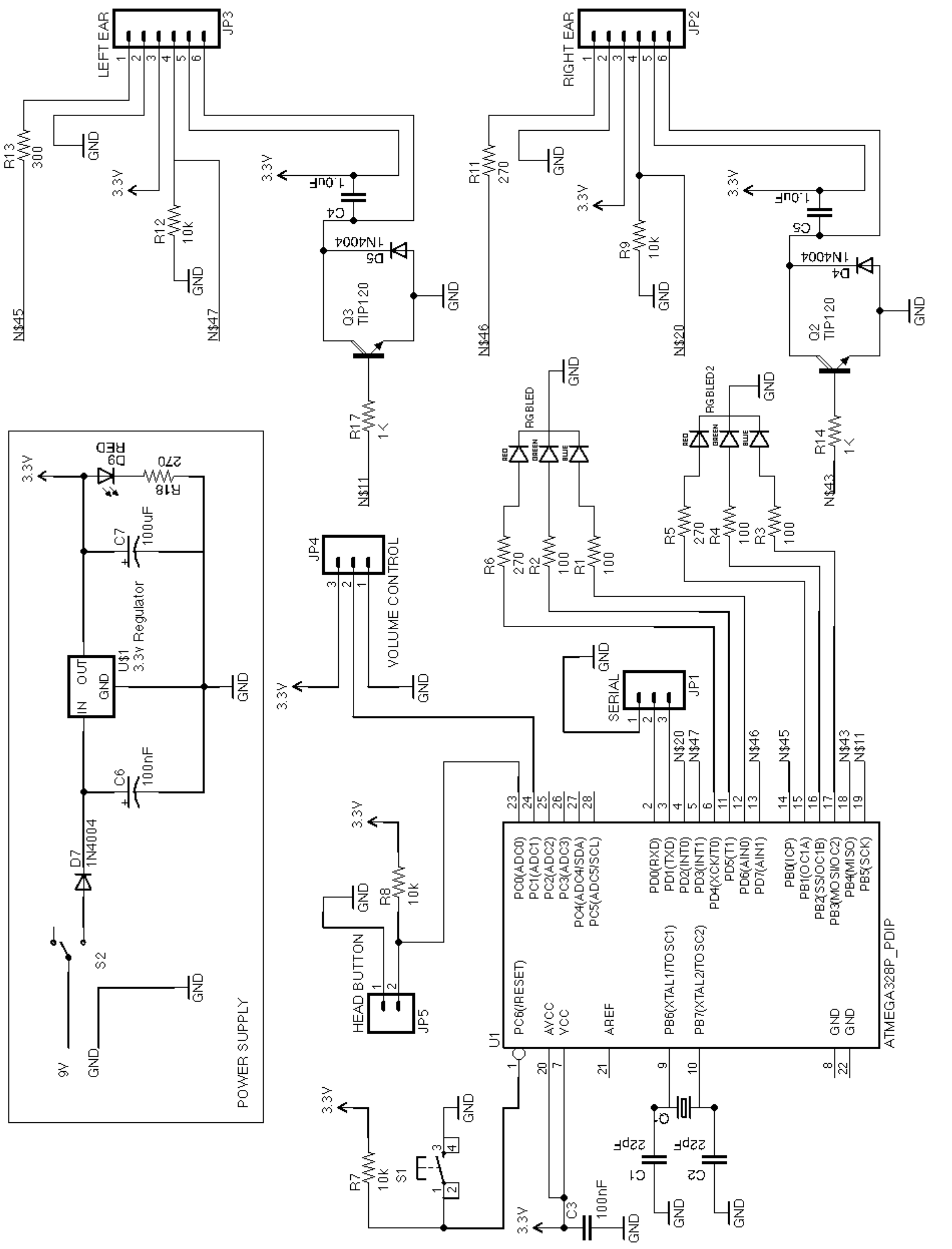


Figure 17: Schematic for custom AVR circuit created to control original I/O hardware.

C Custom AVR Board Prototype

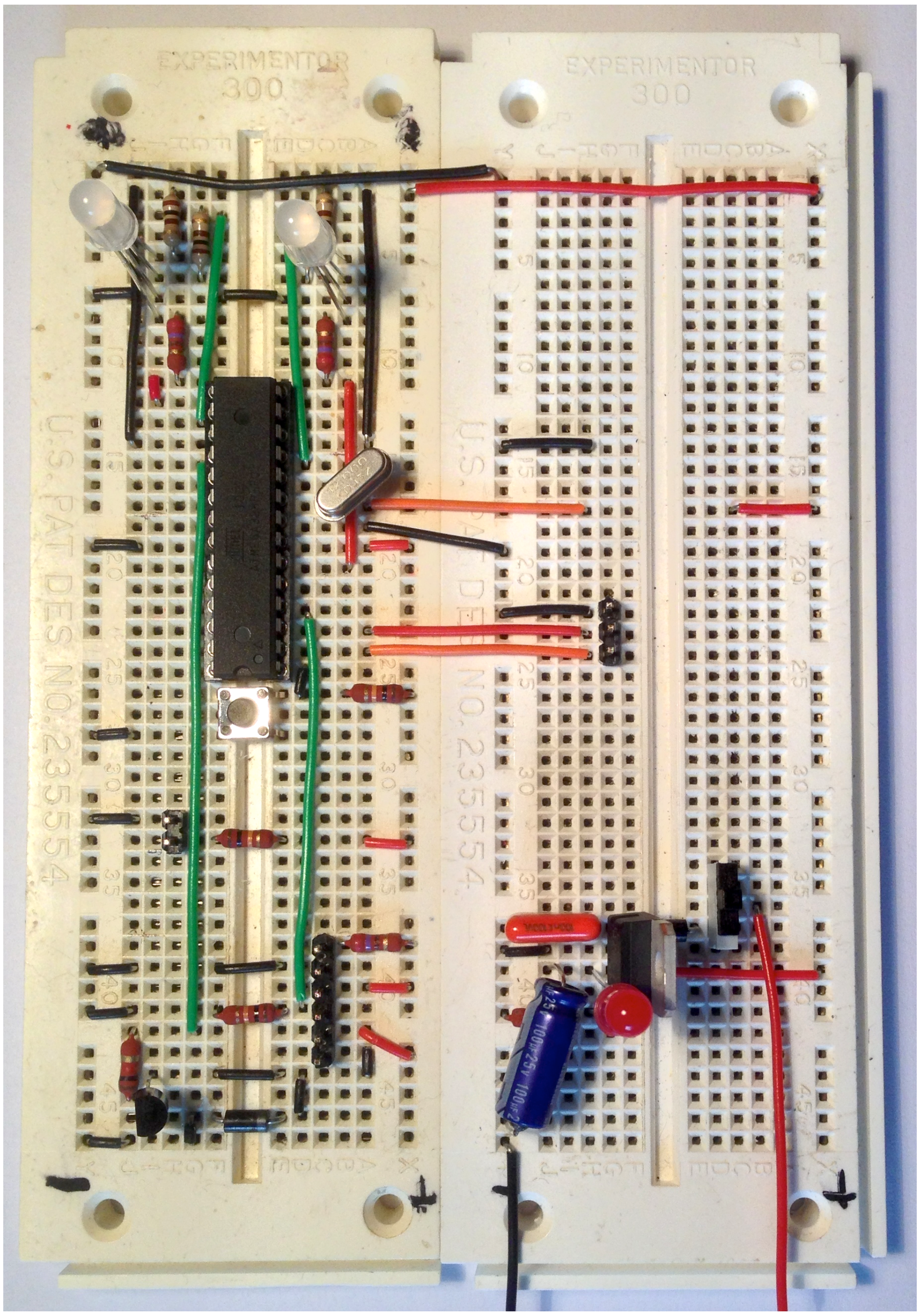


Figure 18: Protoype board created using schematic from Appendix B

D Components List

Component	**Qty.**	**Function**	**Source**
BeagleBone Black	1	Main hardware	Dr. Martinez
BeagleBone Power Supply	1	Main hardware	Dr. Martinez
USB Wi-Fi Dongle	1	Wi-Fi	Dr. Martinez
USB Sound Card	1	Sound	Purchased using budget, cost £7.96
USB Sound Card	1	Sound	Purchased using budget, cost £11.86
4 Port USB Hub	1	Port expansion	Purchased using budget, cost £4.75
2 Port USB Hub	1	Port expansion	Purchased using budget, cost £7.10
Arduino Duemilanove	1	Prototyping	Owned personally
ATMega328-PU	1	Microcontroller	From Arduino Duemilanove
ATMega328-PU	2	Microcontroller	Purchased using budget for spares, cost £7.22
100nF Capacitor	1	Power circuit	UG Electronics Lab
1μF Capacitor	2	Motor circuits	UG Electronics Lab
100μF Capacitor	1	Power circuit	UG Electronics Lab
100Ω Resistor	4	RGB LEDs	Dr. Martinez
270Ω Resistor	7	RGB & IR LEDs, Ears & Power Circuit	Dr. Martinez
1KΩ Resistor	2	Ears	Dr. Martinez
10KΩ Resistor	4	Ears & Switches	Dr. Martinez
RGB LED	2	LEDs	Dr. Martinez
Red LED	1	Power circuit	Dr. Martinez
3.3v Regulator	1	Power circuit	Dr. Martinez
BC547 NPN Transistor	2	Motor circuits	UG Electronics Lab
1N4004 Diode	3	Power & motor circuits	UG Electronics Lab
Momentary Switch	1	AVR Reset	Dr. Martinez

Table 6: Breakdown of components required for project. Total budget use: £38.89

E Serial Communication Protocol Example

```
include <SerialCommand.h>
SerialCommand sCmd;

volatile int leftPinPosition, rightPinPosition;

void setup() {
  Serial.begin(9600);
  sCmd.addCommand("EARPOS", EARPOS);
}

void loop() {
  sCmd.readSerial();
}

/*
EARPOS is called when receiving serial commands of the form
EARPOS [R|L], it returns the position of the ear over the
serial port.
*/
void EARPOS()
{
  char *arg;
  char earPos;

  arg = sCmd.next();
  if (arg != NULL) {
    earPos = *arg;
  }

  switch(earPos){
    case "R":
      Serial.println(rightPinPosition);
      break;
    case "L":
      Serial.println(leftPinPosition);
      break;
  }
}
```

Figure 19: Code excerpt demonstrating use of the SerialCommand [20] serial protocol library.

F Interrupt Routine Example

```
/*
EARMOV controls movement of an individual ear.
The correct interrupt for the ear is enabled, and
variables for correct functioning of the interrupt are set.
*/
void EARMOV(char earSide, int targetPosition){
  int rotaryPin = (targetPosition + 2) % 17;
  switch(earSide){
    case LEFT:
      attachInterrupt(LEFTEAR_INTERRUPT,moveLeftEar,RISING);
      leftEarTargetPosition = rotaryPin;
      seenLeftGap = false;
      leftInterruptTime = millis();
      digitalWrite(LEFTEAR_MOTOR, HIGH);
      break;
      // Repeated case for right ear omitted.
  }
}

/*
Interrupt routine for setting the position of the left ear.
This takes into account the reference gap to ensure that the
pin position remains correct, e.g. 0-17 (inclusive)
*/
void moveLeftEar(){
  leftPinPosition++;
  leftLastInterruptTime = leftInterruptTime;
  leftInterruptTime = millis();
  leftPulseWidth = leftInterruptTime - leftLastInterruptTime;

  if(leftPulseWidth > 500){
    leftPinPosition = 0;
  }

  if(leftPinPosition == leftEarTargetPosition){
    digitalWrite(LEFTEAR_MOTOR, LOW);
    sendEarPosition(LEFT, leftPinPosition);
    attachInterrupt(LEFTEAR_INTERRUPT, leftEarMoved, RISING);
  }
}
```

Figure 20: Code excerpt demonstrating use of interrupts to change ear position.

G Assembled Nabaztag:tag

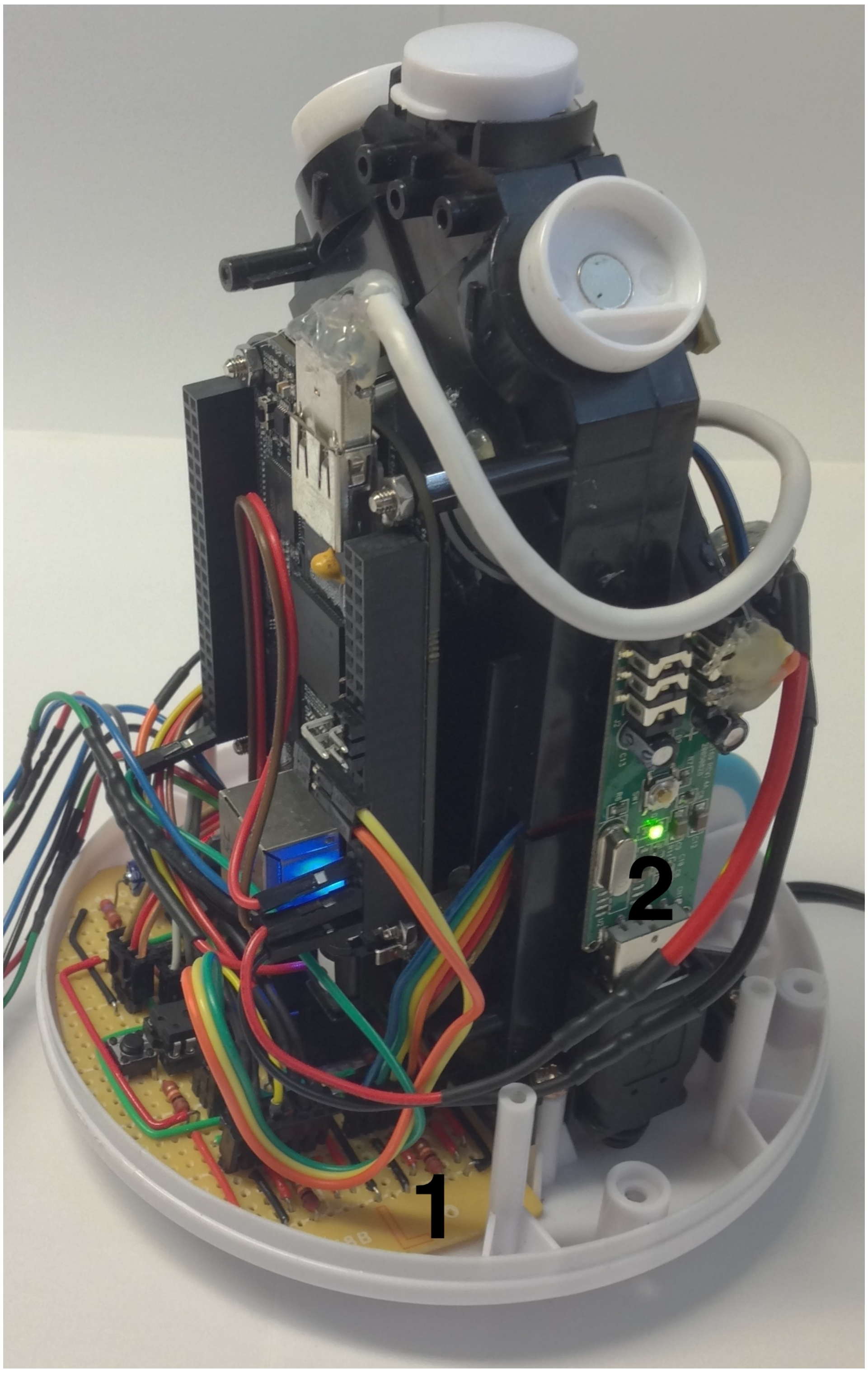


Figure 21: The final assembled form of the Nabaztag:tag replacement hardware. 1) Custom AVR circuit board, 2) USB sound card.

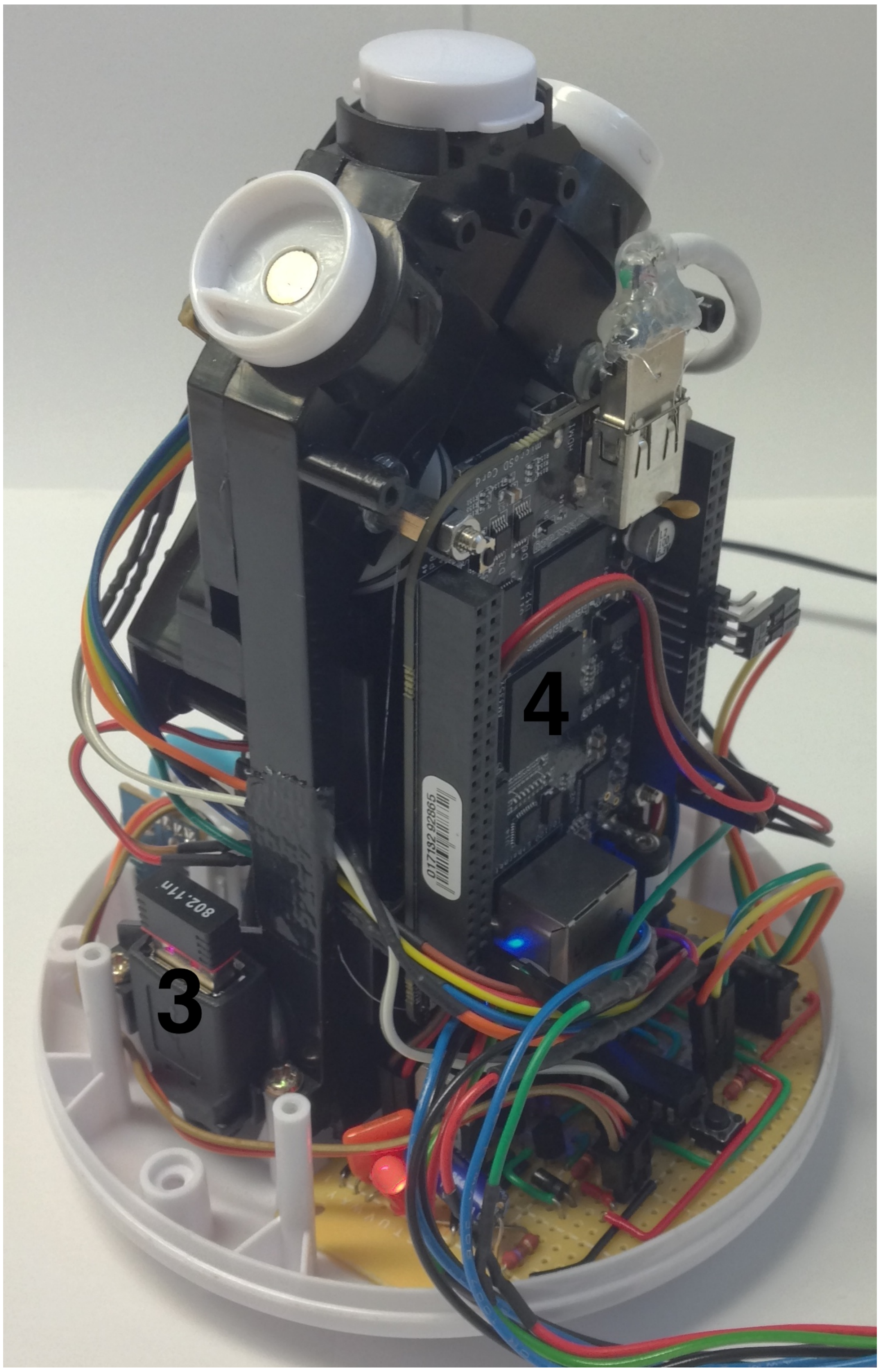
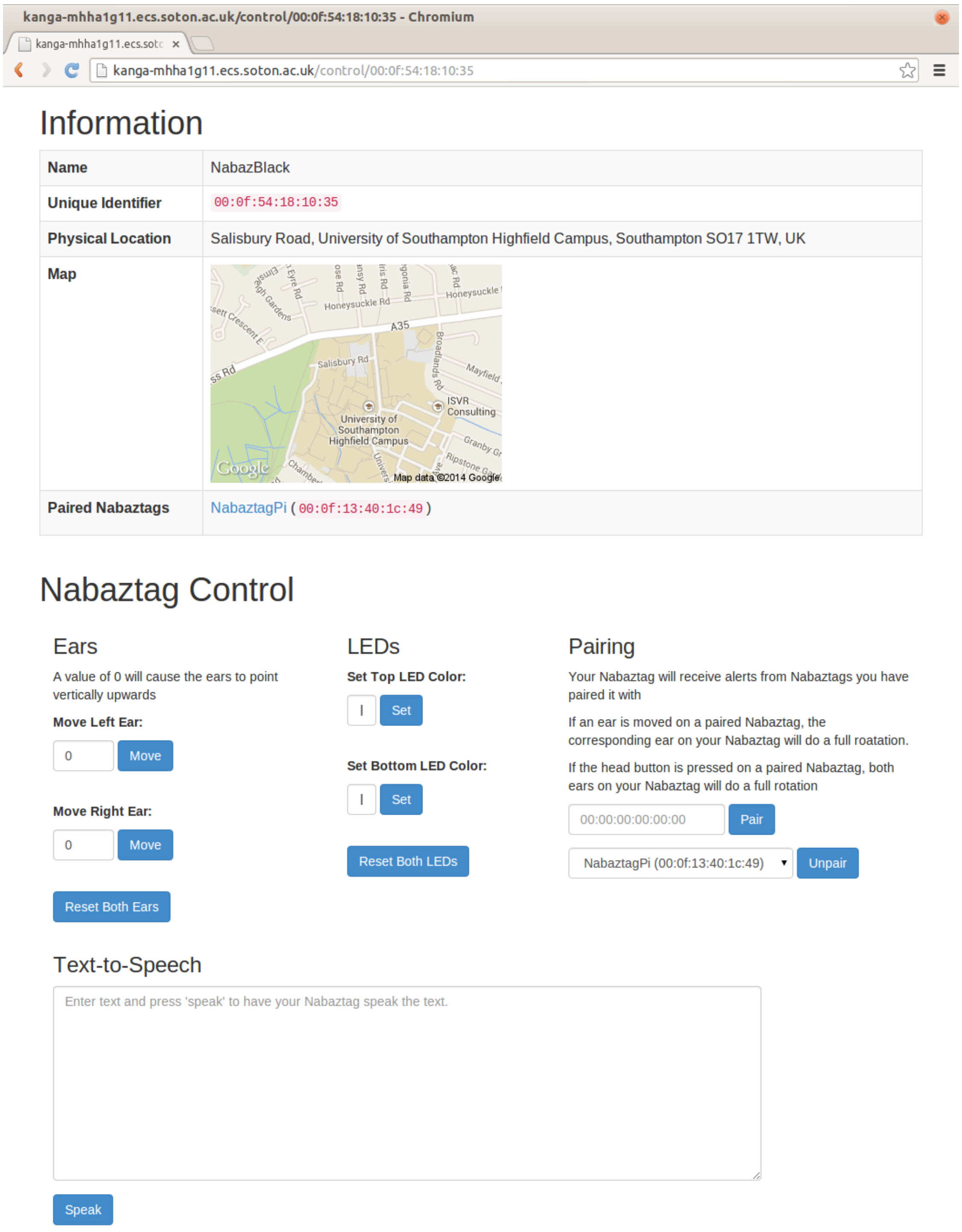


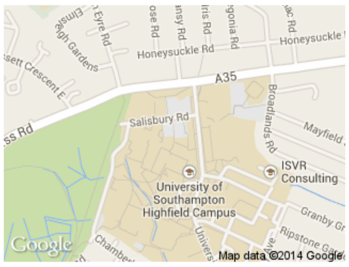
Figure 22: The final assembled form of the Nabaztag:tag replacement hardware. 3) USB Wi-Fi adapter, 4) BeagleBone Black.

H Web Application Control Page


kanga-mhha1g11.ecs.soton.ac.uk/control/00:0f:54:18:10:35 - Chromium

kanga-mhha1g11.ecs.soton.ac.uk/control/00:0f:54:18:10:35

Information

Name	NabazBlack
Unique Identifier	`00:0f:54:18:10:35`
Physical Location	Salisbury Road, University of Southampton Highfield Campus, Southampton SO17 1TW, UK
Map	 Honeysuckle Rd A35 Salisbury Rd Broadlands Rd Mayfield ISVR Consulting University of Southampton Highfield Campus Granby Gr Ripstone Gar Google Map data ©2014 Google
Paired Nabaztags	NabaztagPi (`00:0f:13:40:1c:49`)

Nabaztag Control

Ears

A value of 0 will cause the ears to point vertically upwards

Move Left Ear: 0 Move

Move Right Ear: 0 Move

Reset Both Ears

LEDs

Set Top LED Color: Set

Set Bottom LED Color: Set

Reset Both LEDs

Pairing

Your Nabaztag will receive alerts from Nabaztags you have paired it with

If an ear is moved on a paired Nabaztag, the corresponding ear on your Nabaztag will do a full roatation.

If the head button is pressed on a paired Nabaztag, both ears on your Nabaztag will do a full rotation

00:00:00:00:00:00 Pair

NabaztagPi (00:0f:13:40:1c:49) Unpair

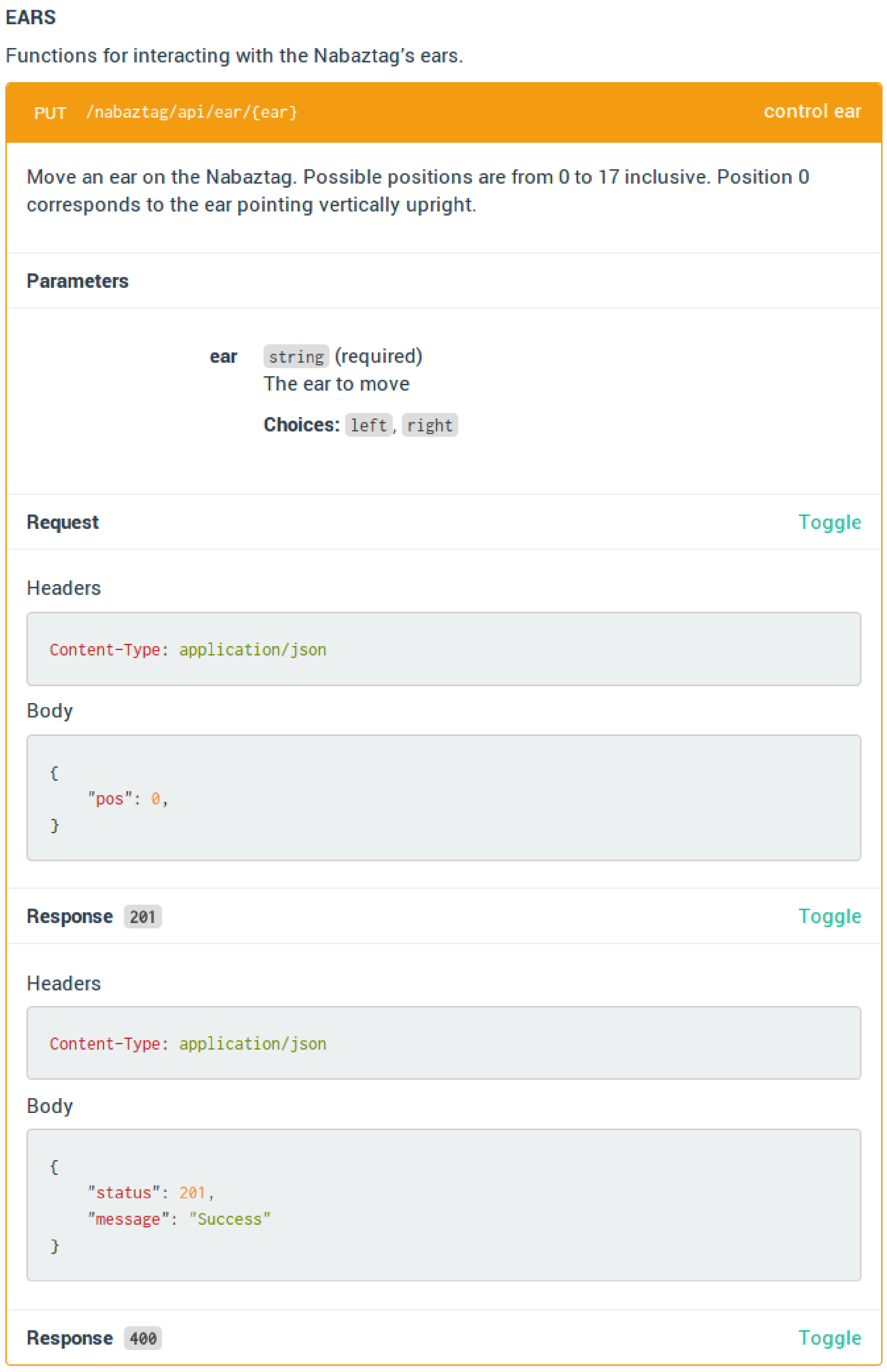
Text-to-Speech

Enter text and press 'speak' to have your Nabaztag speak the text.

Speak

Figure 23: An example of the information and control page for a Nabaztag:tag

I Sample API documentation



EARS

Functions for interacting with the Nabaztag's ears.

PUT `/nabaztag/api/ear/{ear}` control ear

Move an ear on the Nabaztag. Possible positions are from 0 to 17 inclusive. Position 0 corresponds to the ear pointing vertically upright.

Parameters

ear `string` (required)
The ear to move

Choices: `left`, `right`

Request Toggle

Headers

```
Content-Type: application/json
```

Body

```
{
    "pos": 0,
}
```

Response `201` Toggle

Headers

```
Content-Type: application/json
```

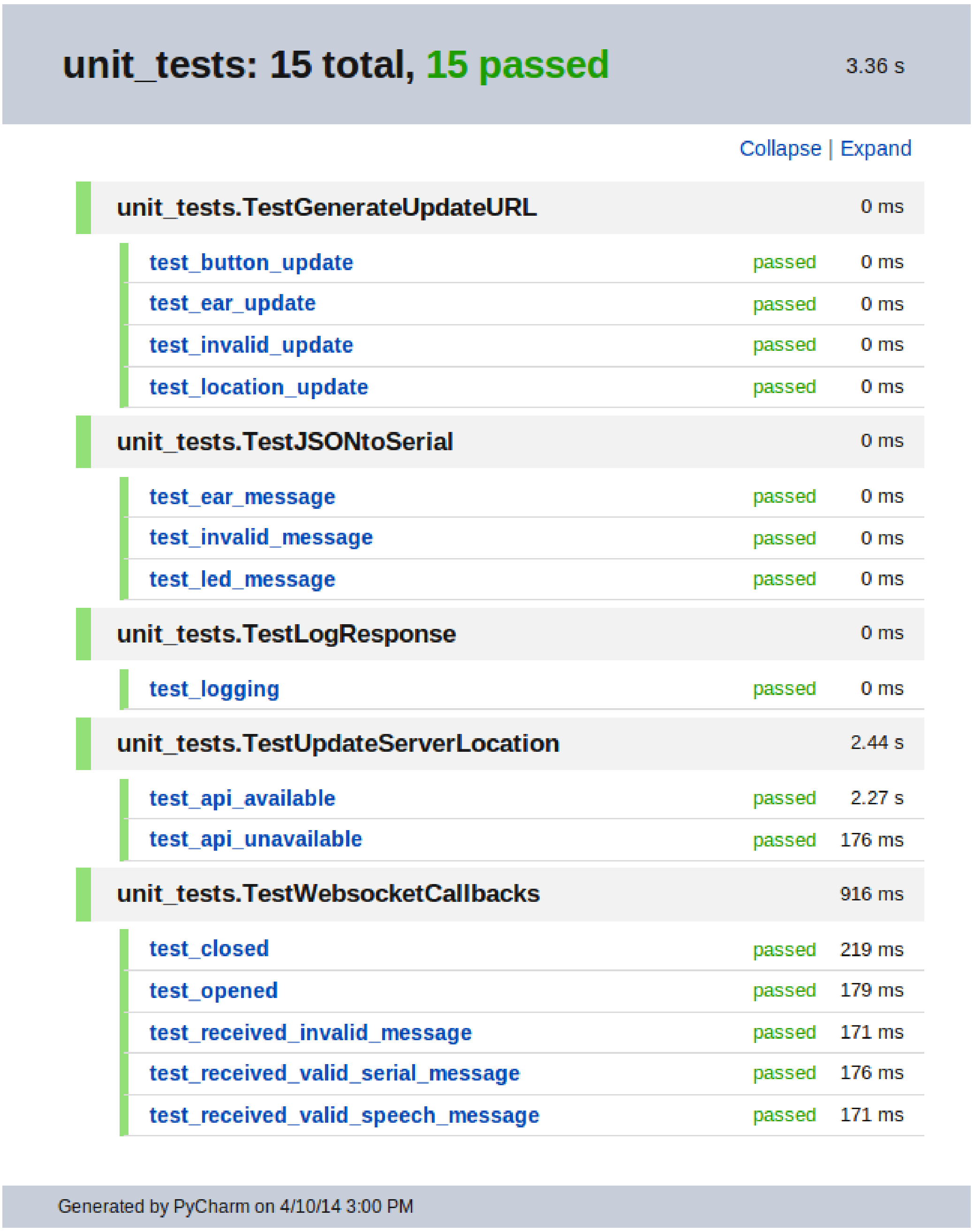
Body

```
{
    "status": 201,
    "message": "Success"
}
```

Response `400` Toggle

Figure 24: An example of the API documentation available from the root of the Nabaztag:tag's API

J Unit Test Results



```
unit_tests: 15 total, 15 passed                                   3.36 s

                                                      Collapse | Expand

unit_tests.TestGenerateUpdateURL                                  0 ms
    test_button_update                              passed        0 ms
    test_ear_update                                 passed        0 ms
    test_invalid_update                             passed        0 ms
    test_location_update                            passed        0 ms
unit_tests.TestJSONtoSerial                                       0 ms
    test_ear_message                                passed        0 ms
    test_invalid_message                            passed        0 ms
    test_led_message                                passed        0 ms
unit_tests.TestLogResponse                                        0 ms
    test_logging                                    passed        0 ms
unit_tests.TestUpdateServerLocation                               2.44 s
    test_api_available                              passed        2.27 s
    test_api_unavailable                            passed        176 ms
unit_tests.TestWebsocketCallbacks                                 916 ms
    test_closed                                     passed        219 ms
    test_opened                                     passed        179 ms
    test_received_invalid_message                   passed        171 ms
    test_received_valid_serial_message              passed        176 ms
    test_received_valid_speech_message              passed        171 ms

Generated by PyCharm on 4/10/14 3:00 PM
```

Figure 25: Unit test results for Nabaztag:tag client application

K Initial Gantt Charts

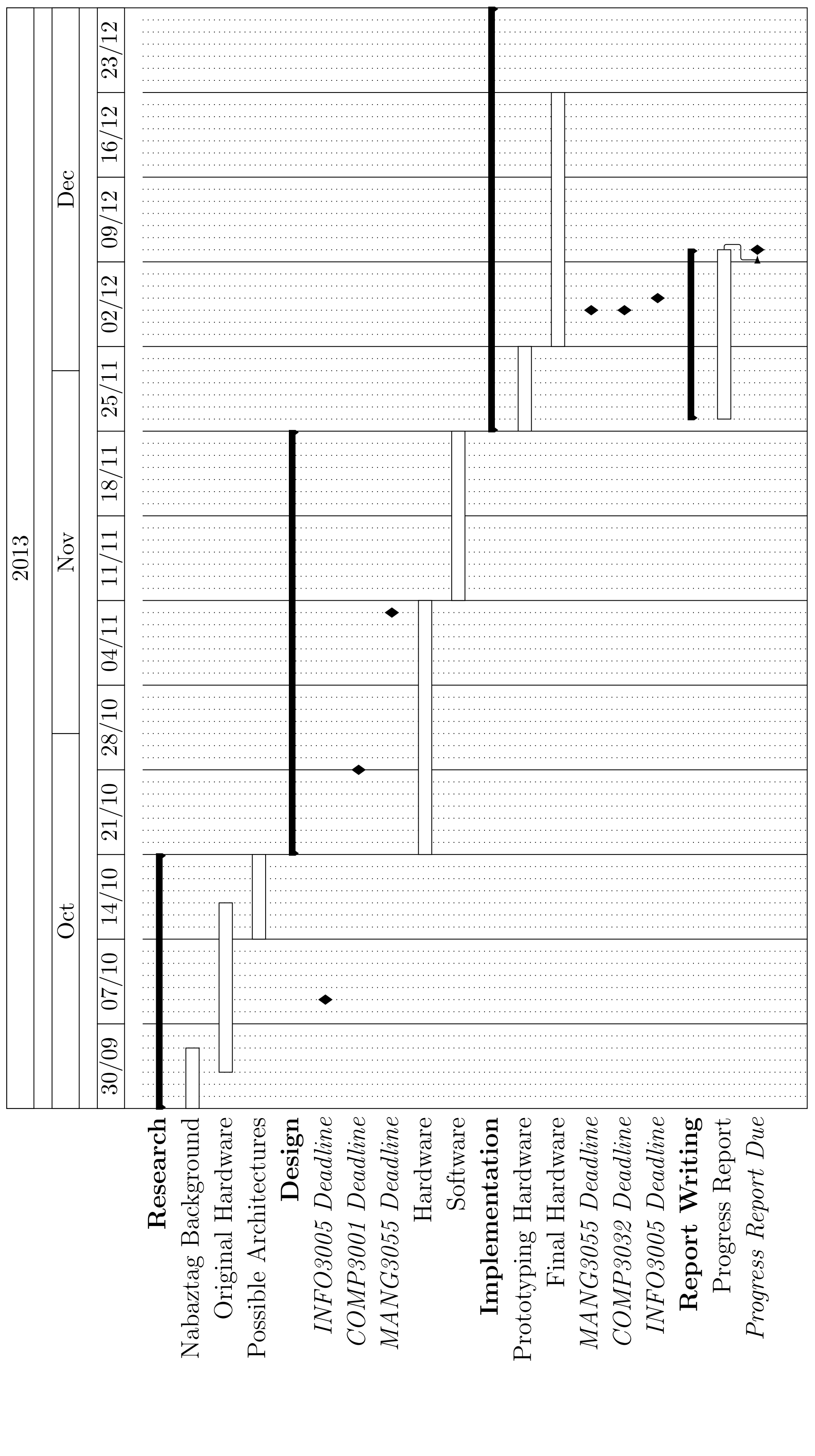


Figure 26: Original Semester 1 Gantt chart

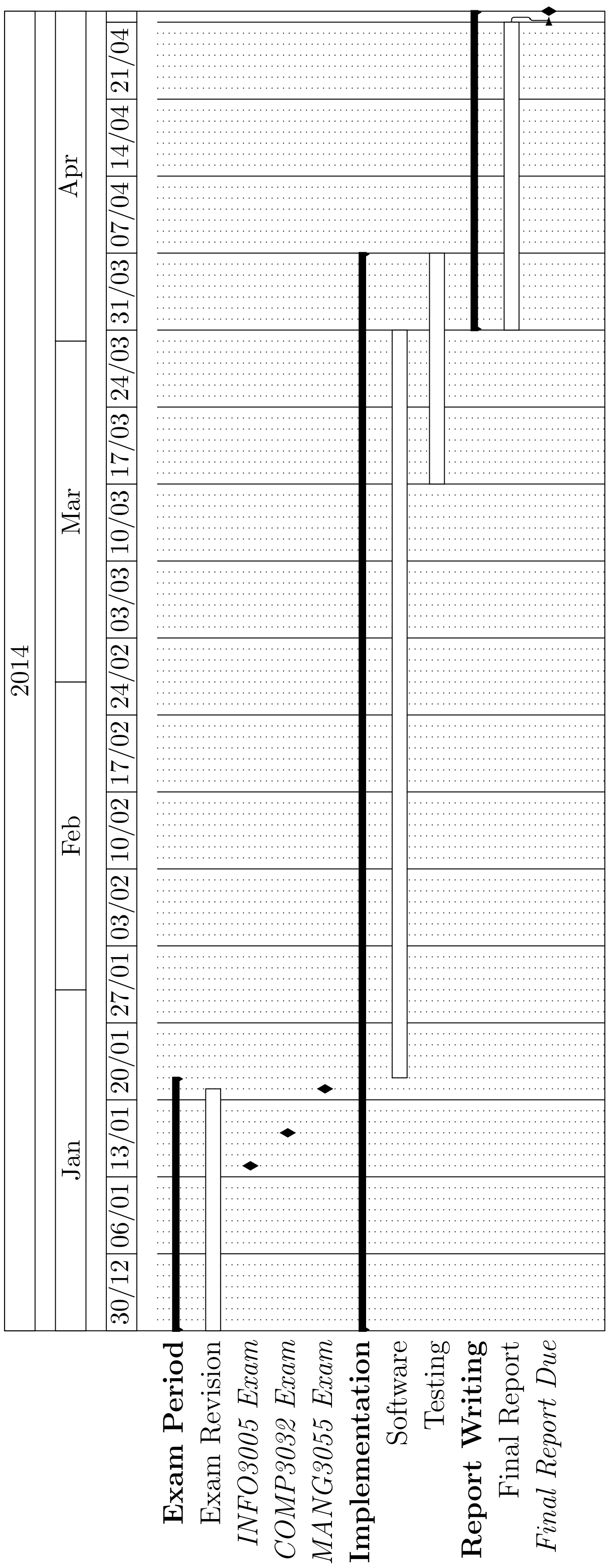


Figure 27: Original Semester 2 Gantt chart

L Design Archive Listing

```
avr_setup
├── boards.txt
└── nabaztag_avr.ino

beaglebone_setup
├── BB-UART1.dts
├── LED_blink_test.py
├── LED_fade_test.py
├── README.md
└── dtc_patcher.sh

nabaztagclient
├── beaglebone
│   ├── __init__.py
│   ├── nabaztag_client.py
│   ├── nabaztag_serial.py
│   ├── nabaztag_update.py
│   ├── nabaztag_upstart.conf
│   └── nabaztag_websocket.py
├── nabaztagconfig.yaml
├── pi
│   ├── pi_button.py
│   ├── pi_client.py
│   ├── pi_update.py
│   ├── pi_upstart.conf
│   └── pi_websocket.py
├── restapi
│   ├── documentation
│   │   ├── nabaztagapi_blueprint.md
│   │   └── nabaztatgapi_static.html
│   ├── nabaztagapi_geolocate.py
│   ├── nabaztagapi_nginx.conf
│   ├── nabaztagapi_server.py
│   ├── nabaztagapi_upstart.conf
│   └── nabaztagapi_weather.py
└── unit_tests.py

nabaztagserver
├── deployment
│   ├── README.md
│   ├── django_uwsgi.ini
│   ├── nginx.conf
│   ├── uwsgi.conf
│   ├── uwsgi_params
│   └── websocket_uwsgi.ini
├── manage.py
├── nabaztag
│   ├── __init__.py
│   ├── admin.py
│   ├── forms.py
│   ├── models.py
│   ├── templates
│   │   ├── control.html
│   │   └── index.html
│   ├── urls.py
│   └── views.py
└── nabaztagserver
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── wsgi_django.py
    └── wsgi_websocket.py
```

Figure 28: Directory listing for project files included in Design Archive

M Original Project Brief

This was the initial brief submitted for the project, it is included for reference:

The Nabaztag, developed by Violet and first released in 2005, was an attempt at introducing pervasive computing to the masses. A wi-fi enabled smart device, shaped to look like a rabbit, with a variety of functions. Nabaztag could convey information through 5 multicoloured LEDs, motorised ears, and built-in speaker, and could receive input from its environment using a push button, a microphone, an RFID reader and by manual adjustment of the position of its ears.

Nabaztag relied heavily on communication with servers operated by Violet, which stored the majority of its configuration information, and provided the only way for a Nabaztag owner to alter the behaviour of their device. In December 2006, when large numbers of purchased Nabaztag devices were activated simultaneously, Violet's infrastructure was unable to cope with the demand, and there were service disruptions for both new and existing Nabaztag owners. In October 2009, after a long period of technical troubles, Violet declared bankruptcy and was purchased by an Australian software house, Mindscape. In July 2011, Mindscape stopped maintaining the Nabaztag, and released the previously closed source code to the public. Efforts to understand the Nabaztag architecture have been hindered by the poor documentation, and by existing documentation being in French.

Several open source projects have attempted to create new web interfaces for the Nabaztag, but they suffer from the requirement that they must interact with the original software running on the proprietary, single-board computer that is at the heart of the Nabaztag. While compatibility with the client software on the Nabaztag is necessary to allow existing Nabaztag owners to return functionality to their devices without any complex modifications, it limits the potential to experiment with new and improved technologies which might be used to address the original issues with the Nabaztag architecture.

In this project, I intend to bring a single Nabaztag back to life. I wish to as a minimum restore its previous functionality, and potentially add new functionality not present on the original device. To achieve this, broadly I will:

- Replace the single-board computer in the Nabaztag with a modern open hardware device such as the BeagleBone Black, whilst maintaining all the original input and output hardware of the Nabaztag.
- Develop a client application, able to run on the open hardware device and control all functionality of the Nabaztag,.
- Develop a server application, able to communicate with the client application using a push notification architecture.
- Develop a web interface to the server application, allowing users to interact with their Nabaztag by pushing updates and configuration changes from server to client.