INITIAL DESIGN REPORT

DECEMBER 2008

BUGRA OKTAY

GIZEM KILIÇ

GÜVEN ISCAN

ÖZKAN AKMAN

Table of Contents

1. INTRODUCTION

1.1 Purpose of the Document

The main purpose of this document is supplying an initial design to our project.

This document is prepared to show the design process in the project. These processes are not the certain processes, they can be changed, but the main lines of the project will take part in this report.

The final processes will be shown in our final report.

1.2 Scope of the Document

The document comprises of our hardware and software requirements, environment, user interfaces, architectural design, project procedures, UML modelings(diagrams), development schedule and planned future works.

The aim of the diagrams are to visualize the details of the project.

2. PROJECT DESCRIPTION

Our project is to design and implement a device to control a patient's health situation. In the project, some sensors will be used to reach the values of the patient.

2.1 Detailed Problem Definition and Project Features

In the project, the patient will carry some sensors on his body, and the data came from the sensors will be transferred via bluetooth. After transferring the data, they will be sent to internet by using GPRS. We are going to create a small database to know the patient's health situation.

The features to be provided;

1. Object-oriented programming
2. Data transfer
3. Database
4. Simple, easily understandable user interface

2.2 Design Constraints and Limitations

We are preparing this project for the project of our department. So, the basic limitation is time. The final date is June, 2009 and from now, we have just six months. The only job we must do is not this project, we have some other jobs, so the time is really a limited time for this project. Besides the final deadline, we have so much intermediate deadlines, and these are taking so much time too.

We have some hard times because we can not find any sensor to get data. We are still trying to find but we think, if we can not fint any sensor to use, we will give the data to the device ourselves as input. Searching for sensors really took so much time.

The security is the other main constraint. We will use encryption to supply the security. We will discuss this in the future steps of the project.

Performance is the other constraint. Because of the project's main issue is health, the result of a little latence of transfer of data can create big problems. So, the project should show a great performance.

2.3 Design Goals and Objectives

USABILITY: The device should be easily usable by everyone. Someone who don't know so much about computer can use the program, because anyone can be a patient and he should be watched. We can not set a computer knowledge requirement for this kind of job.

RELIABILITY: This is a key concern for a project. There shouldn't be any bugs. The program should not be killed by the buggy codes.

SECURITY: Because of the online usage of the program, it should be secure. During the usage of the program, it should not be affected by the threads or anything else.

PORTABILITY: The project will be used in Windows operating system because of its more common usage.

3. HARDWARE AND SOFTWARE REQUIREMENTS

3.1 Hardware Choices

PIC

In our project, we will study with an embedded system, so we will need a pic. We will get a PIC from our department, which is PIC18F252.

Device	On-Chip Program Memory		On-Chip RAM (bytes)	Data EEPROM (bytes)
	FLASH (bytes)	# Single Word Instructions		
PIC18F252	32K	16384	1536	256

This pic satisfies our needs with its 28 pins, 32K Program Memory, 1536 bytes of Data Memory (RAM), 256 bytes of EEPROM Data Memory and embedded analogto-digital converter.

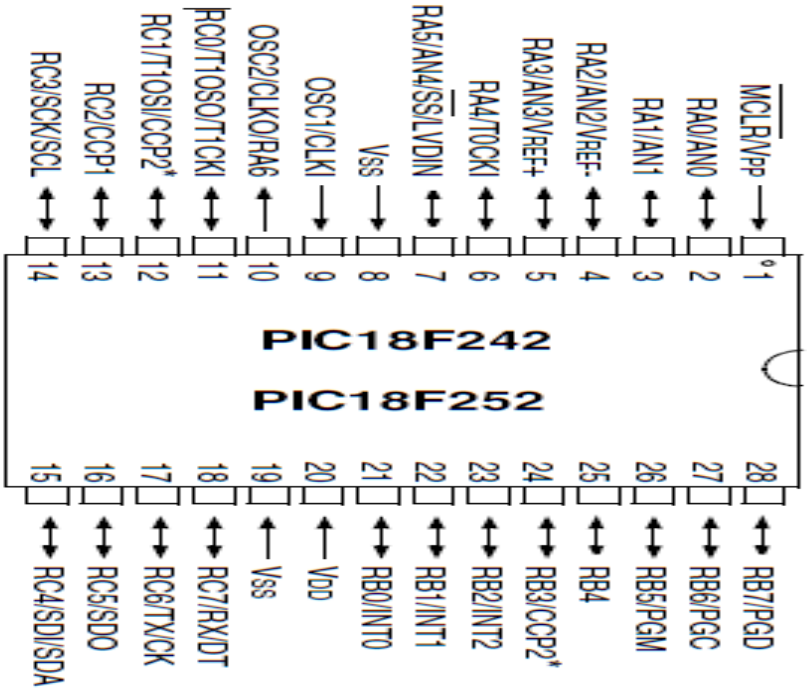


TABLE 1-2: PIC18F2X2 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	DIP	SOIC			
MCLR/VPP	1	1			Master Clear (input) or high voltage ICSP programming enable pin.
MCLR			I	ST	Master Clear (Reset) input. This pin is an active low RESET to the device.
VPP			I	ST	High voltage ICSP programming enable pin.
NC	—	—	—	—	These pins should be left unconnected.
OSC1/CLKI	9	9			Oscillator crystal or external clock input.
OSC1			I	ST	Oscillator crystal input or external clock source input. ST buffer when configured in RC mode, CMOS otherwis
CLKI			I	CMOS	External clock source input. Always associated with pin function OSC1. (See related OSC1/CLKI, OSC2/CLKO pins.)
OSC2/CLKO/RA6	10	10			Oscillator crystal or clock output.
OSC2			O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO			O	—	In RC mode, OSC2 pin outputs CLKO which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
RA6			I/O	TTL	General Purpose I/O pin.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2			
RA0			I/O	TTL	Digital I/O.
AN0			I	Analog	Analog input 0.
RA1/AN1	3	3			
RA1			I/O	TTL	Digital I/O.
AN1			I	Analog	Analog input 1.
RA2/AN2/VREF-	4	4			
RA2			I/O	TTL	Digital I/O.
AN2			I	Analog	Analog input 2.
VREF-			I	Analog	A/D Reference Voltage (Low) input.
RA3/AN3/VREF+	5	5			
RA3			I/O	TTL	Digital I/O.
AN3			I	Analog	Analog input 3.
VREF+			I	Analog	A/D Reference Voltage (High) input.
RA4/T0CKI	6	6			
RA4			I/O	ST/OD	Digital I/O. Open drain when configured as output.
T0CKI			I	ST	Timer0 external clock input.
RA5/AN4/SS/LVDIN	7	7			
RA5			I/O	TTL	Digital I/O.
AN4			I	Analog	Analog input 4.
SS			I	ST	SPI Slave Select input.
LVDIN			I	Analog	Low Voltage Detect Input.
RA6					See the OSC2/CLKO/RA6 pin.

Legend: TTL = TTL compatible input
ST = Schmitt Trigger input with CMOS levels
O = Output
OD = Open Drain (no P diode to VDD)
CMOS = CMOS compatible input or output
I = Input
P = Power

TABLE 1-2: PIC18F2X2 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	DIP	SOIC			
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/INT0	21	21			
RB0			I/O	TTL	Digital I/O.
INT0			I	ST	External Interrupt 0.
RB1/INT1	22	22			
RB1			I/O	TTL	
INT1			I	ST	External Interrupt 1.
RB2/INT2	23	23			
RB2			I/O	TTL	Digital I/O.
INT2			I	ST	External Interrupt 2.
RB3/CCP2	24	24			
RB3			I/O	TTL	Digital I/O.
CCP2			I/O	ST	Capture2 input, Compare2 output, PWM2 output.
RB4	25	25	I/O	TTL	Digital I/O. Interrupt-on-change pin.
RB5/PGM	26	26			
RB5			I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGM			I/O	ST	Low Voltage ICSP programming enable pin.
RB6/PGC	27	27			
RB6			I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGC			I/O	ST	In-Circuit Debugger and ICSP programming clock pin.
RB7/PGD	28	28			
RB7			I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGD			I/O	ST	In-Circuit Debugger and ICSP programming data pin.

Legend: TTL = TTL compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output
P = Power
OD = Open Drain (no P diode to V_{DD})

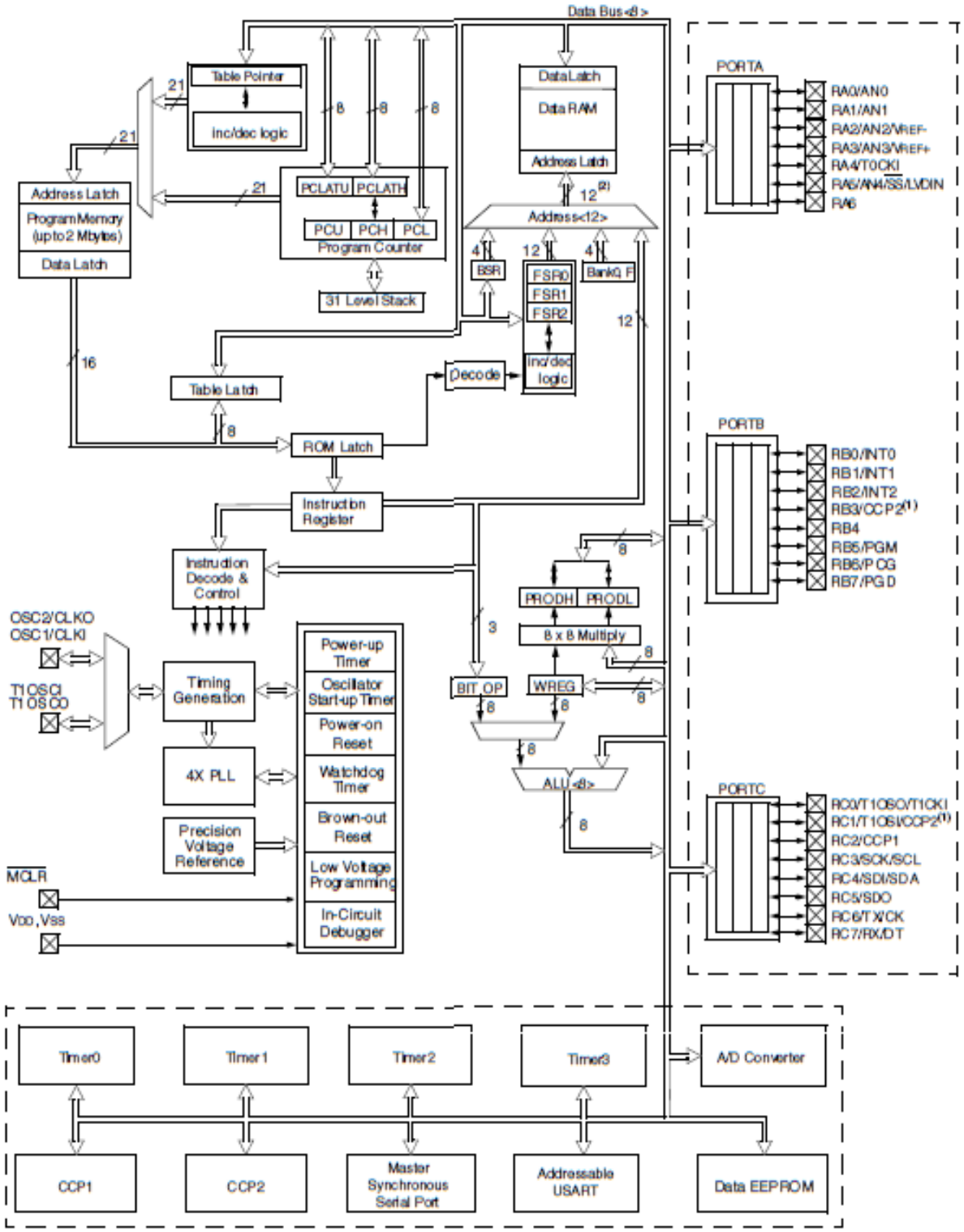
TABLE 1-2: PIC18F2X2 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	DIP	SOIC			
					PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	11	11			
RC0			I/O	ST	Digital I/O.
T1OSO			O	—	Timer1 oscillator output.
T1CKI			I	ST	Timer1/Timer3 external clock input.
RC1/T1OSI/CCP2	12	12			
RC1			I/O	ST	Digital I/O.
T1OSI			I	CMOS	Timer1 oscillator input.
CCP2			I/O	ST	Capture2 input, Compare2 output, PWM2 output.
RC2/CCP1	13	13			
RC2			I/O	ST	Digital I/O.
CCP1			I/O	ST	Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	14	14			
RC3			I/O	ST	Digital I/O.
SCK			I/O	ST	Synchronous serial clock input/output for SPI mode.
SCL			I/O	ST	Synchronous serial clock input/output for I^2C mode
RC4/SDI/SDA	15	15			
RC4			I/O	ST	Digital I/O.
SDI			I	ST	SPI Data In.
SDA			I/O	ST	I^2C Data I/O.
RC5/SDO	16	16			
RC5			I/O	ST	Digital I/O.
SDO			O	—	SPI Data Out.
RC6/TX/CK	17	17			
RC6			I/O	ST	Digital I/O.
TX			O	—	USART Asynchronous Transmit.
CK			I/O	ST	USART Synchronous Clock (see related RX/DT).
RC7/RX/DT	18	18			
RC7			I/O	ST	Digital I/O.
RX			I	ST	USART Asynchronous Receive.
DT			I/O	ST	USART Synchronous Data (see related TX/CK).
VSS	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

Legend: TTL = TTL compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output
P = Power
OD = Open Drain (no P diode to VDD)

Features	PIC18F252
Operating Frequency	DC - 40 MHz
Program Memory (Bytes)	32K
Program Memory (Instructions)	16384
Data Memory (Bytes)	1536
Data EEPROM Memory (Bytes)	256
Interrupt Sources	17
I/O Ports	Ports A, B, C
Timers	4
Capture/Compare/PWM Modules	2
Serial Communications	MSSP, Addressable USART
Parallel Communications	—
10-bit Analog-to-Digital Module	5 input channels
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes
Programmable Brown-out Reset	Yes
Instruction Set	75 Instructions
Packages	28-pin DIP 28-pin SOIC

FIGURE 1-1: PIC18F2X2 BLOCK DIAGRAM



Note 1: Optional multiplexing of CCP2 input/output with RB3 is enabled by selection of configuration bit.

2: The high order bits of the Direct Address for the RAM are from the BSR register (except for the MOVFF instruction).

3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

3.2 Software Choices

As mentioned earlier, the project has three main parts. In the first part, we deal with the sensors and their connection with the pic board. The second part's main concern is the mobile applications and data transfer via Bluetooth between the source device containing the sensors and the receiver device (most probably a mobile phone) which would publish the data. And this brings us to the third and the final part which is based on this action of publishing data over the web for the ones interested with this data. For all of these phases of the project, not despising the hardware side's weight, we would need to use some software as well as we develop some software.

3.2.1 Phase One: Sensors and the Pic

In the first part of the project, most of the time, we would be dealing with maintaining the connection and obtaining the flow of data between the output ports of the different sensors that we would use and the microcontroller that has a Bluetooth transceiver on it.

To be able to work on data the sensors give, without no doubt, we would first need to learn the way which each and every one of the sensors output the data. After having the connection between the output ports of the sensors and the analog or digital ports of the microcontroller, we would pack the data collected over the ports and push them to the Bluetooth device connected.

The important thing is, all through these parts of work on the microcontroller we would need some language and some software to communicate with it and have absolute control over it. We can use PIC C, PIC Basic or PIC Assembly as a language and a proper compiler to be able to obtain the hex files that will tell the microcontroller what to do in a way that it would understand. Apart from those, we may also need simulators and debuggers to detect the errors, make tests easier and in a simultaneous way.

We are planning to code in PIC C and PIC Assembly languages when needed.

An assembly language is a low-level language for programming computers. It implements a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on abbreviations (called mnemonics) that help the programmer remember individual instructions, registers, etc. As an appropriate example, PIC Assembly language is specific to Programmable Interface Controller architecture.

The most popular development tool for PIC Programming is MPLAB IDE by Microchip. MPLAB Integrated Development Environment (IDE) is a free, integrated gcc-based toolset for the development of embedded applications employing Microchip's PIC and dsPIC microcontrollers. The MPLAB IDE runs as a 32-bit application on Microsoft Windows, and includes several free software components for application development, hardware

simulation and debugging. MPLAB also serves as a single, unified graphical user interface for additional Microchip and third party software and hardware development tools.

Both Assembly and C programming languages can be used with MPLAB. Others may be supported through the use of third party programs. Support for MPLAB and tutorials can be found easily on the web. However, MPLAB does not support Linux, Unix, or Macintosh based operating systems.

There are also some open source development tools. The following are available for the PIC family under the GPL or other free software or open sources licenses:

FreeRTOS is a real-time operating system for embedded devices, being ported to several microcontrollers. It is distributed under the GPL with an optional exception. The exception permits users' proprietary code to remain closed source while maintaining the kernel itself as open source, thereby facilitating the use of FreeRTOS in commercial applications. The exception also prevents users from comparing FreeRTOS with other RTOSs, except with permission from the author.

GPUTILS is free and available from the GPUTILS website.

GPSIM is an Open Source simulator for the PIC microcontrollers featuring hardware modules that simulate specific devices that might be connected to them, like LCDs.

SDCC supports 8-bit PIC micro controllers (PIC16, PIC18). Currently, throughout the SDCC website, the words, "Work is in progress", are frequently used to describe the status of SDCC's support for PICs.

KTechlab is a free IDE for programming PIC Microcontroller. It allows one to write the program in C, Assembly, Microbe (a BASIC-like language) and using Flow Chart Method.

For debugging and simulations, we also have various options:

First of all, MPLAB includes a software emulator for PICs. However, software emulation of a microcontroller will always suffer from limited simulation of the device's interactions with its target circuit.

Proteus VSM is a commercial software product developed by Labcenter Electronics which allows simulation of many PIC micro devices along with a wide array of peripheral devices. This method can help bridge the gap between the limited peripheral support offered by the MPLAB simulator and traditional in-circuit debugging/emulating. The product interfaces directly with MPLAB to offer a schematic display of signals and peripheral devices.

KTechLab has a circuit simulator for KDE which features simulating some types of PIC microcontrollers besides many other analog and digital parts.

Piklab is a free and open source IDE for developing PIC software on KDE which is able to simulate and debug PIC software using another free and open source tool called gpsim as backend.

Process, Details and Issues:

The sensors will periodically -10 seconds or less for critical sensors and 60 seconds or more for the others- produce outputs which are going to be collected and checked by the PIC program and then the packed data will be transmitted to the Bluetooth device connected. Throughout this flow of data from the sensors to the BT device, there will be pauses and triggering processes caused by alarm conditions. One of the most important alarm conditions is occurred when the critical sensors output a value outside of the predetermined range. Another one will happen when the patient hits the button integrated on the module having the sensors, PIC and BT device. One other example may be the situation when one or more of the sensors run out of power and stop giving outputs. In an alarm condition, PIC will interrupt the process and send an appropriate alarm code via BT immediately.

During the normal flow of the data there are some points to focus on. Firstly, the format of the sensors data should be standardized. Each sensor may have a different format for outputs and even different output types. One may need an analog pin while the other one requires digital. The PIC program that we will develop will handle all of the possible conditions and transform these unformatted inputs taken from the sensors to a standardized form and then pack them. We need to be more specific about the sensor output types and formats at this point. However, thanks to the company that we had contacted for information about an all-in-one device containing some of the sensors that we would include in our project, we have no further information for now. But we are working on a way of gathering the necessary information without the sensors. We are planning to emulate the sensors in a realistic manner. Another point is the size of the input and the output of PIC. The size should be adjusted for the optimal progressing of the device. By this, we mean the output transmitted to the BT device should be packed so that the size of the packet should not exceed a threshold value but should contain enough data. The frequency of transmitting data and the average time cost of one transfer over BT should be calculated to obtain the cost. The priority issues while having the sensors data is another point to be considered important. The data from vital sensors should be given more priority both in data collection and packaging parts of this phase. Since the size will matter, we need to add the vital data to the packages before any other.

3.2.2 Phase Two: Mobile Application and Bluetooth

In this second and the most challenging part of the project, we will dive into a world that we are not familiar with. Mobile Applications were always out of our scope until this project. The other important point is the Bluetooth connection requirement of this application. Neither the standalone mobile application design part nor the Bluetooth data transfer part is a piece of cake so we know that we should read, learn and work more for this particular phase.

Mobile software is designed to run on handheld computers, personal digital assistants (PDAs), enterprise digital assistants (EDAs), smart phones and cell phones. For mobile software development, we would need lots of options when it has come to development platforms.

The dominant mobile software platform is Java (in its incarnation as "J2ME" / "Java ME" / "Java 2 Micro Edition"). J2ME runs atop a *Virtual Machine* (called the KVM) which allows reasonable, but not complete, access to the functionality of the underlying phone. The JSR process serves to incrementally increase the functionality that can be made available toJ2ME, while also providing Carriers and OEMs the ability to prevent access, or limit access to provisioned software.

This extra layer of software provides a solid barrier of protection which seeks to limit damage from erroneous or malicious software. It also allows Java software to move freely between different types of phone (and other mobile device) containing radically different electronic components, without modification. The price that is paid is a modest decrease in the potential speed of the game and the inability to utilize the entire functionality of a phone (as Java software can only do what this middle-man layer supports.)

Because of this extra security and compatibility, it is usually a quite simple process to write and distribute Java mobile applications (including games) to a wide range of phones. Usually all that is needed is a freely available JDK (Java Development Kit) for creating Java software itself, the accompanying Java ME tools for packaging and testing mobile software, and space on a web server (web site) to host the resulting application once it is ready for public release.

Symbian is very powerful for general purpose development. The Symbian based S60 platform is strongly supported by Nokia with some support from other device manufacturers. In Japan NTT DoCoMo's Symbian based MOAP platform is also well supported by a number of manufacturers (Fujitsu, Sony Ericsson Japan, Mitsubishi and Sharp amongst others). It should be noted, however, that MOAP is not an open development platform. Another Symbian based platform, UIQ, is less well supported (principally by Sony Ericsson and Motorola).

iPhone and iPod touch development with the iPhone SDK is ideal for quickly developing applications for users of the iPhone. iPhone apps must be cleared for approval to Apple before being listed on the app store. The programming language used is Objective C, based on the C programming language. Currently, the iPhone SDK is only available on Mac OS X 10.5.

Lazarus is ideal for prototyping and quickly developing database powered applications. Also useful for porting Object Pascal software to mobiles. Can access the native APIs when translated headers are available.

BREW is ideal for deploying applications for deployment on CDMA-based networks (also supports GPRS/GSM models) with a deployed Brew Content Platform especially if OTA app deployment is desired. Little penetration in Europe.

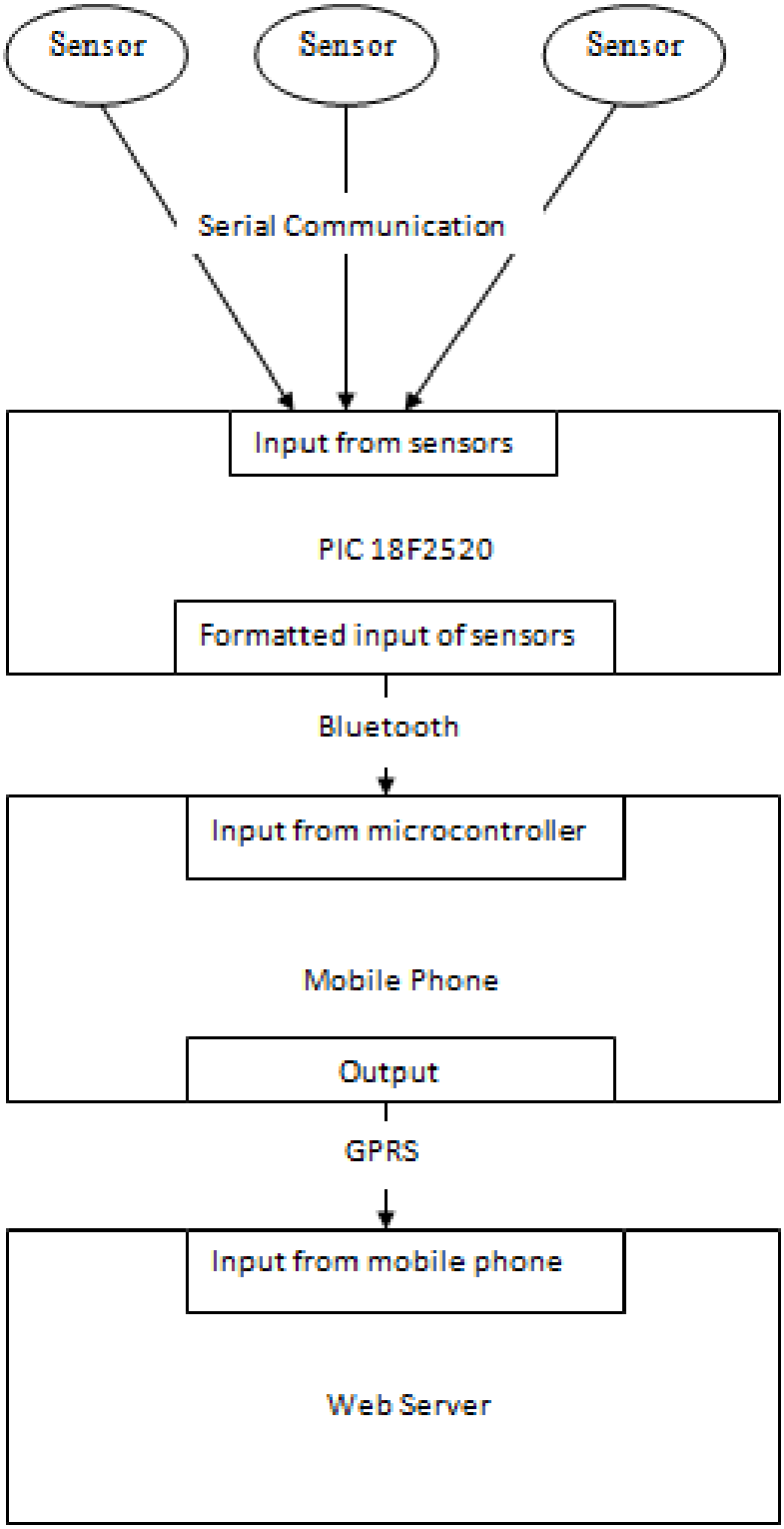
Python is ideal for initial prototyping and concept testing when functionality falls outside Java ME.

Recently announced by the Open Handset Alliance, whose 34 members include Google, HTC, Motorola, Qualcomm, and T-Mobile, Android is a new Linux-based platform currently available only as a developer pre-release. Although it does not yet have any fielded implementations, its support by 34 major software, hardware and telecoms companies makes it likely that it will be rapidly adopted from 2008. The Linux kernel is used as a hardware abstraction layer (HAL). Application programming is exclusively done in Java. You need the Android specific Java SDK. Besides the Android Java Libraries it is possible to use normal Java IDEs.

4. ARCHITECTURAL OVERVIEW AND INTERFACE DESIGN

4.1 Architectural Overview

If we look at the system as a whole it is easy to observe that, it is distributed, and consists of different platforms to develop software. This makes the partitioning of the system easier, but brings the difficulty to specialize in multiple programming paradigms. The abstract architecture of the whole system can be seen in the below figure.


Sensor
Sensor
Sensor
Serial Communication
Input from sensors
PIC 18F2520
Formatted input of sensors
Bluetooth
Input from microcontroller
Mobile Phone
Output
GPRS
Input from mobile phone
Web Server

As can be seen from the figure also, there is not a center point for the whole system where everything is managed. In fact the system can be viewed as a series data transfers starting from the sensors and ending in the web server.

Data flow is the primary process, thus in the figure which is the abstract system only data transfer parts are shown. There are additional parts employed in each of the points where data passes through.

Three different programming platforms will be used to achieve the final product, first is PIC programming, second is mobile application programming, and the last one is web programming. Platforms are quite diverged from each other.

Microcontroller programming part consists of three major parts. In the first part where measurements of the sensors are sent, there should be a robust PIC program to establish a reliable serial connection between the sensors and microcontroller. After that received data should be compressed in an efficient way to be ready to send. In the last part data is put into the ports of microcontroller hence bluetooth device can read and send it to the mobile phone. All of these tasks will be coded in C, MPLab the official IDE of the Microchip provides this feature.

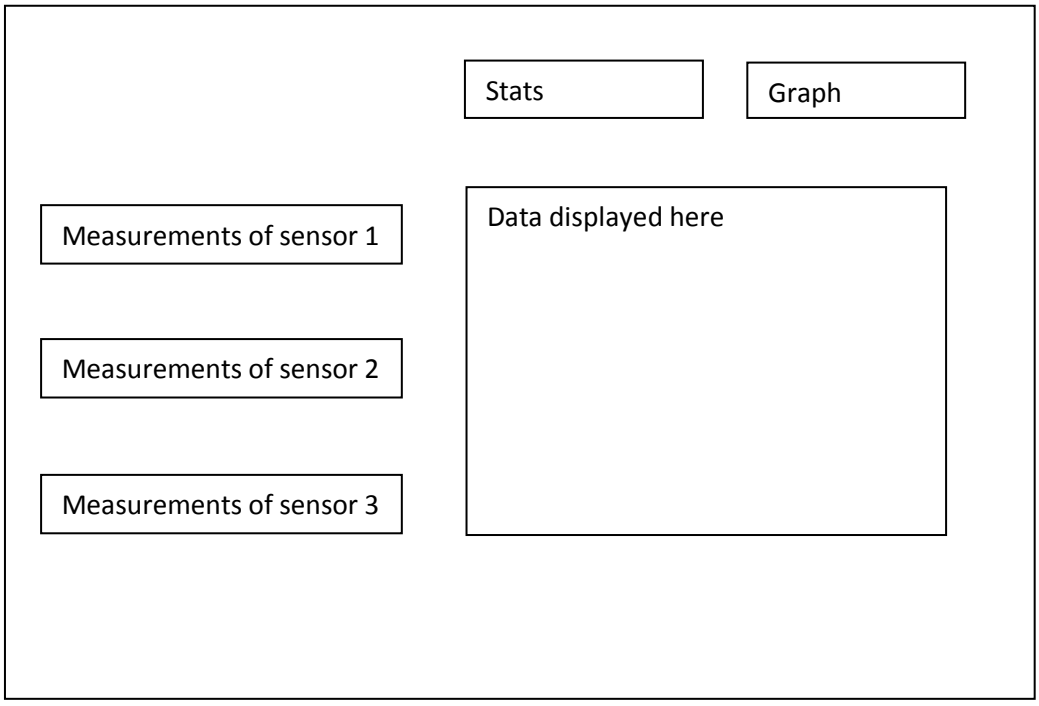
Mobile programming part is the first part where received data is evaluated. Because in the previous parts data was only received, packaged and sent. Our mobile program will check the values in the data package and will run script codes to send notifications to the doctor and relatives of the patient. We'll use built-in methods of J2ME to supply this. But mobile program will contain more vital parts than this. It should receive data from the Bluetooth device and send it readily to web server via GPRS.

Web programming part includes also database programming, otherwise it'd be impossible to monitor the conditions of each patient for such a big system. Data stored in the web server will be combined with a user-friendly web interface to provide users (doctors, patients, and their relatives) an easy way to monitor the latest conditions.

4.2 Interface Design

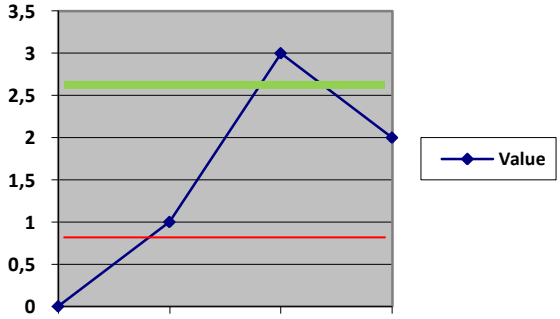
The system, if looked at the big picture, is based on the transferring and monitoring of data. Thus presentation of data (consists of vital values of the patient) is of great importance. Two separate interfaces would be implemented one for the mobile program and the other for the website.

Since screen size and properties of a mobile phone is limited, the interface in the program will be simpler. A generic representation of this interface is in the below figure.

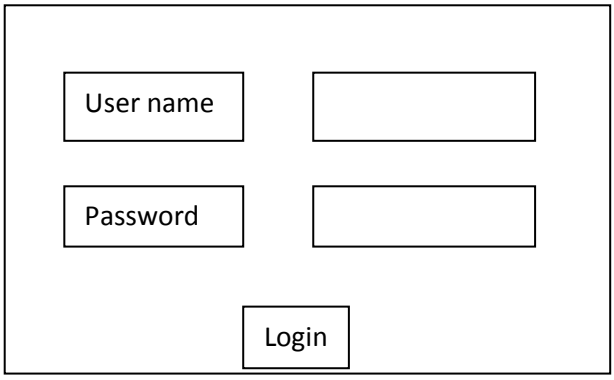


The user can select in which way he/she is going to view the data from the upper part, when stats is selected measurements are displayed line by line in time order. Values bigger or smaller than threshold values are differently colored.

In the graph mode the values will be displayed in a graph, there will be two lines to indicate min and max norms. In this symbolic graph below, green line denotes the max norm value and the red line denotes the min norm value.



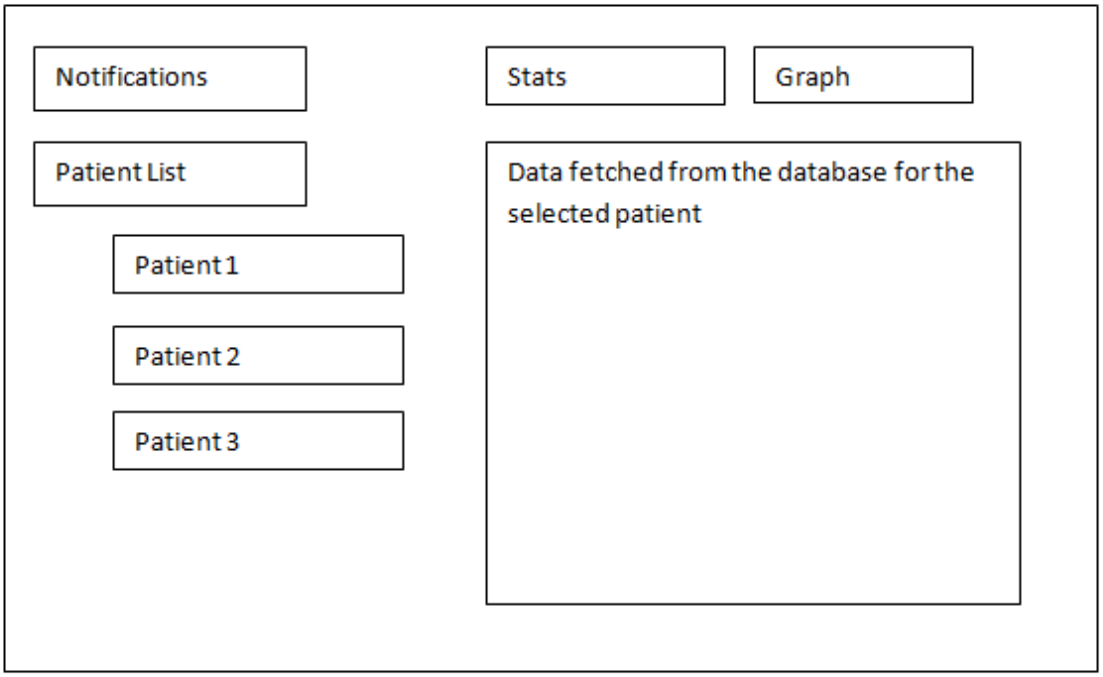
The web server's interface will contain a form application. Initially it will have a login screen when page is displayed.



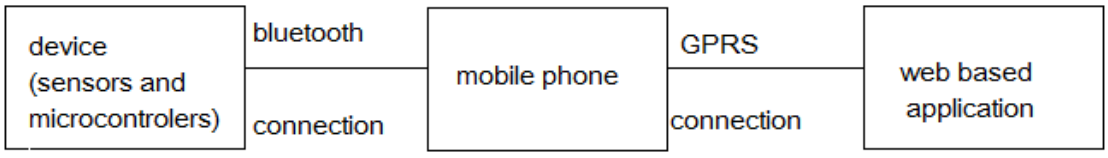
If the user is a patient or relative of a patient a screen similar to the one in the mobile program is displayed, since these users can monitor only one patient's data.

Otherwise if the account is owned by a doctor he/she will have more options. In the notifications section the doctor can view the patients with alarm conditions. It can also be displayed by selecting the patient but notifications will be quick link for the doctor.

Patient list will be expandable/shrinkable so that it will not take a large place in the page if desired. The right frame of the page will also be similar to the page which patient/relatives view. Data fetched from the database can be displayed either in numbers or graphs.



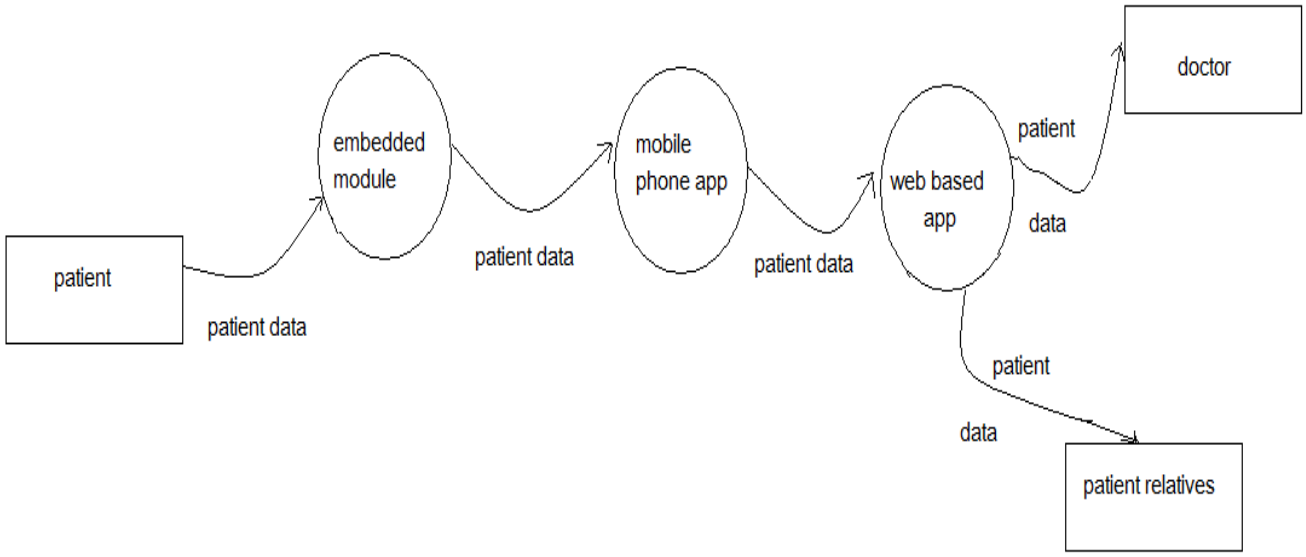
5. DETAILED DESIGN



ARCHITECTURAL OVERVIEW

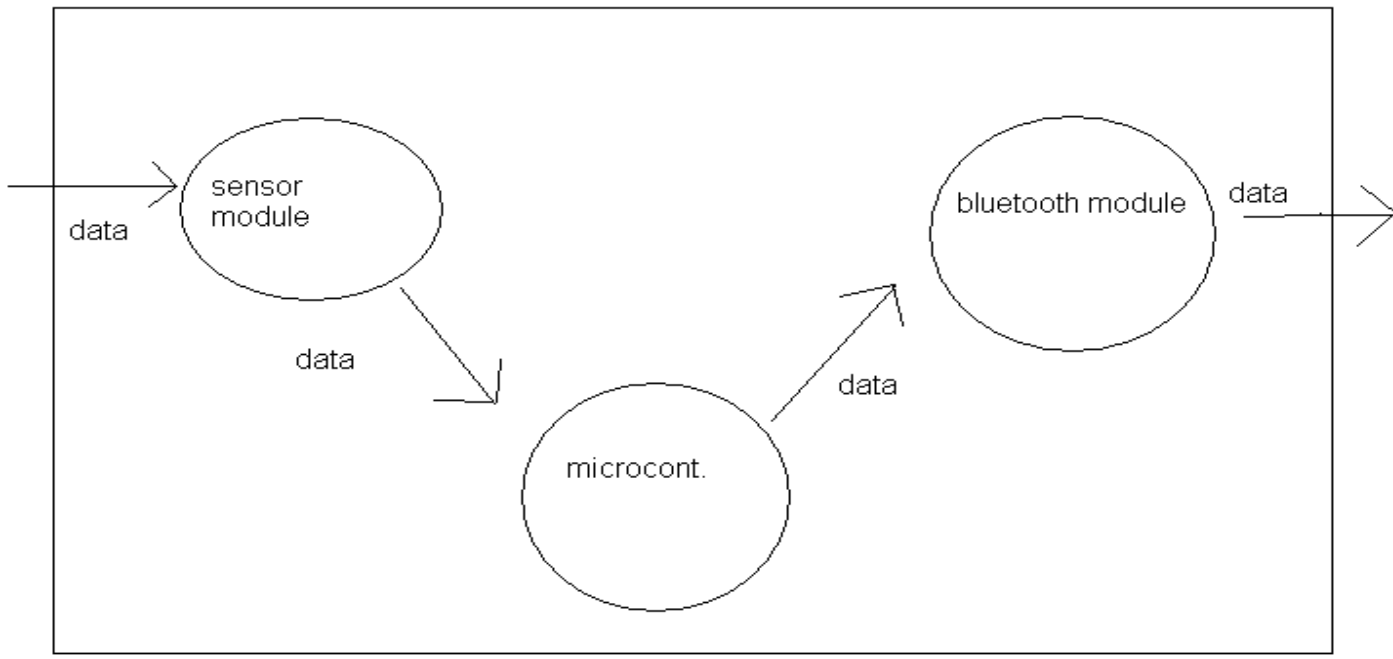
5.1 DATA FLOW DIAGRAMS

DFD LEVEL 0



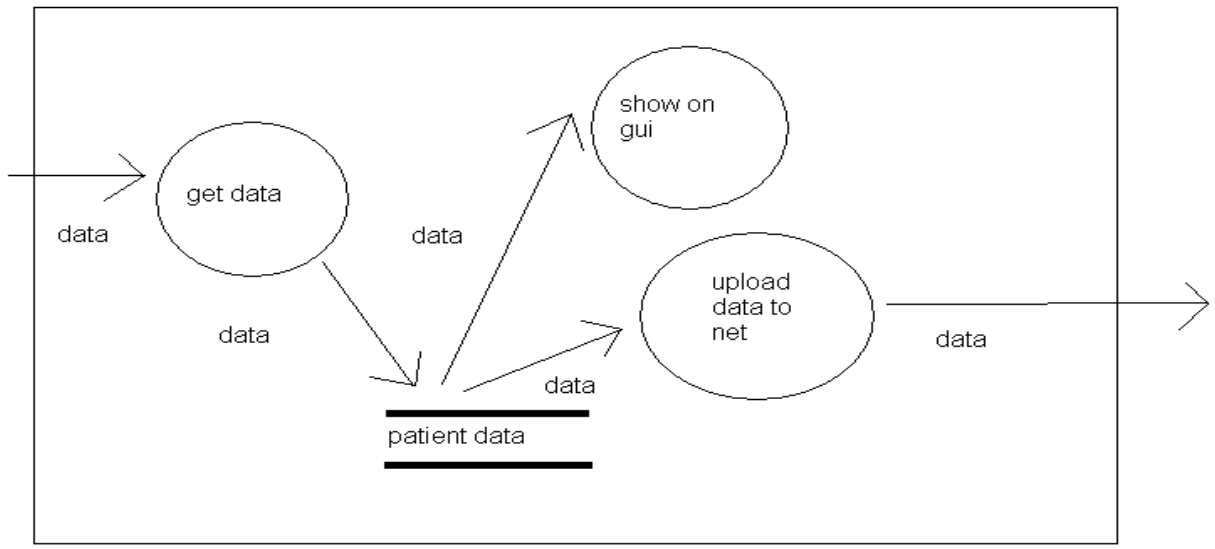
Level 0 DFD

LEVEL: 1 EMBEDDED MODULE DFD



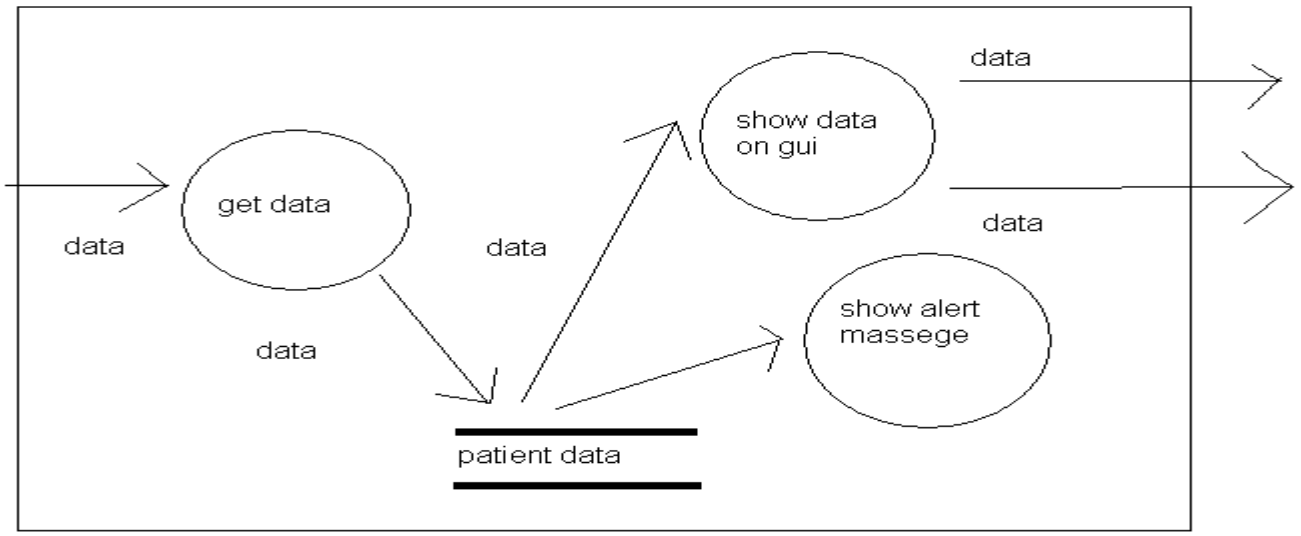
Level 1 Embedded Module DFD

LEVEL: 1 MOBILE APPLICATION



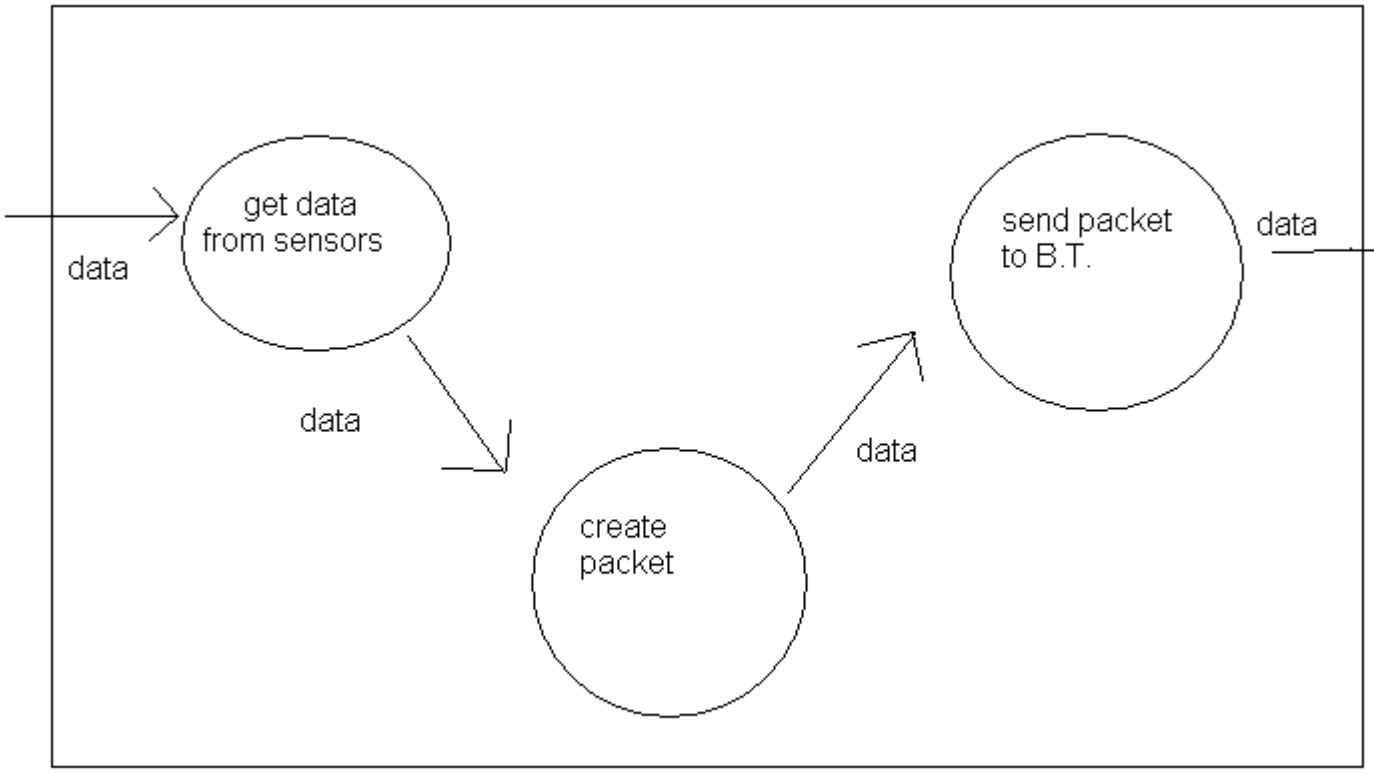
Level 1 Mobile Application DFD

LEVEL: 1 WEB APPLICATION DFD



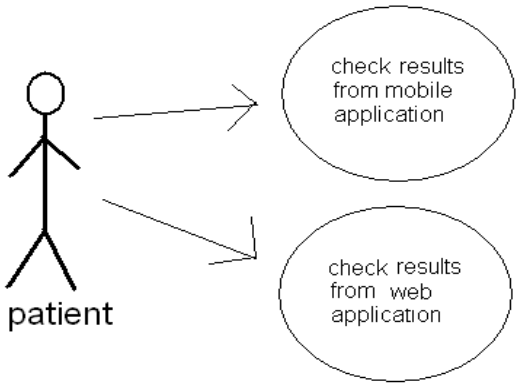
Level 1 Web Application DFD

LEVEL: 2 MICROCONTROLLER DFD

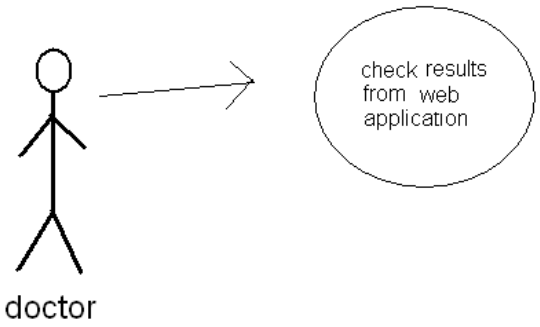


Level 2 Microcontroller DFD

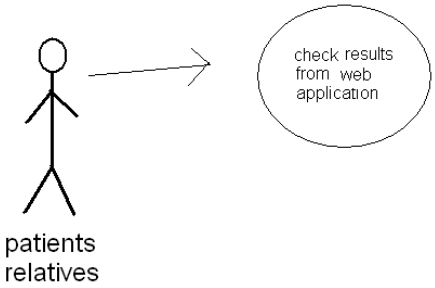
5.2 Use Case Diagrams



Patient Use Case Diagram

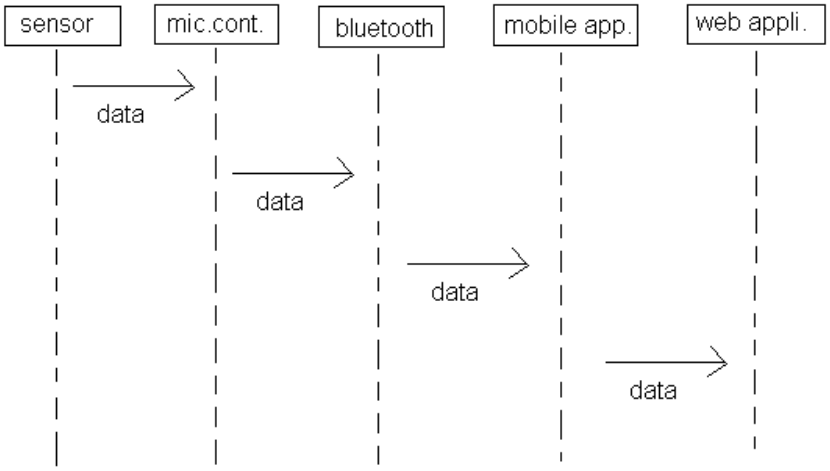


Doctor Use Case Diagram



Relatives Use Case Diagram

5.3 Sequence Diagram



Sequence Diagram

6. CONCLUSION

We prepared this Initial Design Report to create a bridge between the design and implementation parts of the project.

First, we determine the hardware and software requirements and find the best choice of hardware. After searching about them, we create our diagrams which will be reliable guides for the next phases of the project.

We believe that, this report will help us for the future of the project, especially for the Detailed Design Report which is one of the important reports for the process of the project.