

ABSTRACT

Embedded electronics are increasingly becoming part of everyday life, with almost all home appliances now utilising embedded microcontrollers and integrated circuits. Increasingly these embedded electronics also feature network controllers, leading to the proliferation of embedded networks.

This paper documents the creation of a system which makes use of both embedded electronics and embedded networks with an aim to connect medical professionals with their patients in a more seamless fashion.

ACKNOWLEDGMENTS

My good friend, Adam Boulton, who was always on-hand to offer advice and support.

Dr Christopher W. Hodgkins and Nurse Emma Quinn, who both helped me immeasurably during the planning stages of the project, improving my understanding of the existing systems and protocols already in use within hospitals.

Finally, my mother and father, without their endless support and encouragement I would still be writing this report. To them, I owe everything.

TABLE OF CONTENTS

Abstract	1
Acknowledgments.....	2
1. Introduction	6
1.1. Project Code Name	6
1.2. Project Summary	6
1.3. Aims and objectives	7
1.4. Justification	7
1.5. Ethical, Moral and Legal Obligations.....	8
1.6. Proposed Solution	8
1.7. Security Requirements	9
1.8. Deliverables.....	9
2. Report Overview	10
2.1. Audience.....	10
2.2. Definitions	10
3. Investigation.....	11
3.1. Existing Systems	11
3.1.1. The Sensium™ (and Sensium™ ‘Life Pebble’)	12
3.2. Investigation into Suitable Hardware	14
3.2.1. Initial Research.....	14
3.2.2. Further Investigation into Hardware found.....	18
3.2.3. Investigation Conclusion	21
3.2.4. Additional Hardware.....	22
3.3. Software Review.....	24
3.3.1. Development Language(s)	24
3.3.2. Persistent Storage	25
3.3.3. Web Server.....	25
3.3.4. Summary	26
4. Design and development	27
4.1. Design Overview.....	27
4.1.1. High Level Workflow Diagram	27

4.1.2.	High Level UML Sequence Diagram	28
4.1.3.	Design Structure.....	28
4.2.	Design Process.....	29
4.2.1.	Development Methodology.....	29
4.2.2.	Testing Methodology	30
4.2.3.	Coding Conventions	30
4.2.4.	Documentation	31
4.2.5.	Source Code and Project Website	32
4.3.	Hardware Design And Implementation	33
4.3.1.	Overview	33
4.3.2.	Implementing the HRMI	34
4.3.3.	Implementing the Xbee Wireless Shield.....	34
4.3.4.	Implementing the TMP36 Temperature Sensor	35
4.3.5.	Testing and Validation	35
4.3.6.	Issues with Implementation	35
4.4.	Arduino Software Design and Implementation	36
4.4.1.	Designing the Serial Communications Protocol.....	36
4.5.	Database Design and Implementation.....	38
4.5.1.	Overview	38
4.5.2.	Entities, Attributes And Entity Relationships.....	38
4.5.3.	Implementation	39
4.6.	'Back-End' System Design and Implementation	40
4.6.1.	Serial Communications	40
4.6.2.	CollectorEntry Object.....	41
4.6.3.	Development of THE Communications API	42
4.6.4.	Development of Application	44
4.7.	Web Application Design	45
4.7.1.	Overview	45
4.7.2.	Use Cases	45
4.7.3.	Why use a web-based GUI?	46
4.7.4.	Presenting Collected Data.....	47
5.	Evaluation	49

5.1.	Critical Evaluation of the System	49
5.1.1.	Software	49
5.1.2.	Hardware	50
5.1.3.	Conclusion.....	50
5.2.	Critical Evaluation of the Project.....	51
5.3.	Possible Further Enhancements.....	52
5.3.1.	Hardware	52
5.3.2.	Software.....	53
6.	Glossary.....	55
7.	Bibliography	55
8.	References	56
9.	Further Reading	56
10.	Appendices.....	58
10.1.	Appendix A – Source Code	58
10.1.1.	Source Code for Arduino.....	58
10.1.2.	Source Code for API.....	59
10.1.3.	Source Code for Back End Application	60
10.1.4.	Source Code for creation of Database	60
10.1.5.	Source Code for Web App.....	61
10.2.	Appendix B – Coding Conventions.....	61
	Java.....	61
10.3.	Appendix D – Table of Figures	61
10.4.	API Reference Guide.....	62

1. INTRODUCTION

1.1. PROJECT CODE NAME

Throughout this report both the hardware and software systems are referred to under an umbrella name of 'Project Zed'.

1.2. PROJECT SUMMARY

The use of electronic systems to monitor a patient's vital statistics within healthcare institutions is not a new concept. For many years, ECG machines have been in use within operating theatres and high dependency wards or units. These units play a vital role in the wellbeing of a patient, and allow doctors and nurses valuable insight into a patient's overall condition during the course of their stay.

However, the sizeable cost of these ECG machines, coupled with the logistical considerations of employing them on a larger scale, has stopped them from seeing widespread use in all hospital wards.

I benefit from having a family member who is a trained nurse, currently in the employ of a local hospital, and a close family friend, Dr Christopher W. Hodgkins, MD, who is in his final year of residency at Miami University Hospital to become an orthopaedic surgeon.

As such, I have been lucky enough to be able to discuss the problem with both of them at length. After explaining what I had envisioned, both were in agreement that being able to monitor a patient's vital statistics 'around-the-clock' would be of great use.

The nurse commented that she would sometimes have to wake a patient, if their condition was serious enough, to take heart rate and body temperature as part of a monitoring routine. This would unavoidably lead to the patient having their already disjointed sleeping pattern disturbed even further.

Dr Hodgkins was particularly helpful, and remarked that "any attempt at progressing the ability to monitor a patient's vital statistics is a worthwhile endeavour".

Thanks in part to their valuable insight, I see a definite need for a smaller, easier to employ, and most importantly cost effective solution for monitoring and gathering patient's vital statistics.

1.3. AIMS AND OBJECTIVES

In the context of this project, a patient's vital statistics will refer to their body temperature and heartbeat. Monitoring a patient's heartbeat can help medical professionals diagnose and track medical conditions, and provide valuable insight into a patient's overall health condition. It also gives the clearest indication of conditions such as Tachycardia and Bradycardia.

Recording a patient's body temperature helps medical professionals track and monitor a patient's fever in the case of an infection. Fever can be a symptom of many serious conditions, and indeed different types (or patterns) of fever exist that can point to a narrower range of conditions. The ability to view such data would be of great use when attempting to make a diagnosis.

The purpose of this project is to devise a prototype system that can effectively collect, manage, and collate these statistics at a centralised location (such as a nurses station), and allow them to be displayed in a useful manner, with a minimum amount of fuss from the patient's perspective. Hardware attached to the patient will transmit their statistics back to a collection point wirelessly, and then a software based system will process the collected data and make it available for viewing by medical professionals.

1.4. JUSTIFICATION

Currently, the costs and logistics of having every patient connected to an ECG machine far outweigh the possible benefits it would bring. By devising a system at a fraction of the cost, that can be implemented easily and negate the need for the patient to be physically attached to a cumbersome ECG machine, would potentially allow for such a system to be employed on a much wider scale.

There are of course other 'knock-on' benefits brought about by such a system. For example, having a patient's statistics collected automatically will free up a portion of a nurses workday allowing them to provide a better level of care to those who need it.

Instead of routinely disturbing resting patients unnecessarily, a nurse or doctor can instead quickly have access to a patient's vital statistics whenever is convenient from a centralised location.

1.5. ETHICAL, MORAL AND LEGAL OBLIGATIONS

It is crucial in the design of any system that is geared towards automated information assimilation that great care is taken to ensure that certain safeguards against information misuse are in place.

However, due to this system being a prototype, only limited attention will be paid to the security of the system. If the system were to be used in a production environment, then particular attention would have to be paid to ensuring that any information stored about individuals complied with the data privacy laws of the country in which the information is both stored and accessed.

Also, information retrieved via this system should never be considered as bona fide, or absolutely correct, since to do so would mean that the entire system is guaranteed to be without defect.

Information should be treated solely as an indication, upon which further manual investigation can be instigated. Any conclusions should be drawn solely from the result of that manual investigation.

1.6. PROPOSED SOLUTION

A hardware and software system that allows for patient's vital statistics to be collected, and transmitted wirelessly back to a base station which would then store and collate those statistics. A web application would then allow this data to be displayed to an end user (in practice the ward nurse).

The following is a breakdown of the minimum features required for a functional system:

Feature	Description
Data Collection	A patient's vital statistics (their heartbeat and body temperature) will be collected by an independent hardware system.
Data Transmission	All data collected by the aforementioned hardware will be transmitted wirelessly to a 'base station' for processing.
Data Processing	Data transmitted wirelessly by the aforementioned should be collected, processed (including validation), and stored appropriately.
Web Based GUI	An interface which is used to display the collected and processed results in a useful way.

1.7. SECURITY REQUIREMENTS

Due to this system being a prototype, stringent security measures will not be required.

If the prototype system were to be further developed so that it could be put to use in a production environment, independent investigations into the security implications of such a system would have to be made.

1.8. DELIVERABLES

There are four deliverables for this project:

1. A working hardware prototype that allows for the collection of heart rate and temperature data from a user
2. A working software system that stores and processes data from the hardware system
3. A web based application that displays the collected statistics
4. This report

2. REPORT OVERVIEW

2.1. AUDIENCE

The document is primarily aimed at technically literate users, both developers and consumers. However, with the exception of the design and implementation sections, the report should be easily understood by anybody with an interest in a related field.

2.2. DEFINITIONS

Open Source Software – Generally speaking, software that is freely available (though this is largely dependent on the specific open source license it has been released under) and which a user is free to view and modify (again, dependent on the specific license) the original source code. Most open source software allows for the modified code to be redistributed as well.

IDE – This acronym, used within the context of this report, refers to an integrated development environment. An integrated development environment provides a programmer or software developer with a flexible workspace in which multiple development tools are often combined. An IDE usually consists of a source code editor, a compiler and an interpreter for the language being used, build/compile tools, and finally a debugger. Other facilities such as memory profiling have also started to become more prevalent.

Multi-platform - refers to computer software that is available for use or compilation on multiple computer platforms.

3. INVESTIGATION

3.1. EXISTING SYSTEMS

Despite many hours spent searching for existing systems, and indeed companies, that are targeted at the same problem domain I hope to address, I have only ever been able to find one – Toumaz Technology Limited, producers of the Sensium™ range of products.

There are solutions provided by the likes of Philips with their TraceMasterVue software that provide management of ECG results across an enterprise or institution. However the ECG results are gathered using their traditional ECG machines, which are extremely bulky as can be seen with their PageWriter TC50 model, shown in Figure 2.1.



Figure 3.1 - PageWriter TC50 ECG Machine
(Courtesy of Philips Inc.)

Philips also produces a much more portable cardiograph machine (Fig. 2.2), the DigiTrak XT Holter Recorder. This device is designed to be worn by the patient for a number of days, at which time the results are then retrieved via a wired docking station and viewed using the accompanying Holter Monitoring Software.

The results can then be further exported to a larger TraceMasterVue system so that the results can be viewed throughout an organisation.

However, the main drawback of both of the solutions provided by Philips is the fact that they only monitor a patient's heart condition. Also, whilst the bulky PageWriter device supports wireless transmission of ECG data, the portable DigiTrak XT system does not – meaning neither caters for the requirements I aim to fulfil.



Figure 3.2 - DigiTrak XT Device
(Courtesy of Philips Inc.)

3.1.1. THE SENSIMUM™ (AND SENSIMUM™ 'LIFE PEBBLE')

GENERAL SUMMARY

The Sensium™ is said to offer an “ultra-low power sensor interface and transceiver platform for a wide range of applications in healthcare and lifestyle management” (Toumaz Technology, 2009). The two products the company offers are both based around the Sensium TZ1030 ‘system-on-a-chip’ that provides an interface to which external sensors (heartrate, blood glucose etc.) can be attached.

The TZ1030 carries out some preliminary processing of the captured analog measurements with the aid of the programmable ‘Mentor eWarp 8051 Microprocessor’, and then transmits the information wirelessly to a base station over radio frequencies.

ADVANTAGES

The main advantage of the TZ1030 is its extremely low power requirements and its small size. According to the manufacturers specifications, it can last for up to five days on the remarkably small ‘675’ battery (commonly used in hearing aids). It is also small enough that the circuitry can be attached directly to the body using an everyday medical plaster.

Primarily the device is intended for third-party integration, that is, the TZ1030 provides the basic hardware

functionality that is then extended by a third party developer or systems integrator. So for example a sportswear company may decide to use the TZ1030 to collect heart rate data, or a pharmaceutical technology company may decide to use the TZ1030 to power a device that monitors blood sugar levels. The third-party are then subsequently also responsible for providing the whole software system that will then process and store any data collected by the TZ1030.



Figure 3.3 – Sensium™ 'Life Pebble'.
(Courtesy of Toumaz Tech. Ltd.)

However, Toumaz do produce the Sensium TZ203082, or 'Life Pebble' (Fig. 3.3), which incorporates their TZ1030 into a production quality device, complete with sensors to measure heart rate, skin temperature, and movement.

The 'Life Pebble' is worn by the user, and has two electrodes that protrude from the main body of the device that then attach to the users chest to provide ECG functionality (Fig. 3.4). Data collected from these sensors is then transmitted via the TZ1030's low-power RF transmitter back to a USB device that is plugged into a nearby computer.

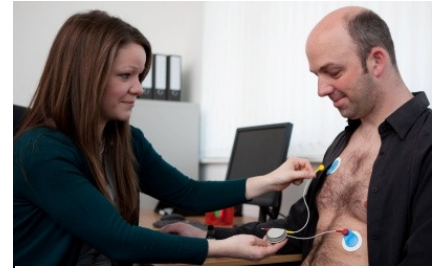


Figure 3.4 - 'Life Pebble' being attached.
(Courtesy of Toumaz Tech. Ltd.)

As a software system isn't provided to interpret, parse and display the collected results, an API is provided with which an end-user would have to devise a new system for doing so, or indeed integrate the 'Life Pebble' into an existing system.

DISADVANTAGES

There are two significant disadvantages to the Sensium TZ1030, and subsequently the 'Life Pebble': the poor signal range of its wireless transmitter, and the lack of a complete system with which data can be viewed.

The poor wireless range of both the TZ1030 (10 meters) and the 'Life Pebble' ("~5m") (Toumaz Technology, 2009) mean that both devices have extremely limited operating conditions. When you factor in radio frequency noise, poor signal attenuation will doubtless be a real problem as a patient moves around a ward.

Whilst it is certainly beneficial that an API is provided to allows a developer to integrate the 'Life Pebble' into an existing system, or devise a completely new one, the fact that there isn't some basic functionality provided to allow for results to be viewed sets it apart from the system that I envisage developing.

3.2. INVESTIGATION INTO SUITABLE HARDWARE

3.2.1. INITIAL RESEARCH

Deciding upon what hardware system would fit the project requirements was the first decision to be made. Because of the complexity of the system envisaged, it will almost certainly require the use of a microcontroller, or indeed a hardware platform that makes use of a microcontroller.

To begin, a list of loose requirements for the hardware was drawn up to aid with my initial investigation. Greater depth is added later on in the investigation.

These initial requirements are as follows:

- A system or controller that would allow me to apply my own program logic, via a higher level language than machine code.
- An easy to implement, well documented process to enable me to interface with a PC.
- A relatively easy to implement, or rather well documented, method of interfacing with 'off-the-shelf' hardware, such as a wireless controller.
- Something that is of a small size, so that it could feasibly be produced into a production model that could be worn by a patient.
- It has to be low-cost; both to satisfy my budget and to reduce costs should the system ever be manufactured on a larger scale.
- A system with a proven track record of use within embedded systems.

Working from these loose requirements, I began my investigation. Through some initial research, and indeed past experience, I knew already of a small number of systems that would possibly satisfy these requirements.

MICROCHIP'S PIC

The PIC microcontroller, produced by the Microchip corporation, has a long and illustrious history. First introduced in 1975, it has seen widespread use in both industry and the hobbyist market, and is still widely used today in everything from household electrical items right through to industrial machinery.

The PIC can be programmed using a number of methods, and extensive documentation along with example projects exist for the beginner to study. Whilst its low cost (typically around \$1-3 per chip) has been instrumental in its wide adoption, it does somewhat mask the 'true cost' of using it as part of a larger system.

For example, if PIC were to be used, a PIC programmer will also have to be purchased or constructed, and an accompanying circuit board (complete with appropriate components) would have to be designed and made for use with the PIC itself.

Whilst it's certainly true that a development circuit board similar to what the Arduino provides can be purchased or constructed, the learning curve involved with this low-level of electronics could possibly be unfeasible for the timescale involved with this project. Moreover, the prototype boards that can be purchased tend to be focused on just one function (for example, a micro-webserver); rather than the modular approach that the Arduino takes with its easily detachable 'shields'.

Designing a program to run on a PIC can be accomplished in a number of ways. The language in which a program can be written depends entirely on the specific PIC chip being used, though the C programming language is the most commonly supported.

Alternatively, the program can be coded in assembler using the PICs relatively simple instruction set (consisting of 35 instructions). Finally, MPLab, a piece of software produced by Microchip, can be used to design the program visually.

Once your code has been written and compiled, the PIC programmer, a hardware device that allows you to interface the chip with your computer, is used to write the code to the on-board memory.

Using a PIC as the 'heart' of the system would offer advantages. Such a system would be custom made to my requirements from the 'ground up' – meaning there would be no unnecessary expenditure and fine grained control over the systems functionality would be achievable, and no doubt a better understanding of the underlying electronics would be gained as a result.

The PIC also benefits from a large and vibrant community. Besides the numerous community-run message boards and mailing lists, Microchip also provide their own thriving message board with nearly 40,000 registered members. At the time of writing there were 51 members and 3442 guests viewing the message boards, illustrating just how popular the Microchip products are.

GUMSTIX

The GumStix platform briefly held my attention. The two products manufactured under the Gumstix name are Overo and Verdex – which are stated as giving “Laptop-like performance and desktop [computer] flexibility in extremely small form factors”. [? – www.gumstix.com]

The most recently released product, the Overo, offers both advanced functionality and a wide range of capabilities (for example: Wireless(802.11G), large amounts of RAM, extremely fast processor). Whilst technically brilliant, the Overo is far too expensive for me to even consider, and if mass-production were to be considered for a large scale deployment it would make costs extremely prohibitive.

ARDUINO

The Arduino platform is a rapid prototyping tool for use with designing embedded systems. It consists of a single-board circuit based around the Atmel AVR ATmega series of microcontrollers, and comes complete with embedded I/O support.

The Arduino hardware platform has seen many design iterations, though for the purposes of this investigation I will be looking exclusively at one of the most recent incarnations, the Duemilanove¹ (Fig 3.5).

¹ Pronunciation: http://arduino.cc/en/uploads/Main/Arduino_Duemilanove.wav

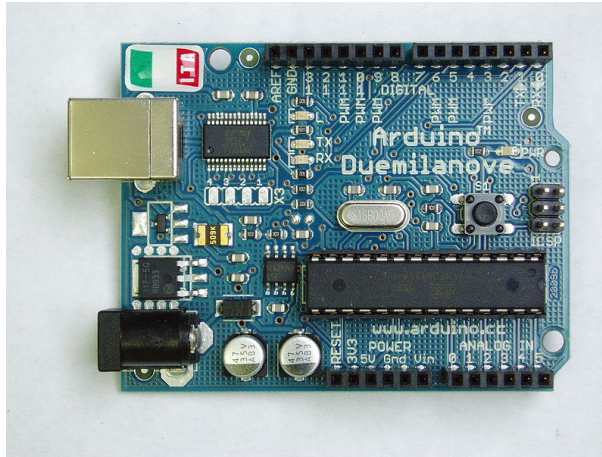


Figure 3.5 – Arduino Duemilanove. (Public Domain).

Interacting with the Duemilanove is made extremely easy by the inclusion of a USB interface, which when connected to a computer is registered as a USB-to-serial adapter. Code can then be uploaded over this serial connection, and indeed further interactions can be made with the Arduino as if it were any other serial device.

Programs are written for the Arduino using the Wiring programming language, which is largely based on C++ but with some simplifications having been made. Designing programs is made easy by an accompanying open-source integrated development environment (IDE). Once a program has been designed in the Wiring language, one click will compile and upload the code directly to the Arduino (or rather the ATmega microcontroller) over its USB interface.

Perhaps most impressive when considering the Arduino hardware platform is the extensibility offered; a large range of 'shields' can be purchased that extend the functionality of the system to offer wireless and wired communication, GPRS communication, touch sensitive pads etcetera.

Most importantly when undertaking a project with this limited timescale, copious and thorough documentation exists for every aspect of the Arduino's use. Where the PIC microcontroller may have in-depth technical documents and countless spurious message board discussions, the Arduino has hundreds of practical step-by-step guides, detailed discussions, and user based reports on various projects of all different guises.

The Arduino is relatively inexpensive, with the standalone Duemilanove costing around £20. It's important to remember that this cost includes everything needed to begin writing the first program to the on-board ATmega chip, and also provides numerous analog and digital I/O ports to use immediately.

Also, this cost need not be applicable if the system were to be put into production. The Arduino also comes in a 'mini' form factor – whereby the size of the circuitry is vastly reduced, and more importantly the price is more than halved with it costing a little over £10.

Furthermore, if production were to be done on a large enough scale, it could well be economical to design custom circuitry (the design of the Arduino circuitry is open-source and freely available), and simply utilise the Arduino bootloader on the ATmega chip.

3.2.2. FURTHER INVESTIGATION INTO HARDWARE FOUND

Despite considerable research, I struggled to find any other platforms that could fit my loose set of requirements. As such, the only real contenders were the Arduino and PIC .

Both will be examined more closely to try and establish which of them will better suit the projects requirements. To do so, a much clearer set of assesment criteria was established:

- Ease of Use
It has to be easy to implement, and indeed understand. Merely programming the device/system need not be a complicated process.
- Well documented
There needs to be thorough documentation, whether it be produced by the manufacturer or 'the community', on-hand and freely accessible.
- Well supported
Comprehensive support needs to be available, again, whether it be from the manufacturer or from a good community (online message boards, mailing lists etc) based around the product.
- Low Cost

Whilst not of paramount importance, cost should be factored into any decision. Costs should not be prohibitive if the system was to actually go into mass production, and initial outlay costs for the prototype system should not be exorbitant.

- Extensible

Methods whereby the functionality of the hardware can be extended to cater for the further requirements of my system, such as wireless communication, must exist. These 'methods' must also fit the 'ease of use' requirement.

With clear requirements now outlined a closer look was taken at both the PIC based solution and the Arduino based solution.

EASE OF USE

It is a little unfair to compare Microchip's PIC microcontroller with the Arduino on an 'ease of use' basis, as the Arduino provides a complete platform upon which embedded systems can be built.

However unfair – it is necessary. In order to program the PIC microcontroller, a 'PIC programmer' would first have to be built or purchased. This would provide an interface (USB is available) through which the connected PC can interact with the microcontroller, and code can be uploaded.

Then, once the chip has been successfully programmed, a custom circuit board would need to be designed and built, requiring considerable research into the appropriate accompanying electronics.

Carrying out this research into the necessary circuitry would take a great deal of time, and as such there would doubtless be a long lead time before the first iteration of a working prototype was built and ready for testing.

In almost complete contrast, the Arduino prides itself on being a 'rapid development system'. Once in receipt of an Arduino, the multi-platform IDE can be downloaded and

installed directly from the Arduino website, and programs can be uploaded directly from the IDE using the on-board USB interface.

Furthermore, the Arduino provides both digital and analog I/O headers with which you can immediately begin using to gather data from sensors or indeed interact with external circuitry.

WELL DOCUMENTED

Both the PIC and the Arduino are extremely well documented. Hundreds if not thousands of documents exist for every detail of each. However, the Arduino also benefits from a great deal of community written documentation that is of a professional standard. When buying a component that is specifically designed for use with the Arduino, almost all of the online stores offer links to detailed guides on how to put the component to use.

Unfortunately, whilst the PIC has an abundance of technical documentation that would doubtless be easily understood and digested by somebody with a keen interest in electronics, there is a distinct lack of documentation that is aimed squarely at the amateur.

WELL SUPPORTED

Support for both the Arduino and the PIC is provided courtesy of vibrant online discussion forums on both manufacturers websites, with each having tens of thousands of registered members. Again, the Arduino benefits somewhat by catering for the amateur, and there are dedicated sub-forums for those just getting started with the platform.

Users are free to make posts describing any problems they are having, and from a cursory glance the help that is provided is of good quality.

LOW COST

Whilst at first it may appear that the PIC is the cheaper of the two solutions, there are many hidden costs involved with its use. The Arduino is readily programmable, with a USB interface provided on-board. The PIC requires a separate programmer be built or purchased, with both options costing a considerable amount of money.

There are then the costs associated with constructing the circuitry to use alongside the PIC. Whilst the majority of components can be acquired cheaply, the extra costs should not be ignored.

The Arduino Duemilanove costs significantly more than the PIC, but provides everything needed on-board to allow the user to program it and begin prototyping a system 'out-of-the-box'. However, whilst the 'shields' available for the Arduino are a great help, they also come at a premium. The Xbee wireless shield for example costs around £30, whereas the individual components that make up this shield can be purchased for far less. Though again there is then the need for a more intimate knowledge of the electronics.

EXTENSIBLE

Both the Arduino and PIC are essentially limitlessly extensible, that is to say, either can be used with any electronic component given the right accompanying circuitry. However, the Arduino does have the great advantage of the ready-made 'shields' which can be plugged directly into the board's header pins.

These shields generally contain all the circuitry necessary to begin using its core functionality, though in some cases (such as the Xbee wireless shield) one remaining component will need to be fitted. This is often the case where a single shield can be used with a range of additional components.

3.2.3. INVESTIGATION CONCLUSION

After considerable research it was decided that the Arduino platform would be best suited to the requirements of the project. The complete platform, from the hardware such as the Duemilanove through to the provided IDE, has been designed from the outset to be put to use developing prototypes, and developing them quickly with a minimum of underlying electronics knowledge.

Rather than being geared towards the electronics enthusiast with a complex interest in the underlying circuitry, it is instead "intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments" (Arduino, 2010). Put more succinctly: it is designed for people who want to build systems, not circuits.

3.2.4. ADDITIONAL HARDWARE

HEART RATE MONITOR INTERFACE

The Heart Rate Monitor Interface (HRMI) is an "intelligent peripheral device that converts the ECG signal from Polar Electro Heart Rate Monitor (HRM) transmitters into easy-to-use heart rate data". Both the HRMI and the Polar Electro HRM (example shown in Fig. 3.6) are readily available for purchase online, and it is with these two devices that heart rate data will be gathered.



Figure 3.6 - Example of a Polar HRM Chest Strap. (Public Domain).

The HRMI uses an onboard Polar Heart Rate Module, that works by registering the 1ms electromagnetic pulse given off by the accompanying Polar chest strap. The HRMI then uses the onboard PIC microcontroller to implement Dan Julio's heart rate algorithm which is designed to "be responsive to changes in heart rate but ignore spurious data", before finally outputting it to the chosen interface.

Comprehensive documentation for the HRMI is available for download, which gives thorough detail on its use in an Arduino based system.

XBEE WIRELESS SHIELD AND TRANSCEIVER

The Xbee Wireless Module utilises the 802.15.4 wireless standard, as defined by the Institute of Electrical and Electronics Engineers (IEEE), which is a standard that defines the lower network layers (such as the physical and media access control (MAC) layer) of a type of Wireless Personal Area Network (IEEE , 2006).

The manufacturers of the Xbee wireless module then build upon this stack to provide a simple to use serial command set for communicating wirelessly with other devices using the same hardware. Furthermore, the Wireless Shield available for the Duemilanove plugs directly into the Arduino's ICSP (In Circuit Serial Programming) interface headers, immediately allowing any serial communications to be transmitted over wireless without the need for code to be implemented or changed.

So for example, the `Serial.print()` command used to print information over the Arduino's USB interface works in exactly the same way over the wireless interface. Not only does this make implementing the wireless hardware easy, but it also means code can be tested using the USB interface first which will be incredibly useful when ruling out the wireless module as the source of any problems.

In addition to the above, it has more than adequate wireless range (up to 120m with a clear line of sight) (Digi International Inc., 2008), and can be used in multi-point networks such as the one this project hopes to create. It also implements 128-bit AES encryption natively, again without the need for any code to be implemented, and has transmission speeds up to 1Mbps. Finally, it also has low power requirements, with the model used in this project operating on as little as 2.1 Volts.

3.3. SOFTWARE REVIEW

Where software is concerned, it has been the preference to use open source solutions wherever possible. Be it with the IDE used to develop code, or the web server used to present the 'front-end' of the system, using open source software means that no time has to be given to worrying about licensing or usage constraints.

3.3.1. DEVELOPMENT LANGUAGE(S)

For any code that has needed to be developed, time constraints have meant using the languages of which prior knowledge and experience is had, namely Java and PHP. In any case, these languages have both proven to be well suited to the project requirements. Both are multi-platform, both are open-source, both are widely used and as a result well documented.

Java, as a strongly typed Object Orientated programming language, would be well suited to the more complex and 'mission-critical' back-end of the system, dealing with the collection and parsing of results.

PHP, a loosely typed procedural programming language, has become one of the most popular languages (The PHP Group, 2010) with which to generate dynamic content on the Internet. Code written in PHP is interpreted 'on-the-fly' by the web server as a user requests a web page, meaning that the data presented to the user is generally always the most recent.

It is used by some of the most popular sites on the Internet, most notably Facebook, and has been instrumental along with JavaScript in the rise of the 'Web 2.0' phenomenon. As a result, vast amounts of code examples and tutorials exist for its use in almost any application, and as such, it would be well suited to powering the web-based graphical user interface (GUI).

Quite an extensive knowledge of JavaScript has been developed during the course of the project as more time was spent on developing the web application. The W3Schools website² proved to be an invaluable resource from which I could draw knowledge of three of the four languages used in the project, namely SQL, JavaScript, and PHP.

3.3.2. PERSISTENT STORAGE

Storing any collected patient data, and indeed usernames and passwords for the GUI, could well be done in a simple text file. However, this immediately brings about a myriad of problems concerning how data should be organised, and how access can be controlled.

Instead, a relational database will provide the means by which data will be stored, specifically, MySQL.

MySQL is an open source, multi-platform, relational database management system (RDBMS) that provides multi-user, concurrent access to any number of relational databases. Again, it has seen wide adoption and an almost meteoric rise in popularity (Oracle Corporation, 2010), with the likes of Google Inc., Wikipedia, and Facebook using it to power their websites.

Almost all contemporary languages support its use through the provision of APIs, and PHP and Java are no different, with native drivers and APIs available for both languages.

3.3.3. WEB SERVER

Whilst PHP is multi-platform, and interpreters are available for the majority of web servers, the most well supported and by far the most widely used is Apache's HTTP server.

Apache as seen as a key part in the exponential growth of the WWW, and as of April 2010 it is in use on over 112 million websites (Netcraft Ltd, 2010). With it responsible for serving

² <http://www.w3schools.com>

over 50% of the web pages as of 2009 (Netcraft Ltd, 2010), it is hailed as one of the biggest open source successes.

3.3.4. SUMMARY

For storage, web server, and generation of the GUI, the open source 'XAMPP' software stack will be utilised. XAMPP is an acronym that stands for: X – Multi/Cross-Platform; A – Apache HTTP Server; M – MySQL RDBMS; P – PHP Interpreter; P – Perl Interpreter (which will go unused).

XAMPP is freely available, and has been designed from the outset to provide developers with a rapid deployment solution for testing and developing new web applications. It will provide me with a complete, and more importantly portable environment with which I can store collected statistics (using MySQL) and present a graphical and dynamic front-end to the system (using Apache and PHP).

Development of the 'back-end' system, which will be responsible for the core logic and collection of patient statistics, will be done using the Java programming language. Java benefits from being a strongly typed language, meaning that each variable is assigned a type (Integer, String, Boolean etc), and to try and assign a value to a variable that does not match its type would generate an error, or rather an exception. This means that Java as a language has inherent validation, something that is of great importance in a system such as this whereby the validity of data is paramount.

With Java enjoying popularity both in enterprise systems and for smaller scale projects, support is widely available through a number of avenues. There are countless forums and mailing lists where developers can ask for the community's help on all manners of problems.

Also of great importance is the huge amount of libraries available, both native and third party. A quick Internet search quickly revealed a third-party library, RXTX, which looks ideal for communicating over a USB/serial interface.

4. DESIGN AND DEVELOPMENT

4.1. DESIGN OVERVIEW

4.1.1. HIGH LEVEL WORKFLOW DIAGRAM

The basic high-level workflow for the entire system can be seen in the following diagram.

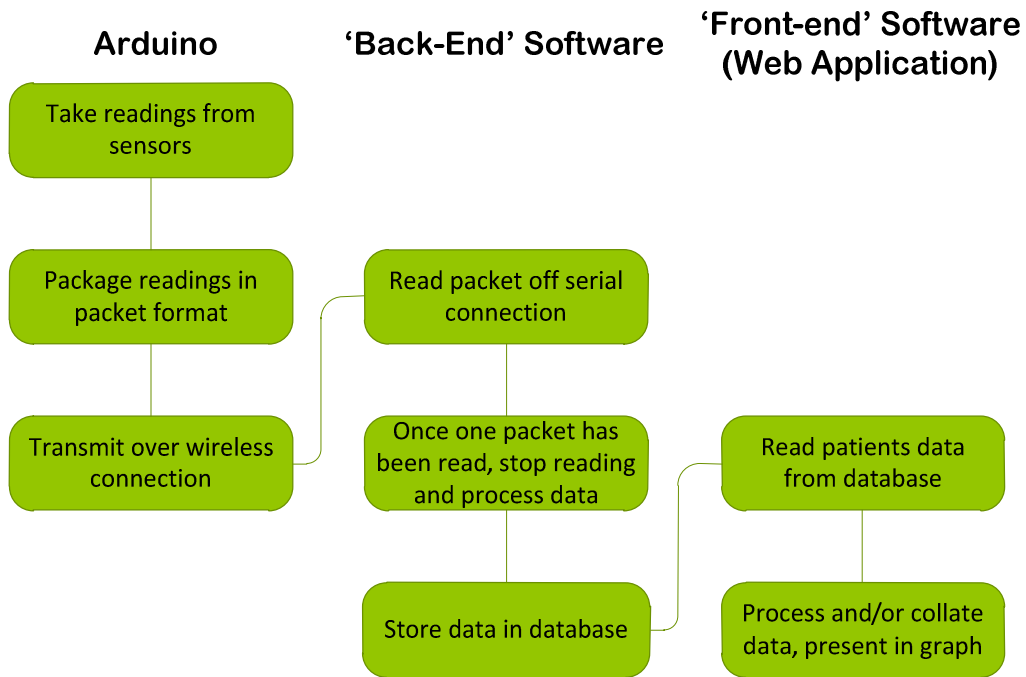


Figure 4.1 - High Level Workflow Diagram

4.1.2. HIGH LEVEL UML SEQUENCE DIAGRAM

The UML Sequence Diagram below (Fig. 4.2) gives a broad overview of the entities and processes involved with the system, and the order in which they'll interact with each other.

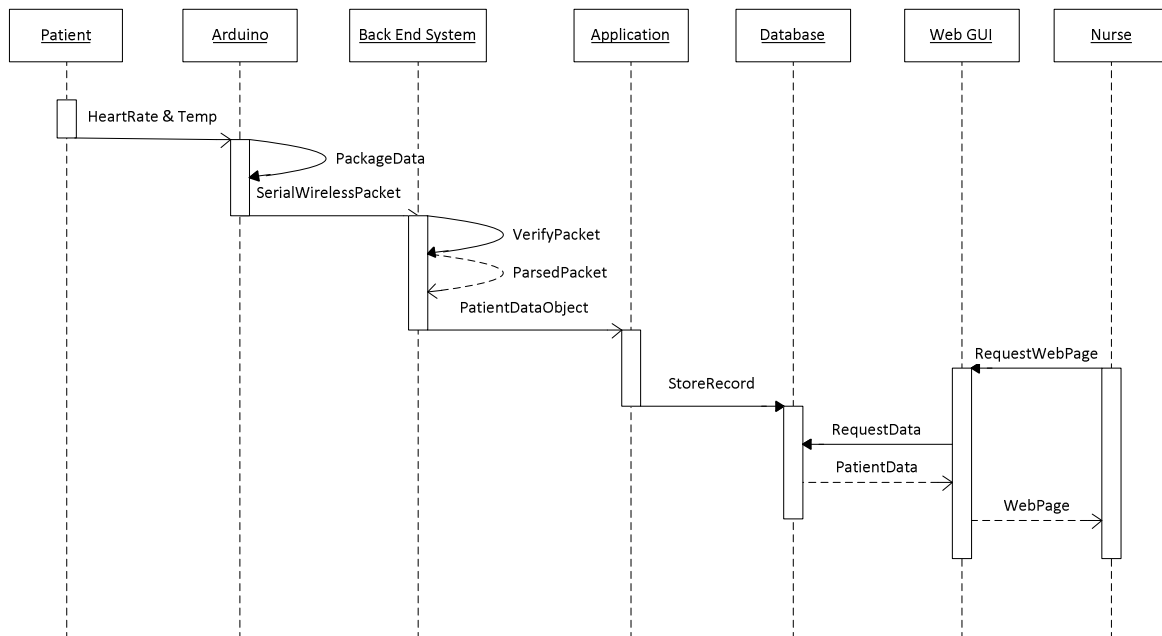


Figure 4.2 - UML Sequence Diagram, showing interactions in the system

4.1.3. DESIGN STRUCTURE

The design stage will be structured into four parts:

1. Designing the hardware circuitry to enable me to collect data from the sensors and transmit data wirelessly back to a base station.
2. Designing the on-chip software for the Arduino to read data from the sensors and transmit it across the wireless network.
3. Designing the database for storing the collected data
4. Designing the software for receiving and processing data using a serial connection.
5. Designing the web application that will be used to present the collected statistics to the end-user.

4.2. DESIGN PROCESS

4.2.1. DEVELOPMENT METHODOLOGY

For all stages of development, it was decided that an iterative development process (Fig. 4.2), as described by Dr Alistair Cockburn (Cockburn, 2008), would be adopted. This meant that following the initial planning stage, incremental stages of each part of the design were implemented and tested, and anything learnt during one of the stages could therefore be used in the subsequent stages.

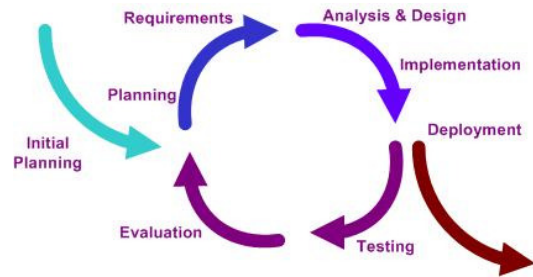


Figure 4.3 - Iterative Development Model.
(Public Domain).

So for example, for developing the back-end software, it started with a small Java class that carried out one function - receiving communication and dumping it to the Java console. Once it had been tested and confirmed that serial communication was being received reliably, it then progressed to creating a Java class that received communication and stored it in a StringBuffer object.

This continued with small changes being made to the design with each iteration, affording the project real flexibility with regards to the end result, until eventually an entire API for handling communications using my own high-level protocol had been designed and implemented - something which would have never featured in the original design.

Compare this to what would have resulted if a methodology such as the 'waterfall model' had been used, whereby the entire system is meticulously planned and designed during the initial stages, and then isn't allowed to be changed or re-evaluated until the entire system has been implemented and tested. As the author of the concept said himself, "[it] is risky and invites failure" (Royce, 1970).

Of course there are a myriad of other design methodologies that could have been adopted, 'Agile Development', 'Spiral', and 'Extreme Programming' to name but a few that are

currently enjoying popularity. However, it was felt that the 'Iterative Design Model' best suited both the project and the timeframe.

4.2.2. TESTING METHODOLOGY

As had just been described, testing was a continual part of the development. Typically each 'feature' required would be split into smaller objectives. So for example the development of the PHP code that is instrumental in the web application began with just making a successful connection to the database. Once that had been tested and proved as working, the code was changed to implement more complex SQL queries, and then further tests were run.

Every facet of the script was tested individually. Not only did this allow for rapid changes in the design, but it also saved valuable time with debugging. If an error was encountered, it could have only been the last line or function to be implemented.

Contrast this with if the 'waterfall model' design methodology had been used. The entire script would have been implemented, with several big changes in design being missed (like the inclusion of functions), and then if an error were to occur the entire script would have had to have been debugged.

4.2.3. CODING CONVENTIONS

Throughout my project strict coding conventions will be adhered to in order to provide consistency throughout the code base. Furthermore, adhering to coding conventions is important to ensure accessibility; I intend to release my code under a suitable open source license, and as such I want to make it as easy as possible for developers to view and understand my code, therefore making it easier to modify and enhance.

Coding conventions cover everything from how packages and classes are named, through to how code is indented and comments made.

When coding the back-end system in Java, I will largely be adhering to Sun's coding conventions as can be found online (<http://java.sun.com/docs/codeconv/>), save for a few enhancements of my own. Any enhancements I make will be documented in Appendix B.

It could be argued that to deviate from Sun's coding conventions is unnecessary; however, I feel that the enhancements made significantly improve the readability of the code. One such example can be seen below:

SUN'S CONVENTION

```
private static void main (String args[]) {
    // Comments go here
    if (true) {
        System.out.println("Hello world!");
    }
}
```

MY CONVENTION

```
private static void main (String args[])
{
    // Comments go here
    if (true)
    {
        System.out.println("Hello world!");
    }
}
```

Whilst it may seem somewhat trivial, I have found that this simple change can help a great deal when trying to find a missing brace, or indeed where a for loop or if statement ends when trawling through lines of code. The same is true even if the IDE in use features code highlighting.

4.2.4. DOCUMENTATION

Throughout every stage of the design and implementation process, extensive commentary was made in-line with all Java, PHP, or JavaScript code. Not only does this help with remembering exactly what a function does or what a variable is used for, but it will help immeasurably if anybody comes to extend or implement the code in the future.

In addition to this, significant inroads have been made to providing comprehensive documentation for the API

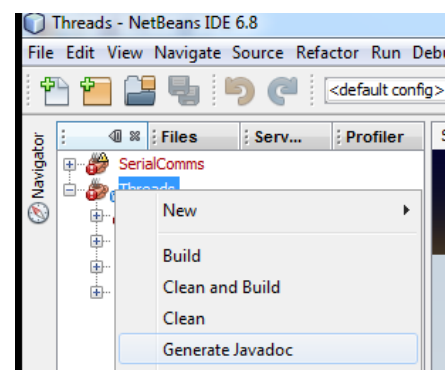


Figure 4.4 - Generating Javadoc within IDE

through the use of the JavaDoc documentation generator, which generates documentation in HTML format from the so called 'JavaDoc comments' in the source code.

To generate or view the JavaDoc documentation the developer has to first make use of the JavaDoc tool, available from Sun Microsystems³, to compile to documentation from the source code. However, most modern IDE's incorporate this tool into their own functionality, so often it is a trivial matter of 'right-clicking' on a project and selecting to 'Generate JavaDoc', as is shown in Figure 4.4.

Furthermore, the JavaDoc documentation has been made readily available at the 'ProjectZed' website (<http://projectzed.org/javadoc>) for third-party developers to study at their leisure.

4.2.5. SOURCE CODE AND PROJECT WEBSITE

SOURCE CODE

Throughout the design and implementation stages a private Subversion code repository was setup and used in order to keep a navigable copy of all design and implementation revisions, and towards the end of the project it was decided that all of the developed code should be released into the public domain.

With the first import of the project into its Google Code Subversion repository, the code was officially open source, released under the GNU General Public License v3⁴. This allows for anybody to use, share, modify and redistribute the code however they like.

WEBSITE

Around the project a website has been constructed (<http://projectzed.org>) that showcases the different facets of the system, particularly the API. It's primary aim is to demonstrate how the software and hardware can be used as part of a larger system, and the accompanying videos explain every facet of the system to the beginner, and will hopefully inspire somebody to use the code in their own project.

The website also provides links to:

- Where the Source Code can be accessed (Google Code repository)
- Runnable, pre-compiled JAR files for the API and the main back-end application

³ <http://java.sun.com/j2se/javadoc/>

⁴ <http://www.gnu.org/licenses/quick-guide-gplv3.html>

4.3. HARDWARE DESIGN AND IMPLEMENTATION

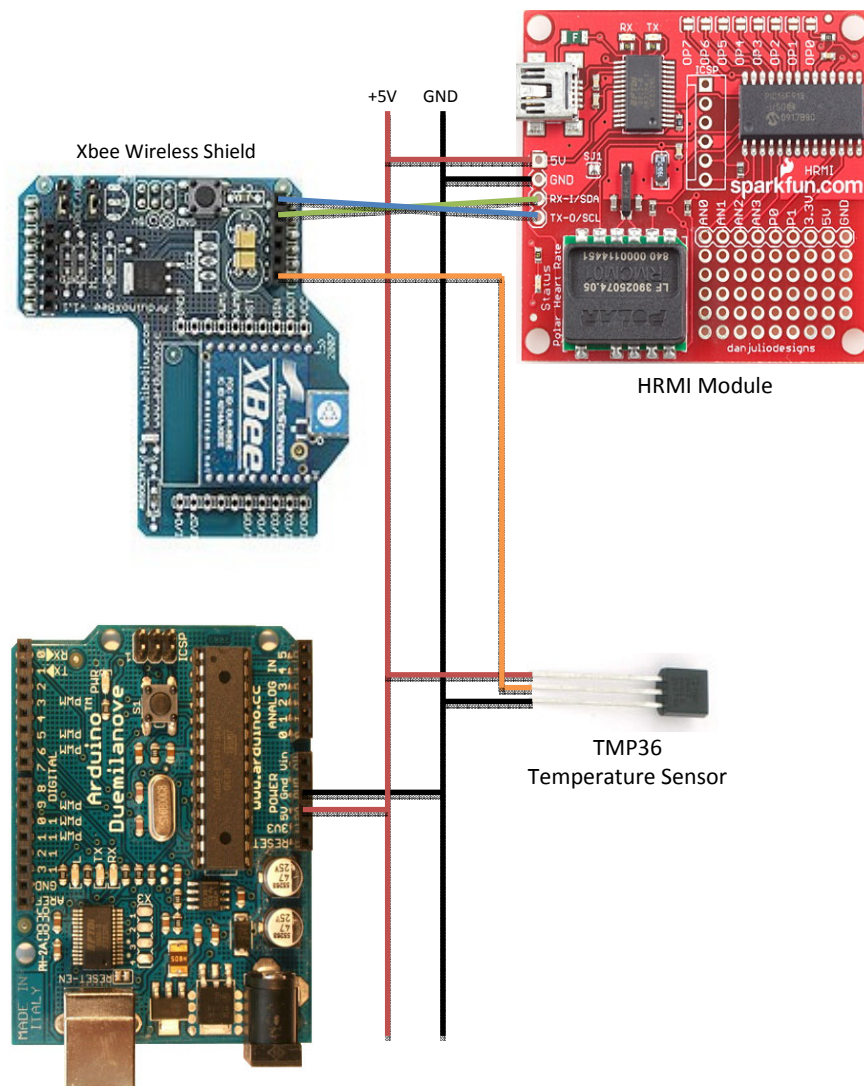
4.3.1. OVERVIEW

The hardware circuitry involved is relatively simplistic: the wireless shield is plugged in to the ICSP headers of the Duemilanove, and then the HRMI serial interface (two wires: one transmit, one receive) is connected to analog pins 4 and 5 of the pass-through headers provided on the wireless shield.

Finally, the Vout pin of the TMP36 temperature sensor is connected to analog pin 1, again provided by the pass-through headers on the wireless shield.

A +5V rail and Ground rail are provided for use by both the HRMI and the TMP36.

A pictorial diagram of the wiring of the circuit can be seen below:



4.3.2. IMPLEMENTING THE HRMI

Implementing the HRMI was a somewhat trivial matter as comprehensive instruction was provided as part of the units documentation.

I did run into one issue, however, as the documentation failed to make it clear that a jumper on the circuit had to be de-soldered, and another soldered, for it to communicate using the I2C interface rather than the USB interface. After re-reading the documentation I finally spotted the instruction in amongst the technical description of each jumper.

4.3.3. IMPLEMENTING THE XBEE WIRELESS SHIELD

Configuring the two Xbee Series 2.5 Wireless modules was made relatively simple by the ample documentation available online, both from the manufacturer Digi and (anonymous!) community members⁵. As has been mentioned previously, the Xbee's allow for multi-point networks arranged in a star configuration, whereby there is a central 'coordinator' to which several nodes, or 'routers', can communicate.

As a result, the Xbee's fit perfectly within the confines of my system: a base station receives information from a number of Arduino's attached to individual patients. The Xbee uses a network access method identical to Ethernet, known as 'Carrier sense multiple access with collision detection' (CSMA/CD), which results in reliable communication even in busy networks.

Configuring the Xbee modules is done using a separate USB interface board and software provided by the manufacturer. The 'patient nodes' are configured with an address upon which to transmit, and the 'coordinator' or receiver is then configured to listen to broadcasts on that address.

⁵ Nameless author documentation: <http://www.humboldt.edu/~cm19/XBee%20setup.pdf>
Manufacturer documentation: http://ftp1.digi.com/support/documentation/90001003_A.pdf

4.3.4. IMPLEMENTING THE TMP36 TEMPERATURE SENSOR

Implementing the TMP36 temperature sensor was possibly the simplest aspect of the entire project. When powered, the unit outputs 10 millivolts per degree centigrade on the signal pin (with a 500mV offset to allow for the measurement of freezing temperatures - so 0°C = 500mV).

It is then a simple case of connecting this output pin to one of the Arduino's analog pins, and taking a reading. The Arduino's built in math functions then make light work of carrying out the necessary conversions.

4.3.5. TESTING AND VALIDATION

Testing the validity of the data provided by the Arduino and its sensors couldn't have been simpler: an 'off-the-shelf' digital thermometer and heart rate monitor were purchased, and the results given by the Arduino were cross checked with those given by the production units.

4.3.6. ISSUES WITH IMPLEMENTATION

It was during the course of testing the Arduino and the data it was producing that it was noted that there was a slight delay, or rather a lag, between the heart rate given and the actual heart rate.

For example, exercise would begin, and it would take around 6-7 seconds for the increase in heart rate to be reflected in the data being produced by the Arduino. After some investigation it was realised that the delay was introduced by the buffer used by the HRMI. This buffer is used in order to have a large enough set of data to give an accurate, average heart rate.

4.4. ARDUINO SOFTWARE DESIGN AND IMPLEMENTATION

An Arduino compatible program must implement two functions at a minimum: `setup()` and `loop()`. The `setup()` function is used as would be imagined, to initialize any libraries or methods, or set pin modes etcetera. The `loop()` function again does as it describes, it executes the code and logic contained within in a continuous loop for as long as the Arduino has power.

The HRMI module that will be used to interface with the Polar heart rate monitor comes with its own API, which is provided as a C++ library that can be included in the Arduino program.

The simple command/response interface provided by the library is documented in detail in the provided documentation. There will be two commands needing to be used from the HRMI library: `hrmiCmdArg` and `hrmiGetData`.

The first command, `hrmiCmdArg`, will be used to tell the HRMI to read a number of values (number is passed as an argument) from the heart rate history buffer. The second command, `hrmiGetData`, is used to read those values into an array (a memory reference to which is provided as an argument).

More details of the implementation can be read in the in-line comments of the Arduino source code provided in section 10.1.1 of this report.

4.4.1. DESIGNING THE SERIAL COMMUNICATIONS PROTOCOL

For effective and reliable communications over the serial connection a protocol had to be designed with which an efficient yet effective error checking routine would be built upon.

Each transmission received from the hardware collectors will equate to one row of collected results in the database. These singular transmissions are effectively packets (of information) as is understood in a modern networking context, and will be referred to as such from now on.

Each complete packet will be comprised of the following essential data fields:

- Hardware ID – the unique identifier for that individual hardware collector
- Sequence Number – used in case packets are collected or processed out of order
- Temperature – in degrees Celsius
- Heart Rate – in beats per minute

Similar to how the RS232 protocol uses a start and end bit, my protocol will encapsulate or surround the above fields with a start and end tag. They will be used to ensure the completeness of a transmission; if a packet is missing either the start or end tag, it will be regarded as corrupt and subsequently discarded.

Therefore, a packet adhering to the serial communications protocol will be in the following format (with examples of possible field contents given in ASCII in the proceeding row):

Start Tag	Hardware ID	Seq. Number	Temperature	Heart Rate	End Tag
ST	123	1234	38.6	101	ET

A field delimiter will be required to enable the back-end logic to distinguish one field from the next. Commas could have been used, as is done with the popular CSV (comma separated values) format, but it was decided instead to use a colon as it was felt it would be more readable during debugging.

4.5. DATABASE DESIGN AND IMPLEMENTATION

4.5.1. OVERVIEW

The design of the database is limited in its complexity. To provide the basic functionality required by the project only two tables were strictly necessary: a table to store collected data, and a table that provided a link between patients and their hardware device.

It had originally been envisaged that a third table would be needed to store details of patients (name, address, contact details etc), but it was quickly decided that this was outside of the scope of this project.

Moreover, the likelihood would be that the application would interact with an external Patient Management System and retrieve their details from there if the system were ever to be used in a production environment, much like how the Philips TraceMasterVue system operates.

4.5.2. ENTITIES, ATTRIBUTES AND ENTITY RELATIONSHIPS

Below is a UML Entity Relationship diagram from the early stages of development.

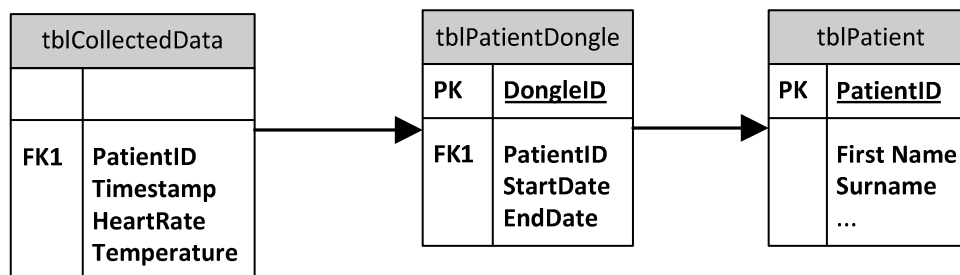


Figure 4.5 - UML ERD Diagram for the system

As is evident above, in the early revisions of the database design the **tblPatient** table was still included. One 'packet' of information received from the Arduino equates to one record in the **tblCollectedData** table, though as is shown the 'DongleID' field that makes up part of the packet transmitted by the Arduino is never stored in the table.

Instead, upon receiving a packet, a lookup query is performed by the back-end system that crosschecks the **DongleID** with the **PatientID**, and so the **PatientID** is stored along with the

data. This lookup query also makes use of the SQL BETWEEN command (Fig. 4.6) to ensure that the patient is in fact still present at the hospital.

```
// Build connection and prepare statement
PreparedStatement checkQuery = MySQLConnector.getConnection().prepareStatement(
    "SELECT PatientID FROM tblPatientDongle WHERE " +
    "DongleID = ? AND ? BETWEEN StartDate AND EndDate");
checkQuery.setInt(1, dongleId);
checkQuery.setTimestamp(2, timestamp);

// Execute prepared statement
ResultSet results = checkQuery.executeQuery();
```

Figure 4.6 - SQL Statement used to cross reference PatientID with DongleID

This has been done as to make the data more usable and readable. Whilst the lookup could have been done in reverse, that is when a nurse instigates a request via the web interface, it was felt that it would be better served by having the back-end software carry out the lookup.

4.5.3. IMPLEMENTATION

The implementation of the database was done via the 'PHPMyAdmin' web application that is bundled as part of the XAMPP software stack. The necessary source code to build the database can be found in Appendix A - 10.1.4.

Unfortunately, due to time constraints, the relationships described in Figure 4.5 were never implemented. If more time were allowed it would be spent on implementing a fully relational database. However, on a project of this scale and timeframes, the importance of having fully relational data was lost.

4.6. 'BACK-END' SYSTEM DESIGN AND IMPLEMENTATION

4.6.1. SERIAL COMMUNICATIONS

Achieving reliable serial communications was possibly the biggest challenge to overcome during the course of this project.

Sun Microsystems no longer support their `javax.comm` package, which includes the necessary classes for communicating over a serial connections. So instead an API named 'RXTX'⁶ had to be used, which is a `javax.comm` clone that took over from Sun after the package was deprecated.

Several of the early attempts at implementing it involved having a loop that read a predetermined number of bytes(/chars) from the input stream, and then passed that String on to another part of the program to be processed. However, in the time that it took to pass that String to another method, several packets of information may have already been missed.

Worse still, when the loop returned to carry on reading from the serial connection, it may well be in the middle of a packet transmission, resulting in a corrupt packet being read.

The initial solution implemented was to increase the number of bytes read to equal the length of two packets, meaning that at least one packet would be guaranteed to be read from the input stream. Validation would then discard the 'corrupt' parts of the received data, leaving one usable packet of data for every read operation.

However, there were even more problems introduced when the packets contained dynamic data: the heart rate may be three chars long (100), or it may be two (80), which could eventually lead to the read operation growing gradually more out of sync until it came down to luck whether a complete, usable packet was ever read.

⁶ <http://users.frii.com/jarvi/rxtx/>

This problem demanded attention for a week or more, until whilst reading through the RXTX documentation mention of using an event based model was found, whereby an event is generated every time data was received on the serial connection. It was plain to see that this was the correct way to implement the RXTX API within the project, and from that 'eureka moment' a reliable method of serial communication was developed and tested within a couple of hours.

Once an event has been generated, data is read from the serial connection until either a carriage return or a line feed is detected (generated by the `serial.println()` command in the Arduino code), at which point it will stop reading and pass the gathered information on to `commsParser` to begin further processing/validation.

Using this method, it is almost guaranteed that a valid packet will be received as the read begins with the first byte sent by the Arduino and finishes with the carriage return and line feed ASCII codes being sent after a whole packet has been transmitted.

During testing there have only ever been a handful of malformed or corrupt packets received using this method, which occurred as the hardware device was taken in and out of range of the receiver. However, these corrupt packets are quickly discarded due to the validation contained later in the program.

4.6.2. COLLECTORENTRY OBJECT

From an early stage in the design and development of the back-end application, a `CollectorEntry` JavaBean was used to store information parsed from a received packet. The `CollectorEntry` has a default constructor that requires it be passed a `String` upon it being instantiated. This `String` is a single 'packet', created by the `CommsEngine` class from the information received over the serial connection.

Once instantiated, the constructor first checks that the `String` contains a valid packet using the `PacketUtil.validate()` method. If true, it then parses the `String`, extracting the individual

packet fields with the aid of String's split() method, and storing the data using its own setter methods (as per the JavaBeans specification⁷).

Once an instance of CollectorEntry has been created, only the getter methods are publically accessible, meaning that neither the object or the datafields within can be manipulated once it has been created, thus further assuring the validity of the data it contains.

Using the CollectorEntry to store the 'packets' received over the serial connection has produced a flexible, fluid method with which data can be moved around the application. Instead of every class or method having to split the String containing the packet, or parse an array, they can instead rely on CollectorEntry's built-in functions to gain access to data.

So instead of packetString.split(":")[5], they can use packetObject.getHeartRate(), leading to much cleaner, readable code.

4.6.3. DEVELOPMENT OF THE COMMUNICATIONS API

Relatively recently it was decided that separating the code responsible for receiving and collecting 'packets' from the main body of the back-end application would be a valuable exercise. Moving the code out of the main application and subsequently developing an API would allow for other developers to utilise my code wherever possible, in projects that may have completely dissimilar end results. By developing an API for receiving and parsing structured data over a serial line, other developers are free to implement the API in any number of other applications.

For example, a developer may decide that they want to use the API in a project for use inside a gym, which simply triggers an alarm if a person exceeds a given heart rate threshold. And instead of storing their data in a MySQL database as has been done with this project, they may decide they want to store data in a text file, or possibly not at all.

⁷ <http://tinyurl.com/365tseb>

To implement the API, the developer simply has to implement the CommsReceiver interface provided by the API, and the subsequent required method incomingEntry() which has to take a CollectorEntry object as an argument.

The developer would then create a new instance of CommsEngine as provided by the API, passing it two arguments: a String which contains the device name of the appropriate COM port (for example "COM3" within Windows, or "/dev/tty1" within Linux) and an instance of the object that implements the CommsReceiver interface.

The CommsEngine class would then use a callback to call the incomingEntry method of the instance of the object that the developer has passed previously.

This use of a callback routine to make the application event driven negates the need for cluttered and inefficient code that would be needed if a process such as polling was used instead. This makes implementing the API 'clean' and efficient, with a minimal amount of code needed.

The developer using the API need not know how data is read off the serial line, nor how the data has been validated. Instead they simply deal with CollectorEntry objects passed to them by the CommsEngine, and can busy themselves with processing the data within as they see fit.

So within the context of this projects application, I have developed the mysqlEntry class that accepts CollectorEntry objects and stores the relevant datafields from within in a MySQL database. I could have just as easily stored that object on disk, or indeed integrated it into a Patient Management System.

VALIDATION

Validation is carried out courtesy of the PacketUtil.validate() method, which checks for a number of things to ensure the packet isn't corrupt or malformed:

- Checks that the packet contains six fields (delimited by a colon)
- Checks that the first field of the packet contains the String 'ST' (the start tag)
- Checks that the last field of the packet contains the String 'ET' (the end tag)

- Checks that all of the data fields contain the correct datatypes (so the DongleID field is an Integer, the Temperature field is a float etc.)

If any of the checks fails, the `PacketUtil.validate()` method returns Boolean false, signifying that the packet was not valid. Processing and parsing of the packet is then stopped as a result.

4.6.4. DEVELOPMENT OF APPLICATION

With the serial communications side of the back-end software being packaged into a reusable API, it was then implemented in an application that would provide the rest of the functionality required. Namely, storing the collected data.

Once the callback has been issued, the `incomingEntry()` method of `EntryProcessor` is passed a `CollectorEntry` object.

MYSQL INTERACTION

The `MySQLEntry` class of the back-end application is responsible for storing data. It has a `saveToDb()` method that accepts a `CollectorEntry` object, which it first extracts the `DongleID` datafield from using `CollectorEntry`'s `getDongleId()` method. It then uses the dongle ID to establish which patient has been assigned that dongle at that present moment in time using an SQL query that makes use of the `BETWEEN` clause.

Finally, now in possession of the patient ID along with the patient's heart rate and temperature courtesy of `CollectorEntry`'s `getHeartRate()` and `getTemperature()` methods, it creates a new record in the `tblCollectedData` table.

4.7. WEB APPLICATION DESIGN

4.7.1. OVERVIEW

The web application makes up only a small part of the overall application, but an important part nonetheless.

It consists of three main parts: the user interface, that is, what is displayed to the user; the JavaScript that fetches data and displays it with the aid of the Highcharts API; and finally the PHP script that is used to generate the dynamic data using SQL queries on the MySQL database.

The amount to which the user can interact with the interface is limited. Their interaction is constrained to setting heart rate and temperature thresholds, which are checked with every data update (once per second), and should they be exceeded a (visual) alarm will be triggered.

The web application also presents a collection of average values such as the average heartbeat over the course of the past hour, and the equivalent for temperature. The hourly heart rate average is also used within the program logic to trigger a visual alarm if it is below 50 (indicating suspected Bradycardia) or above 100 (indicating suspected Tachycardia).

4.7.2. USE CASES

System User Use Case Diagram for use of the Web Application:

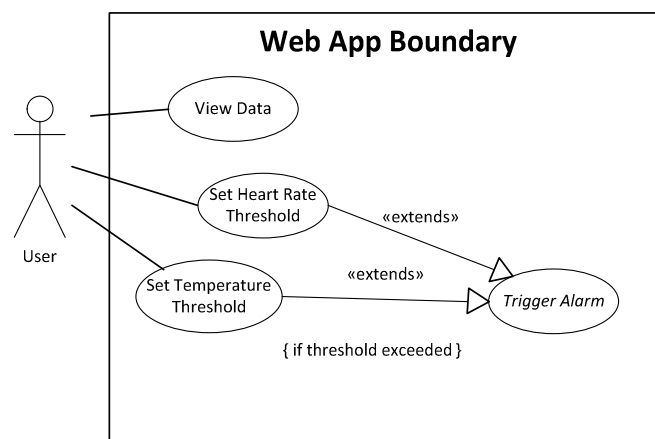


Figure 4.7 - Use Case Diagram depicting use of Web Application

The diagram above (Fig. 4.7) depicts the use cases for the web application system. Not shown is the previously described built in functionality of the web application, that triggers an alarm to signal suspected Tachycardia or Bradycardia based on the average heart rate for the past hour.

4.7.3. WHY USE A WEB-BASED GUI?

There is an emerging trend within computing as a whole that has seen applications moving from a per-computer, static installation, to a dynamic web-based application that can be accessed remotely by the majority of web browsers.

There are far too many advantages brought about by this technique to list, but perhaps the two most important are: platform independence and ease of access.

Platform independence refers to the fact that the web application can be accessed on any platform utilising any operating system, providing that platform's web browser supports any technology utilised by the web-app (typically JavaScript). This means, for example, that a user using the Microsoft Windows operating system (OS) will (typically) be provided with an identical view of the application as a person sitting next to them using the Linux OS.

Ease of access refers to the fact that an end-user does not need to install an application to their computer in order to use it. It also refers to the fact that the application is available to them from any computer as long as they're connected to the same network as the server which hosts it (be it the Internet or a small, local area network).

There are also advantages for the developer when deciding to develop a web-based application. Designing a typical desktop application is fraught with complications; perhaps the most renowned being providing a consistent user experience, something made even more complicated when a multi-platform application needs to be developed. The continued development of web standards is slowly seeing a consistent experience being delivered to the user.

4.7.4. PRESENTING COLLECTED DATA

HIGHCHARTS API

It was decided that displaying the collected statistics in a 'real-time' dynamic graph would be the best way of making the data easily digestible. After a modicum of research, the Highcharts JavaScript API⁸ was discovered, which offered just that functionality along with a robust API with which all manners of graphs can be created.

The Highcharts API is open source software, and is provided under a 'Creative Commons Attribution-NonCommercial 3.0' license, which

means it is free to use permitting the website it's used on isn't of a commercial nature. It produces exceptional looking graphs, whilst all the while remaining highly functional.

Using the comprehensive documentation, I implemented a curved line graph upon which two data series are plotted - heart rate and temperature.

DYNAMIC, REAL-TIME CONTENT USING PHP AND JAVASCRIPT

The two data series were generated dynamically, using a mixture of JavaScript and PHP. The JavaScript is embedded in the web application, and is executed by the client viewing the web page. It calls a PHP script, live-server-data.php, every second which returns a JavaScript array containing both the data used in the graph data and the average values that are visible underneath the graph (Fig 4.8).

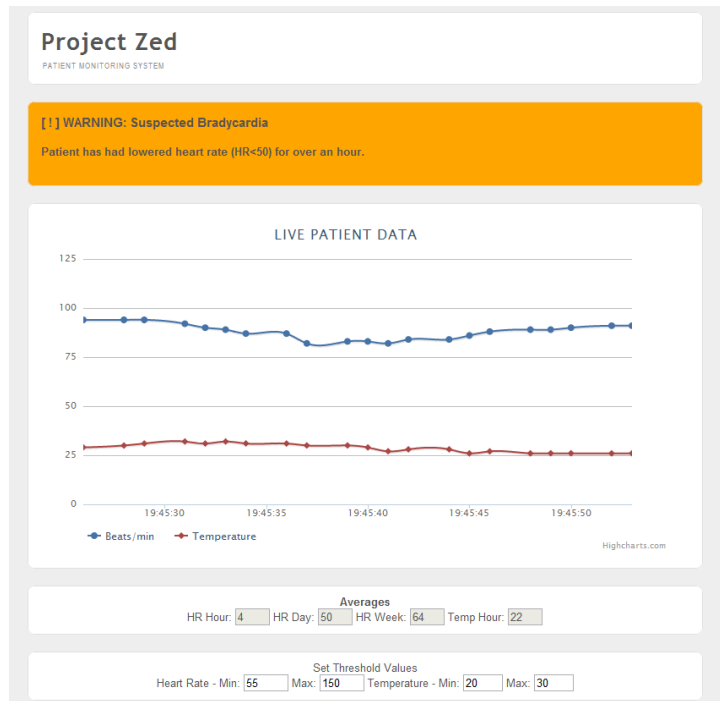


Figure 4.8 - The Web Application

⁸ <http://www.highcharts.com>

Behind the scenes, the PHP script executes several queries upon the MySQL database in order to generate the 'live' data. To establish the average heart rate and temperature values, quite a complex SQL statement that makes use of both the BETWEEN clause and the AVG function is used (Fig 4.9).

```
// Get average heart rate over the past hour for patient 123
function getHeartHourAvg()
{
    // Get dates in correct format
    $today = getTodaysDate();
    $hourago = getOneHourAgo();

    // SQL Query to retrieve average
    $hearhourquery = "Select AVG( `HeartRate` ) FROM `tblCollectedData` WHERE `PatientID` = '123' AND `Timestamp` BETWEEN '" . $hourago . "' AND '" . $today . "'";

    // Execute query
    $hearhourresult=mysql_query($hearhourquery);

    // Extract and format result
    $avgheartbeat=mysql_result($hearhourresult,0);
    $avgheartbeat_hour = sprintf("%d",$avgheartbeat);

    // Return value
}
```

Figure 4.9 - SQL query to calculate average heart rate for the past hour

As with the Java code that has been developed, all the PHP and JavaScript code used has been commented thoroughly in order to aid clear understanding by anybody who may want to reuse it.

5. EVALUATION

5.1. CRITICAL EVALUATION OF THE SYSTEM

Firstly, I must say I am extremely proud of what I have accomplished with the development of this system. Whilst not every facet of the design originally envisaged (particularly the database schema) has seen fruition, a great deal of what I set out to do has been done, and whatsmore I believe it to have been done to a good standard.

5.1.1. SOFTWARE

The development of the API, whilst it still requires some work as evident by the several "TODO" comments scattered throughout the code, is perhaps the part of the project of which I am most proud. More has been learnt and discovered about Java, and indeed object orientated programming in general, in the six months spent developing this system than had been during two years of a Network Programming module.

Furthermore, I am of the belief that the standard of coding is generally of good quality, and should be easily understood by any third-party developer thanks to the attention given to documentation and commentary.

The use of an 'AJAX-like technique' to generate the dynamic data for the web application also performs very well, better than was expected when originally planning the interface.

Whilst the use of the third party Highcharts API could be construed as to be 'cheating' (as if in cutting corners), it has been integrated into my solution well, and provides a much better solution than could have been developed within the same timeframe by myself.

During the late stages of development it was discovered that a bug exists that stops the web application rendering within Internet Explorer. Throughout the development I made a habit of ensuring that the web application worked in the three most common browsers (Microsoft Internet Explorer, Mozilla Firefox, Google Chrome) - however a recent change has resulted in neither the Highcharts graph or the average figures being displayed, which points towards a bug in the JavaScript.

However, the powerful 'FireBug' extension for the Mozilla Firefox web browser gives no warning or error as to what the possible cause of the problem could be. Unfortunately due to time constraints I was unable to investigate this problem any further.

5.1.2. HARDWARE

One part of the problem domain that I have paid very little attention to is powering the prototype hardware system. For the device to be usable within a hospital environment, and to be truly portable, it would need to have its own independent power supply.

Whilst battery packs similar to those found in mobile phones are readily available for use with the Arduino, and as such powering the device presents no real problem, it would have been advantageous to test the prototype system with one; even if it was just an exercise to prove that it works.

The size of the hardware system is also considerably bigger than what I had originally foreseen, and it would be infeasible to expect a patient to wear a system as large as the one I have produced during their stay in hospital. With that said, the system is designed merely as a prototype to prove the concept, and the possible miniaturisation of the hardware is discussed in section 5.3.1.

5.1.3. CONCLUSION

Ultimately, the developed solution caters for all the requirements that were laid out in the Deliverables section of the Introduction, though it's also clear that not all that was originally envisaged has 'seen the light of day'. Some things have been lost due to practical, worthwhile reasons (for example, the sequence number field in the packet protocol), whilst others have had to be sacrificed in order to have a working system to demonstrate (multi-patient display within the web application).

5.2. CRITICAL EVALUATION OF THE PROJECT

In contrast to the successes achieved during the development of the system, the project was much more fraught with problems.

Perhaps the greatest problem or issue experienced was caused by me not keeping *organised* notes throughout the design and development stages of the project. Whilst copious notes have been kept, and almost every reference or resource noted, it has been done so in a disorganised manner.

Three A5 pads containing a jumble of notes, kept in a loose chronological order, presented a considerable headache when it came time to compile the project report.

Time management, or rather the allocation of time, also presented a problem. With hindsight, too big a proportion of my time was spent developing the application, rather than the report.

If I were to repeat the project, I would begin immediately with the report, writing up the investigation and design sections into a usable format, instead of making notes and drawing diagrams by hand to refer back to later, before even beginning to work on the practical side of the project.

The final issue I would correct if I were to repeat the project would be to try and find a contact within the field of medicine who has more time to spare. Whilst Dr. Hodgkins gave me valuable insight and encouragement during the early stages of the project, he was unfortunately too busy with his studies towards the end of the project to carry out any sort of final appraisal. I have however e-mailed him links to the Project Zed website with an aim to have him watch the videos available there, and to at some point receive feedback would be great.

5.3. POSSIBLE FURTHER ENHANCEMENTS

5.3.1. HARDWARE

Whilst I am pleased with the finished hardware design (Fig 5.1), there are a number of things that I would like to see adapted or changed should I be given more time.

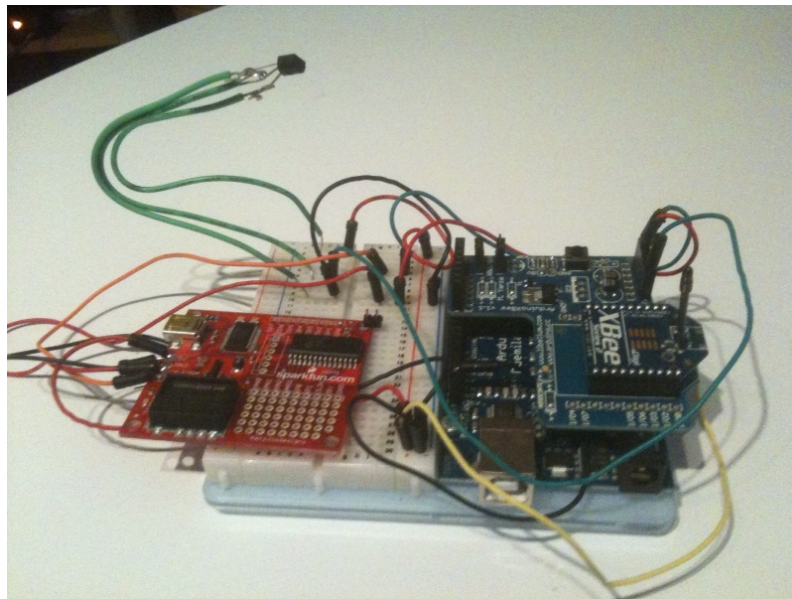


Figure 5.1 - Completed, working hardware prototype

The housing and circuitry for the prototype would be one of the areas I would most like to invest more time in. Prototype housing could be constructed, that would conjoin the Polar heart rate monitor chest strap with the rest of the circuitry. If it could be done to a sufficient standard, this could see the prototype becoming an 'all-in-one' unit as I had originally envisaged.

Of course, if the circuitry were ever to even be used in clinical trials, vast improvements in the unit's size would have to be made. It would probably be best to 'start again' where the design of circuitry was concerned. Beginning with just the ATmega 328 microcontroller (preloaded with the Arduino boot loader), the circuit could be designed without the need of the USB interface provided on the Duemilanove, or indeed the headers, push buttons, LEDs, voltage transformers etc.

The same can be said for both the HRMI circuit and the XBee wireless shield. Whilst both have been instrumental in the rapid development of the prototype, they also provide an abundance of unnecessary circuitry to provide headers, jumpers etc.

Then there is the ATmega 328 microcontroller itself, which is actually available in a much smaller form factor. The 32A version of the chip measures an extraordinary 0.95cm x 0.95cm (Atmel Inc., 2010), compared with the 3.4cm x 0.82cm of the 28P3 version of the chip used on the Duemilanove.

Helpfully, there are variants of the Arduino produced which utilise this much smaller ATmega chip, and enjoy similar vast reductions in size. If I were to produce a second generation prototype, I would probably make use of the 'Arduino Pro Mini' development board, which measures 1.8cm x 3.3cm, compared with the Duemilanove's 6.5cm x 4.5cm dimensions, making it almost a fifth of the size.

This much smaller board could then be incorporated onto a custom printed circuit board, along with the components (available individually) that are used on the Xbee shield and HRMI development board.

The end result being that a unit could be produced that incorporated equally functional hardware in roughly half of the size of the Duemilanove.

5.3.2. SOFTWARE

BACK-END

Given more time I would have liked to have packaged up the back-end system to include all dependencies (RXTX, MySQL JDBC driver), rather than have to distribute them as separate JAR libraries. Of course this would be dependent on the license used.

Also I would have liked to have implemented a properties file for use within the API, so settings such as those for the serial connection could be easily changed by the end-user.

FRONT-END

The front-end has been relegated to becoming somewhat of a prototype system. Whilst I'm extremely pleased with my implementation of the Highcharts JavaScript API, I would have liked to have realised my vision of multiple patient's graphs being displayed on the same page to demonstrate just how useful it could be in a hospital environment.

With that said, the implementation of the front-end application became somewhat of an afterthought as I quickly became engrossed in the development of an API that could be used by others.

As such, I was happy to be able to demonstrate a simple implementation of the API, that sees just one patient's statistics displayed on screen. Developing this further to meet my original goal would be a trivial matter, and one that I aim to complete in my own time.

6. GLOSSARY

Concurrency - concurrency describes a system within which several simultaneous computations are being carried out.

API - Application Programming Interface, an interface provided by a piece of software to enable interaction with other software.

7. BIBLIOGRAPHY

Arduino. (2010, February 23). *HomePage*. Retrieved March 2010, from Arduino:
<http://arduino.cc>

Atmel Inc. (2010). *ATmega328 Product Summary*. Retrieved April 14, 2010, from Atmel:
http://www.atmel.com/dyn/resources/prod_documents/8271S.pdf

Cockburn, D. A. (2008, May). *Crosstalk - The Journal of Defense Software Engineering*. Retrieved January 4, 2010, from Software Technology Support Center, U.S. Air Force:
<http://www.stsc.hill.af.mil/crosstalk/2008/05/0805Cockburn.html>

Digi International Inc. (2008). *Xbee / Xbee-PRO Znet 2.5 OEM Module - Product Manual*. Retrieved October 29, 2009, from Digi International:
http://ftp1.digi.com/support/documentation/90000866_C.pdf

IEEE . (2006, September 8). *802.15.4 Specification-2006*. Retrieved March 21, 2010, from Institute of Electrical and Electronics Engineers:
<http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf>

Netcraft Ltd. (2010, April). *April 2010 Web Server Survey*. Retrieved April 19, 2010, from Netcraft: http://news.netcraft.com/archives/web_server_survey.html

Oracle Corporation. (2010, April). *MySQL :: Market Share*. Retrieved April 4, 2010, from MySQL: <http://www.mysql.com/why-mysql/marketshare/>

Royce, D. W. (1970). *Managing the Development of Large Software Systems*. Retrieved March 21, 2010, from

http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf

The PHP Group. (2010, April 22). *Usage Stats for April 2007*. Retrieved April 2, 2010, from PHP: <http://php.net/usage.php>

Toumaz Technology. (2009). *Sensium Introduction*. Retrieved December 2009, from Toumaz Technology: http://www.toumaz.com/public/page.php?page=sensium_intro

8. REFERENCES

Highcharts Documentation - How to Use

<http://www.highcharts.com/documentation/how-to-use>

HRMI manual and tutorial

http://danjuliodesigns.com/sparkfun/hrmi_assets/hrmi.pdf

9. FURTHER READING

Igoe, T. (2007). *Making Things Talk: Practical Methods for Connecting Physical Objects*. Make.

A detailed look at the various methods that can be employed to connect physical devices via an array of mediums. It discusses in-depth the protocols and systems used when developing proprietary and non-proprietary networks, and also gives practical examples illustrating how the reader can develop their own. Also discusses the use of the XBee wireless module which is extremely pertinent to my project.

Sullivan, O. (2004). *Physical Computing*. Premier Press.

A great overview and study of the bridge that exists between the world of computer software and computer hardware. Examines how closely intertwined the two are, even

though to the layman they may at first appear completely separate. Also discusses the meteoric rise of the embedded computer, and its acceptance by society.

Farley, J. (1998). *Java Distributed Computing*. O'Reilly Media.

A thorough synopsis of how distributed systems can be developed using the Java programming language. Also gives detail on how Java can be used to develop database-driven distributed systems, and also gives great detail on how Java is designed from 'the bottom up' to cater for distributed systems.

Moggridge, B. (2006). *Designing Interactions*. MIT Press.

A detailed analysis of what constitutes HCI, and more importantly, how to design with HCI in mind. Discusses why some hardware devices have become immensely popular, and analyses the mistakes made by some of those that haven't. Relevant to my project because of the hardware design stage.

Galitz, W. O. (2007). *The Essential Guide to User Interface Design*. John Wiley and Sons.

A review of successful methodologies that can be employed when designing user interfaces. Useful for when analysing the design of my application 'front-end'.

Fitzgerald, B. & Kenny, T. (2004). *Developing an Information Systems Infrastructure with Open Source Software*. IEEE Computer Journal – January Edition.

A review of the authors work towards developing open-source information systems within a hospital environment. Extremely pertinent because of the context even though my project will be developing an entirely different system.

10. APPENDICES

10.1. APPENDIX A – SOURCE CODE

10.1.1. SOURCE CODE FOR ARDUINO

```
/*
 * ProjectZed - www.projectzed.org
 *
 * Program for reading heartrate from a Polar heartrate monitor (via
 HRMI module)
 * and body temperature from a TMP36 Temperature sensor, and
 transmitting that
 * information as a packet across a wireless network using the Xbee
 wireless
 * module.
 *
 * Version 0.69 - 18/04/10
 */

// Include the Wire and HRMI libraries
#include "Wire.h"
#include "hrmi_funcs.h"

/*
 * Configuration Information
 *
 * The BAUDRATE should match that with which the receiving device is
 set to
 * communicate at.
 *
 * The HRMI_I2C_ADDR should be set to whichever address the HRMI
 module has been
 * set to. By default this value is 127.
 */

#define BAUDRATE 9600
#define HRMI_I2C_ADDR 127

/*
 * Global variables
 */

int numHeartReq = 1;           // Number of HR values to request
int numResponseBytes;         // Number of Response bytes to read
byte ResponseArray[3];        // Byte array sized to store single
heart rate
int temperaturePin = 0;        // The anaolog pin with which the
// Vout from the TMP36 is connected.
int deviceId = 4;             // [!] Device Unique ID
byte hrmi_addr = HRMI_I2C_ADDR; // I2C address to use

/*
 * Arduino initialization code segment
 */
```

```

void setup()
{
  // Initialize the I2C communication
  hrmi_open();
  // Initialize the serial interface
  Serial.begin(BAUDRATE);
}

/*
 * Arduino main code loop
 */
void loop()
{
  // Read the voltage from TMP36
  float temperature = getVoltage(temperaturePin);
  // Change mV reading into degrees C
  temperature = (temperature - .5) * 100;

  // Request heart rate value
  hrmiCmdArg(hrmi_addr, 'G', (byte) numHeartReq);
  // Set how many bytes to retrieve from the HRMI (3)
  numResponseBytes = numHeartReq + 2;

  // Pass hrmiGetData the i2cRspArray mem reference, which
  // if data is available will be populated with heart rate
  if (hrmiGetData(hrmi_addr, numResponseBytes, ResponseArray) != -1)
  {
    Serial.print("ST:"); // Print Start tag
    Serial.print(deviceId); // Print Device ID Number
    Serial.print(":"); // Print field separator
    Serial.print("1234:"); // Print seq. number (unused)
    Serial.print(temperature); // Print temperature float
    Serial.print(":"); // Print field separator
    Serial.print(ResponseArray[2], DEC); // Print latest heart rate in
    decimal format
    Serial.print(":ET"); // Print end tag
    Serial.println(); // Print carriage
    return/linefeed
  }

  delay(1000); // Delay 1 second before
  fetching new result
}

float getVoltage(int pin)
{
  // Convert from digital range (0-1024) to a voltage (0-5v), and
  // return that value.
  return (analogRead(pin) * .004882814);
}

```

10.1.2.SOURCE CODE FOR API

Source code for the API is included on disk which is located at the end of this report.

It is also available for viewing and download via Google Code:

<http://code.google.com/p/project-zed>

10.1.3.SOURCE CODE FOR BACK END APPLICATION

Source code for the back-end application is included on disk which is located at the end of this report. It is also available for viewing and download via Google Code:
<http://code.google.com/p/project-zed>

10.1.4.SOURCE CODE FOR CREATION OF DATABASE

The following is produced using the MySQL Dump utility:

```
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

CREATE DATABASE `projectzed` DEFAULT CHARACTER SET latin1 COLLATE
latin1_swedish_ci;
USE `projectzed`;

CREATE TABLE `tblcollecteddata` (
  `PatientID` int(11) NOT NULL,
  `Timestamp` datetime NOT NULL,
  `HeartRate` int(11) NOT NULL,
  `Temperature` float(10,0) NOT NULL,
  KEY `fkPatientID` (`PatientID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE `tblpatient` (
  `PatientID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` text NOT NULL,
  `Gender` text NOT NULL,
  `Age` int(11) NOT NULL,
  `ThresholdTemp` int(11) NOT NULL,
  `ThresholdHeart` int(11) NOT NULL,
  PRIMARY KEY (`PatientID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

CREATE TABLE `tblpatientdongle` (
  `PatientID` int(11) NOT NULL,
  `DongleID` int(11) NOT NULL,
  `StartDate` datetime NOT NULL,
  `EndDate` datetime DEFAULT NULL,
  PRIMARY KEY (`PatientID`,`DongleID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

10.1.5.SOURCE CODE FOR WEB APP

Source code for the web application is included on disk which is located at the end of this report. It is also available for viewing and download via Google Code:

<http://code.google.com/p/project-zed>

10.2. APPENDIX B – CODING CONVENTIONS

JAVA

When enclosing code in braces {}, Sun's coding convention was deviated from so that the braces were always on their own line:

```
private static void main (String args[])
{
    // Comments go here
    if (true)
    {
        System.out.println("Hello world!");
    }
}
```

10.3. APPENDIX D – TABLE OF FIGURES

Figure 3.1 - PageWriter TC50 ECG Machine (Courtesy of Philips Inc.)	11
Figure 3.2 - DigiTrak XT Device	11
Figure 3.3 – Sensium™ 'Life Pebble'.....	12
Figure 3.4 - 'Life Pebble' being attached. (Courtesy of Toumaz Tech. Ltd.)	13
Figure 3.5 – Arduino Duemilanove. (Public Domain).	17
Figure 3.6 - Example of a Polar HRM Chest Strap.....	22
Figure 4.1 - High Level Workflow Diagram	27
Figure 4.2 - UML Sequence Diagram, showing interactions in the system	28
Figure 4.3 - Iterative Development Model.	29
Figure 4.4 - Generating JavaDoc within IDE.....	31
Figure 4.5 - UML ERD Diagram for the system	38
Figure 4.6 - SQL Statement used to cross reference PatientID with DongleID	39
Figure 4.7 - Use Case Diagram depicting use of Web Application.....	45
Figure 4.8 - The Web Application	47

Figure 4.9 - SQL query to calculate average heart rate for the past hour.....48
Figure 5.1 - Completed, working hardware prototype.....52

10.4. API REFERENCE GUIDE

JavaDoc documentation for the API can be viewed at:

<http://projectzed.org/javadoc>

A preliminary guide and examples of how the API can be implemented can be found at:

<http://code.google.com/p/project-zed>