DESIGN OF AN EMBEDDED COMMUNICATION FRAMEWORK FOR AN EMERGENCY TELECARDIOLOGY SYSTEM

by

ONUR YILDIRIM

B.S., Electronics and Communication Engineering, Istanbul Technical University, 2003

Submitted to the Institute of Biomedical Engineering
in partial fulfillment of the requirements
for the degree of
Master of Science
in
Biomedical Engineering (or Biomedical Science)

Boğaziçi University
June 2006

ACKNOWLEDGMENTS

First and foremost, I would like to thank my thesis advisor, Dr. Ahmet Ademoğlu, for all the help and mentoring he has provided me to complete this thesis. And, I would like thank him again for the patience he has shown in last few weeks as I was struggling to complete the typesetting of the thesis.

I also would like to thank my colleagues at the EKGNET team, Mr. Tuğrul Anıldı, Mr. Adnan Kurt and dear friend Baran Dilber. Without the hard work they put in EKGNET this thesis would not even exist. I would like to thank Mr. Tuğrul Anıldı especially for writing the hardware related firmware modules.

I would like to thank Dr. Ata Akın and Dr. Murat Gülsoy from the Biomedical Engineering Institute for all the interest and help they have provided in the course of the last two years.

I would like to thank Dr. Tamer Demiralp from Istanbul University, Physiology Department for all his support during the project.

Finally, I would like to thank my family for all their support and encouragements during the course of my life.

ABSTRACT

DESIGN OF AN EMBEDDED COMMUNICATION FRAMEWORK FOR AN EMERGENCY TELECARDIOLOGY SYSTEM

Acute myocardial infarction (AMI or MI) occurs when a part of the heart muscle dies because of sudden total interruption of blood flow to that area. It is a life-threatening medical emergency which demands immediate activation of the emergency medical services. This thesis proposes the development of an embedded communication framework designed to enable quick diagnosis of AMI and immediate activation of emergency medical services targeted to it. The system consists of an embedded communication software along with a TCP/IP based server software for a GSM based ambulatory ECG device. Both, the software components running on the ECG device and the communication server enable the device to be remotely interfaced by the call center software and controlled by the cardiologists.

Keywords: Mobile ECG device, AMI, Pre-hospital thrombolysis, 12-lead ECG over GSM

ÖZET

ACİL TELEKARDİYOLOJİ SİSTEMİ İCİN GÖMÜLÜ İLETİŞİM ÇERÇEVESİ TASARIMI

Akut Miyokard İnfarktüsü (AMI veya MI) kalp kasının belli bir bölgesine giden kan akışının aniden kesilmesi sonucu bu bölgedeki kas dokusunun ölümünden kaynaklanır. Yaşamı tehdit eden bu durum acil tıp hizmeti örgütlenmesinin hızla harekete geçirilmesini gerektirir. Bu tez, erken AMI tanısı ve ilgili acil tıp hizmetlerinin harekete geçirilebilmesine olanak tanımak amacıyla geliştirilen gömülü bir iletişim çerçevesi önermektedir. Sistem, GSM tabanlı taşınabilir ECG aygıtı için TCP/IP temelli sunucu yazılımıyla, aygıt üzerinde çalışan gömülü iletişim yazılımından oluşmaktadır. ECG aygıtı üzerindeki yazılım bileşenleri ve iletişim sunucusu cihazın çağrı merkezinde kardiyologlarca denetlenen yazılımla uzaktan arayüzlenmesini sağlamaktadır.

Anahtar Sözcükler: taşınabilir ECG aygıtı, AMI, hastane öncesi tromboliz, GSM üzerinden 12 kanal ECG iletimi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADC	Analog to Digital Converter
AMI	Acute Myocardial Infarction
ACLS	Advanced Cardiac Life Support
CTS	Clear To Send
DCD	Data Carrier Detected
ECG	Electrocardiogram
FAT	File Allocation Table
FSM	Finite State Machine
GSM	Global System for Mobile communication
I/O	Input/ Output
IP	Internet Protocol
ISR	Interrupt Service Routine
LED	Light Emitting Diode
OOP	Object-oriented programming
RS-232	Recommended Standard 232
SPI	Serial Peripheral Interface
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TDD	Test-driven development
TÜMAR	Türkiye Akut Miyokard İnfarktüsü Araştırması
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
WHO	World Health Organization

1. INTRODUCTION

This thesis aims to develop a viable pre-hospital thrombolysis method for immediate treatment of heart attacks. To achieve this goal, an infrastructure for correct assessment of coronary disease patients has to be constructed. Two components of a proposed infrastructure, an embedded communication software running on a ambulatory GSM based ECG device and a TCP/IP based communication server software, has been developed within the scope of this thesis.

1.1 Motivation

Acute Myocardial Infarction (AMI) is a life-threatening medical emergency which demands immediate activation of the emergency medical services. Immediate transport by ambulance to a hospital where advanced cardiac life support (ACLS) and modern AMI therapeutics are available is of crucial importance. The more time that passes before medical attention is sought, the more severe the permanent heart damage is likely to be, and the less likely survival will be [1].

Classical cases of myocardial infarction are often identified by ambulance staff or emergency room doctors without further investigations. Nevertheless, for a complete diagnosis, the medical history, combined with electrocardiogram results and blood tests, is vital [1].

In 2004, the total cost of cardiovascular disease in the USA was estimated at $368.4 billion. This figure includes both direct costs (physicians and other professionals, hospital and nursing home services, medications, home health care and other medical durables) and indirect costs (calculated from lost productivity resulting from morbidity and mortality) [2].

AMI is the most important mortality cause in Turkey and coronary mortality rate in Turkey is the highest among other European countries. This is mainly due to the fact that in Turkey, %71 of the AMI cases arrive to hospitals in 6 hours, %13 of the cases arrive between 6-12 hours and %16 arrive after 12 hours [3]. Considering the fact that most of the emergency therapeutics of AMI are only effective within 2-3 hours after the onset of AMI symptoms, developing an effective emergency medical service to improve the effectiveness of modern AMI therapeutics proves to be very important.

1.2 Goals

This thesis proposes the development of embedded communication framework for a GSM based 12-lead clinical ECG device and a Cardiology Call Center specifically designed to enable quick diagnosis of AMI and immediate activation of AMI targeted emergency medical services. The goal of this thesis is to develop a communication server software and an embedded software as the firmware of the mobile ECG device. These two components will communicate with each other by the rules of an application protocol and together they establish the embedded communication framework for the proposed infrastructure.

The embedded software accepts commands from the remote service headquarters. These commands are transmitted over a TCP/IP based communication channel between the device and the server, enabling the device to be remotely controlled by cardiologists. The main function of the embedded software is to accomplish data acquisition, data storage, data processing, data compression, data encoding and establishment of a communication channel between the device and the service headquarters to receive commands and transmit ECG data. It also checks for device failures and responds to user interface events. Server software is responsible for authenticating devices and users, decoding ECG data sent by the device and provide a device interface for the call center software allowing the device to be remotely controlled from the headquarters or a cardiologist connected to the internet.

1.3 Organization of the Thesis

Chapter 2 presents the contextual background behind the accomplished work. For that purpose, EKGNET project and two related components are thoroughly explained. Chapter 3 then presents an overview of the firmware running on EKGNET Patient Unit. The firmware consists of modules and some details about these modules are given. Chapter 4 focuses on the server software and its design criteria. Detailed information about clients and related software classes are also provided. Chapter 5 describes the application protocol used for the transfer of the ECG data over TCP/IP based communication channel. Finally, Chapter 6 discusses the results and concludes the thesis toward a future outlook.

2. BACKGROUND

2.1 Acute Myocardial Infarction

Acute myocardial infarction (AMI or MI), commonly known as a heart attack, is a serious, sudden heart condition usually characterized by varying degrees of chest pain or discomfort, weakness, sweating, nausea, and vomiting, sometimes causing loss of consciousness. It occurs when a part of the heart muscle dies because of sudden total interruption of blood flow to that area [1]. Coronary heart disease, the main cause of acute myocardial infarction, is today the single largest killer worldwide. It has become a true pandemic and its incidence is still rising. According to the WHO, in 2002 7.2 million deaths worldwide resulted from coronary heart disease [4]. Electrocardiogram (ECG) findings suggestive of MI are elevations of the ST segment and changes in the T wave. After a myocardial infarction, changes can often be seen on the ECG called Q waves, representing scarred heart tissue [1].

Pre-hospital prediction of the final diagnosis is based only on a snapshot of the clinical history and a single ECG recording, but can be reasonably accurate. With clinical assessment alone, the diagnostic accuracy of experienced clinicians is about 75%. With the addition of the ECG, accuracy may be increased to 90-95% [5]. The presence of ST-segment elevation on ECG appears mostly within minutes after onset of symptoms and is the most sensitive and specific marker for AMI. New ST-elevations are found in 80-90% of AMI but only 30-40% of patients have ST-elevations on the hospital admission. ST-elevations are more marked in men than in women. ST-depressions indicate myocardial ischaemia but only 50% of these patients will eventually develop an AMI. Symmetrical T-wave inversions are a non-specific sign which indicate ischaemia, myocarditis and pulmonary embolism. About one third of the patient with this sign and chest pain will develop an AMI. About one third of patients admitted with acute chest pain to the emergency department have a normal ECG, yet 5-40% of them have an evolving AMI. The early case fatality rate is highest among patients with ST-

elevation, intermediate among patients with ST-depression and lowest among patients with T-wave inversion [6].

2.2 Pre-hospital Thrombolysis

Thrombolytic, or clot-busting drugs rapidly destroy unwanted blood clots. They can be used in the early treatment of some heart attacks. Heart attacks are caused by a blood clot blocking the supply of blood to an area of the heart muscle. As a result the affected area of heart muscle is damaged or dies, and the patient may develop longer term heart problems or may die [7]. Thrombolytic drugs work by dissolving the fibrin mesh that binds the blood clot together. This can greatly improve the flow of blood to the affected area of the heart and prevent ill health or death in some cases [7]. One reason why people should get to hospital as quickly as possible after a suspected heart attack is because thrombolytic drugs can only be used in the first few hours after a heart attack. The sooner they are used the more likely they are to work well [7]. According to the WHO, in both developed and developing countries, 40 to 75% of all heart attack victims die before reaching the hospital [8]. According to the TÜMAR research conducted in Turkey, a median of 4 hours passes until a AMI patient is treated [3].

The objective of pre-hospital thrombolysis is to resuscitate the patient, if necessary, and to reduce myocardial damage and complications following an AMI, prior to arrival to hospital [9]. Pre-hospital thrombolysis aims to shorten time to treatment. The average elapsed time from symptom onset to treatment has been three hours ever since the GUSTO I trial in 1993 [10]. However, the ASSENT-3 PLUS trial of pre-hospital tenecteplase has broken that barrier. In ASSENT-3 Plus, a 1639-patient multinational study, elapsed time from symptom onset to treatment was shorter by 47 minutes compared with the ASSENT-3 in-hospital trial. This was achieved with a 30-day mortality rate that dipped as low as 6%, a remarkable achievement, considering that the patients in this trial were at higher risk than those enrolled in previous ASSENT studies [11].

The safety of pre-hospital thrombolysis is strongly dependent on correct diagnosis, usually by means of a standard 12-lead ECG that can either be transmitted via telephony for interpretation by a cardiologist or interpreted on-site by specifically designed computer programs. However, once an appropriate infrastructure is in place, the benefits of pre-hospital thrombolysis are unequivocal. It has been estimated that each 30-minute delay in giving thrombolytic therapy reduces patients' life expectancy by an average of one year [12].

2.3 Telecardiology

The notion of using telecommunications in the healthcare industry goes back to early 1900s. There had been experiments using radio telecardiology (from the 1910s), telephone-mediated telestethoscopy (from the 1920s), and radiology image transfer and videophone experiments (from the early 1950s) [13]. Einthoven's initial works presented in 1906 described the transmission of ECG information over telephone wires [14]. Although at the time this was not labeled as a telemedicine system, or more specifically *telecardiology*, the principles of remote ECG transmission were demonstrated. The underlying fundamentals of the process have changed little in virtually a century. However, technological advancements have allowed the development of lightweight, portable recording systems (in comparison with the string galvanometer), the application of mobile phone technology as methods of transmission (in comparison with telephone wires) and standard desktop personnel computers (PCs) as the display medium (in comparison with optical projection systems) [15].

Telecardiology is the practice of cardiology over telemedicine systems enabling cardiologists and patients to reside in remote locations. Currently most of the telecardiology applications focus on remote monitoring of cardiac patients via mobile holter devices. Holter monitoring is an important tool since the 1970s and it has become a very useful tool for analyzing intermittent arrhythmias [16]. However, a certain probability remains that arrhythmias can symptomatically occur outside of the monitoring period. In these cases, transtelephonic monitoring has proven to be useful in therapy:

If an arrhythmia occurs, the patient may initiate ECG recording and transmission [17]. As of today, some of the holter devices are capable of recording and transmitting low resolution ECG recordings to cardiology centers where arythmias and other disorders are tracked. A one channel 3 leads transtelephonic ECG signal acquisition system along with a temperature sensor was developed by Yüksel Yazıcı on his work for his Master's thesis [18]. This device is very suitable to be used as a holter device.

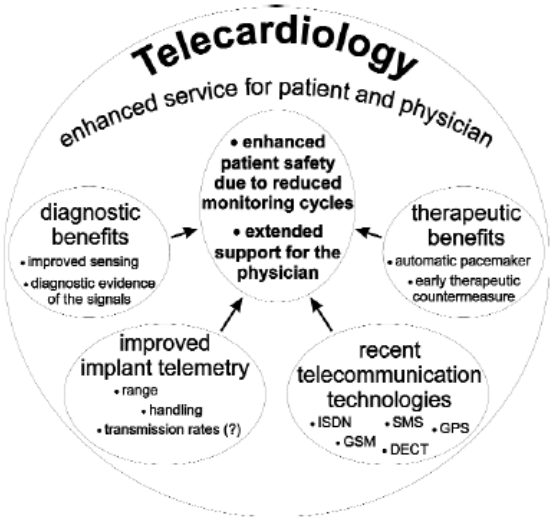

Figure 2.1 Telecardiology: modules and aspects

Another important field of application for telecardiology proves to be the recording and transmission of 12-lead clinical ECG data for interpretation. In this field, some specific applications are designed to be utilized in emergency situations. Most widely used emergency services targeted system is the MobiMed system produced by Ortivus AB [19]. But due to high total cost of ownership of these systems, most emergency healthcare services cannot record, interpret, or transmit 12-lead ECG.

2.4 EKGNET

The main purpose of EKGNET project is to create an infrastructure for treating AMI patients in pre-hospital settings. EKGNET aims the transfer of 12-lead, clinical quality ECG signals recorded in emergency situations to a cardiac consultation center to enable quick diagnosis and pre-hospital treatment of cardiac patients. In addition

to that the TCO of overall system is designed to be significantly lower than currently existing solutions like MobiMed. The components of EKGNET is shown in Fig. 2.2

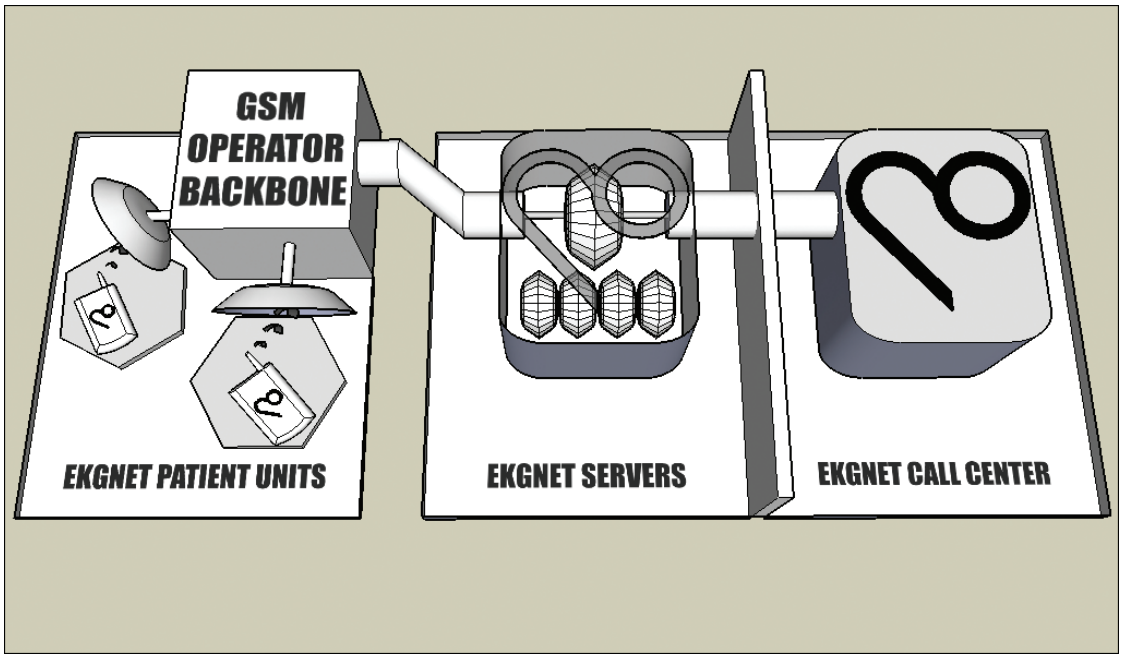

Figure 2.2 EKGNET Overview

The first component, the EKGNET Patient Unit as shown is a mobile 12-lead clinical ECG device capable of recording high quality ECG data and transmit this data through GSM. Further information about Patient Unit is provided in section 2.4.1. The second component, the EKGNET Communication Server routes calls from devices to Call Center cardiologists. More information about this component is given in section 2.4.2.

The third component, the EKGNET Call Center is an emergency cardiac call center with resident cardiologists responding to the AMI cases 7 days 24 hours. Details about the Call Center are provided in section 2.4.3.

2.4.1 Patient Unit

EKGNET Patient Unit is a mobile and battery powered, 12-lead clinical ECG device. The unit is capable of recording high quality 12-lead ECG data and transmit-

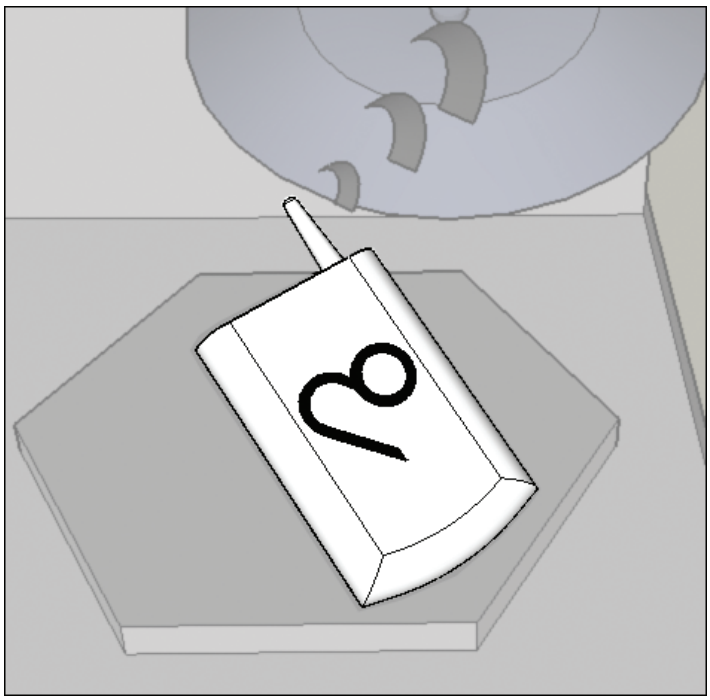

Figure 2.3 EKGNET Patient Unit

ting it via GSM network. It is designed to record and transmit high quality ECG data in ambulatory environments. Patient Unit is show in Fig. 2.3. Operation of Patient Unit is very easy. It has a single button on its panel which turns on the device and initiates connection to Call Center. Operating cardiologist at the Call Center intercepts this connection request and takes control of the device remotely. After this point, Patient Unit is operated directly from the Call Center by the cardiologist. Thus, the person handling the device has to focus only on the correct placement of ECG electrodes.

Technical details of Patient Unit are as follows:

- 12-lead clinical ambulatory ECG device,
- simple and failsafe operation,
- simultenaous recording of all channels,
- store-and-forward operation,
- separate 24bit sigma-delta ADC for each channel,

- 500 Hz sampling rate,
- 0.2 uV effective resolution,
- DC-250Hz signal bandwidth,
- low power operation, very long sleep span,
- embedded 900/1800 MHz GSM modem with TCP/IP stack,
- remotely controlled from EKGNET Call Center,
- powered by two standard AA batteries.

2.4.2 Communication Server

EKGNET Communication Server manages the communication channel between the Patient Unit and Call Center application. GSM operators adhere to strict security measures to protect their customers privacy. Because of these measures, creating a connection within to the operator backbone is prohibited. Communication Server works as a proxy server between the Patient Units and Call Center. When the Patient Unit is activated, a GRPS call is initiated to the GSM operators backbone. GSM Operator transfers this call to Communication Server and any data transferred from the device is forwarded to Communication Server. After a communication channel is securely created, data transmission to the Patient Unit is also possible.

2.4.3 Call Center

EKGNET Call Center is an emergency cardiac call center with resident cardiologists responding to the AMI cases. The ECG recording taken from the patient in ambulance will be transferred to this Call Center. This recording will be interpreted by the cardiologist and proper feedback about the case and if needed, authorization to carry out specific procedures -such as thrombolytic treatment- will be given to the ambulance crew. For this purpose, a special software is developed. This software gives the

operating cardiologist full control over the patient unit. ECG recordings and transmissions are initiated by the operating cardiologist at the call center. This enables to call center cardiologists to quickly inspect ECG data and give feedback to the ambulance crew.

This software was developed by Baran Dilber and detailed information can be found in his treatise about the subject [20].

3. PATIENT UNIT FIRMWARE

3.1 Overview

This chapter details the embedded software running on the EKGNET Patient Unit. More specifically, in what follows, we discuss the software modules developed for the Patient Unit Firmware. First an overview of the software structure is given. Then each module is detailed within following subsections. The firmware of the Patient Unit is composed of several modules. All of these modules are developed in ANSI C language with the sole exception of very tightly time constrained ADC Hardware Interface Module which is developed in Assembly language.

3.2 Firmware Structure

The firmware of Patient Unit consists of several modules exposing public functions to be called from other modules. The Main module runs on top of other modules and manifests the operation of the Patient Unit. main module contains a `for(;;){}` loop which begins to run after the device is started and initialized. This loop creates a single thread processing tasks stored in an event queue. In each iteration of the loop the task with highest priority is retrieved from the event queue and processed. Remaining modules are satellite modules called by the main module. These modules are either hardware interface modules or application control modules. Hardware interface modules abstract the hardware layer from the rest of the firmware by exposing hardware independent functions to be called from other modules. Application control modules provide functions to accomplish tasks like reading data from AD converters or sending commands to the GSM modem. By this design application control modules can be used with different hardware settings only by modifying the hardware abstraction layer. When the device is turned on, all hardware is initialized to their default states and checks for failures are carried out. Then `for(;;){}` loop in main module starts to

run. By default, Patient Unit connects to Communication Server automatically after start up. Steps of this default action is controlled by the main module. Main module governs the operation logic of the Patient Unit. Every event is handled within main module. After connection the main module begins to process commands received from EKGNET Call Center.

Table 3.1
EKGNET Patient Unit firmware modules overview

Module Name	Function	Related Section
System	initializes and controls core hardware components	3.2.1
Event Control	manages tasks and events	3.2.2
Keyboard Interface	manages user input	3.2.3
LED Interface	supplies feedback about status of device	3.2.4
GSM Modem Interface	configures and initializes GSM modem	3.2.5
UART Interface	communicates with the GSM modem	3.2.6
AT Commands	send commands to GSM modem	3.2.7
ADC Interface	reads a sample from ADCs	3.2.8
Data Acquisition	controls reading of many samples	3.2.9
Memory Interface	provides access to persistent storage medium	3.2.10
File System	manages a FAT on persistent memory	3.2.11
Packet Control	decodes incoming packets, creates outgoing packets	3.2.12

Detailed information about firmware modules are provided in related sections as shown in Table 3.1.

3.2.1 System Module

System Module initializes and controls main hardware components of Patient Unit such as power supplies, timers, reset circuitry. All power management, start-up and shutdown operations are controlled through this software module. System Module executes the logical steps to gracefully start and stop the system. Any hardware failures such as unexpected resets due battery level disturbances are monitored and logged by this module. System Module also includes the Interrupt Service Routines

(ISR) for timer interrupts. Two timers of the microcontroller are configured to generate different interrupts at specific intervals. Interrupts generated by these timers are handled through multiple ISRs providing a flexible timing framework for the rest of the firmware modules.

3.2.2 Event Control Module

Event Control Module manages tasks for the single thread of firmware and provides a simple and failsafe tasking environment for other modules. To keep the tasking system as simple and as failsafe as possible a single threaded event handling mechanism was designed. Tasks of the system are stored in a static event queue event queue to be processed in the main loop. Multiple tasks can be posted on the queue with different priorities. Main loop chooses the most important event in each iteration and processes the event. Processed events are removed from the event queue. Every event in the queue has 2 bytes of data associated with that event. These 2 bytes can contain any data related with the event. If a larger data block has to be associated with the event, these 2 bytes is used as a pointer to a specific memory location. Also, these 2 bytes can be used to store a function pointer. The pointed function is called by the main loop while processing the event.

3.2.3 Keyboard Interface Module

This module handles the user input from the keyboard of the device. Device has a single button which is used to turn on the Patient Unit. This button needs to be continuously pressed for four seconds. After four seconds a device reset is initiated. If the button is pressed again for four seconds then turn off sequence is initiated and all resources are gracefully released by the system. This means all timed events are removed from the queue, GSM modem is removed from the network if registered.

3.2.4 LED Interface Module

LED Interface Module controls the light emitting diodes of the Patient Unit. LEDs provide proper feedback about the status of the device and its operation. This software module exposes functions to turn on/off the LEDs, program a blinking pattern to be repeated over a certain period of time.

3.2.5 GSM Modem Interface Module

This hardware interface module configures and initializes GSM modem and related hardware modules like the USART module of the microcontroller used to communicate with the GSM modem or the power supply unit supplying power to the GSM modem. It also exposes functions to manage and monitor the operation of the GSM module. It contains all ISRs to handle events caused by the GSM module. There are several hardware signals issued by the GSM modem to indicate the status of operation. For instance, the Voltage Interrupt indicates the status of the power GSM modem is receiving. During startup, this line signals the microcontroller that there is no problem at the power. Another important interrupt is the CTS interrupt. This is a standard RS232 interrupt. During normal operation while sending data to modem from the microcontroller this line stays low to indicate that the GSM modem can receive data from microcontroller. If internal communication buffers are full, this line goes to a logic high to indicate the microcontroller should stop sending data. Also at startup, this line indicates the firmware of GSM modem started correctly. Another important interrupt is the DCD Interrupt. This interrupt is issued upon connection to a remote host. If this line goes high, the connection to the remote host is lost.

3.2.6 UART Interface Module

This module is a hardware interface module which abstracts RS-232 based serial communication from the rest of the firmware. The other firmware module need to call

one simple function to transmit data to the GSM modem. Also, received characters are put into a receive buffer for further processing. The module contains functions to communicate with the GSM modem over RS-232 protocol. It exposes functions to initialize the communication channel between the microcontroller and GSM modem. The interfacing USART module of the microcontroller works with two interrupts. One of them is the transmit interrupt, occurring whenever transmission of one byte is completed. Other interrupt is the receive interrupt, occurring whenever a new byte is received from the GSM modem. The transmit ISR copies a new byte to outgoing byte register after transmission of previous byte is complete. The receive ISR copies newly received byte from incoming byte register after a new byte is received from GSM modem.

3.2.7 AT Commands Module

This module exposes functions to send AT commands to the GSM modem. GSM modem is operated with AT commands. All characters sent to the GSM modem through USART module is interpreted as AT commands until the modem connects to the Communication Server. After connection data sent to GSM modem is relayed directly to the Communication Server. Also, data sent from Communication Server is received by the GSM modem.

3.2.8 ADC Interface Module

This is the only module written in assembly language. This module interfaces ADC hardware and carries out reading of a single sample to a memory location. ADC hardware is interfaced by I/O ports of the microcontroller. Through emulation of SPI protocol by rapidly tweaking these I/O ports, module exposes a function which reads a single sample from ADC hardware. Each call to this function retrieves one sample and writes it to given memory location.

3.2.9 Data Acquisition Module

ADC Interface Module allows reading out of a single sample. Data Acquisition module is used to read as many samples as needed. In background it uses ADC Interface Module to read sequential samples with specified timing requirements. It takes parameters like sampling frequency, sampling duration and channels to convert from and carries out analog-to-digital conversion as requested. It also calibrates the ADC hardware as needed and constantly monitors any conversion faults. This module also implements an ISR for monitoring the power supply of ADC hardware for any power failures.

3.2.10 Memory Interface Module

Patient Unit has embedded serial flash memory with 16Mbits of capacity which is used as persistent storage medium for the Patient Unit. Microcontroller of the Patient Unit interfaces flash memory through one of its USART modules configured in SPI mode. SPI is a general-purpose synchronous serial interface and flash memory accepts commands via SPI. This software module abstracts the flash memory hardware from the firmware by providing general purpose data management functions. These functions are designed to be compatible with most of persistent memory operation needs. Thus, if the need to use another flash module arises, only this portion of the firmware needs to be rewritten. Memory Interface Module exposes functions for initializing, reading data, writing data and checking the status of the flash memory. Flash memory is segmented in 4096 pages. Writing and reading data is conducted per page basis. Each page is `512 + 16 = 528` bytes long. 512 bytes of space is used for data storage and remaining 16 bytes is used to store related CRC checksum. Total size of flash memory is 17 301 504 bits.

3.2.11 File System Module

This module controls the file structure on the flash memory. Flash memory of the Patient Unit is divided in fixed-size partitions. A File Allocation Table (FAT) maintains the begin and end segments of files. This module also includes functions to check the validity of FAT. Any change in file structure is persisted by updating the FAT as needed. If errors occur in reading and writing of file allocation data a new FAT structure is created.

3.2.12 Packet Control Module

This module parses and decodes packets received from the Call Center and creates response packets to be sent back to the Call Center. Incoming packets are forwarded to main module for further processing. Main module inspects the contents of the packets and takes appropriate action against the commands from the Call Center. This module also creates response packets to be sent back to the Call Center Application. Once communication between device and server is initiated periodic receive task is posted to the event queue. Any data received afterward is considered as part of a valid packet. Only one packet can be actively transmitted or received. This design simplifies packet processing logic and eases debugging. On the reception of a valid packet an Ack Packet is sent back to the server. Details of Ack Packet are given in section 5.2.1. Invalid packets are not processed therefore no Ack Packet is sent back. This enables the Call Center Application to detect transmission failures and retransmit same packet if needed.

4. COMMUNICATION SERVER

4.1 Overview

In this chapter details of EKGNET Communication Server will be presented. This server provides a communication channel between the Patient Units and Call Center. In section 4.1, an overview of the software structure will be given. Communication Server software is written in C# which is an object oriented language, therefore section 4.2 will present details about classes used as the building blocks of the Communication Server.

Because of the security measures taken by GSM operators, creating a connection to the operator backbone from outside of their network is prohibited. Instead, Patient Units have to create connections from inside to outside. These connections between the GSM operator and EKGNET Communication Servers are carried through a VPN tunnel for maximum security.

Communication Server works as a proxy server between the Patient Units and Call Center. When the Patient Unit is activated, a GRPS call is initiated over the GSM network. GSM Operator transfers this call to EKGNET Communication Server via TCP/IP and any data transferred from the device is forwarded to Communication Server. After a Patient Unit connects to the Communication Server, the Call Center application is notified by this incoming call and an alert is generated. Upon this alert an operating cardiologist intercepts this call by connecting to the Communication Server and requesting the call to be switched to his/her console. EKGNET Communication Server evaluates this request by authorizing supplied user information on the authorized users database. If authorization succeeds any data received from the Patient Unit is forwarded to the console of the call center operator. Also, any data sent from the operator console is relayed to the Patient Unit by the Communication Server. Software for the Communication Server was developed with Test-Driven Development technique.

Test-Driven Development (TDD) is a computer programming technique that involves repeatedly first writing a test case and then implementing only the code necessary to pass the test [21]. A sample test suite code for Communication Server is given in A.

Communication Server is written in C# which is an object-oriented programming language developed by Microsoft as part of .NET initiative. C# was later approved as a standard by ECMA and ISO. C# has a procedural, object oriented syntax based on C++ that includes aspects of several other programming languages (most notably Delphi, Visual Basic, and Java) with a particular emphasis on simplification (fewer symbolic requirements than C++, fewer decorative requirements than Java) [22].

4.2 Classes

In object oriented programming, classes the unit of definition of data and behavior (functionality) for some kind-of-thing. For example, the 'class of Dogs' might be a set which includes the various breeds of dogs. A class is the basis of modularity and structure in an object-oriented computer program. A class should typically be recognizable to a non-programmer familiar with the problem domain, and the code for a class should be (relatively) self-contained and independent (as should the code for any good pre-OOP function). With such modularity, the structure of a program will correspond to the aspects of the problem that the program is intended to solve. This simplifies the mapping to and from the problem and program [23].

EKGNET Communication Server consists of several Classes shown in table 4.1. These classes are hierarchically and functionally bound together. The details about each class is given in following sections. Detailed class diagrams are given in Appendix B.

Table 4.1
EKGNET Communication Server classes

Class Name	Function	Related Section
Server	listens for new clients, manages operation	4.2.1
Client	contains the socket to communicate with client	4.2.2
ClientManager	maintains autorization of clients	4.2.3
SessionClient	contains identity information and client socket	4.2.4
Session	contains two SessionClients	4.2.5
SessionManager	processes data transfer for active Sessions	4.2.6

4.2.1 Server Class

This is the main class governing the operation of the Communication Server. Upon instantiation, this class creates three threads. First thread listens for connection requests on a specific port. Second thread handles connected yet unidentified clients. Third thread transmits data between connected and authorized Patient Units and Call Center Consoles. Server Class manages overall timing by sequentially handling each thread and calling related methods of other classes to do processing if needed. Main processing logic of these tasks are encapsulated in following classes. Class diagram for Server class is given in Appendix B.1.

4.2.2 Client Class

When a host (Patient Unit or Call Center Console) connects to the Communication Server an Listener thread creates an instance of this class containing newly created socket connection. This socket is used to communicate with the connected host. Class diagram for Client class is given in Appendix B.2.

4.2.3 ClientManager Class

After a connection request is handled by Communication Server a new Client is created and passed to ClientManager class. ClientManager maintains two lists for active Clients. One list contains all anonymous clients and new clients are put in this list. Clients of this list have to send their identity information to ClientManager until their anonymous connection times out. When Server demands the ClientManager to process connected clients, every client in this list is checked for incoming data. If any of the clients has sent correct identification information it is removed from the anonymous clients list and is added to the authenticated clients list. Call Center Application is notified of any new addition to the authenticated clients list. This means a new Patient Unit has connected to the server and needs to be intercepted. Available Call Center cardiologist request the control of the Patient Unit by connecting to the Communication Server and requesting a new session. If authentication information from Call Center Console is accepted, requested Patient Unit client is removed from authenticated clients list and a new session is created. Class diagram for ClientManager class is given in Appendix B.3.

4.2.4 SessionClient Class

After a session is successfully started sockets contained in Client classes are transferred in SessionClient classes. SessionClient contains additional information about the type and identity of the connected client. This kind of information is only available when a Session is successfully started. Class diagram for SessionClient class is given in Appendix B.4.

4.2.5 Session Class

Session class contains two SessionClient instances. These are two ends of the ongoing communication session between a specific Patient Unit and Call Center Con-

sole. This class is created upon successful retrieval of an incoming Patient Unit call by a Call Center Console. Until disconnection all data transfer between the two specific clients are considered to be conducted in current session. Class diagram for Session class is given in Appendix B.4.

4.2.6 SessionManager Class

SessionManager Class maintains a list of ongoing Sessions. When called by the Server, this class processes each Session in its Sessions list. Processing a session is forwarding any received data from one host to the other one. Class diagram for SessionManager class is given in Appendix B.5.

5. APPLICATION PROTOCOL

5.1 Overview

EKGNET Application Protocol governs the communication between the Patient Unit and Call Center Application. All request and response transmissions between two clients adhere to the rules set by this protocol. EKGNET Application Protocol is a request/response protocol between Patient Units and Call Center Application. Call Center Application sends task requests as predefined commands encoded in control packets. EKGNET Communication Server does not interfere with the communication between the Patient Unit and Call Center Application, in other words the Application Protocol is transparent to the Communication Server. Only logic related with this protocol residing in Communication Server is used to parse Control Packets that are initially sent upon connection to the Communication Server because these packets from both Patient Unit and Call Center Application contain authentication information used by the Communication Server to authorize the association of two clients.

Upon connection of the Patient Unit and Call Center Application through Communication Server, Patient Unit responds to commands from Call Center Application after accomplishing requested task. The response is sent back to the Call Center Application as a Data Packet if the request was to sample ECG data. All other responses are sent back as Control Packets from the Patient Unit containing the result of the requested task.

5.2 Packets

Binary data transmission between Patient Unit and Call Center Application is encoded as Packets. Rules of forming of packets and parsing them is set by Application Protocol. General structure of a Packet shown in Fig. 5.1.

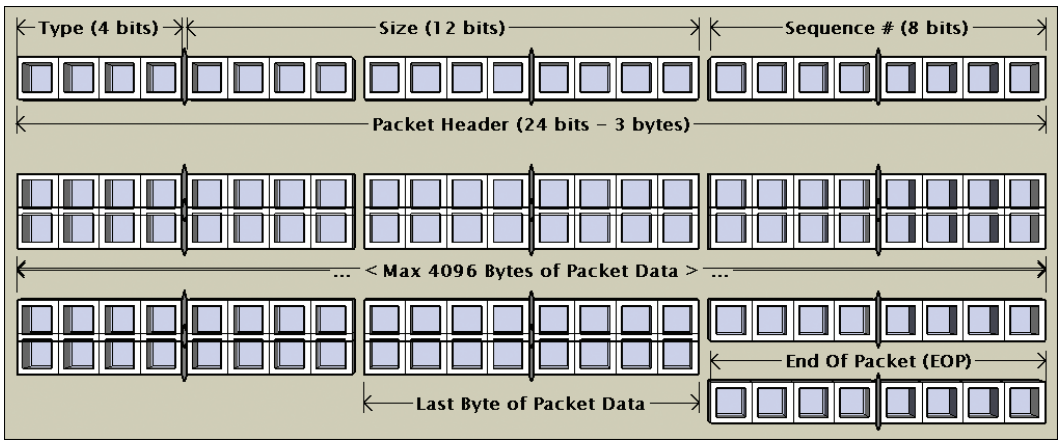

Figure 5.1 Structure of a Packet

First four bits of every packet contain a packet type indicator. This allows 16 different types of packets. Following 12 bits indicate the length of total packet. Following byte contains the sequence number of the packet. Sequence number is used to keep track of binary data streams. During the transfer of a large amount of binary data, every packet sent has an incremented sequence number. Thus receiving end expects to receive packets in suggested order by checking the sequence number.

5.2.1 Ack Packet

Acknowledgment of correct reception of packets are signalled to other host by Ack Packets. These packets contain information about the received packet like the type of received packet and the sequence number of received packet. An Ack Packet consists of three bytes. First four bits of these bytes designate the packet as an Ack Packet. These four bits are `[1111]` (`0xF` in hexadecimal notation). The next four bits contain the type information for successfully received packet. The next byte contains the sequence number of received packet. Last byte contains the End of Packet code. Structure of an Ack Packet shown in Fig. 5.2.

After sending a packet sending hosts expect to receive related Ack Packet in a certain amount of time. If no Ack Packet is received in that period, a connection failure is assumed. This enables to keep track of connections established by TCP.

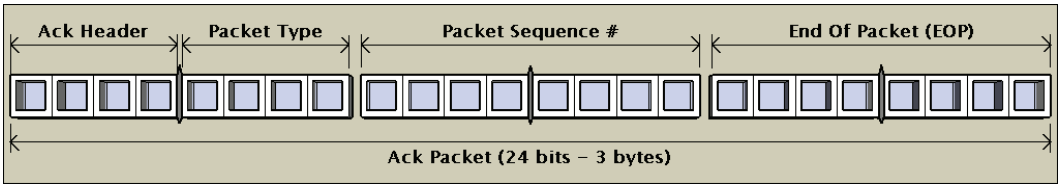

Figure 5.2 Structure of an Ack Packet

5.2.2 Data and Control Packets

Data packets are used to transfer bulk data between the Patient Unit and Call Center Application. In most cases, data flows from Patient Unit to Call Center Application. In rare cases, like remote updating of Patient Unit firmware, data flow is reversed. The type nibble denoting a Data Packet is `[0000]` (`0x0` in hexadecimal notation). Control packets are used to exchange requests and responses between the Patient Unit and Call Center Application. Four type bits denoting a Control Packet are `[0001]` (`0x1` in hexadecimal notation).

5.3 Packet Parsing

Creating and sending packets is a straightforward encoding process. But correct decoding of packets from incoming byte stream is more complicated. Because data transmission between two hosts introduces delays and transmission failures, parsing of received byte stream requires a carefully constructed and well tested finite state machine (FSM). Initially Packet Parser is in idle state. Upon reception of data, Packet Parser begins to parse incoming data. Incoming byte stream can contain a single complete packet, more than one packet and/or a fragment of a packet. If incoming data contains a single packet, Packet Parser switches to idle state after the packet is successfully reconstructed at the receiving end. If incoming data contains a fragment of a packet, Packet Parser stays in parsing state until new data arrives and any received data in that state is considered to be part of the previous packet. If incoming data contains more than one packet, Packet Parser parses all of the packets contained in received data. After all packets are parsed, Packet Parser moves to either idle state or

stays in parsing state if parsing of last packet is not complete.

6. DISCUSSIONS, CONCLUSIONS AND FUTURE WORK

6.1 Discussions

There are some pitfalls to be avoided in the design of an ambulatory ECG device, most important one being the prevention of movement artifacts during ECG recordings. The patient in an ambulance generally has a very slim chance of persisting a motionless state thus enabling high quality ECG recordings. EKGNET project primarily targets patients in acute conditions and therefore conducting artifact free ECG recordings in mobile conditions was our primary design criteria for EKGNET Patient Unit. For that purpose a 24bit ADC hardware was chosen. By this design, ECG signal is allowed to float without saturation within a very large dynamic signal range. The absence of signal saturation risk allows us to drop the offset removing analog high-pass filter from the input stage of our ECG device. This removes unwanted baseline drifts due to motion artifacts. Our latest prototype performs extremely well in ambulatory situations and high quality ECG recordings are possible even if the patient is mobile.

Our second important concern was to make the Patient Unit as simple as possible. In current setting, our ECG device has a very simple user interface and it is extremely simple to operate. The Patient Unit is has only a single button which turns on the device. Rest of the operation is remotely controlled by the cardiologist at the EKGNET Call Center. General system structure and communication framework gives the cardiologist total control over the ECG device after the device connects to EKGNET Communication Server. This design enables the ambulance crew to concentrate on correct placement of ECG electrodes.

Remote controlled operation scheme simplifies device firmware as well. In embedded programming, software components related to graphical user interface are very hard to develop, especially for an ECG device. By relocating data visualization tasks from embedded environment to PC environment, a very simple and effective implemen-

tation has been achieved. Also, overall cost of both hardware and software components were reduced considerably. By our experiments with the current prototype we investigated that even with a constant latency overhead induced by the GSM network, conducting and sending a 3 seconds length ECG recording takes 10-15 seconds. This means even if the first recording attempt fails, relaying of consecutive ECG recordings until a successful one will be completed in a time frame of 1 minutes. Our inquiries with emergency medical service providers indicate that this worst case time frame is more than acceptable within their operation standards.

Our TCP/IP based communication framework allows us to utilize next generation communication technologies very easily. By changing the GSM modem with a 3G enabled modem, we can achieve greater bandwidth. In a few years, after 3G technology becomes available, we will be able to exploit the advantages of this new technology very quickly. Also, scalable and maintainable system design and implementation allows us to answer capacity increase needs as they occur.

The hardware abstraction layer of the firmware provides great flexibility on changing hardware components of the Patient Unit. If components of the hardware are to be upgraded related hardware revision will have minimum impact on Patient Unit firmware.

Test-driven development is cited as a best approach to create virtually bug free and easily maintainable code. By employing this technique to create core components of our software we greatly increased our efficiency in debugging and maintaining the code. This was also a huge win in the development process.

Lastly, one of our primary concerns was to develop a system with low total cost of ownership. This criteria clearly separates EKGNET project from similar solutions like Ortivus' MobiMed system [19], which is far too expensive to implement in countries like Turkey. We currently estimate the TCO for EKGNET project will be significantly lower than MobiMed system.

6.2 Conclusions

The EKGNET as a technology driven project has a strong motive for improving emergency health care quality in Turkey by enabling quick diagnosis of AMI and immediate activation of targeted emergency medical services related to it. The use of telecardiology systems permitting rapid interpretation 12-lead ECG recordings by highly skilled cardiologists, promises to be a very effective for diagnosing cardiac disorders. As mentioned in section 2.2, recent thrombolytic agents are very effective in pre-hospital scenarios. Thus EKGNET system will create the opportunity to equip most ambulances with an affordable solution for treating AMI patients with next generation thrombolytic agents and increase their chances of survival.

In order to provide cardiac consultancy services to a larger population, the capacity of current servers should be increased. However, scalable design criteria of EKGNET project will allow for accomplishing this task without great difficulty.

6.3 Future Work

After EKGNET Call Center becomes operational as a cardiac consultancy service center, EKGNET project will provide the infrastructure necessary to help many patients through the means of telecardiology services. Such an infrastructure can be expanded upon in a number of interesting ways. Some possibilities include:

- Scaling and expanding the system to help with non-acute conditions,
- Providing a cardiac consultancy center for non-cardiologist MDs,
- Modifying Patient Unit to decrease its cost and becoming accessible by the public,
- Providing hotels, big corporations and schools etc. a means for correct diagnosis of coronary diseases,

- Employing 3G technology to conduct more efficient and higher resolution ECG transfer,
- Employing 3G technology to transfer various other diagnostically important signals along with the ECG data,

APPENDIX A. SAMPLE UNIT TEST: COMMUNICATION SERVER

```
using System;
using System.Net;
using System.Net.Sockets;
using MbUnit.Framework;
using Teknofil.EkgNet.Server;
using System.Threading;
using Teknofil.EkgNet.Communication.Packets;
namespace Teknofil.EkgNet.Server.UnitTest
{
  [TestFixture]
  public class ServerTest
  {
    private Server server = new Server(IPAddress.Any, 23);
    AutoResetEvent[] eventSignals;
    private int eventIndex;
    int[] deviceIds;
    int[] userIds;
    int[] connectedDeviceIds;
    int[] connectedUserIds;
    [TestFixtureSetUp]
    public void FixtureSetUp()
    {
      server.ClientConnected+=new EventHandler(server_ClientConnected);
      server.NewDevice+=new NewDeviceEventHandler(server_NewDevice);
      server.NewSession+=new NewSessionEventHandler(server_NewSession);
    }
```

```
// Test the server start method
[Test]
public void StartServer()
{
  this.server.Start();
  Assert.AreEqual(this.server.Up, true);
}

// Simulates a single client connection
[Test]
public void SingleClientConnection()
{
  TcpClient[] clients = CreateClients(1);
  AssertConnections(clients, "Connect single client failed.");
  DisconnectClients(clients);
}

// Simulates multiple client connections
[Test]
public void MultipleClientConnection()
{
  TcpClient[] clients = CreateClients(64);
  AssertConnections(clients, "Connect multiple clients failed.");
  DisconnectClients(clients);
}

// Simulates a single session
[Test]
public void SingleSession()
{
  TcpClient[] devices = CreateClients(1);
  AssertConnections(devices, "Device could not connect to server");
```

```
  TcpClient[] users = CreateClients(1);
  AssertConnections(users, "User could not connect to server");
  AssertSessions(devices, users);
  DisconnectClients(devices);
  DisconnectClients(users);
}

// Simulates multiple sessions
[Test]
public void MultipleSessions()
{
  TcpClient[] devices = CreateClients(64);
  AssertConnections(devices, "Device could not connect to server");
  TcpClient[] users = CreateClients(64);
  AssertConnections(users, "User could not connect to server");
  AssertSessions(devices, users);
  DisconnectClients(devices);
  DisconnectClients(users);
}

// Simulates data transfer within a session
[Test]
public void SingleSessionDataTransfer()
{
  TcpClient[] devices = CreateClients(1);
  AssertConnections(devices, "Device could not connect to server");
  TcpClient[] users = CreateClients(1);
  AssertConnections(users, "User could not connect to server");
  AssertSessions(devices, users);
  Random random = new Random();
  byte[] data = new byte[1024];
  random.NextBytes(data);
```

```
  NetworkStream deviceStream = devices[0].GetStream();
  NetworkStream userStream = users[0].GetStream();
  DataPacket dataPacket = new DataPacket(0, data);
  byte[] readData = new byte[dataPacket.Bytes.Length];
  deviceStream.Write(dataPacket.Bytes, 0, dataPacket.Bytes.Length);
  Thread.Sleep(100);
  userStream.Read(readData, 0, dataPacket.Bytes.Length);
  DataPacket receivedPacket = DataPacket.ParsePacket(ref readData);
  Assert.AreEqual(dataPacket, receivedPacket);
  readData = new byte[dataPacket.Bytes.Length];
  userStream.Write(dataPacket.Bytes, 0, dataPacket.Bytes.Length);
  Thread.Sleep(100);
  deviceStream.Read(readData, 0, dataPacket.Bytes.Length);
  receivedPacket = DataPacket.ParsePacket(ref readData);
  Assert.AreEqual(dataPacket, receivedPacket);
  DisconnectClients(devices);
  DisconnectClients(users);
}

// Simulates data transfer with multiple sessions
[Test]
public void MultipleSessionDataTransfer()
{
  TcpClient[] devices = CreateClients(64);
  AssertConnections(devices, "Device could not connect to server");
  TcpClient[] users = CreateClients(64);
  AssertConnections(users, "User could not connect to server");
  AssertSessions(devices, users);
  AssertDataTransfer(devices, users);
  DisconnectClients(devices);
  DisconnectClients(users);
}
```

```
// Helper function to check integrity of transferred data
private void AssertDataTransfer(TcpClient[] devices,
                                TcpClient[] users)
{
  if (devices.Length != users.Length)
  {
    throw new Exception("Devices and Users are not same size.");
  }
  DataPacket[] dataPackets = new DataPacket[devices.Length];
  Random random = new Random();
  byte[] data = new byte[1024];
  NetworkStream deviceStream;
  for (int i = 0; i < devices.Length; i++)
  {
    deviceStream = devices[i].GetStream();
    random.NextBytes(data);
    dataPackets[i] = new DataPacket((byte) i, data);
    deviceStream.Write(dataPackets[i].Bytes,
                       0,
                       dataPackets[i].Bytes.Length);
  }
  byte[] readData;
  DataPacket[] receivedDataPackets=new DataPacket[devices.Length];
  NetworkStream userStream;
  for (int i = 0; i < devices.Length; i++)
  {
    readData = new byte[dataPackets[i].Bytes.Length];
    userStream = users[i].GetStream();
    userStream.Read(readData, 0, readData.Length);
    receivedDataPackets[i] = DataPacket.ParsePacket(ref readData);
  }
```

```
  ArrayAssert.AreEqual(dataPackets, receivedDataPackets);
}

// Helper function to check the integrity of simulated sessions
private void AssertSessions(TcpClient[] devices, TcpClient[] users)
{
  if (devices.Length != users.Length)
  {
    throw new Exception("Devices and Users are not same size.");
  }
  deviceIds = new int[devices.Length];
  userIds = new int[users.Length];
  connectedDeviceIds = new int[devices.Length];
  connectedUserIds = new int[users.Length];
  this.eventSignals = new AutoResetEvent[devices.Length];
  this.eventIndex = 0;
  for (int i = 0; i < devices.Length; i++)
  {
    this.eventSignals[i] = new AutoResetEvent(false);
  }
  NetworkStream deviceStream;
  for (int i = 0; i < devices.Length; i++)
  {
    deviceStream = devices[i].GetStream();
    byte seqNo = (byte) i;
    byte[] data = new byte[] {ClientTypes.Device,
                              (byte) i,
                              (byte) i}
```

```
  ControlPacket deviceIdPacket = new ControlPacket(seqNo,
                                                   data);
  deviceStream.Write(deviceIdPacket.Bytes,
                     0,
                     deviceIdPacket.Bytes.Length);
  this.deviceIds[i] = (int) ((ushort) (i << 8 | i));
}
bool index = WaitHandle.WaitAll(this.eventSignals, 1000, false);
Array.Sort(this.deviceIds);
Array.Sort(this.connectedDeviceIds);
ArrayAssert.AreEqual(this.deviceIds, this.connectedDeviceIds);
this.eventSignals = new AutoResetEvent[devices.Length];
this.eventIndex = 0;
for (int i = 0; i < devices.Length; i++)
{
  this.eventSignals[i] = new AutoResetEvent(false);
}
NetworkStream userStream;
for (int i = 0; i < users.Length; i++)
{
  userStream = users[i].GetStream();
  ControlPacket userIdPacket=new ControlPacket(
                                                 (byte) i,
                                                 new byte[]
                                                {ClientTypes.User,
                                                (byte) (i + 1),
                                                (byte) (i + 1),
                                                (byte) i,
                                                (byte) i});
  userStream.Write(userIdPacket.Bytes,
                   0,
                   userIdPacket.Bytes.Length);
```

```
    this.userIds[i] = (int) ((ushort) ((i + 1) << 8 | (i + 1)));
  }
  index = WaitHandle.WaitAll(this.eventSignals, 1000, false);
  Assert.AreEqual(index, true, "Session was not started.");
  Array.Sort(this.userIds);
  Array.Sort(this.connectedUserIds);
  ArrayAssert.AreEqual(this.userIds, this.connectedUserIds);
}

// Helper function to check integrity of established connections
private void AssertConnections(TcpClient[] clients,
                               string errorMessage)
{
  int previousClientCount = this.server.ClientCount;
  ConnectClients(clients);
  bool index = WaitHandle.WaitAll(this.eventSignals, 100, false);
  Assert.AreEqual(index, true, errorMessage);
  Assert.AreEqual(server.ClientCount,
                  previousClientCount + clients.Length);
}

// Helper functions which created clients
private TcpClient[] CreateClients(int count)
{
  this.eventSignals = new AutoResetEvent[count];
  TcpClient[] clients = new TcpClient[count];
  for (int i = 0; i < count; i++)
  {
    this.eventSignals[i] = new AutoResetEvent(false);
    clients[i] = new TcpClient();
    clients[i].LingerState.Enabled = false;
  }
```

```
  this.eventIndex = 0;
  return clients;
}

// Helper function which connect clients to server
private void ConnectClients(TcpClient[] clients)
{
  for (int i = 0; i < clients.Length; i++)
  {
    clients[i].Connect(IPAddress.Loopback, this.server.Port);
  }
}

// Helper function which disconnects clients from server
private void DisconnectClients(TcpClient[] clients)
{
  for (int i = 0; i < clients.Length; i++)
  {
    clients[i].Close();
  }
}
// Code running at the end of unit test
[TearDown]
public void TearDown()
{
   // this.server.Stop();
   // this.server.Start();
}

// Code running at the ent of the test fixture
[TestFixtureTearDown]
public void FixtureTearDown()
```

```
    {
      this.server.Stop();
      Assert.AreEqual(this.server.Up, false);
    }
    // Event handler code running when a client connects to server
    private void server_ClientConnected(object sender, EventArgs e)
    {
      this.eventSignals[eventIndex++].Set();
    }

    // Event handler code running when a new device is authenticated
    private void server_NewDevice(object sender, NewDeviceEventArgs e)
    {
      try
      {
        this.connectedDeviceIds[eventIndex] = (int) e.DeviceId;
        this.eventSignals[eventIndex++].Set();
      }
      catch (Exception ex)
      {
        string deneme = ex.Message;
      }
    }

    // Event handler code running when a new session is started
    private void server_NewSession(object sender, NewSessionEventArgs e)
    {
      this.connectedUserIds[eventIndex] = (int) e.UserId;
      this.eventSignals[eventIndex++].Set();
    }
  }
}
```

APPENDIX B. COMMUNICATION SERVER CLASS DIAGRAMS

B.1 Server Class

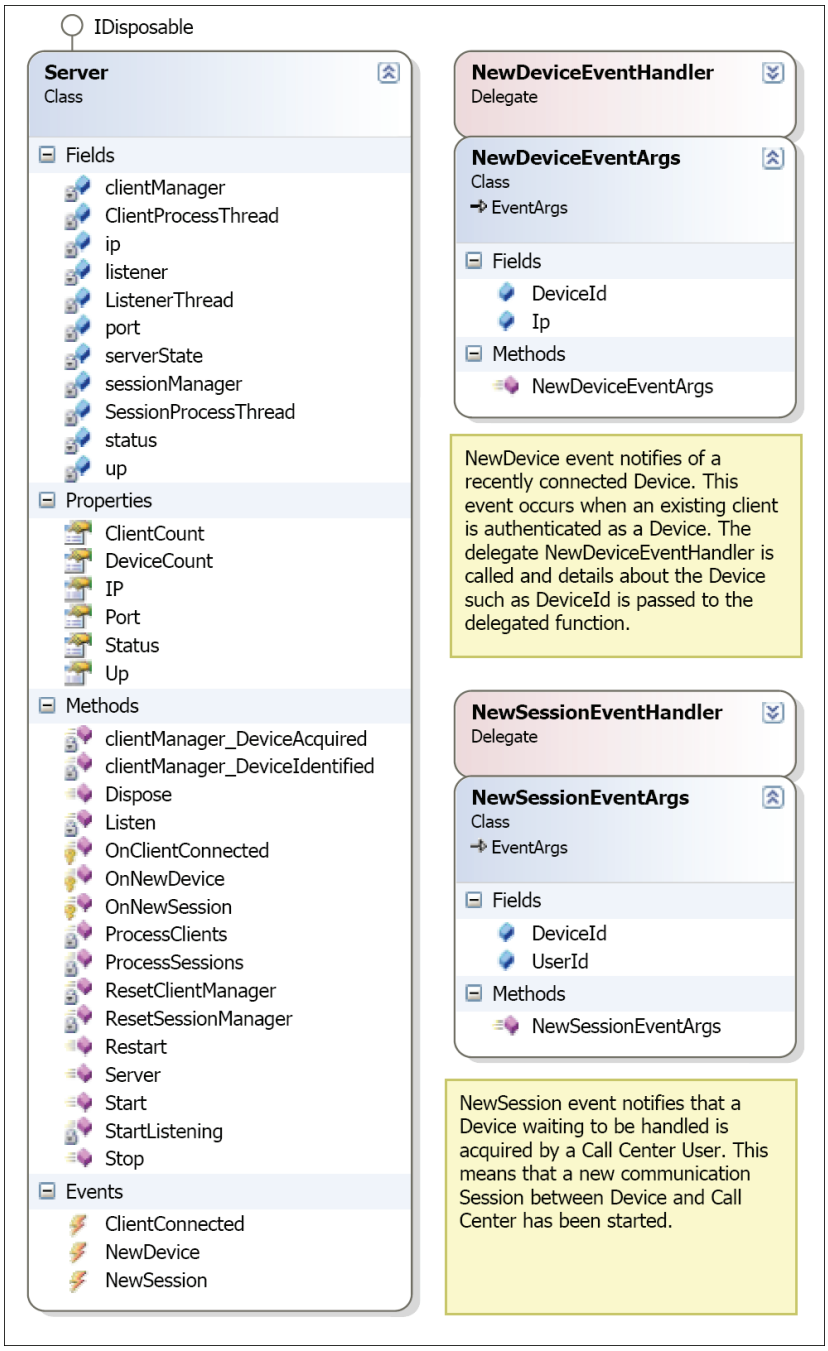

Figure B.1 Server Class Diagram

B.2 Client Class

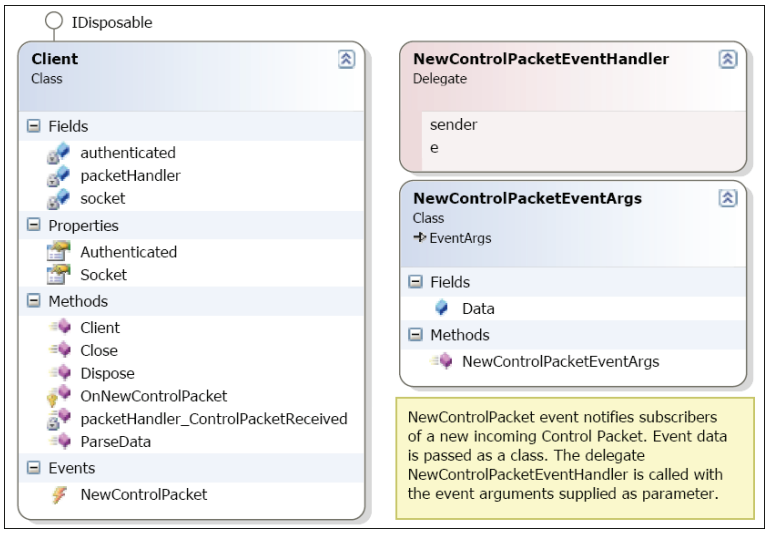

Figure B.2 Client Class Diagram

B.3 ClientManager Class

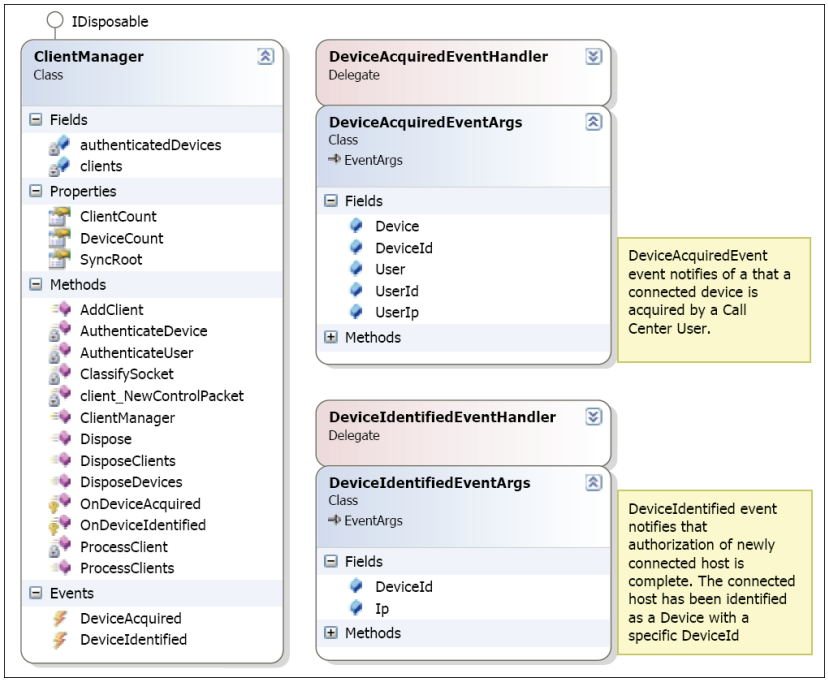

Figure B.3 ClientManager Class Diagram

B.4 Session and SessionClient Classes

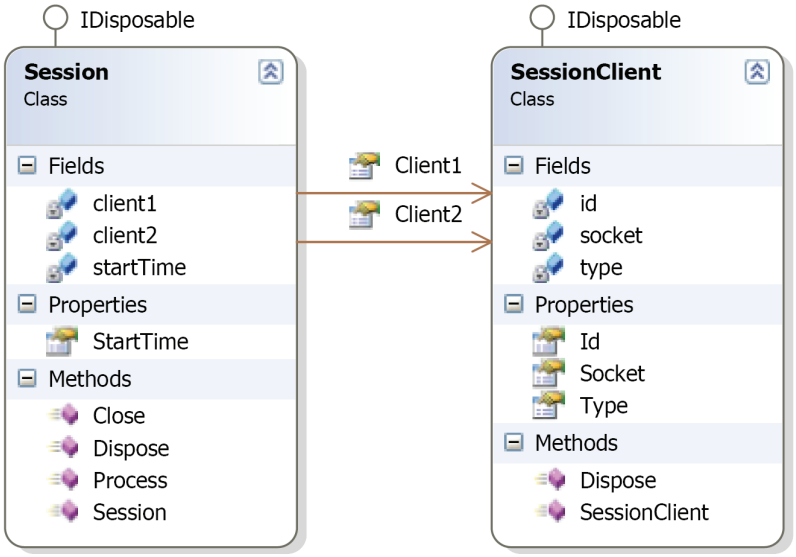

Figure B.4 Session and SessionClient Clas Diagrams

B.5 SessionManager Class

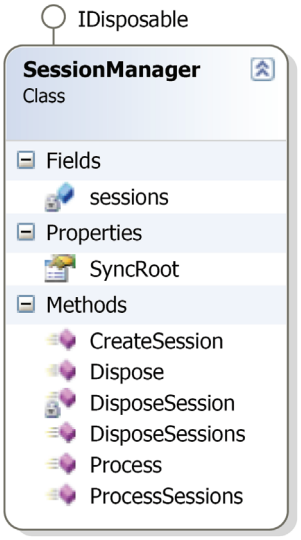

Figure B.5 SessionManager Class Diagram

REFERENCES

1. Wikipedia, The Free Encyclopedia, *Myocardial Infarction*, en.wikipedia.org: Wikipedia, 2006. Available: `http://en.wikipedia.org/wiki/Myocardial_infarction`.

2. American Heart Association, *Heart Disease and Stroke Statistics - 2004 Update*, Dallas, Tex.: American Heart Association, 2004. Available: `http://www.americanheart.org/downloadable/heart/1079736729696HDSStats2004UpdateREV3-19-04.pdf`.

3. TÜMAR Çalışma Grubu, *Türkiye Akut Miyokard İnfarktüsü Araştırması*, Istanbul, Turkey: MI Kulübü, 1999. Available: `http://miclub.org/home/kitap04.shtml`.

4. WHO, *Cardiovascular Diseases*, Washington, USA: World Health Organization, 2003. Available: `http://www.who.int/cardiovascular_diseases/en/cvd_atlas_14_deathHD.pdf`.

5. A. Leizorovicz, J.P. Boissel, D. J. A. C., and M. Haugh., "Esc task force report," *European Health Journal*, Vol. 19, pp. 1140–1164, 12 1998.

6. Boehringer Ingelheim, *The electrocardiagram (EKG) for pre-hospital triage and treatment*, Boehringer Ingelheim International GmbH, Germany: Boehringer Ingelheim, 2005. Available: `http://www.metalyse.com/com/Main/myocardial_infarction/symptoms/diagnosis/index.jsp`.

7. Ingelheim, B., *What is Thrombolysis?*, Boehringer Ingelheim International GmbH, Germany: Boehringer Ingelheim, 2005. Available: `http://www.metalyse.com/com/index.jsp`.

8. Pedley D. K., Bissett K., Connolly E. M., Goodman C. G., Golding I., Pringle T. H., McNeill G. P., Pringle S. D., Jones M. C., "Prospective observational cohort study of time saved by prehospital thrombolysis for st elevation myocardial infarction delivered by paramedics.," *BMJ*, Vol. 327, pp. 22–26, Jul 5 2003. PMID: 12842951.

9. Boehringer Ingelheim, *Basic pre-hospital treatment*, Boehringer Ingelheim International GmbH, Germany: Boehringer Ingelheim, 2005. Available: `http://www.metalyse.com/com/Main/myocardial_infarction/symptoms/diagnosis/index.jsp`.

10. The GUSTO Investigators, "An international randomized trial comparing four thrombolytic strategies for acute myocardial infarction.," *N Engl J Med*, Vol. 329, pp. 673–682, Sep 2 1993. PMID: 8204123.

11. Assessment of the Safety and Efficacy of a New Thrombolytic Regimen (ASSENT)-3 Investigators, "Efficacy and safety of tenecteplase in combination with enoxaparin, abciximab, or unfractionated heparin: the assent-3 randomised trial in acute myocardial infarction.," *Lancet*, Vol. 358, pp. 605–613, Aug 25 2001. PMID: 11530146.

12. Rawles JM, R. L., "Thrombolysis in peripheral general practices in scotland: another rule of halves.," *Health Bull (Edinb).*, Vol. 57, pp. 10–16, Jan 1999. PMID: 12811860.

13. California Telemedicine & eHealth Center, *The History of Telemedicine*, California Telemedicine & eHealth Center, 1215 K Street, Suite 800, Sacramento, CA 95814: California Telemedicine & eHealth Center, 2006. Available: `http://www.cttconline.org/telemedicine_history.html`.

14. Einthoven, W., "Le télécardiogramme," *Arch Int Physiol*, Vol. 4, no. 132, 1906.

15. Scanlon, W. G., "Using wireless technology to develop an effective personal telemedicine service," tech. rep., University of Ulster, Shore Road, Newtownabbey, Co. Antrim, N. Ireland,, 2000.

16. Kennedy, H. L., and D. G. Caralis, "Ambulatory electrocardiography: A clinical prospective," *Ann. Intern. Med.*, Vol. 87, pp. 729–739, 1977.

17. M Schaldach, H. H., "Telecardiology - optimizing the diagnostic and therapeutic efficacy of the next implant generation," *Progress in Biomedical Research*, pp. 1–2, February 1998.

18. Yazıcı, Y., "Design of a transtelephonic ECG and thermometer device using the mobile phone," Master's thesis, Bogazici University, Istanbul, Turkey, 2005.

19. Ortivus AB, *MobiMed*, Danderyd, Stockholm Sweden: Ortivus AB, 2005. Available: `http://www.ortivus.com/templates/Tmpl_Page____2217.asp`.

20. Dilber, C. B., "Design and implementation of an ecg based emergency telediagnostic system," Master's thesis, Bogazici University, Istanbul, Turkey, 2006.

21. Wikipedia, The Free Encyclopedia, *Test-driven development*, en.wikipedia.org: Wikipedia, 2006. Available: `http://en.wikipedia.org/wiki/Test_driven_development`.

22. Microsoft Corporation, *The C# Language*, One Microsoft Way Redmond, WA 98052-6399: Microsoft Corporation, 2006. Available: `http://msdn.microsoft.com/vcsharp/programming/language/`.

23. Wikipedia, The Free Encyclopedia, *Object-oriented programming*, en.wikipedia.org: Wikipedia, 2006. Available: `http://en.wikipedia.org/wiki/Object_oriented_programming`.