LAPPEENRANTA UNIVERSITY OF TECHNOLOGY

Department of Information Technology

Designing a modular, open and expandable Bluetooth-platform

Master of Science Thesis

The subject of the thesis has been approved 10.04.2002 in a meeting by Department Council of Information Technology Department

Supervisor: Professor, D.Sc. Jouni Lampinen

Sami Merilä

Sami Merilä

Lapinkaari 21a9 33180 Tampere FINLAND

+358 40 5389034

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology
Sami Merilä

Designing a modular, open and expandable Bluetooth-platform

Master's thesis
2002
88 pages, 21 figures, 6 tables, 3 appendices
Supervisor: Professor, D.Sc. Jouni Lampinen
Keywords: Bluetooth, Wireless application, robot, platform

In the near future, the business opportunities of wireless systems will be enormous. To research the upcoming needs, this thesis gives some consideration how to design and implement open wireless client-server system.

It was decided that the system would use Bluetooth. Of the inspected wireless standards, Bluetooth suits best a multi-purpose, battery-powered device. In addition, the standard has huge business expectations and one of the goals of the thesis was to study if these are realistic. It was discovered that Bluetooth has a lot of hype attached to it and it is quite complex. However, it has many features and with different usage profiles, it can be suited for wide variety of tasks.

The designed system runs a socket server on top of Bluetooth connection. Dedicated sockets provide needed expandability. The socket server was implemented as a Linux thread and it controls the Bluetooth protocol stack as well as applications running on the server. The services provided by these applications are available through Bluetooth.

TIIVISTELMÄ

Lappeenrannan Teknillinen Korkeakoulu
Tietotekniikan Osasto
Sami Merilä

Modulaarisen, avoimen ja laajennettavan Bluetooth-alustan suunnittelu

Diplomityö
2002
88 sivua, 21 kuvaa, 6 taulukkoa, 3 liitettä
Tarkastaja: Professori Jouni Lampinen
Hakusanat: Bluetooth, langaton sovellus, robotti, alusta

Lähitulevaisuudessa langattomien järjestelmien kaupalliset mahdollisuudet tulevat olemaan valtavia. Tutkiaksemme tulevia tarpeita, tässä diplomityössä esitellään kuinka voidaan suunnitella ja toteuttaa avoin langaton asiakas-palvelin järjestelmä.

Järjestelmänä päätettiin käyttää Bluetooth:ia. Tutkituista langattomista standardeista Bluetooth sopii parhaiten akkukäyttöiselle laitteelle, jonka tulee olla monipuolinen. Lisäksi Bluetooth:iin on liitetty suuria kaupallisia odotuksia ja yksi työn tavoitteista olikin tutkia, ovatko nämä odotukset realistisia. Bluetooth:iin havaittiin liittyvän paljon ylimainontaa ja, sen todettiin olevan monimutkainen. Sillä on kuitenkin paljon ominaisuuksia ja erilaisten käyttöprofiilien avulla sitä voidaan käyttää monenlaisiin tehtäviin.

Suunniteltu järjestelmä ajaa socket-palvelinta Bluetooth-yhteyden päällä. Tietyntyyppiseen liikenteeseen erikoistuneet socket:t tarjoavat vaaditun laajennattavuuden. Palvelin toteutetiin Linux-säikeenä ja se hallitsee Bluetooth protokollapinoa sekä sovelluksia, joita suoritetaan palvelimella. Näiden sovelluksien palvelut ovat muiden käytössä Bluetooth:n kautta.

TABLE OF FIGURES AND TABLES

Terms and Abbreviations:

Term	Description
ACL	Asynchronous ConnectionLess
AM_ADDR	Active Member Address
ARQ	Automatic Repeat Request
ARQN	An acknowledgement bit in ACL packets that tells that previous packet was received correctly.
BB	Bluetooth Baseband
BD_ADDR	Bluetooth device's unique address
BR	Bluetooth radio
CDMA	Code-Division Multiple Access
CRC	Cyclic Redundancy Check
DECT	Digital Enhanced Cordless Telephony
DLL	Dynamic Link Library
DTMF	Dual Tone Multiple Frequency
EPOC	Operating system designed for small, portable computer-telephones with wireless access to phone and other information services.
FEC	Forward Error Correction
FHS	Frequency Hop Synchronization
FHSS	Frequency Hop Spread Spectrum
FTP	File Transfer Protocol
GAP	General Access Profile
GSM	Global System for Mobile communication
HAN	Home Area Network
HCI	Host Controller Interface
HEC	Header Error Check
HomeRF	Home Radio Frequency
HTTP	Hypertext Transfer Protocol
IAC	Inquiry Access Code
IEEE	Institute of Electronic and Electrical Engineers
IP	Internet Protocol
IR	Infrared
IrDA	Infrared Data Association
ISA	Industry Standard Architecture
ISDN	Integrated Services Digital Network
ISM	Industrial Scientific Medical
ITU	International Telecommunication Union
ITU-T	Telecommunication Standardization Sector of the International Telecommunications Union
L2CAP	Logical Link Control and Adaptation Protocol
LAN	Local Area Network
LIAC	Limited Inquiry Access Code

LM	Link Manager
LMP	Link Manager Protocol
MAC	Medium Access Control
Master	Any device that is currently controlling another device, a slave.
MCP	MOE Central Process
MMI	Man Machine Interface
MOCOMA	MOE Command Application
MOE	Mobile Open Expandable platform
MOEP	MOE Protocol
MPI	MOCOMA Parallel Interface
MTU	Maximum Transmission Unit
MVC	Model-View-Controller
OBEX	Object Exchange protocol
OS	Operation System
OSI	Open System Interconnection, a reference model for network protocols
PAN	Personal Area Network
PDA	Personal Data Assistant
PDU	Protocol Data Unit
PI	Protocol Interface
PIN	Personal Identification Number
PM_ADDR	Parked Member Address
Power Class	Three different categories of Bluetooth devices, based on transmission power. Power Class 3 is the most common.
PPP	Point to Point Protocol
Q.931	ISDN's connection control protocol
QoS	Quality of Service
RFCOMM	Protocol for RS-232 serial cable emulation
Scatternet	Bluetooth network formed from several overlapping piconets.
SCO	Synchronous Connection Oriented
SDP	Service Discovery Protocol
SIG	Special Interest Group
Slave	Any device that is controlled by another device called the master.
SWAP	Shared Wireless Access Protocol
TCP	Transfer Control Protocol
TCS	Telephony Control protocol Specification
TDM	Time Division Multiplexing
TS	Telecom Standard
UDP	User Datagram Protocol
UI	User Interface
URL	Universal Resource Locator
USB	Universal Serial Bus

UUID	Universal Unique Identification
VAU	Video/Audio Application
VTP	Video Transfer Process
WAP	Wireless Application Protocol
WCDMA	Wideband code-division multiple access
WESA	Wireless EPOC Software Application, infrared controlled Lego robot
Wi-Fi	Wireless Fidelity, another name for IEEE 802.11b
WLAN	Wireless Local Area Network
WUG	Wireless User Group

FOREWORD AND ACKNOWLEDGEMENTS

This thesis has been written from February to December 2001 in Tampere, Finland.

I would like to thank my supervisor, Professor Jouni Lampinen for his help, comments and support. A warm thank you to all my co-workers at TietoEnator Wireless SWP for support and feedback. Especially great appreciation for all those, who took part in the MOE Project in one-way or another – Karoliina Heikkilä, Elina Humaloja, Sanna Santanen, Vesa Kuoppala, Antti Hannonen, Antti Huokko, Ville Aittomäki, Ville Somppi, Kimmo Leppäaho and Jani Alakoski. Additionally, I would like to thank Ville Aittomäki for guidance and supportive ideas throughout the project. I'd also like to thank my superior, Harri Luuppala, for letting me into his fine team. Finally, I'd like to thank my friends and family who have been supportive in their own unique way.

Sami Merilä, 03.05.2002

1 INTRODUCTION

In this chapter, some background for the thesis is provided. In addition, some of the main characteristics of Bluetooth and wireless networks are detailed and the objectives of the thesis are given.

1.1 Objectives of the thesis

The main objective of this thesis is to design and implement a Bluetooth platform that is easy to expand on. The architectural design of the platform is to be efficient, yet allow variety of applications to be used. The platform is to be as open and modular as possible to permit maximal reusability and portability in forthcoming projects and expansions. If the design for platform needs to be heavily modified whenever a new application is added to the system, the goals set in this thesis are clearly failed.

As the upcoming market for Bluetooth devices will be enormous, it is very important that application vendors commit themselves to the new technology. One of the key aspects of this thesis is to learn - to learn about Bluetooth and to learn about open server design and implementation. Learning about Bluetooth standard is especially important, since the Bluetooth standard is huge. The total page count of Core and Profile parts is approximately two thousand pages. This thesis concisely describes the most important issues of the Bluetooth in a fraction of page count.

Another goal is to provide guidelines and tips to design and implement wireless systems. The guidelines are best suited for systems that use Bluetooth for transmitting data over the air. The guidelines are not complete, but cover the issues that were raised when designing and implementing this thesis.

The result of this thesis, MOE System, will be used as a proof of TietoEnator's knowledge of and experience with wireless systems when marketing company's competence to the clientele. To prove the versatility of the platform, a simple example application and user interface is also included in the thesis.

The first commercial solution based on the results done for this thesis is being designed. It is based on concept of MCP given in this work, but the application(s) and user interface are to be quite different. Indeed, the solution is to be machine-to-machine communication system.

1.2 Background

Due to complexity of this project, some information about what was the starting point of the project is first provided.

MOE-project is based on earlier WESA (Wireless EPOC Software Application)-project. WESA was an educational project to study how people who are not familiar with EPOC (currently known as Symbian OS) operating system, could easily learn to program applications with it. Because of WESA-project a simple EPOC-based application was created that was able to control a small robot. In WESA, the wireless link between the robot and the controlling device was an IR-solution.

As WESA was used to demonstrate skills and abilities of the Wireless R&D department, it was soon decided that a more mature and professional solution was needed. The IR-link of WESA was to be replaced with Bluetooth. At the same time, more experience about the various new wireless techniques was also required. Thus, MOE-project was born.

With MOE project, the goal was set to be a wireless platform that would be easy to expand when needed. The project was to be properly designed, so that documentation could be used

on commercial projects as guidelines. Even though MOE shared seemingly similar finished product with WESA, none of the components of the WESA project could be used for MOE project. However, documentation of that project was used as guidelines for MOE.

The first stage of the MOE-project was finished up during April 2001, when Markus Penttilä completed his Master's Thesis about the technology and hardware requirements needed by the implementation and design of the MOE-project. This Master's Thesis is based on the requirements and suggestions given in that work. In short, the hardware specifications for both the client and server end were defined there. [PENT01]

The second stage of MOE-project was started during the latter phases of the first stage. Second stage consists of the issues specified in this thesis.

There is also a planned third stage of MOE-project. This consists of designing and implementing video camera application and integrating it into existing MOE System. In this stage, the replacement of RCX-unit with homespun hardware would be implemented as well.

1.3 Bluetooth

What is Bluetooth? A very simplified definition could be that Bluetooth represents a possibility for devices to communicate without cables. [PALO01]

However, behind all this simplicity lies a complex and full-featured system, with many components and layers of abstraction. A more exact and technical definition is that Bluetooth is a low cost, low power, microwave wireless link technology. It is designed to replace awkward communication cables between electronic devices. [NOKI01]

In order to actually succeed in replacing cables, a couple of requirements are needed:

1. Technology cannot be much more expensive than current cables.
2. Since Bluetooth is aimed for mobile devices, it must be able to run on batteries. Therefore, it should be low power and be able to run with low voltages.
3. It must be as easy and convenient to use as plugging a cable in.
4. It must be as reliable as the cable it replaces.

Bluetooth specification is open and it defines the complete system from the radio up to application level. Specification is divided into two parts – Core part that details how the technology works and Profiles part that focuses on how to build interoperable devices using Bluetooth technology. [PALO02]

Version 1.0 of Bluetooth specification was published in July 1999, and it was soon followed by version 1.1 that included specifications for cordless telephony, synchronization, Internet access and several new profiles. In addition, several errors and unclear issues from previous standard version were solved. To qualify a product for Bluetooth version 1.1 was made a lot more difficult, to avoid compatibility issues. Version 2.0 is in the works and was initially planned to be released during summer 2001, but has now been postponed.

Bluetooth specification is promoted and defined by Bluetooth Special Interest Group (SIG) that is formed by a group of companies, such as Ericsson Mobile Communictions AB, Nokia Mobile Phones, IBM Corp, Intel Corp, Microsoft, Lucent, 3Com and Motorola. The total number of companies in Bluetooth SIG is currently over 2100. [ERIC01]

It is estimated that there are half a billion Bluetooth devices in use by 2004, with a total market for Bluetooth components worth 2 billion dollars that same year. Therefore, the economic situation for Bluetooth looks quite attractive. In order to meet these estimates, the price of a single Bluetooth chip should be low. The price of a mass produced Bluetooth chip

is forecasted to be less than five dollars, but current solutions are still much more expensive. [BRAY01]

The Basics of the Bluetooth technology is provided in chapter two and in chapter four the Bluetooth protocol stack is examined.

1.4 Personal Area Network (PAN)

"*There's a plenty of space available in a shoe for circuitry – no companies are yet fighting over access to your shoes, but they will ... shoe computers are no joke.*"

Neil Gershenfeld [MIT99]

Bluetooth and other short-range communications technologies have brought up the issue of personal area network (PAN). PAN is a close range ("personal bubble") wireless network in which devices use ad-hoc networking for easy connections. [PAN01]

Ad-hoc network means that devices are part of the network only for the duration of the communication session, or when a mobile device is in close proximity to a local area network (LAN).

PAN differs from LAN and from home area network (HAN) in that it is based around its' user, not around the devices in the network. Indeed, roaming is one of the key aspects of PAN. Typical devices in PAN are mobile phones, laptops, headsets and wireless controlling devices. However, in the near future devices that look like normal items, such as watches and clothes will diversify these devices. [SYMB01]

1.5 Outline of the thesis

The chapter two will introduce Bluetooth technology and give details about why use Bluetooth technology in a project such as this. The chapter is collected information from

various sources, most important of which are Bluetooth Standard [CORE00] and “Bluetooth – connection without cables” [BRAY01]. In chapter three, the structure of the architecture of the MOE System is revealed. This chapter is the heart of MOE design and wholly creation of this thesis’ author. Chapter four detail Bluetooth protocol stack and how MOE System uses it. Apart from details of MOE Protocol, this has been compiled from various sources. Chapter five describes the server end of the MOE System. Chapter six will then shed light on client end of the MOE System that is used to access services provided by the server end. Finally, in chapter seven conclusions and future considerations are provided. Last three chapters are mostly written solely for this thesis.

2 BLUETOOTH

The basics of Bluetooth are depicted in this chapter. Additional details of the Bluetooth protocol are described in the Chapter 4.

2.1 Introduction to Bluetooth

Bluetooth operates in the globally available ISM band from 2400 MHz to 2483.5 MHz – using 79 channels (in earlier version of the standard some countries, such as Spain, France and Japan used only 23 channels). This band is generally reserved for use of Industrial, Scientific and Medical (ISM) applications. In this band, Bluetooth is able to transport data with flow lower than one megabit per second – peak rate is approximately 721 Kbps. This is actual data rate without headers - raw data rate is 1 Mb/s. [RFI01]

Bluetooth devices can function, either in an asynchronous packet switched mode (ACL link) with flow from 57.6 Kbps to 721 Kbps, or in a synchronous circuit switched mode (SCO link) with maximum flow of 64 Kbps. The operating range for Bluetooth devices is approximately from 10 centimeters to 10 meters, with power consumption of 1mW (0dBm). However, higher ranges, up to 100 meters are possible with higher Power Classes. These, however, consume more power (up to 100mW (20dBm)). [RFI01]

2.1.1 Slot, Packet and Frame

Bluetooth is a Time Division Multiplex (TDM) system, where the basic unit of operation is a slot of 625 microseconds (μs). Time Division Multiplex means that communications channel is split up among several users. One user at a time uses a channel during his/her time slot, and then another user is switched active and so on. [CORE01]

In transferring and receiving data, all operations occur in one, three or five slots. In some operations (for example inquiry or paging), transmission and receive can occur in half a slot.

No matter what the length of the data operation, for simplicity's sake, the transmission of data is called a packet. Therefore, packet pair consisting of transmit and receive packets, can be 2, 4, 6, 8 or 10 slots long. A packet pair is also known as a frame. [INTEL01]

Since all packets have the same header and data overhead, sending long packets is more efficient. On the other hand, long packets are more susceptible to interference and therefore the whole packet can more often be discarded, resulting a retransmission, which can have a significant impact on the efficiency of the system.

In Figure 1 is depicted in a simplified way the differences between slots, packets and frames. Note that a 20-microsecond uncertainty window is allowed around the exact receive time in order receiver to have time to synchronize with the transmitter.

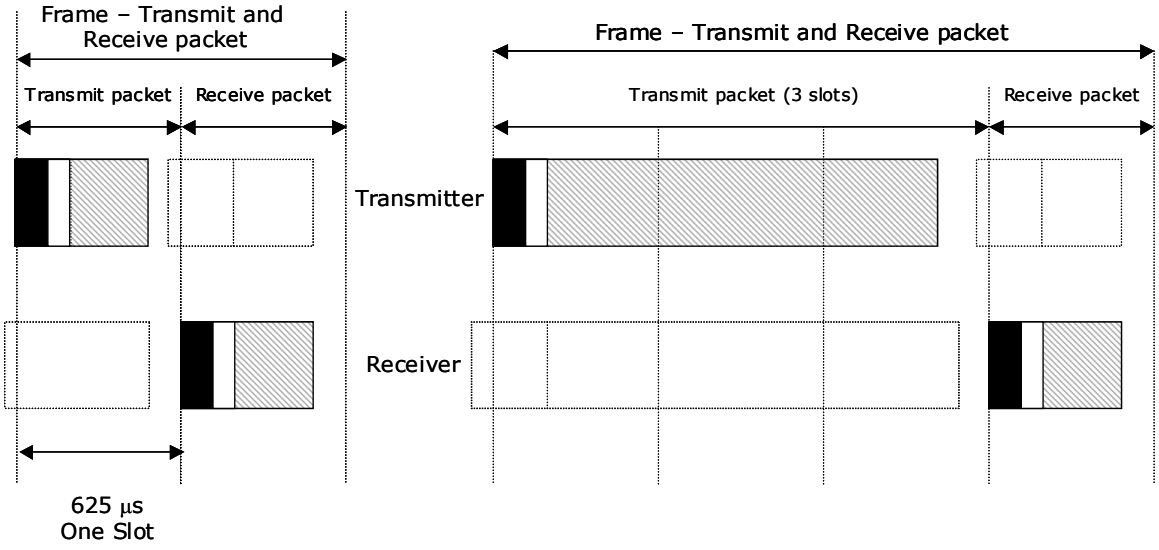

Figure 1 - Slots, packets and frames

2.1.2 Frequency Hopping

Bluetooth devices utilize 79 one MHz wide channels. On these, they periodically hop using pseudo-random algorithm. Randomness is based on device address of controlling device in

the network. Hopping rate is 1600 hops per second. Each device will hop once per packet. Due to nature of hopping, it makes Bluetooth network secure and reliable – if certain channel is badly disturbed by radio interference; another channel is less likely to be so affected. [RFI01]

2.1.3 Roles

Each Bluetooth device can either be in a Master, or in a Slave role. These roles are defined as follows:

Master – Device initiates and controls the data exchange. Master transmits packets on even numbered slots.

Slave – Device, which responds to a Master. Slaves transmit packets on odd numbered slots, but it can only answer to Master's transmissions (except in reserved SCO slots). A Slave has to always respond to Master's transmission. Slaves use Master's timing information to synchronize hopping.

No device can simultaneously be a Master and a Slave in the same piconet (piconet is a Bluetooth network, see next chapter). However, a device can function as a Master in one piconet, and as a Slave in another. It will change roles on TDM basis.

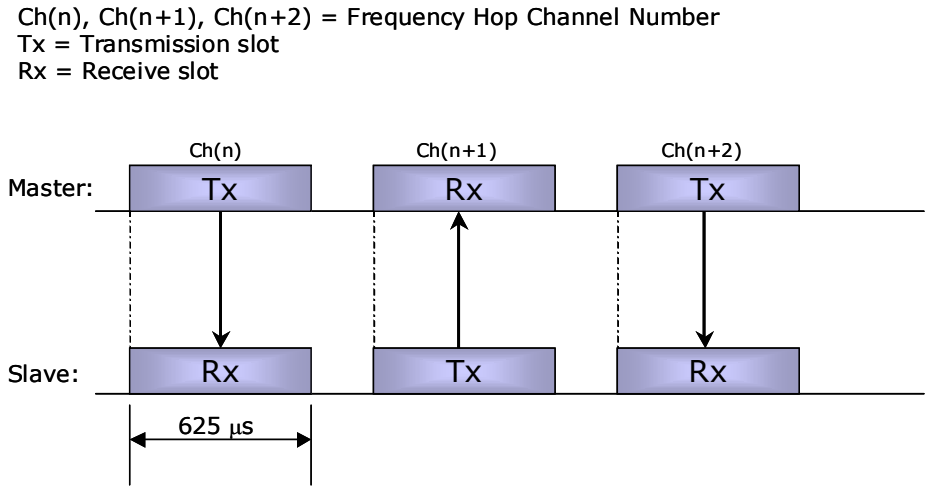

Figure 2 – Basic communication between Master and Slave

As is shown in Figure 2 - a Slave receives a packet in a receive slot (Rx) from a Master on some channel CH(n). A slot later, both devices re-tune to another channel CH(n+1) and now Slave is required to transmit a respond in a transmit slot (Tx). Another slot later CH(n+2), Master can transmit again, but is not required to do so. However, Slave is not allowed to transmit again until Master has transmitted to it first. [BRAY01]

Master decides what to transmit (in 1, 3 or 5 slot packet) and when to transmit it to Slave(s). Slave waits and listens for packets with its' own address. When Slave recognizes from received packet header its own address (Active Member Address) or broadcast indicator (address is zero), Slave receives the whole packet and responds to it. Broadcasting means, that Master is sending the same data to all other devices, and all the other devices will receive it. [BRAY01]

2.1.4 Piconet

A Bluetooth network is called a piconet. Each piconet can contain up to eight Bluetooth devices. Of these, one functions as a Master and the rest of the devices function as Slaves. In addition, several Slaves (up to 255 in each piconet) can be in Park mode (see Chapter 4.3.2.3.2) and only periodically listening to the Master. However, these Slaves cannot receive

or transmit data before they re-activate themselves again by receiving an Active Member Address from the Master. [BRAY01]

One key aspect of Bluetooth is that network is omni-directional, with no line of sight required between devices. As such, it is ideally suited to work indoors or within crowded spaces such as public areas.

If the piconet consists of a single Slave and a Master, the connection is called point-to-point (see Figure 3a). This network typology can be understood as a replacement for IrDA's infrared connection. For example, it can be to synchronize a PDA device with a PC. With several Slaves, the connection is called point-to-multipoint (see Figure 3b). The Master can send data to each of the Slaves (and vice versa), or to broadcast message to all Slaves in the piconet.

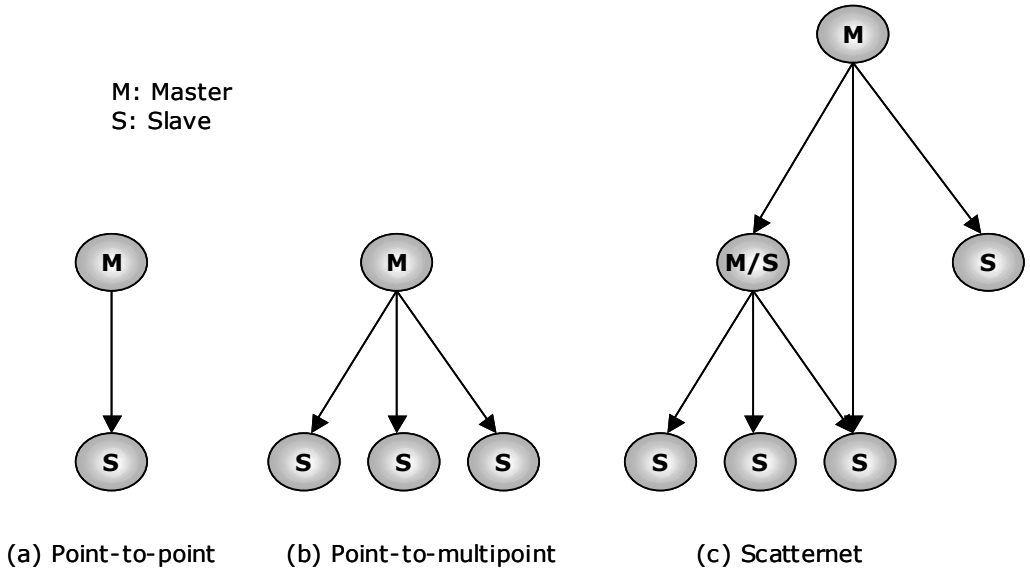

Figure 3 – Various Bluetooth piconets

As in figure 3c, each Master can also be a Slave in another piconet, thus forming a scatternet. Such a device switches roles on TDM basis – being a Master in its own piconet for a time and switch to be a Slave in another Master's piconet. [INTEL01]

Up to ten piconets can overlap – linking up to 80 Bluetooth devices. When two or more piconets are overlapping, they form a scatternet. Beyond 80 devices, the network saturates, since there are only 79 transmission channels available. [BRAY01]

A device that is active in two or more piconets must keep track of several timebases. It takes some time, switching from one timebase to another. This limits how many piconets can be linked together. Each switch from piconet to another requires sending of guard slots before starting to receive and transmit on the new piconet. If there are SCO links in one of the piconets, it means that at least every third frame (the widest spacing of SCO frames) is reserved. Therefore, there are only four slots (two to Master, and two to Slaves) available for other transmissions. Of these, two are reserved for guard slots. This means that only two piconets can be linked when there is an active SCO link in either of the piconets. [BRAY01]

2.1.5 Addresses

Bluetooth devices have a number of special addresses. First, they have a unique Bluetooth Device address that no other Bluetooth devices in the world possess. This is hereafter referred as BD_ADDR. BD_ADDR is a 48-bit IEEE MAC (Medium Access Control) address. This is used to initialize a number of functionalities in the Bluetooth hardware (for example HEC and CRC calculations, frequency hopping). In active connection, Master's BD_ADDR forms a part of Access Code. When in other modes of operation, a special device address (such as Inquiry Access Code) can be used. [PALO03]

Second, an active Slave in a piconet has an Active Member Address (AM_ADDR) that is used to distinguish a device in a piconet. It is assigned by the piconet's Master to a Slave after successful paging procedure. Maximum number of Slaves is seven, because AM_ADDR field in packets is only three bits long and address 000 is reserved for

broadcasting identification when Master is sending. Address 000 is also reserved when Slave is transmitting, but in this case, it means that the packet is directed to the Master. [PALO03]

Third, a parked slave has a Parked Member Address (PM_ADDR) that is used to separate parked devices from one another. It is eight bits long, so there can be 255 parked Slaves. PM_ADDR is valid for the time that Slave is parked. [PALO03]

2.1.6 Role Switching

During the initial connection, or at any time while connection is active, Bluetooth devices can request to switch Master and Slave roles. Either of the devices can make this request. Of course, a Slave cannot initiate communication, but it may request role switch after Master has transmitted to it. Role switch can even occur with a Master that already has several Slaves. This kind of situation is detailed in Figure 4. First, Slave B initiates role switching to which Master A agrees. Then the single piconet is split in two. In the first one, A is still the Master having two slaves (Slave C and Slave D), and in the second piconet, A becomes a Slave for a new Master B. [BRAY01]

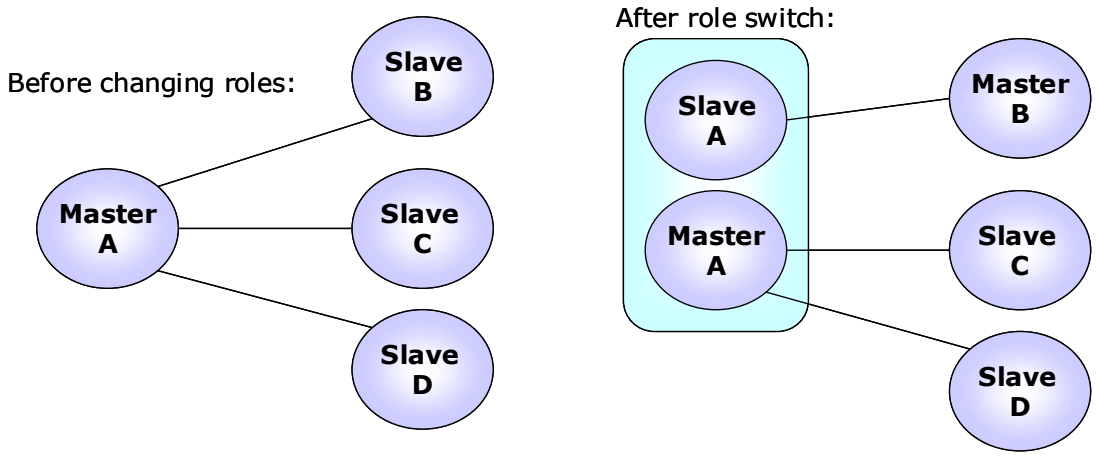

Figure 4 - Role switch with a Master that has Slaves

2.1.7 Links

Links are formed between a Master and a Slave. A link can carry over itself several multiplexed channels. Bluetooth supports two types of links. For asynchronous connections, there exists an ACL link. It should be used whenever transmitting data. For synchronous connections, SCO link is to be used. It should be utilized whenever having voice connections. [BRAY01]

2.1.7.1 Asynchronous link

An ACL link is created between a Master and a Slave immediately after a connection has been established. The Master can have a number of ACL links between several Slaves, but only one ACL link between each Slave. If a Master transmits an ACL packet to a Slave, it *must* answer with ACL packet in the next available slot. In point-to-multipoint connection, ACL packets can also be broadcasted, so that each Slave can receive the same data simultaneously. [BRAY01]

ACL links carry data for higher layers, such as LMP or L2CAP. User data is carried through L2CAP and passed to the baseband. With ACL link, maximum baseband data rate in one direction is 723,2 kb/s. This is achieved by sending data out in 5-slot packets, and replies are received in 1-slot packets. Short slot when receiving causes that the reply rate is only 57,6 kb/s. If 5-slot packets are used for transmissions and receiving, the data rate for both directions is 433,9 kb/s. [BRAY01]

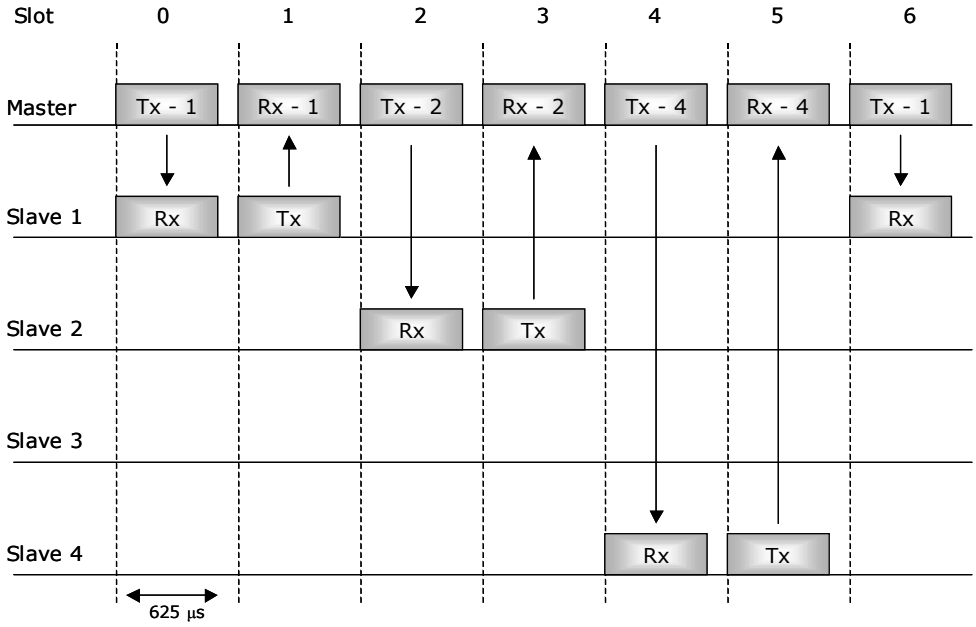

Figure 5 - Piconet formed with a Master and four Slaves with ACL traffic.

In Figure 5 is inspected how asynchronous data is transferred in a piconet formed from a Master and four Slaves. First, the Master sends a packet to a Slave 1, which is required to answer and does so in the next slot. Then the Master sends a packet in slot two to a Slave 2, which is also required to answer immediately. In slot four, the Master sends a packet to a Slave 4, which also answers at once. Last, in slot six, the Master sends data again to the Slave 1. Note that Slave 3 doesn't receive any packets and thus cannot send any.

2.1.7.2 Synchronous link

Synchronous connection between devices is used to transfer constant flow of data, such as audio or video stream. In Bluetooth, this type of connection is called SCO (Synchronous Connection Oriented) link. The maximum data rate for SCO links is 64 kb/s. Due to rather low data rate SCO links are not suited for delivering high quality music or speech. [BRAY01]

SCO link is always created on top of existing ACL link. Either a Master or a Slave can request to set up a SCO link. When the link is formed, slots that are separated by SCO slot interval are reserved for transmission for this SCO link – this is a regular interval and will be enforced until SCO link is dismantled. SCO slot interval can be every slot pair, every second slot pair or every third slot pair. Therefore, a Master can have up to three SCO links active at the same time. These links can be to a single Slave, or to several Slaves. [BRAY01]

During the reserved slots, the Master can only send data to the Slave it created a SCO link with, except when sending out LMP broadcast message, which will take precedence over the SCO link. Due to the nature of SCO data, the SCO packets are never re-transmitted. Erroneous SCO packets are simply discarded. [BRAY01]

Since SCO links use dedicated slots they reduce the efficiency of other operations, such as inquiry, inquiry scan, page or page scan. With a single SCO link, 60% of inquiring time in a piconet is lost. This increases up to 90% with two SCO links. Paging and page scanning are even worse affected. [BRAY01]

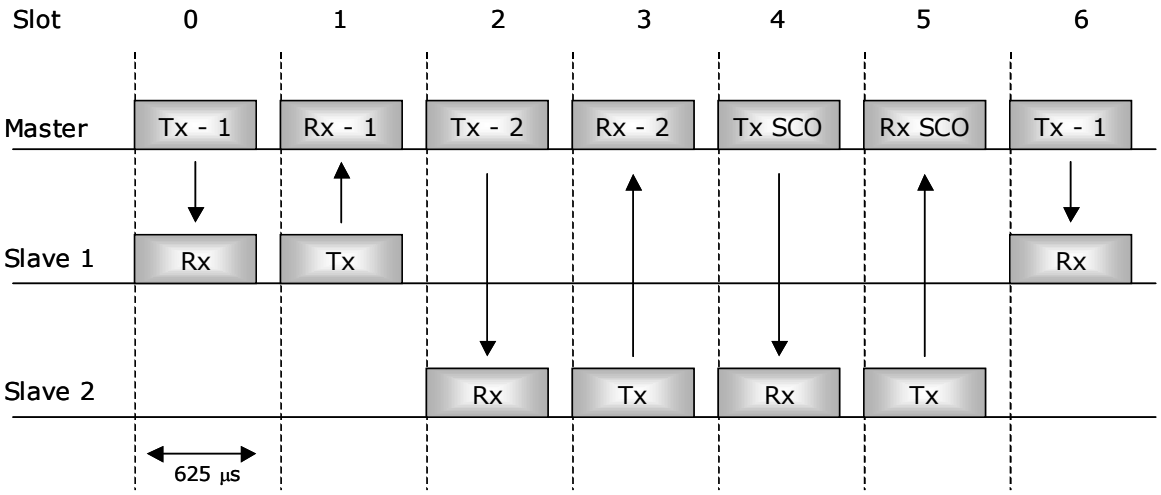

Figure 6 - Piconet with mixed ACL/SCO traffic.

In Figure 6 a Master has one Slave that has a SCO connection and another Slave that has an ACL connection. The SCO slot interval is every third in this example. Just before sending the SCO packet, the Master sends a normal packet in slot 2 and waits for the Slave 2's answer in slot 3. After this, the SCO data is sent from the Master. Because both sides of the SCO connection know when to expect SCO packet, Slave 2 can answer in slot 5 even if the Master's packet in slot 4 was lost. Reserved SCO slots are represented in the picture by "SCO" text in the packet.

2.2 Security in Bluetooth

Bluetooth has several layers of security. It is suggested that the security procedures should be implemented into the upper layers of the Bluetooth protocol stack. It should be emphasized that the complex security mechanisms should be hidden from the user in order to allow ease of use of Bluetooth devices.

The first security mechanism of Bluetooth is the high speed pseudo-random hopping. It is very difficult to eavesdrop. Short range of Bluetooth makes this even harder. This cannot be trusted, however, since the predicted volume of Bluetooth chips is huge, so that sensitive data could get into wrong hands. [INTEL01]

2.2.1 Pairing

Pairing is a procedure that authenticates two devices, based on a PIN, and subsequently creates a common link key that can be used as a basis for a trusted relationship or a (single) secure connection. The procedure consists of the steps: creation of an initialization key, authentication based on the initialization key and creation of a common link key. The devices that took part in the procedure use the link key for future authentication when exchanging information. [TRÄS00]

2.2.2 Authentication and Authorization

Authentication means a process of verifying the identity of the other side of the connection. Bluetooth uses challenge-response scheme for authentication, where it is used to check whether the other party knows a shared identical secret key (a symmetric key). The protocol checks that both devices have the same key, and if they do authentication is successful. If the authentication fails, there is a delay until a new attempt at authentication can be made. This can subsequently increase or decrease depending on the results of previous authentication attempts. To achieve this, Bluetooth employs 128-bit cipher algorithm SAFER+ for authentication.

Note that secret key is never transmitted on air. Instead, it can be on hardware, or it can be entered as a PIN (Personal Identification Number) code, so that the user can validate that he/she can access transmitted data. Once Bluetooth radios have authenticated themselves, encryption of the data can occur. [TRÄS00]

2.2.3 Encryption

The Bluetooth encryption system systematically encrypts the payload of each packet. A simple three-stage conversation is required for encryption. First, the Master requests what encryption mode to use. Available choices are none, all packets and only point-to-point packets. Slave approves this and then devices negotiate agreeable key size. Last, the Master requests the Slave to turn on the encryption. [TRÄS00]

2.2.4 Security Levels

Bluetooth devices can either be trusted, or untrusted. Trusted device can use the other side's services unrestricted, though an authentication may have to be done. Untrusted devices have only limited access to the other side's services. A newly discovered device is always deemed as an untrusted device. [TRÄS00]

2.3 Profiles

The second part of Bluetooth standard, Profiles, defines different ways to utilize Bluetooth technology. Since each Bluetooth device must support at least one profile, profiles ensure interoperability by providing well-defined set of higher layer procedures and standardized ways to use the lower layers of Bluetooth. Each profile matches to usage model. [BRAY01]

In Figure 7 is shown how the Bluetooth Profiles are organized. Simply, first a broad group Generic Access Profile is provided. Within it exists groups that are more specialized at certain functionality. Boxes with gray background are so specialized that they don't have sub-groups within them. All profiles are detailed later on this chapter. The upcoming 2.0 version of the Bluetooth standard is likely to add several new profiles, such as a Car profile.

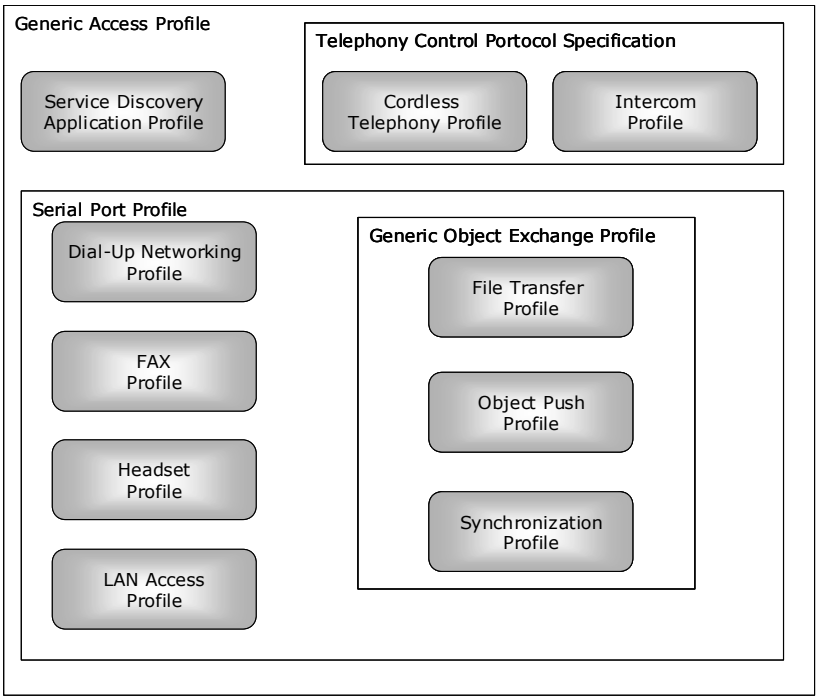

Figure 7 – Bluetooth Profiles

2.3.1 Generic Access Profile

Generic Access Profile (GAP) is the most basic of the various Bluetooth profiles. All other profiles are built upon GAP and they use its definitions. The main purpose of GAP is to provide possibilities for all the devices to establish a Baseband link. In order to manage this, GAP defines General requirements that *must* be implemented in all devices. In addition, it defines the generic procedures related to discovery of Bluetooth devices, link management aspects of connecting to Bluetooth devices, different security levels and how to use them and finally, common format requirements for parameters accessible on the user interface level. [PROF01]

One of the main issues of GAP is that it provides naming conventions to use with user interfaces. [BRAY01] In addition, GAP defines certain modes of operation. These are collected into Table 1 below.

Table 1 – Modes defined in the Generic Access Profile

Mode Name	Description
Discoverability	Device is either in **discoverable** or **in non-discoverable** mode. When a device is in non-discoverable mode it will not respond to inquiry. Therefore, it can never be discovered. There are two different ways to be discoverable – limited or general. Devices in limited discoverable mode only respond to inquiries with Limited Inquiry Access Codes (LIAC). Device in discoverable mode will from time to time start inquiry scan. [PROF01]
Connectability	Device is either in **connectable** or in **non-connectable** mode. When a device is in non-connectable mode, it will not respond to paging. Therefore, it will only connect to devices when it has itself initiated the connection. Device in connectable mode will from time to time start to page scan. [PROF01]
Pairability	Device is either in **pairable** or in **non-pairable** mode. In pairable mode, a device is capable to set up a link key with another device. Then these devices are bonded. This means a trusted relationship, which can be used at higher layers for authentication. In non-pairable mode a device will bond to other devices. [PROF01]
Security	There are three security modes [BRAY01]: **Mode 1** – security is never initiated (non-secure). **Mode 2** – security is initiated when creating L2CAP channel and enforced according to the requirements of the service (service level security). **Mode 3** – security is initiated when ACL link is created (link level security).

2.3.2 Service Discovery Profile

This profile describes how applications running Service Discovery Protocol (a protocol dedicated to discovering nearby Bluetooth devices and their services) use that protocol to discover available Bluetooth services within range. Profile contains a number of primitives that can be used to drive service discovery. In addition, this profile includes examples of messages and sequences between devices using Service Discovery Protocol. Service Discovery Profile requires that device supports pairing and authentication. [PROF01]

2.3.3 Serial Port Profile

This profile defines RS-232 serial cable emulation. The profile can be used for existing devices to start using Bluetooth instead of serial port without making any modifications. The serial port profile uses RFCOMM to provide serial port emulation. [BRAY01]

2.3.4 Dial-Up Networking Profile

The Dial-up networking profile provides a dial-up data connection. This allows laptop to use cellular phone as a wireless modem or to receive data calls via it. Two types of connection are possible, a remote access service or a cordless modem. [BRAY01]

2.3.5 FAX Profile

The FAX profile defines procedures for sending and receiving faxes. It is very similar to dial-up networking profile. Three classes of faxes are supported, FAX Class 1 (ITU T.31), FAX Class 2.0 (ITU-T T.32) or manufacturer specific. [BRAY01]

2.3.6 Headset Profile

This profile defines how to make and receive hands-free voice calls between Bluetooth devices. There are two roles defined in this profile: audio gateway and headset. [BRAY01]

2.3.7 LAN Access Profile

This profile allows Bluetooth device to access fixed network using Bluetooth link to a LAN Access Point. LAN access is secured using a PIN code. [PROF01]

2.3.8 Generic Object Exchange Profile

This profile uses IrDA's Object Exchange (OBEX) and defines how it is used within Bluetooth. The profile defines two roles: server and client. Server is a device from/to objects are pulled or pushed. Client is a device that can push or pull objects from a server. [BRAY01]

2.3.9 File Transfer Profile

File transfer profile provides wireless data transfer between devices. This profile, as all other profiles upon Generic Object Exchange Profile, use OBEX for transactions. Additional features that are only available in File Transfer profile are concepts of files and folders. Folders can be browsed, created, deleted and transferred. Files can be browsed, pulled, pushed and deleted.

2.3.10 Object Push Profile

This profile defines ways to exchange business cards between devices. As with all OBEX transactions, client initiates the operations. Pushed data has to conform into one of the predefined formats (for example, vCard). [BRAY01]

2.3.11 Synchronization Profile

The Synchronization profile provides a standard way to synchronize data between two Bluetooth devices. Synchronization can be triggered automatically when Bluetooth devices are within range of each other. Synchronization can be protected with user entered PINs, or using trusted devices. [BRAY01]

2.3.12 Cordless Telephony Profile

The Cordless Telephony profile enables Bluetooth devices to connect to various telephony base stations. This profile defines two roles: gateway and terminal. Gateway connects to external network and receives incoming calls. It functions as a Master for the piconet. Up to seven terminals can connect to a gateway simultaneously, just like in a normal piconet. Of these, only three SCO links can be active at the same time. Terminal receives the calls from gateway and provides speech and/or data links to the user. [BRAY01]

2.3.13 Intercom Profile

The Intercom profile provides support for direct voice connection between two Bluetooth devices. It is most likely to be used with higher Power Class devices that provide ranges up to 100m. [BRAY01]

2.4 Comparison of wireless technologies

There are several other new wireless technologies besides Bluetooth. In the next few sub-chapters, key aspects of these systems are examined and compared with the Bluetooth technology.

2.4.1 Bluetooth vs. IrDA

IrDA (Infrared Data Association) is a communication system based on infrared light. Therefore, it is limited by the line of sight and it is unable to penetrate obstacles such as walls and furniture. On the other hand, this ensures private communication.

At first glance, IrDA and Bluetooth seem to be competing of the same market niche. With careful examination, however, Bluetooth and IrDA are quite different and suited for different tasks. One of the greatest weaknesses of IrDA is that it has a narrow (30° cone) and short (up to two meters) visibility. When compared to Bluetooth's no line of sight network and 10 meters of range, it seems, that Bluetooth is highly superior to IrDA. However, the data transmission speed in IrDA is significantly faster than in Bluetooth (up to 16Mb/s vs. close to

1Mb/s). In addition, IrDA technology has been in use for few years, (the IrDA work group was created in 1993 and there are more than 50 millions IrDA products installed world-wide) so most of the glitches and problems of the technology have been fixed. On the other hand, Bluetooth provides security mechanisms that are not part of IrDA. For consumer markets, the price of IrDA chip (around one dollar) is excellent. [SUVA00]

The differencies between Bluetooth and IrDA are collected to Table 2 that is presented later in this chapter.

Even if Bluetooth would become the de facto standard for wireless communications, IrDA has its place for sending of large amount of data with point-to-point connections. In addition, the application framework of Bluetooth is aimed to achieve interoperability with IrDA's OBEX. Therefore, Bluetooth does not aim to override IrDA, but to co-exist with it.

2.4.2 Bluetooth vs. IEEE 802.11

IEEE 802.11 workgroup is defining and maintaining specification for Wireless Local Area Network (WLAN). Currently the specification is divided into three parts: IEEE 802.11-1999, IEEE 802.11a-1999 and IEEE 802.11b-1999.

The original specification IEEE802.11-1999 uses frequency hop spread spectrum (FHSS) with 2.4 GHz ISM band, which is the same band that Bluetooth is using. The hopping scheme is similar to Bluetooth's, but with much slower hopping rate. This allows the technology to be easier to implement. The IEEE 802.11 is capable of transmitting up to two Mb/s, but this rate is expected to rise with future add-ons to the specification. However, when raising the bandwidth the noise-sensitivity increases. This can be avoided with careful and expensive radio design and implementation. [IEEE01]

The IEEE 802.11b ("Wi-Fi" – Wireless Fidelity) includes the possibility to provide 5.5 or even 11 Mb/s bandwidth in the same network. Currently new versions for standards of IEEE 802.11a and HomeRF (or SWAP) are putting Wi-Fi into an awkward position. Wi-Fi is insecure and likely to be interfered by both Bluetooth and micro ovens. In addition, Wi-Fi lacks support for voice and streaming media. Due to these problems, it is likely that IEEE 802.11b will not make a great impact on a wireless market. Last, the equipment needed for this technology is complex and not in the low-cost end of Bluetooth and IrDA. [IEEE01]

The IEEE 802.11a specification provides a 5GHz bandwidth with up to 54Mb/s data rate, but the specification is complex and thus the chip cost is likely to be quite high. However, due to broader bandwidth and using non-crowded frequency the IEEE 802.11a should be robust and reliable. [IEEE01]

Since IEEE 802.11 and IEEE 802.11b, use the same frequency range as Bluetooth, there is likely to be some interference between these technologies. The quick hopping scheme of the Bluetooth devices makes them robust on this crowded band, but WLAN products operating near-by can be affected with lost packages and network errors. Some estimations claim that WLAN devices might have their throughput cut in half. Bluetooth devices will become somewhat more inefficient due to lost packets. [PROX01]

Differences between the three different IEEE 802 technologies and Bluetooth have been collected into the Table 2 presented later in the chapter.

2.4.3 Bluetooth vs. SWAP

SWAP (Shared Wireless Application Protocol) is intended to carry both data and voice by combining most convenient parts of DECT (Digital Enhanced Cordless Telephony) and 802.11 FHSS. SWAP is designed by HomeRF Working Group to be low-cost by using simple radio specifications, but otherwise using the same hopping scheme as 802.11. In

addition, complex parts of the protocol were removed – except for 802.11's TCP/IP support. SWAP can operate either as an ad-hoc network or as a regular network with connection points. [CHIN01]

At first glance, SWAP seems to be a rival technology for Bluetooth, but with more throughout examination the differences between these two are clear – SWAP is intended for Home Area Networking (HAN), whereas Bluetooth is based on the concept of Personal Area Network. Though superior in bandwidth and range, the price of combined DECT and 802.11 units is more expensive than a single-chip Bluetooth. In addition, the size of SWAP device is inconvenient for mobile devices. Finally, SWAP device's power consumption is not suited for battery-powered devices. [CHIN01]

Again, the differencies between Bluetooth and SWAP are collected into the Table 2 presented later in the chapter.

Currently there are more installed SWAP devices than Bluetooth devices, but this is assumed to change quite soon, since most of the major mobile telephone manufacturers are presenting their latest models with Bluetooth. As with comparison with IrDA, SWAP is designed for a bit different area than Bluetooth [HoRF01].

Table 2 - Bluetooth vs. other wireless technologies

Technology	Maximum data rate (Mb/s)	Range (meters)	Maximum nodes	Cost of chip (US$)	Special features
IrDA	Up to 16	1-2	2[1]	1-2	Limited visibility (angle of 30° degrees), line of sight
Bluetooth	Up to 1	Up to 100[2]	8	5+	Omni-directional, ad-hoc connections
IEEE 802.11	Up to 2	a. 100	10 per access point	50+	Noise-sensitive, high-speed
IEEE 802.11a	Up to 54	a. 100	10 per access point	150+	Reliable, expensive
IEEE 802.11b	Up to 11	a. 100	10 per access point	75+	Low security, high-power consumption, no voice/telephone support
SWAP	1.6[3]	50	127	10+	Good support for voice and data, variable levels of security

Notes:

Maximum data rate – how many megabytes per second the technology is able to transmit.

Range – to how far the technology is able to send data reliably.

Maximum nodes – how many devices can use connection simultaneously.

Cost of chip – how much, in US$, does the mass production of the chip cost.

Special features – any noteworthy special features of the technology.

1 – IrDA uses point-to-point connection, so only two devices are communicating with each other.

2 – Normally, the maximum range is only approximately 10 meters, but with Power Class 1 or 2 devices, the range can be up to 100m.

3 – This is for version 1.0, the upcoming version 2.0 will increase the data rate to 10Mb/s.

2.4.4 Reasons for choosing Bluetooth

In MOE-project Bluetooth was decided as the used wireless technology, since it was new and unknown and there is a very high business expectation attached to it. The basis of the project was to get rid of IR-link, so it would have been against the goals of the project to choose IrDA. The other wireless technologies inspected were not suited for small mobile devices, because they have rather high power consumption, making them illogical choices for battery-powered system. In addition, the physical size of Bluetooth chip is convenient in MOE Project for client and server end. Last, mobile telephone manufacturers have stated that their upcoming products will use Bluetooth. This is especially important, since TietoEnator's Wireless SWP is in close co-operation with Nokia Mobile Phones. This way, the competence gained from this project can be used for good effort.

3 SYSTEM ARCHITECTURE DESIGN

MOE System is a rather normal client-server system, where the server functions as a platform for the MOE applications. However, inclusion of Bluetooth makes the system interesting, as the server end and the client end of the system are not physically connected to each other in any way.

3.1 Sub-systems

3.1.1 MOE Client

The user end of the MOE System is physically a handheld controlling device that has a Bluetooth card. This is hereafter referred as MOE Client. MOE Client consists of UI that allows user to send commands to MOE Server over Bluetooth. In addition, parameters that control Bluetooth operations and protocol stack are set from UI.

3.1.2 MOE Server

The server end of the system is a normal PC, possible powered by batteries that is running on Linux and has Bluetooth hardware. This is called MOE Server. MOE Server is slightly more complex than MOE Client is. MOE Server consists of hardware, on top of which runs a controlling application, MOE Central Process (MCP). Applications (such as MOCOMA command application, or VAU Video stream application) can be started on MOE Server. Immediately on start-up, they connect to MCP via sockets. After initial connection, these applications indicate to the MCP the services they provide and how to access them. Now, MCP can market these services to nearby Bluetooth devices using Service Discovery Protocol from Axis' Bluetooth stack. If a nearby Bluetooth device wishes to use certain service, MCP allows connection and starts to pass data between connecting Bluetooth device and application running on MOE Server.

3.1.3 MOE Protocol

MOE Protocol is used to standardize data transfers between MOE Applications, MCP and MOE Client. Each end of the system has a module that is capable of understanding MOE Protocol and is able to issue commands with it.

3.2 System overview

As a Bluetooth system, MOE is designed to be point-to-point. Only a single connection to each MOE Application is allowed, though a single MOE Client can be connected to a MOE Server with both ACL and SCO link.

From Bluetooth profiles point of view, MOE Server is designed to support LAN Access Profile. Therefore, it is also supporting Serial Port Profile and Generic Access Profile. In addition to this, MOE Server supports Service Discovery Profile.

MOE Server is running a number of applications, whose services are available with Service Discovery Protocol to all Bluetooth devices within range. Initially, only one of these applications, MOCOMA, will be implemented. Applications register to MOE Central Process (MCP) using sockets. MCP then commands SDP to make the service available. MCP is connected to all active applications with sockets. Without a connection to the MCP, applications running on MOE Server are not available for other Bluetooth devices; they are just normal local processes. Connection to MCP is only achieved if the application supports MOE Protocol. Otherwise, it cannot understand what the MOE Server is requesting from it after a socket connection to MCP has been created. Immediately after creating a socket connection, MCP requests to know the location of service record. Service record is a description of provided service. In addition, service record includes information required to establish connection to the service. With Axis's Bluetooth stack service records are depicted as XML-files. The stack can send out service record information to the client on request via SDP.

MOE Client is running software that provides user interface for the applications running on MOE Server. MOE Client is supporting MOE Protocol, so it can send commands to MOE Server. On MOE Client, the functionality of the Bluetooth stack is hidden from the user interface by providing a layer that hides the technical details of the connection. In fact, it looks to the UI that it is in socket connection, when in actuality it is having a Bluetooth connection.

In Figure 8 is depicted the high-level architecture of the MOE System.

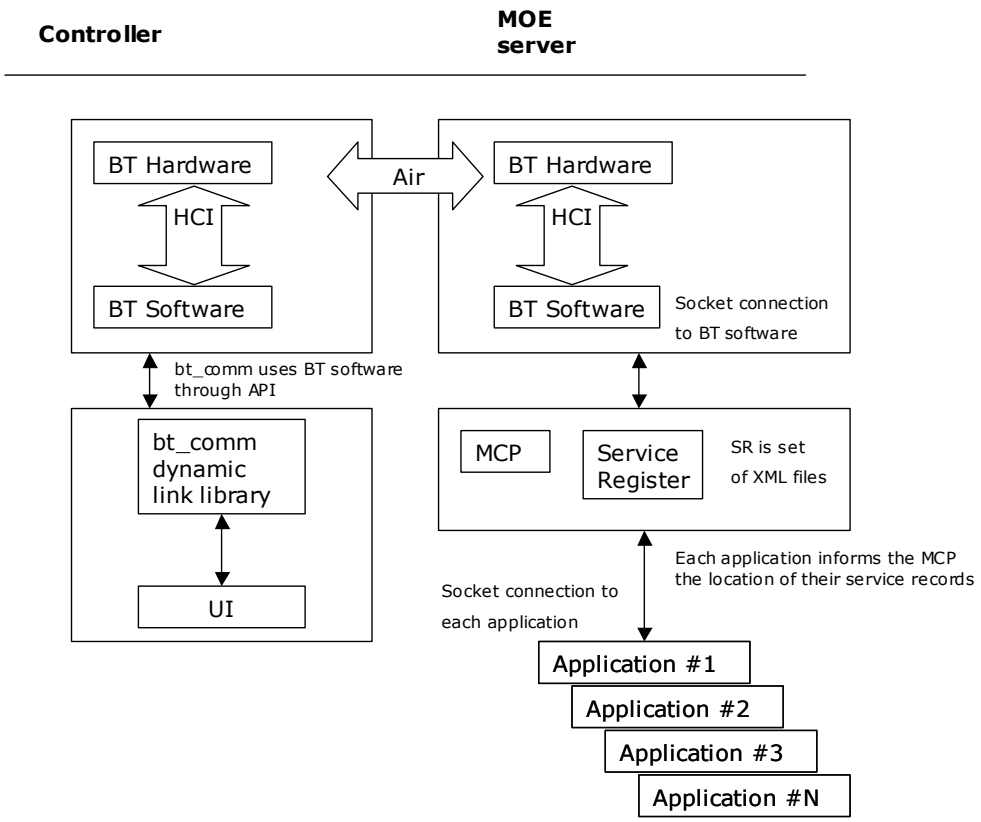

Figure 8 - Architectural view of the MOE System

4 PROTOCOL DESIGN AND IMPLEMENTATION

To clarify different protocols in the Bluetooth protocol stack, lets first see how Bluetooth compares to Open System Interconnection (OSI) reference model. After this, the different layers of Bluetooth protocol are detailed. This is followed by short description of protocols that Bluetooth uses, but which are not part of Bluetooth standard. Finally, MOE Protocol and its relation to the Bluetooth protocol are described.

4.1 OSI reference model

OSI reference model is a standard description for how messages should be consistently transmitted between any two points in a telecommunication network. The purpose of OSI is to define guidelines for implementers so that maximal inter-working capabilities are reached. The model defines seven layers of functions that take place at each end of the communication. The model is symmetrical on both (sending and receiving) ends. [TANE96]

The seven layers and their basic functionalities are:

1. **Physical layer** is responsible for conveying bitstreams through the network. Therefore, it provides hardware for sending and receiving data on a carrier.
2. **Data link layer** is responsible for synchronization of the physical level. It also manages transmissions.
3. **Network layer** is responsible for routing and forwarding the data.
4. **Transport layer** is responsible for end-to-end connection management and error checking.
5. **Session layer** is responsible for setting up, coordinating and terminating data exchanges between the applications on each end.
6. **Presentation layer** is responsible for converting incoming and outgoing data from one system to another.
7. **Application layer** is responsible for identification of each end, authentication and constraining data.

In Figure 9 is shown how Bluetooth protocol stack approximately compares to OSI reference model.

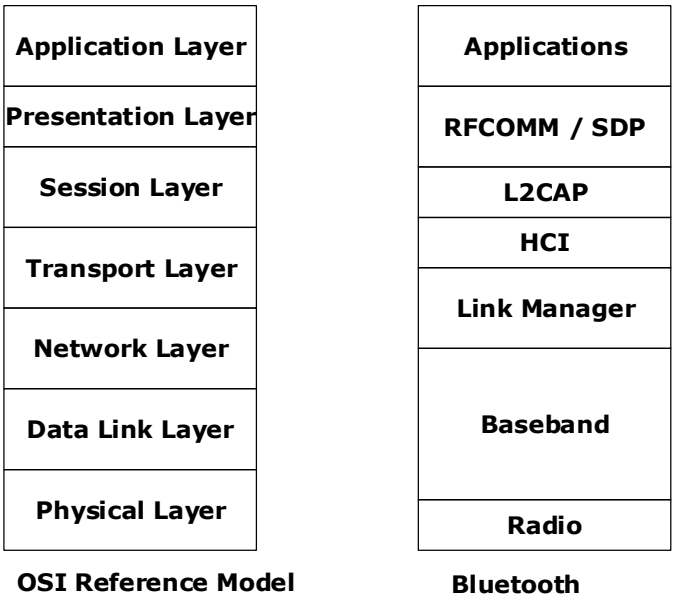

Figure 9 - OSI reference model vs. Bluetooth stack

Since Physical layer is responsible for the electrical interface to the communications media (including modulation and channel coding), it covers all the functionality of the Radio and some of the Baseband layer. Data link layer covers rest of the Baseband functionalities. After this, it gets less easy to compare the protocol stacks. Link Manager contains functionalities that are covered in the OSI model by Network and Transport layers. Host Controller Interface (HCI), however, contains some operations that are in these two layers in the OSI model. Session layer provides management and data flow control services that are covered by Logical Link Control and Adaptation Protocol (L2CAP) and the lower ends of the RFCOMM and Service Discovery Protocol (SDP). Presentation layer's functionalities are covered by the RFCOMM and SDP. Last, Application layer matches perfectly with Bluetooth's applications. [BRAY01]

4.2 Bluetooth protocol

The Figure 10 conveys the protocol stack of Bluetooth. It can be divided into two parts. First, lower layers that are composed of Radio part, Baseband and Link Manager – these are always implemented in the Bluetooth module (the hardware part of the Bluetooth solution). The rest of the stack forms the higher layers that are implemented to the host processor (memory resident software). In between the two parts is Host Control Interface that connects the lower layers to the higher layers. HCI's main tasks are to provide command interface and to access data on hardware. Bluetooth also uses non-Bluetooth protocols, such as IrDA's OBEX. These are shown in the Figure 10 between RFCOMM and Applications.

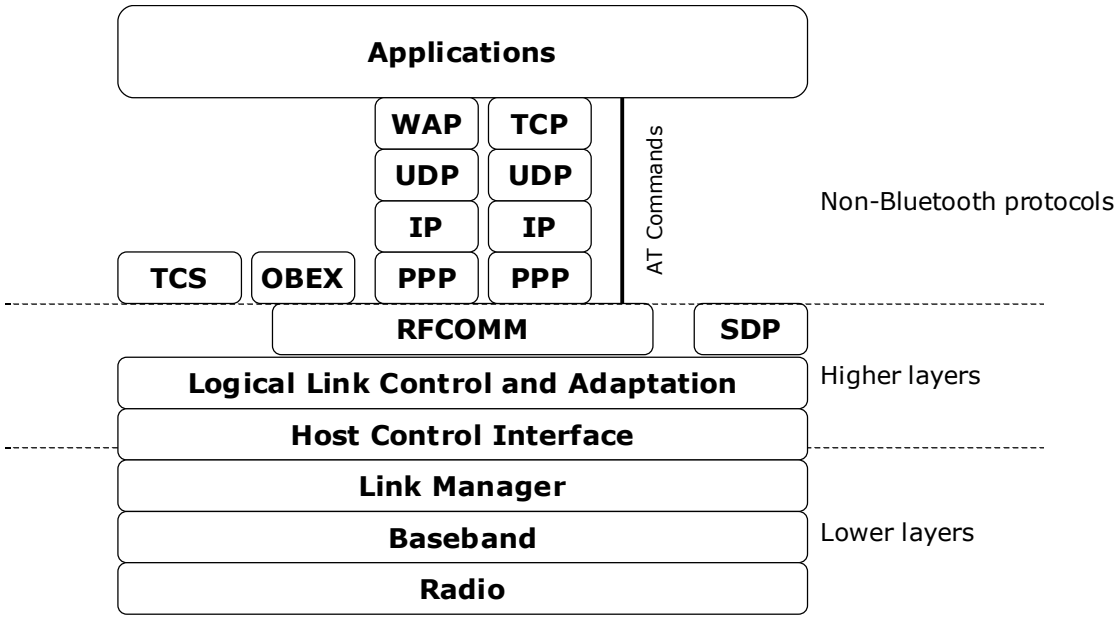

Figure 10 - Bluetooth protocol stack

4.2.1 Bluetooth radio

Bluetooth Radio (BR) sends and receives modulated bit streams. It also handles power emissions in the piconet. Bluetooth standard defines three power classes (Power Classes 1, 2 and 3), of which, Power Class 3 is the most common as it consumes the least power. The other Power Classes have higher power consumption, but also greater operational range (up to 100 meters with Power Class 1 devices). Only Power Class 1 devices have mandatory

power control. This means that the receiver can request the sender to increase or decrease transmission power. [BRAY01]

4.2.2 Baseband

Most of the issues (such as role-switching and piconet operations) described in Chapter 2.1 are Baseband's responsibilities.

Baseband (BB) retrieves data from channel and passes it upwards on the Bluetooth stack. To achieve this, it needs to code and decode the channel, as well as manage low-level timing. It also handles flow control and frames packages by adding addressing and link control fields to raw payload. Finally, Baseband provides error detection and correction. [CORE01]

4.2.2.1 Packets

Baseband sends data in packets. Each packet consists of an access code, a header and a payload as shown in Figure 11. **Access code** is used for detecting the packet and for addressing it to the correct destination. Access code is partly formed from BD_ADDR. When not in connection, access code is predefined, such as Inquiry Access Code (IAC). However, during active connection it identifies the packet to/from specific Master. It consists of preamble, synchronization word and trailer. Preamble is fixed (4 bits) – it is 0101 for synchronization words, which start with a zero, 1010 otherwise. Fixed preamble allows more time to set up reliable clock signal to sample and clock reminding data. Synchronization word is formed from the BD_ADDR. Trailer is similar to preamble, but it is optional. Certain special packets can consists of just access code (ID packet), or from access code and header (NULL and POLL packets).

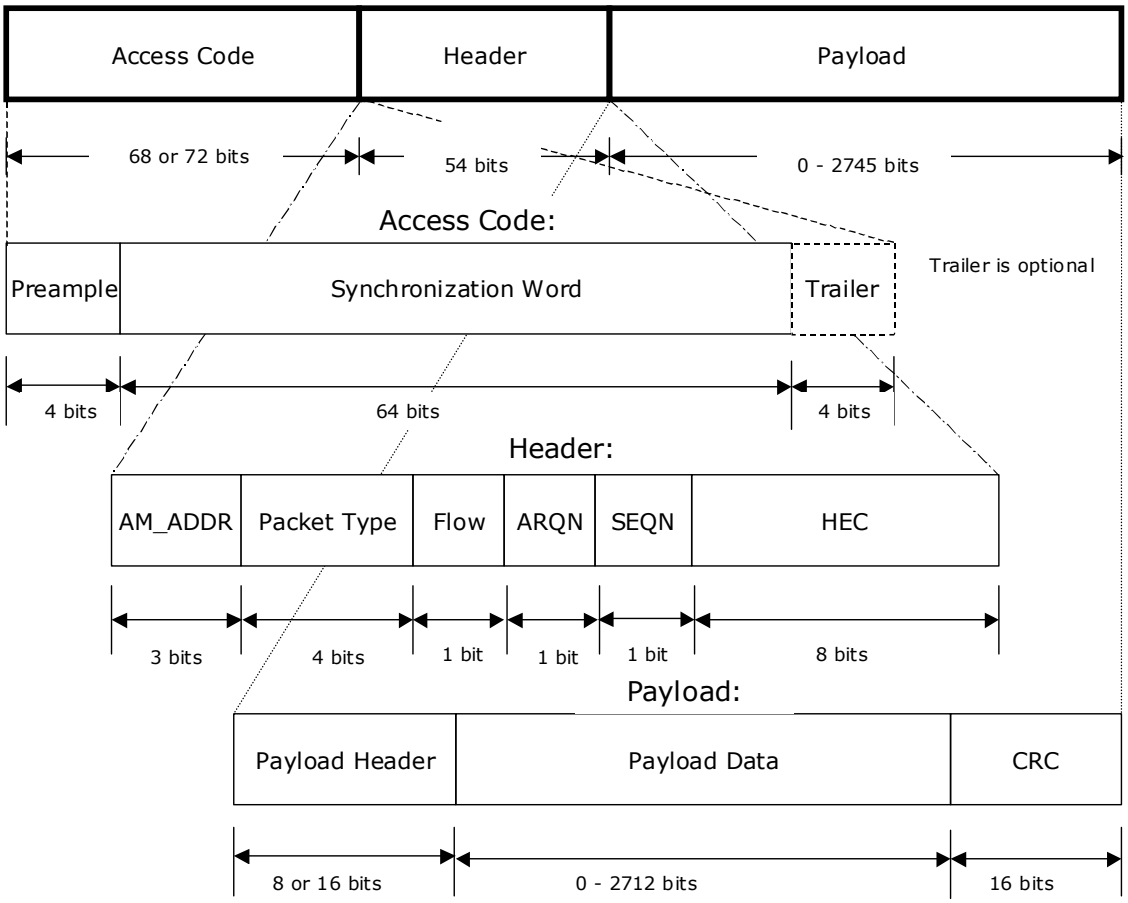

Figure 11 - Bluetooth packet structure

Header contains the control information of the packet and the associated link (for example, address of Slave that the packet is intended to). FEC 1/3 encoding is applied to a header, so data is replicated three times (18 bits => 54 bits). Header consists of AM_ADDR, Packet type, Flow flag, ARQN Flag, SEQN Flag and HEC. Packet type tells what kind of traffic is transmitted, there are sixteen different packet types defined. Most important are collected to Table 3. Packets differentiate by traffic type (ACL or SCO) as well as how many slots the transmission takes (1, 3 or 5 slots). Flow Flag is set to zero by a device whose receiving buffers are full, therefore being unable to receive any more data. Once buffers are empty, Flow flag is set to one again. ARQN flag notifies that previous sending was successful – Cyclic Redundancy Check (CRC) was passed. SEQN flag is always toggled when re-sending

occurs. Finally, HEC (Header Error Check) is a CRC check sum performed on the header. It allows recipient to ignore remainder of the packet if HEC operation fails. [CORE01]

Payload contains the data of the packet. Payload consists of variable sized Payload header, payload (0 to 2712 bits) and 16-bit CRC checksum. Header is one byte for single slot packets and two bytes for multi slot packets. For voice data, header is non-existent.

Table 3 - Some of the most important Baseband packets

Packet Name	No. of Slots	Description
ID	½	Contains only access code. Used before actual connection.
NULL	1	Contains only access code and packet header. Used for acknowledging transmissions or to pass flow control back to the other device.
POLL	1	As NULL, but needs to be acknowledged.
FHS	1	Sent by an inquiring device to an inquirer during inquiry, by Master to Slave during page, or by Master to Slave when changing roles. FHS packet provides information from sender to recipient allowing frequency hop channel synchronization and correct device access code.

SCO packets differ from above in that they do not need Flow, ARQN or SEQN flags in the packet header. No CRC is made for SCO data, since SCO packets are never transmitted, therefore CRC field is absent as well. Payload size is fixed to 240 bits, of which 80, 160 or the whole payload is source data, depending on what FEC was used (1/3, 2/3, none). [BRAY01]

4.2.2.2 Error correction

Baseband handles error detection and correction. There are three error correction schemes available for Baseband – 1/3 rate FEC, 2/3 rate FEC and Automatic Repeat Request (ARQ). FEC means that parity bits are added to the source data. FEC 1/3 means that bits are simply repeated three times. 1/3 FEC is always applied to a packet header. FEC 2/3 uses shortened Hamming code to form from 10 input bits 15 output bits. With a channel that has significant

amount of interference, 1/3 FEC is usually applied to SCO packets, whereas ACL packets are detected and corrected with 2/3 FEC. If a damaged packet is received with ARQ bit set on, the receiver will request the sender to re-transmit the packet, unless the damaged packet is a SCO packet, which are, of course, just discarded. [CORE01]

4.2.2.3 Link Control

Baseband is responsible for link control such as paging, page scanning, inquiring and inquiry scanning. It is also responsible for link level operations over several data packet durations in response to higher-level commands. In addition, it manages multiple links with different devices or even different piconets and it is responsible for maintaining a link after setup. [CORE01]

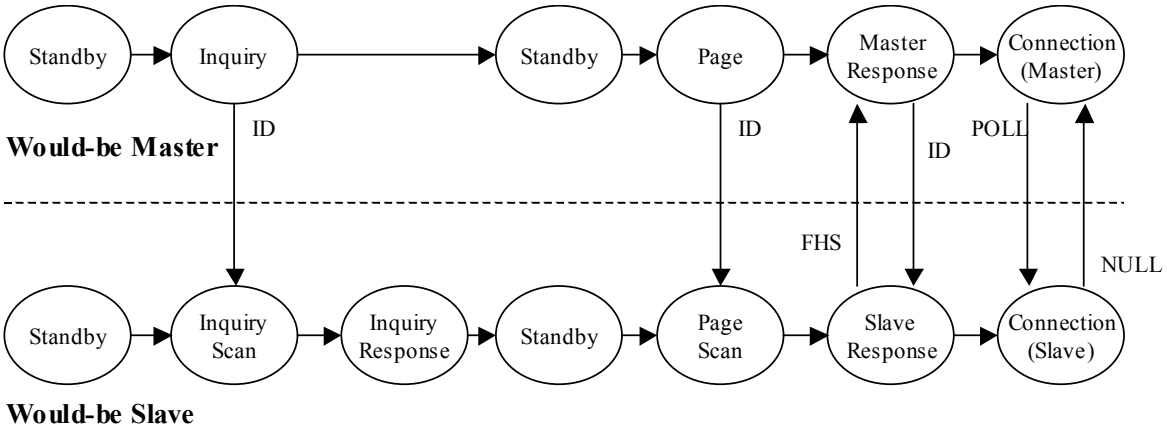

Figure 12 - Baseband state transitions

Link control functionality is easiest to describe using a state machine. A device is always in one of the states described in Table 4. All devices start in Standby state when powered up. Note that each device has as many state machines, as there are connections with it in separate piconets. Figure 12 illustrates how devices move from Standby to Connection.

Table 4 - Link Control states

State	Description
Standby	Device is inactive, no data is transferred and radio is turned off. Therefore, device consumes very little power.
Inquiry	Device tries to discover all other discoverable Bluetooth devices in the vicinity. It sends out ID packet twice per slot (ID packet can be transmitted in half slot) to decrease discovery time. Device compiles list of found devices that are available.
Inquiry Scan	Device makes itself available to the inquiring devices.
Inquiry Respond	Inquiry scanning device will send FHS packet to the inquirer after random delay when it has discovered an inquiring device. Delay is due to make sure that that inquiry scanning devices in the vicinity do not interfere each other.
Page	Would-be Master device transmits paging packets to the intended Slave device using ID packet information received earlier.
Master Respond	After acquiring Slave's respond to paging message, Master sends out FHS packet.
Page Scan	Device enters this state to allow devices to establish connection with it.
Slave Respond	After receiving paging message from the would-be Master, a device acknowledges paging with ID packet and starts to wait for the FHS packet. Once it receives FHS packet, it freezes hopping and then uses the received data to update information of Master's timing and frequency hopping scheme.
Connection	Slave switches to Master's timing sequence and frequency hop scheme. Master transmits POLL packet to verify the connection, to which the Slave answers with any packet.

It is not necessary to execute inquiry if the Bluetooth Device Address (BD_ADDR) of the other device is known. In this case, no accurate estimate of would-be Slave's timing scheme is available and therefore link setup takes more time. [BRAY01]

Once a link has been established between two devices, these may enter one of the following connection sub-states:

- **Active** - When Master has an active connection, it must send periodically something to Slave(s), so that these stay synchronized to Master's timing info. If a Slave receives a packet header and the packet's Active Member Address (AM_ADDR) is not its own, Slave aborts receiving the remaining packet. [CORE01]
- **Hold** - A device in Hold mode ceases to support ACL traffic for a defined period. It retains its AM_ADDR. This frees bandwidth for other functions, such as inquiry scanning. After the period has passed, Slave synchronizes to Master's frequency hopping and starts to listen to ACL traffic again. [CORE01]
- **Sniff** – Slave only periodically listens to traffic in Sniff mode. If it receives a packet that matches its AM_ADDR, it will listen for as long as packets directed to it are received. Once it receives a packet intended for other device, it will stop listening traffic for a while. [CORE01]
- **Park** - In Park mode Slave gives up its AM_ADDR. From time to time, it will synchronize to Master's hopping scheme, but most of the time it remains in low power mode. [CORE01]

4.2.3 Link Manager

Link Manager (LM) translates Host Controller Interface commands into operations at the Baseband level. The Link Manager carries out link setup, authentication and link configuration. To perform its service provider role, the Link Manager uses the services of the underlying Baseband. It discovers peer Link Managers and communicates with them via the Link Manager Protocol (LMP). LMP essentially consists of a number of PDUs (Protocol Data Units), which are sent as single-slot packets from one device to another. One of the main tasks of the LM is allocation of Active Member Addresses and keeping track of these. [PALO04]

4.2.3.1 LMP PDU

Each LMP message starts with transaction identification. This is a one-bit field that is not set for Master device but is set for a Slave device. This is followed by seven-bit Operation Code that identifies the type of LMP message. This allows for 128 different PDUs, but only 27 are

currently defined in Bluetooth standard version 1.1. Last, there is variable length of parameters. No flow control is used when passing LMP PDUs. [BRAY01]

4.2.4 Host Controller Interface

Host controller interface (HCI) is an interface between the lower and upper layers of the Bluetooth stack. Usually the lower layers are implemented on a Bluetooth module (hardware) and upper layers consist of software running on a processor, on host. This allows reply time of microsecond(s) for interrupts, since lower levels are on hardware and allows chip cost to remain low, since complex, higher level layers, which are not so time-critical are software based. Since HCI is running between hardware module and software stack, it allows mixing of different manufacturer's solutions. HCI can be used to test and verify the lower layers. HCI is not a mandatory part of Bluetooth stack. [PALO05]

HCI standard defines three types of packets.

Command packets are used to control and monitor the module from host. After issuing a command packet, the module always answers with an event. The host is able to command the module to mask certain events, if it is not interested of all of them. [BRAY01]

Data packets are used to pass ACL and SCO information across HCI. Each data packet can carry up to 65535 bytes of information. It is likely that inexpensive Bluetooth modules do not have enough buffers for this. Bluetooth standard requires that all Bluetooth devices must support buffers at least capable to contain 255 bytes of data. For SCO data, the maximum data length is always 255 bytes. This is to avoid lag between SCO links. [BRAY01]

Structure of **event packets** is similar to command packets. They are issued after each command packets as a reply to a command. Some of the event packets may be masked out, if the recipient is not interested of them. [BRAY01]

4.2.4.1 Flow Control for packets

HCI supports higher data rates than what can likely be supported on module. Module could buffer incoming data, but eventually it would run out of buffer space. A partial solution would be to slow down the HCI, but this would also mean that commands are issued slower. Therefore, the whole Bluetooth stack operability would be affected. Instead of slowing the whole HCI, it is only slowed down when buffers are overloaded. Initially the module will only accept one command packet. The event that is responding to this command carries with it the available buffer size. Now, host can send out this many commands. Once this amount is reached, another event is used to inform the available space on the buffers again and so on. In case of a module being faster than a host, the module could buffer events and data in the same way. [PALO05]

For data packets, the flow control is similar, though a separate command is issued to find out how many ACL and SCO packets can be buffered. Response contains values for each buffer. Usually SCO data is not buffered at all – it is either sent or discarded. [PALO05]

4.2.4.2 HCI Commands

HCI Commands allow control of Bluetooth module. All these tasks are initiated from HCI, though the actual process is performed by the Baseband. HCI Commands are below in bold.

A peer device's response to **inquiry** results to an event that includes the BD_ADDR of the responding device, page scan repetition mode, page scan period mode, page scan mode, class of responding device and clock offset. As there can be several inquiry scanning Bluetooth devices in the vicinity, inquiry responses can be filtered only to include new devices, certain class of devices or specific BD_ADDR. Filtering can also be done in applications, but to do it

in HCI saves bandwidth. Inquiry will be halted once wanted number of responses is received, or inquiry time has passed. In addition, a host can always cancel the inquiry. Inquiry should only be done in short bursts, to save battery and to avoid cluttering the radio bandwidth. Therefore, host only periodically puts module into inquiry mode. This is done at random intervals, so that all the nearby Bluetooth devices can be discovered. [BRAY01]

Other devices can discover devices in inquiry scan mode. **Inquiry scanning** device checks for IAC (either General IAC or Limited IAC) in received packet's payload. Inquiry scan drains a lot of batteries, therefore device goes into idle state from time to time and the actual inquiry scanning is done in short bursts. For example, the GAP defines that device should inquiry scan every 2.56 seconds for 10.625 milliseconds. However, a device is in inquiry scan mode far more often than other devices are in inquiry mode. This way, there is a very good chance that the device is discovered. Again GAP defines that inquiry scan should last three times more than inquiry mode. [BRAY01]

Once **page-scanning** device receives its own ID packet is received, the **page**d device raises a connection request event to the host. If the host accepts the request, Baseband and LMP finalize the connection setup. Once connection has been setup, another event indicating that connection is completed is raised. In this event a connection handle is returned. Page scan is used in short bursts to save battery and to allow other modes to be used as well. There are three different page scan repetition modes: continuous (0), every 1.28 seconds (1) and every 2.56 seconds (2). [BRAY01]

4.2.5 Logical Link Control and Adaptation Protocol

Logical Link Control and Adaptation Protocol (L2CAP) passes data packets (no audio) either to Host Controller Interface, or in a hostless system, directly to Link Manager. One of the most important tasks of the L2CAP is multiplexing between higher layer protocols, allowing them to share lower layer links. L2CAP also segments larger packets going down the protocol stack and reassembles lower layer packets coming up the stack. Additionally, it

manages groups of devices, providing broadcast capabilities. Finally, it is responsible for quality of service in ACL links for the higher layer protocols. L2CAP can also be used to request information from local or peer L2CAP entity. Higher layers are dependant on L2CAP, making it compulsory part of Bluetooth system. [PALO06]

4.2.5.1 Multiplexing

With multiplexing L2CAP allows several higher layer links to use a single ACL link for passing data. L2CAP uses channel number to distinct between packets, so that it can route them to correct places. Channel numbers are reserved when setting up a connection. L2CAP packets can have up to 65535 bytes of data. [BRAY01]

4.2.5.2 Signaling

L2CAP signaling is used to send control information between L2CAP peer entities. Whenever L2CAP is sending out a message, it starts a timer and if no response has arrived when timeout occurs, message is re-sent. Several L2CAP commands can be sent in a single packet. However, the packet size can be restricted by setting Maximum Transmission Unit (MTU). If receiver receives a packet that is longer than its MTU, the packet is discarded. The sender is sent a special reject packet with a reason why the transmission was rejected. All implementations of L2CAP have to support at least packet size of 48 bytes of length. [BRAY01]

4.2.5.3 Operations

Higher layer always starts the process of creating a L2CAP **connection**. If there is no existing ACL link, it must be created. Once this has been done, ACL link is used to carry L2CAP packets. First message sent is always connection request that includes information which protocol uses this L2CAP channel. Connection may not immediately be created due to pending operation (for example authorization). Connection can also be refused due to security or lack of resources. [CORE01]

After a link has been successfully set up, it must be **configured** so that both ends of the connection agree on used parameters (MTU, flush timeout and quality of service). This can take some time, since the process continues until both sides agree, or timeout occurs. After the channel has been successfully configured, data can be transmitted on it. [CORE01]

L2CAP channel is dismantled when requested by higher-level layer, or when timeout occurs. When a **disconnection** requests is received from a higher layer, L2CAP entity stops sending and receiving data on the requested channel number. [CORE01]

4.2.6 RFCOMM

RFCOMM can emulate serial cable line settings and status of an RS-232 serial port. It is based on GSM TS 07.10 with minor differences. It should mainly be used by devices that use Bluetooth to replace wire connections. RFCOMM can provide multiple concurrent connections by using L2CAP. In theory, up to 30 data channels can be supported. RFCOMM is dependant on Baseband to provide reliable in-sequence deliverance (i.e. flow control) of byte streams, since RFCOMM does not have any error correction abilities. It distinguishes two types of devices [BRAY01]:

- **Class I** – Internal emulated serial port (e.g. printer)
- **Class II** – Intermediate device with physical serial port (e.g. modem)

RFCOMM frames are sent as payload on L2CAP packets. Therefore, L2CAP channel to the peer entity has to be set up before RFCOMM connection. Once L2CAP channel is established, RFCOMM control frames are sent to RFCOMM multiplexer. After this has been created, RFCOMM connection can be created. [CORE01]

4.2.7 Service Discovery Protocol

Imagine a scenario, in which user walks up to a certain area, uses his Bluetooth device to discover a near-by printer and uses the printer to print a document from his device. Service

Discovery Protocol (SDP) is the part of Bluetooth that allows user to discover near-by services – such as printing services offered by a printer.

SDP is a client/server model and it relies on L2CAP links being established between SDP Client and SDP Server. SDP Server offers services, which are stored on its own database. SDP Clients use services provided by SDP Servers. Bluetooth devices can simultaneously be a SDP Client and SDP Server. Bluetooth standard does not define a Man Machine Interface (MMI) for SDP, instead it defines SDP commands and the way data is represented. [CORE01]

Each service is described in SDP database with service attributes that provide information about the service. Service information may include data, such as URLs for executables, icons or additional documentation. SDP Clients may have to follow the URL to fetch additional information about the service.

To find and use a service, Bluetooth devices need to create a L2CAP connection for SDP traffic. Once this has been done, device either searches for a specific class of services (identified by UUID – Universal Unique Identifications) or browses discovered services. From selected service, attributes needed to use it are retrieved. After retrieval, a separate (non-SDP) connection is established to use the service. Finally, it must be decided whether to drop the existing SDP connection, or to leave it open if additional SDP-services are required. [CORE01]

4.2.8 Non-Bluetooth Protocols

Next are described protocols that are not part of Bluetooth standard, but which are commonly used by Bluetooth devices. Most of these protocols are well known, so only short and precise description is given for each protocol.

4.2.8.1 Point-to-Point Protocol

Point-to-Point Protocol (PPP) is used for communication between two computers using a serial interface, typically a personal computer connected by a phone line to a server. PPP uses the Internet protocol (IP), but has support for others as well. Essentially, it encapsulates TCP/IP or WAP packets and forwards them to the RFCOMM. PPP can handle synchronous as well as asynchronous communication. [CISC99]

4.2.8.2 Internet Protocol

Internet Protocol (IP) is used to send data from one computer to another on the Internet. Each computer on the Internet has at least one IP address that uniquely identifies it. Each packet contains both the sender's and the receiver's IP addresses. Packet is first sent to a gateway computer that understands a small part of the Internet. The gateway computer reads the destination address and forwards the packet to an adjacent gateway that in turn reads the destination address and so forth across the Internet until one gateway recognizes the packet as belonging to a computer within its immediate neighborhood or domain. That gateway then forwards the packet directly to the computer whose address is specified. Because a message is divided into a number of packets, each packet can be sent by a different route across the Internet. Packets can arrive in a different order than the order they were sent in. It's up to the Transmission Control Protocol (TCP) to put them back in the right order. IP is a connectionless protocol; each packet that travels through the Internet is treated as an independent unit of data without any relation to any other unit of data. [CISC99]

4.2.8.3 Transmission Control Protocol

Transmission Control Protocol (TCP) is used with the Internet Protocol to send data in the form of message units between computers over the Internet. While IP takes care of handling the actual delivery of the packets, TCP takes care of keeping track of the individual packets for efficient routing through the Internet. TCP is a connection-oriented protocol. TCP is responsible for ensuring that a message is divided into the packets that IP manages and for reassembling the packets back into a complete message at the other end. [CISC99]

4.2.8.4 User Datagram Protocol

User Datagram Protocol (UDP) is an alternative to the TCP. Like TCP, UDP uses the Internet Protocol to actually get a data unit from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets and reassembling it at the other end. Specifically, UDP doesn't provide sequencing information of the packets. This means that the application program that uses UDP must be able to make sure that the entire message has arrived and is in the right order. Network applications that want to save processing time because they have very small data units to exchange and therefore very little message reassembling to do may prefer UDP to TCP. UDP provides two services not provided by the IP layer. It provides port numbers to help distinguish different user requests and a checksum capability to verify that the data arrived intact. [CISC99]

4.2.8.5 Object Exchange

Object Exchange (OBEX) is part of IrDA standard. In Bluetooth, it is re-used to save effort of defining similar new protocol. This is possible because the lower levels of Bluetooth stack share quite a many similarities with IrDA. OBEX utilizes client/server-architecture, where client pushes data to the server, or pulls data from server. OBEX needs RFCOMM, since OBEX packets are carried in RFCOMM frames. If using several OBEX servers at once, each requires its own RFCOMM server channel. [BRAY01]

4.2.8.6 Telephony Control protocol Specification

Telephony Control protocol Specification (TCS) specifies the way telephony calls can be made across Bluetooth link. TCS is loosely based on ITU-T Recommendation Q.931 that gives guidelines how to signal between point-to-point and point-to-multipoint calls, as well as how to send DTMF (Dual Tone Multiple Frequency) tones over Bluetooth. TCS signals are sent on an L2CAP channel. A separate bearer channel is established to carry the call. Once TCS call has been created, DTMF tones can be sent on the TCS signaling channel. However, TCS does not provide means how to make a handover of calls from one device to another. It also lacks definitions how to set up conference calls, since only point-to-point links are supported. Limited point-to-multipoint calls can be achieved using Wireless User

Group (WUG), where piconet Master can distribute information from one slave to another and provide parameters for the Slaves to create direct point-to-point connections. [BRAY01]

4.2.8.7 Wireless Application Protocol

Wireless Application Protocol (WAP) provides similar features as the IP, but it is intended for mobile devices. It takes into account devices limited capabilities, such as a small display and resolution. WAP can use Bluetooth stack as a bearer layer, in the same way as it is using GSM, CDMA or any other wireless service. WAP stack is attached to the Bluetooth stack using UDP, IP and PPP. [WAP01]

4.3 MOE Protocol

MOE Protocol (MOEP) is simple ASCII text based protocol (such as FTP) that is running on top of TCP/IP protocol. MOE Protocol is capable of delivering two distinctly different data packet types.

1. **Single word commands** that might have few parameters. These commands are received from user interface, so they are very sporadic. The size of these packets is small, so it is convenient to send it in one package. Overall, this type of data is best suited for packet switched network.
2. **Continuous video stream** from the MOE Server to a controlling device or the other way around. This creates quite a lot of load to the Bluetooth devices. This is best suited for circuit switched network.

As can be seen from above, the two data types are in conflict with each other concerning which type of network is best suited for them. Fortunately, the MCP handles two types of connections.

MOEP runs directly above TCP/IP. This is due to, MOE applications utilize sockets for communication – Bluetooth stack is hidden behind socket layer. Sockets on the other hand

are a common way to access TCP/IP protocols. MOEP can also use TCP's error correction and re-transmission features.

4.3.1 The MOEP model

Each side of the MOE System has a software module that implements the MOEP mechanism. A module that implements MOEP is responsible for initializing, configuring and managing the MOEP application connection. It is also responsible for sending and interpreting commands and replies. Finally, it needs to direct Video Transfer Process (VTP) to set up video connection.

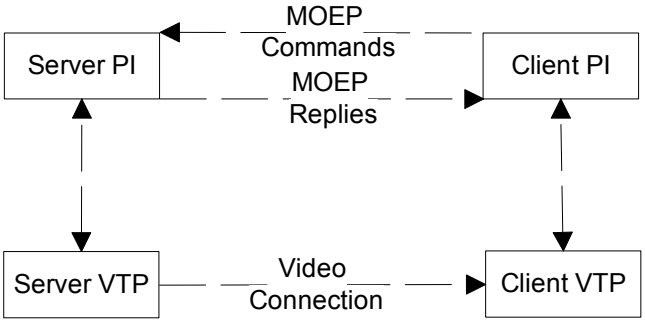

Figure 13 - MOEP model

As is shown above in Figure 13 the Client Protocol Interpreter (Client PI) initiates the control connection. The control connection follows normal Telnet protocol, except that End of File and End of Line characters aren't used. At the initiation of the client, standard MOEP commands are generated by the Client PI and transmitted to the server process via the control connection. Standard replies are sent from the Server PI to the Client PI over the control connection in response to the commands.

The client and server sides of the protocol have distinct roles implemented in a Client PI and a Server PI. Video transfer process establishes and manages audio-video connections. It is a dedicated socket pair.

4.3.2 Operations

The communication channel from the Client PI to the Server PI is established as a TCP connection from the user to the server port using default port number. The server protocol interpreter listens on default port. The client protocol interpreter initiates control connection. It is responsible for sending MOEP commands and interpreting the replies received. The Server PI interprets commands, sends replies and directs its VTP to set up the video connection and to start the video transfer. Additionally, it controls Client VTP to set up connection to Server VTP and it handles Quality of Service (QoS) for this connection.

The server at user's request closes the control connection. If there is an existing video transfer connection, it is first closed, followed by control connection. Before closing connection, any unsent replies are completed.

4.3.3 MOEP Commands

In Table 5 are all the currently supported commands in MOE Protocol. MOEP commands and replies do not have an upper limit to their size. Internet Protocol splits data into convenient sized packets.

Table 5 – Commands of the MOE Protocol

MOE Protocol command	Description
CONNECT	A request of connection between MOEP entities.
DISCONNECT	A request to dismantle connection between MOEP entities.
SERVICE_QUERY	A request to receive information about offered services.
SERVICE_REGISTER <service>	A request to register selected service. Name of the required service is indicated by <service>.
MOVE_FORWARD, MOVE_BACKWARD, MOVE_TURN_LEFT, MOVE_TURN_RIGHT	A request to move the MOE Robot.
MOVE_STOP, MOVE_SPEED <speed>	A request to set the speed of MOE Robot.
VIDEO_QOS <fps> <xres> <yres> | <bandwidth>	A request to change the quality of service for the video stream on MOE Server. The following parameters are used: <fps> frames per second, <xres> resolution of x dimension and <yres> resolution of y dimension, or replacing all three <bandwidth> that sets the maximum byte rate.
VIDEO_START <ip-address>	A request for MOE Server to start sending video stream to a given (<ip-address>) address.
VIDEO_STOP	A request for MOE Server to stop sending video stream.
AUDIO_STATE <state>	A request to change the audio setting. Possible values for <state> are either On or Off.
AUDIO_VOLUME <volume>	A request to change the volume on the MOE Server. Accepted values for <volume> are 1-9.
NOTIFICATION <string>	A catchall notification message. Parameter <string> can contain any string data.

4.3.4 Replies

Replies to MOEP commands ensure the synchronization of requests and guarantee that the client process always knows the state of the server. Only the return code is transferred, not the explanation of the command. Currently available codes are listed in Table 6 . Replies that start with number four have an optional <reason> string field, in which a reason of the reply may be passed. For example, a possible reply to the command "*SERVICE_REGISTER FOO*" may be "*414 Unknown service FOO*".

Table 6 - MOEP Reply codes

Reply code	Parameter	Description
210		Command OK
211		Command not implemented
212		Available services
213	<service>	Service registered
214	<service>	Service released
220		Control connection established
221		Control connection closed
222		Video connection established
224		Video connection closed
410	<reason>	Can't complete requested action
411	<reason>	Can't return service list
412	<reason>	Service not available
414	<reason>	Can't register service
416	<reason>	Can't release service
421	<reason>	Can't open control connection
422	<reason>	Can't open video connection
423	<reason>	Can't close control connection
424	<reason>	Can't close video connection
500		Syntax error, command unrecognized
501		Syntax error in parameters or arguments

4.4 Integrating MOE Protocol to Bluetooth Protocol Stack

MOEP protocol is set on top of TCP/IP, as can be seen from Figure 14. Actual connection is done through sockets. MOEP data is sent to socket, where it is encapsulated into IP packets and sent across TCP/IP network. PPP passes the data to RFCOMM like in normal serial connection. RFCOMM in turn, sends the data through Bluetooth stack. Peer entity reassembles data using Bluetooth stack, TCP. Finally, MOEP data is extracted from a socket.

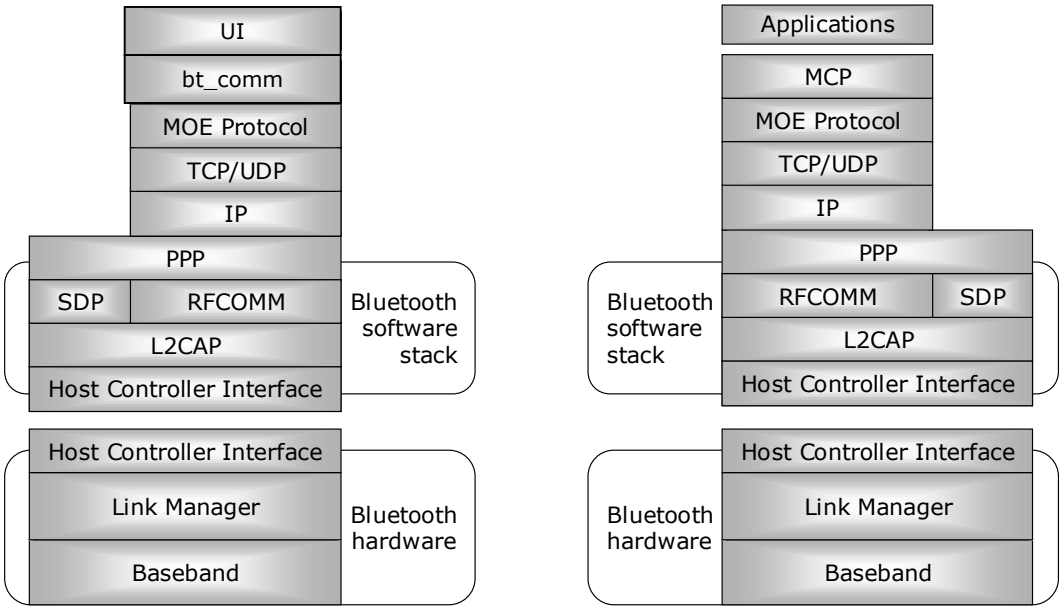

Figure 14 - Protocol stack in MOE applications

Technically speaking, the MOE System contains only two Bluetooth applications. First, MCP, on the server end, is the heart of the MOE System. Secondly, the controlling device has a layer that hides connection to Bluetooth, which can be considered as an application.

Additionally, MOE Server has several applications that are not visible beyond MCP. These are services, or MOE Applications running behind MCP on the MOE Server. They are using Bluetooth connection of MCP to market out services in a Bluetooth piconet.

5 SERVER DESIGN AND IMPLEMENTATION

In this chapter, server end of the MOE System is detailed. Apart from predefined hardware requirements, facts presented in this chapter were defined for the MOE project.

For implementation, it was decided that a demo that fully covers the workings of WESA would be sufficient. Therefore, an application needed to control the movements of a small robot was required. This application was named MOCOMA and it has the following functionalities: turn robot, move robot to either direction, set the speed of the robot.

5.1 Predefined requirements

The following requirements were defined in the Penttilä's Master's Thesis and were the basis for starting designing MOE System. [PENT01]

1. MOE Server would use Linux as Operating System. This would require much less computational power on the hardware.
2. Penttilä in his Master's Thesis defines quite specifically, what hardware is needed for optimal performance.
3. To avoid mechanical construction and building up motor servos, Lego RCX-Unit was to be used as a chassis for the MOE Server.

5.1.1 Linux Operating System

Linux is UNIX-like operating system that was designed to provide personal computer users a free or very low-cost operating system. Linux has a reputation as a very efficient and fast-performing system. Unlike Windows and other proprietary systems, Linux is publicly open and extendible by contributors. The power, reliability, flexibility, and scalability of Linux, combined with its support for a multitude of microprocessor architectures, hardware devices, graphics support, and communications protocols have established Linux as an increasingly popular software platform for a vast array of projects and products. [LNX01]

There are quite a number of different Linuxes available. In the early stages of the project, it was decided that the used Linux should have at least the following features:

- Free
- Minimal set of tools
- No graphical environments
- No need for real time features

Due to support of CompactFlash drives (MOE Server has an IBM microdrive with a CompactFlash connector), it was selected that MOE-project would use PeeWeeLinux (used version v0.61.0), which is based on RedHat Linux 6.2. [PW01]

5.1.2 Physical hardware

Server's hardware was required to consist of x86-processor and motherboard, a hard drive with enough space for fully functional Linux OS and additional space for future options, a soundcard, a Bluetooth card, a battery that would provide power for at least 15 minutes and Lego RCX-unit for controlling motor servos. [PENT01]

The selected hardware's technical specifications are as follows:

- Digital Logic PC/104 MSMP5SEV Motherboard with 266 MHz x86 Processor
- 64 MB of memory
- 340MB IBM microdrive
- Digital Logic PC/104 MSMM104 ISA-soundcard
- Battery with 2200mAh 7,2V
- Connectors for LPT1, COM1, COM2, floppy disk, speaker, mouse, keyboard, IDE, CF, USB, ISA and 100/10BASE-T LAN

5.1.3 Lego RCX-unit

To avoid making a chassis that requires mechanical skills and welding, it was decided that MOE-project would use LEGO technical bricks to construct the chassis. An additional benefit is that it is possible to use LEGO RCX-unit to convey commands from software to the chassis. This way, no hardware is needed to control motor servos. The downside is that the MOE Robot is a bit heavier, since RCX-Unit requires its own set of batteries. In addition, a control program is needed into RCX-Unit. This program sets on and off various servos. The program must be transmitted from PC using infrared connection to RCX-unit. The program receives control impulses from motherboard's parallel port interface and according to these impulses controls the movements of the chassis. It sets servos on and off and sets speed for motors.

5.2 Design

The main design issue with MOE Server was to provide an open and expandable platform. The result of the design should not constrict future add-ons and upcoming needs.

Two major design goals were set:

- **Versatility**: MCP's communication features are implemented using Internet sockets. Thus, Bluetooth links are easily exchangeable.
- **Expandability**: MCP's architecture is not limited to a certain number of clients and applications. Its functionality will remain the same regardless of how many connections are established.

MCP is able to handle several MOE Applications and MOE Clients at the same time. Same types of connections share a common Master Socket. This socket creates a connection between MOE Application and MOE Client by creating an additional socket dedicated to this connection. The upper limit of connections can be set if the user wants to speed up the system.

At first, there is only one MOE Application (MOCOMA – robot controlling application) and one MOE Client connection. Soon, additional applications (VAU – sending of video or audio data over Bluetooth) and several client connections will be added.

5.2.1 Bluetooth stack

Axis' Bluetooth software stack is free, open Bluetooth stack developed by the Axis Communications for Linux OS. Currently it supports SDP, RFCOMM, HCI and L2CAP. This is adequate support for MOE Project, since lower layers below HCI are used directly from Bluetooth hardware. It somewhat limits the selection of Bluetooth modules (hardware), since it does not work with all of them (at least Cambridge Silicon Radio Bluetooth modules are not currently supported). However, since the stack is open, anybody can make fixes for the software. [AXIS01]

Axis stack was chosen mainly because it is free. Another major reason was the openness. It fits thematically MOE Project quite well. Due to openness, it gives a good glimpse how Bluetooth host software stack functions and therefore suits the MOE Project's educational purpose. Last, with third party stack, it is possible to test how well the Bluetooth module and host from different solution providers interoperate.

5.2.2 MOE Central Process

MOE Central Process (MCP) is designed to control the whole MOE Server. Its main functionality is to act as a middleman between applications running on MOE Server and MOE Clients requesting to use their services. To reach this design goal, MCP needs to control the Bluetooth stack.

MCP contains four so-called Master Sockets. These normal TCP/IP sockets are listening a defined port. Whenever they receive a connection request, they create new socket for the connection and bind the requester to it. There is one Master socket for each type and direction of traffic. MCP's state machine is depicted in Appendix 3.

5.2.3 Expandability

Several things provide expandability. First, the hardware for the Server is from quite high-end for the simple services required. Currently, most of the hard drive space is not used. In addition, there are ample of connectors still available. To save memory and hard drive space, a more compact version of Linux could be installed, but for now it has been thought that it is good that the OS on board of MOE has almost all the normal Linux operations available. The PC/104-board has good expandability possibilities – if for example, adapters run out, a new board layer can be attached on top of the motherboard with additional adapters.

On software side, the expandability has been provided by using sockets. There is no upper limit how many socket connections can be from MOE Clients or from MOE Applications. However, only one per application is approved, unless both types of traffic (ACL and SCO) are needed. On startup, an upper limit can be set, to quicken the system. MCP swaps between these socket connections. To allow fast swapping from Master Sockets, they are dedicated to a specific style of traffic: ACL data from MOE Client to MOE Application, SCO data from MOE Client to MOE Application, ACL data from MOE Application to MOE Client and SCO data from MOE Application to MOE Client. Thus, each Master Socket functions only for one purpose.

Additionally, each MOE Application is required to indicate the location of its Service Record to the MCP. This XML-file details the services provided by the application and parameters required to use them.

5.3 Implementation

MOE Central Process was implemented with ANSI compliant C++. MCP is a stand-alone application that becomes a Linux daemon process after startup functionalities. Therefore, after startup it will run on background until it is terminated. Startup functionalities cover such things as creation of log-file and setting up four Master sockets.

Each MOE Application is also a stand-alone application. They become connected to MCP on start-up. MCP can disconnect them at any point.

5.3.1 MOE Central Process

Main tasks of MCP are controlling Bluetooth stack and of four Master Sockets – two for MOE Applications (such as MOCOMA) and two for MOE Client communications. These pairs consist of one socket for SCO link and one for ACL link. In addition, MCP maintains service registry. For tracing, one task of MCP is logging of events on MOE Server. It can be done on several levels, from all incoming and outgoing data to only fatal errors.

5.3.2 MOCOMA

Like MCP, MOCOMA becomes Linux daemon process after startup functionalities been covered.

MOCOMA has a very simple functionality – it translates MOE Protocol commands for the RCX-Unit. This is achieved using a hardware that is connected from motherboard's parallel port into RCX-Unit (hereafter referred as MOCOMA parallel interface, MPI). In addition to this, it can understand other MOE Protocol commands. A MOCOMA state-machine is depicted in Appendix 1.

When MOCOMA receives a command through a socket from MCO, it first tries to interpret it. If it is recognizable command and the number and type of parameters are correct, it checks

what type of command it is. If it is a command necessary to control the application, MOCOMA handles it immediately and sends reply. If, however, it is a command to affect the movement of the chassis, MOCOMA sets a bit array and passes the information to MPI that in turn, activates one or more of the servomotors on the chassis. Last, MOCOMA sends a reply to MCP.

6 CLIENT DESIGN AND IMPLEMENTATION

In this chapter, client end of the MOE System is detailed. For implementation, it was decided that a demo that fully covers the workings of WESA would be sufficient. Therefore, a user interface that allows controlling of the movements of a small robot was required. The User interface would also have means to search and connect to a Bluetooth service.

6.1 Design

The client end of the MOE System is used to send commands over the Bluetooth to the various applications running behind MCP. Due to the expandable nature of MCP, the user interface is likely to change whenever adding an implementation of a new application to the server end. Therefore, features that are not likely to change, such as methods for controlling the Bluetooth stack, were collected to a library named bt_comm. This has the added benefit because the same library hides the Bluetooth interface from UI designers and implementers. For them, it looks like they are using a normal serial connection. One very important design issue is that the implementation running on client end is easy to port to different systems, if ever needed. Due to this all non-UI methods were also included into bt_comm library.

6.1.1 Functionality

The functionality of the MOE Client depends on what services it subscribes from MCP using Bluetooth. Initially, the server will only run MOCOMA application. This application commands the movements of the robot. Due to this, the interface in the client end is quite simple. Required functionalities are the following:

1. Setting of various options to set up client software.
2. Setting of various options to create connection over Bluetooth.
3. Sending commands to MCP and to MOCOMA using MOE Protocol.
4. Receiving commands from MCP and from MOCOMA using MOE Protocol.

6.1.2 Platforms

The initial client design and implementation is for Windows CE 3.0. However, several design techniques were used in order to achieve easy portability when, if ever, need arises.

The following are some of the design techniques used to achieve maximum portability and low latency times on the controlling device. [GIGU00]

1. Unnecessary communication with the server is avoided.
2. Client end of the application is very simple.
3. Most functions have only single action path.
4. Use Model-View-Control (MVC) technique. In MVC, model contains data; view is representation of that data and controller can be used to change either model or view.

The initial intention was that only the user interface of the MOE Client is Operation System (OS) specific code. However, due to some limitations of Windows CE, part of the code is going to be Windows CE specific. However, major parts of the MOE Client have been coded with ANSI standard compliant C++ code. The OS specific parts and parts that control the Bluetooth card on MOE Client are separated to own classes, to minimize changes, if switching OS comes to be a necessity.

6.1.3 User Interface

When designing UI for mobile or handheld device, a number of additional criteria must be understood and taken into account. The most important of these is that the screen size is small and resolution is quite limited. Therefore, a design should avoid user interface that either requires a lot components or that requires fine manipulation. In addition, handheld devices usually have quite limited processing power. That is why a small and efficient design is required so that application is easy to use. Last, to avoid spending significant amount of battery power for running applications (battery power should be conserved for important issues, such as using a telephone in the handheld device), computation-heavy implementation

should be avoided, or if it is possible, to be moved to server side. In short, user interface for handheld device should be simple and small. [GIGU00]

There are quite many options to control various functionalities in the MOE Client. Therefore, more than one dialog was required – especially, when after setting these options or features, user wouldn't need to refer to those values until something changes in the MOE System set-up. In UI for MOCOMA, settings were split across three dialogs and an additional dialog is used as a central settings dialog for these. Additionally, dialogs for searching Bluetooth devices and their services as well as connecting and disconnecting to these were needed. These dialogs are re-usable for most upcoming user interfaces as well. Last, a dialog for client status was designed. This dialog is used to track the connection status to MOE Server.

6.1.4 Other components

Common functions required by the UI and wrapper layer for the Bluetooth software were collected to a common component library. The initial idea was only to isolate Bluetooth stack from user interface, but it was later deemed that several functionalities from the user interface were required to be moved here. MOE Client's class hierarchy is depicted in the Appendix 2.

No software was developed for the Bluetooth stack, instead driver and libraries of the selected Bluetooth Card vendor were used as is.

6.2 Implementation

MOE Client was partly implemented with Windows CE specific code. Re-usable parts were done according ANSI C++ standard.

6.2.1 Implementation techniques

Due to limited nature of the controlling device, it was decided that several good design techniques would be needed in order to reach a passable solution. Most important of these are [GIGU00]:

1. There should be as few as possible public members for each class.
2. Run time memory usage should be minimized.
3. Use scalar variables if possible (q.v. setsize(int, int) vs. setsize(Dimensions)).
4. Help garbage collector by setting to be deleted class to null.
5. Release resources as early as possible.
6. Avoid exceptions.
7. Do not refer to arrays, instead copy values to local variables.
8. If possible, use low-level functions instead of high-level class methods.
9. Do not show public data. Use accessor functions instead.

6.2.2 Hardware

The client end of the system is running in Compaq's iPAQ Pocket PC H3600 with an extension pack for PCMCIA card. Used Bluetooth card is AnyCom's Bluetooth Card CC3010-PPS that has a PCMCIA adapter.

6.2.3 User Interface

The UI is Windows CE specific code, and as such not directly re-usable for other operating systems. However, for future MOE Clients that are implemented with Windows CE, the code is highly re-usable. The screen in Pocket PC is quite small. Therefore, only a few items can be shown on one dialog. When necessary, there are sub-dialogs to further detail certain parts of the interface.

The most important part of the UI is the initial main window (class CMOERemote), from where the robot movement commands can be issued. Simply put, user can set the speed of

the robot and the direction of travel from this menu. Commands cannot be sent until a connection with MOE Server has been established. Handheld device's arrow keys are used to issue movement commands. Main window is depicted in Figure 15.

Figure 15 – Main window

In connect dialog (class CMOEConnectDlg), the devices and services that SDP has found are fetched. These services are filtered to show only services provided by certain device name, MOE Server. This is to avoid confusing the user, since user can only use these functions through the handheld device. User can choose to request a connection from this dialog, or choose to ignore found device and return to main window. In the cases that SDP has returned a device with unknown services, or has not found any services, user can also request to find another device. A screenshot of the connection dialog is shown in figure 16.

Figure 16 – Connection dialog

In options dialog (class CMOEOptionsDlg), user can set the various options concerning hardware. For future additions, means to navigate to sound or video options are provided. These are however, not implemented yet. It is also possible to set Bluetooth card options from here. This calls for Bluetooth card vendor specific dialog that is not part of the MOE Project. Screenshot from options dialog is shown in figure 17.

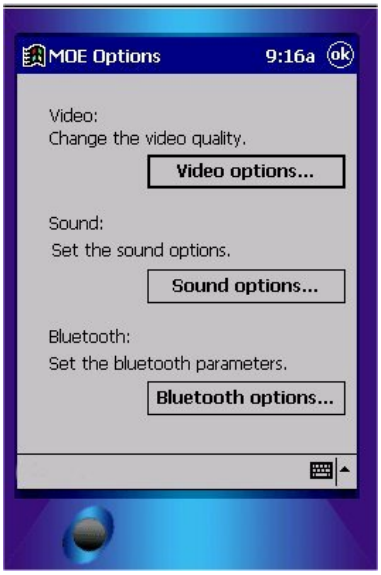

Figure 17 –Options dialog

In status dialog (class CMOEStatusDlg), user can see currently available services, as well as some statistics about the current connection, such as average reply time in milliseconds. Frame rate and resolution are future options for audio and video data. Screenshot of the Status dialog is shown in figure 18.

Figure 18 –Status dialog

6.2.4 Other Components

All the common functionalities are included in a couple of classes that are coded according to ANSI C++ standard. In these classes, also a wrapper layer for communication with Bluetooth software is provided. The most important of these classes is named bt_comm and it takes care of the following functions.

1. Makes an inquiry for nearby Bluetooth devices and their services. Returns immediately when one device has been found.
2. Returns additional details (name and status) about a wanted Bluetooth device.
3. Tries to register a found service.
4. Frees registered service(s).
5. Returns status of all available services.
6. Sends a command to an available registered service.
7. Requests the server to shutdown.

8. Receives and transmits data to buffers.
9. Receives a video or audio stream from server.

Another common component class CMOELog handles all the file operations concerning logging. The level of detail can be varied from all received and sent data to no logging at all or some level in-between.

Socket operations are hidden behind a Socket_connection class. This way the bt_comm just needs to send data onwards without worrying about technical details. If sockets are sometimes replaced with another connection technique, the Socket_Connection class only needs to be replaced with a class that implements a similar interface– other classes need not to be modified.

Finally, there is Vector class that is used to provide buffered services.

MOE Client employs full MOE Protocol (see chapter 4.4). MOE Protocol operates on top of normal TCP/IP sockets, which in turn are stationed on top of Bluetooth's RFCOMM.

This is implemented by the Bluetooth card supplier's Bluetooth stack. Interface into Bluetooth stack, into Bluetooth card and into TCP/IP sockets are hidden by using wrapper layer bt_comm.

7 CONCLUSIONS AND FUTURE CONSIDERATIONS

Last chapter collects lessons learned from the project and details what could be done in the future for the MOE System.

7.1 Conclusions

In ten months since the beginning of this thesis, the availability of Bluetooth devices has not improved the way forecasted in early spring. Bluetooth devices are still hard to come by on open markets, and their prices are still way above those expected. It seems that the Bluetooth revolution, if it ever comes, will have to wait until 2002. This has been somewhat caused by rigorous approval of Bluetooth devices by the Bluetooth SIG. Although this slows the flow of approved items into the market, it helps with interoperability of the devices.

The MOE-Project was partly affected by the availability of Bluetooth devices. Since MOE Project had committed to get Bluetooth devices from open markets, MOE Project couldn't use development kits and such that were readily available. The devices were initially expected in early June, but the actual acquisition was finally done in very late autumn. The delivery of the devices took even longer. The availability of Bluetooth devices caused months delay to the project.

The interoperability of Bluetooth devices is not yet as good as it would need to be. For example, using a third party Bluetooth stack on a given Bluetooth hardware causes some problems. In addition, devices implemented according to Bluetooth standard version 1.0 might not work with version 1.1 devices. This is due to that earlier version had many ambiguous areas that device providers solved differently. The upcoming standard version needs to take care that it doesn't cause problems with already existing hundreds of thousands of version 1.1 devices. After all, most of the version 1.0 devices were development kits that

can be forgotten, but there are now quite many commercial devices already in the markets. It would be foolish to ignore and forget these devices.

Overall, the actual design and implementation of the system was done quite smoothly. Only small changes were required into the initial design. For example, it was initially devised that a single socket in MOE-Server would be enough to handle all the traffic. However, the routing of incoming data proved to be problematic since each data needed to be examined for type, sender and receiver. With four sockets, the type of data is not needed, as the socket already defines the data type and direction of flow.

The resulting platform is reasonable efficient when using ACL-links. The amount of data sent over the system is, currently however, very small, since MOCOMA uses only short text commands. SCO-links have not been tested properly, due to not having a client with video/audio capabilities ready.

The platform is as open and modular as required. Adding a new application is quite easy. The application only needs to implement a connection to MCP using sockets and sending of data in MOEP format. The client end of the system is as easy. Client needs to implement a normal way of creating Bluetooth connection and sending of data in MOEP format over this connection. The portability of the system looks also good. System side of the system is implemented according to ANSI C++ standard. Some parts of the Client were required to be implemented to be Windows CE specific code, but porting to another system shouldn't be too difficult nor time consuming. The library functions and connection to Bluetooth should be the easiest to port.

One feature of the Bluetooth standard that really stood out during the implementation is that it is quite complex. The Core and Profiles parts of the standard version 1.1 have page count of more than two thousand pages. That is a lot for a standard that is designed to replace

communication cables between devices. However, the standard is presented clearly, layer-by-layer. In addition, there is no need to examine Profiles part until designing system's supported profiles. Each Bluetooth device should support at least one of the profiles presented in that part of the standard.

The inclusion of Lego chassis and RCX-unit was an error. The Lego chassis is quite heavy, though easily modifiable, but the Lego servos and motors are inefficient. With a fully featured miniature PC and battery, the total weight of the robot is too much for them. The situation is made worse by the RCX-unit's own battery. It would be easy to design and implement additional hardware, so that RCX-unit would get its power from the main battery. However, removing Lego-chassis and replacing it with lightweight aluminum chassis, makes the robot lighter by half and more durable. Added durability is important, since the robot's motherboard and accessories are expensive and therefore needed to be protected as well as possible.

One of the goals was to learn about Bluetooth. The MOE Project has introduced Bluetooth to at least ten other persons that are working at the same department as the author. In addition, guidelines created during the MOE Project will help upcoming Bluetooth and wireless projects. Parts of this thesis will be used to create a Bluetooth guide that encapsulates two thousand pages long communication standard into a couple of dozen pages.

7.2 Future considerations

To simplify the current architecture, Lego RCX-unit should be removed. With RCX-Unit removed, converter hardware, RCX-unit's software and RCX-unit's batteries could also be removed. Of course, this would require changes to MOCOMA, so that it would convert incoming commands into serial port pin activations. Then these would be deducted using new hardware that would send activation signal to one of the servomotors. Overall, this is the likely way for MOE to develop.

Once RCX-Unit is removed and special hardware has been constructed, there is little need to use Lego technical bricks for anything – they are quite heavy, big and restrict the actual physical character of the device. Additionally, chassis made of them is not very protective. Instead, a chassis constructed from aluminum should be used – it is light and durable, and it is quite easy to modify. On top of this, it is also cheap. Almost tailor made for MOE Server are various chassis available for remote control devices.

Foremost, the designed video/audio (VAU) application that sends video and/or audio data from MOE Server to the controlling device should be implemented. The MOE architecture gives full support for it, so it won't need a lot of effort. Sending stream data has been initially tested on MOE and the initial test results were very encouraging. Playing audio/video data in controlling device might cause some concern, though. To implement VAU, it would require that MOE Server be equipped with small video camera. There are quite a number of net cameras available, which would be convenient for this functionality. This feature would allow MOE Server to be used as mobile net camera, or as surveillance equipment.

Once sending of stream data from MOE Server to MOE Client can be done, it would be an interesting idea to send stream data from MOE Client to MOE Server. This way, MOE Server could play audio tracks, and therefore be able to "speak". This would require additional hardware, a microphone and speakers on the server, but they are easy to install and very cheap. Current MOE Server has even a sound card ready with ISA adapter.

Another interesting idea would be to employ fuzzy logic and Lego's touch sensors in such a way, that MOE Server could circumvent an obstacle. Touch sensors could also be used together with sound effects to give a hint to the user that the MOE Server has collided with something. However, this is unlikely, since it seems that by removing RCX-Unit, most of the hardware and weight problemes are solved.

Another rather interesting idea would be, to use MOE Server's video camera for pattern recognition. Human faces are rather hard to recognize easily, but MOE Server could be developed to recognize a safe-place ("MOE-garage"), where it could go on command. If RCX-Unit is kept, it can used to follow routes that are color linked. For example, a robot can follow black line on white paper. This feature could be used to guide stray robot into safe area.

Finally, voice recognition could be employed to give audible commands to MOE Server. This could be done with on-board microphone (so no handheld device would be needed), or through a handheld device using TCS.

Appendix 1: MOCOMA State machine

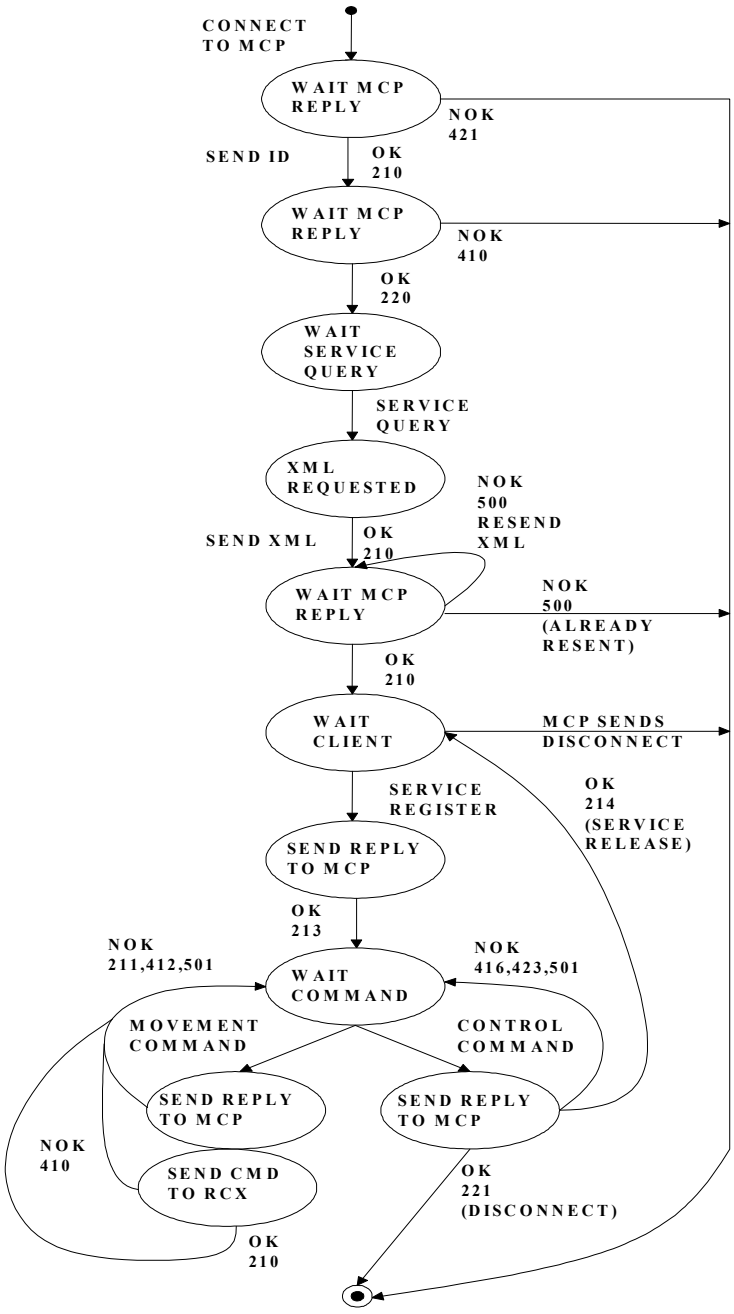

Figure 19 - MOCOMA state machine

Appendix 2: MOE Client Classes

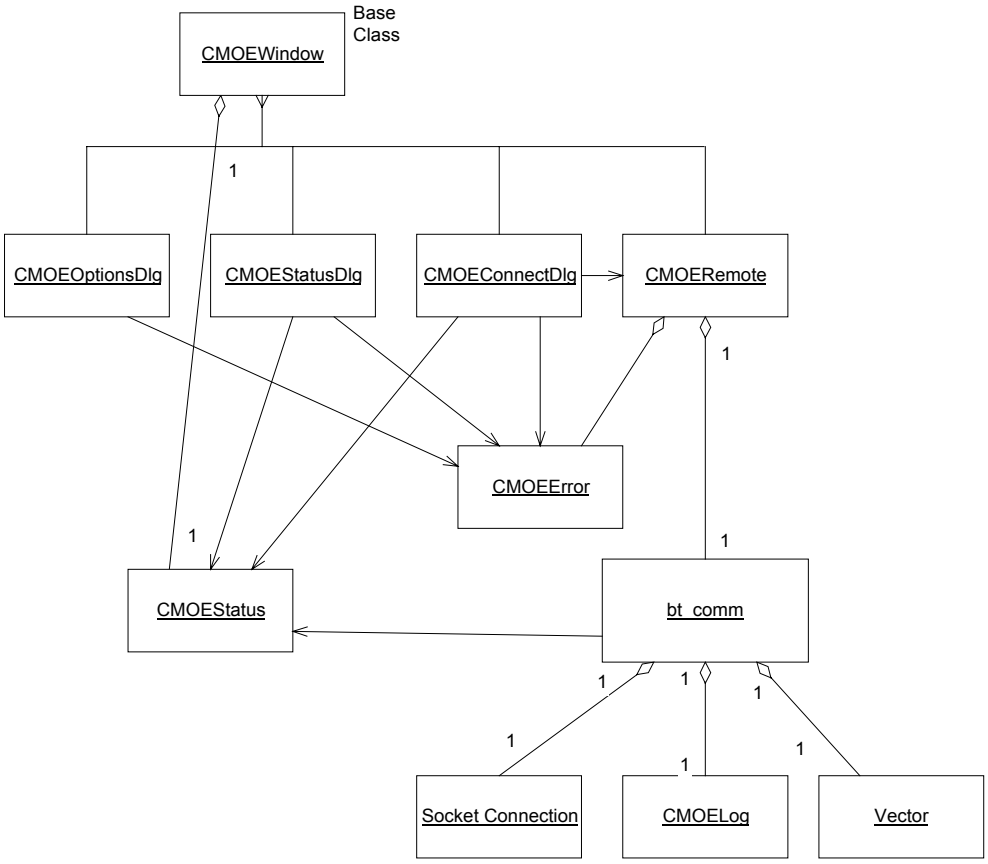

Figure 20 - MOE Client Classes

Appendix 3: MCP State Machine

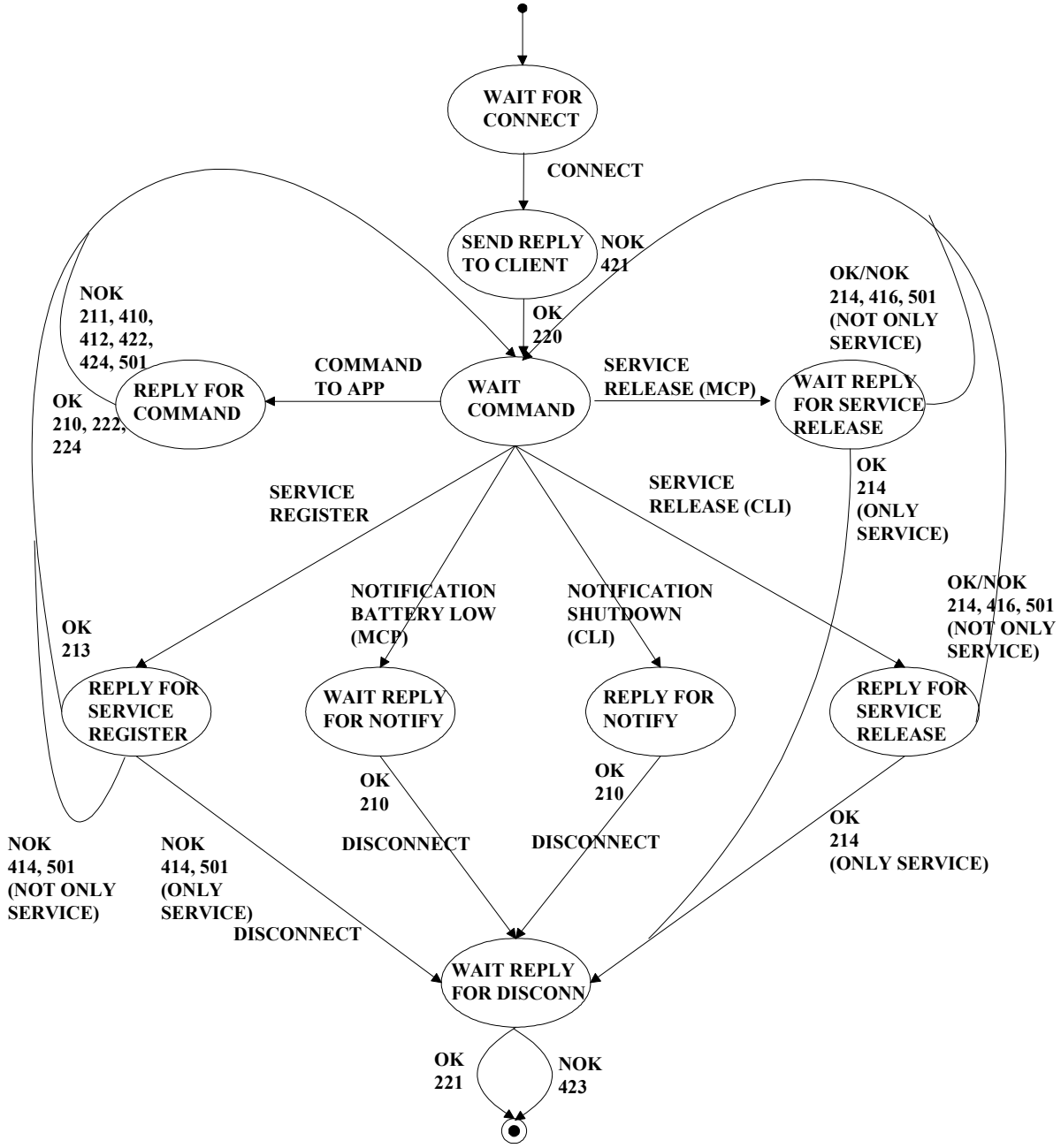

Figure 21 - MCP state machine

REFERENCES

[AXIS01] Axis OpenBT Stack homepage [web publication], Axis Communications AB 2001, [cited 13/09/2001]. Available: http://developer.axis.com/software/bluetooth

[BRAY01] Bray, Jennifer; Sturman, Charles F. "Bluetooth – Connect without Cables", 1st Edition, Prentice Hall 2001, 495 pages. ISBN 0-13-089840-6.

[CHIN01] Chinitz, Leigh. "HomeRF Technical Overview" [web document], HomeRF Working Group Inc., February 2001, 19 pages [cited 11/06/2001]. Available: http://www.homerf.org/data/events/past/pubseminar_0501/tech_overview.pdf

[CISC99] Cisco documentation "Internetworking technology overview"[web document], Cisco Systems Inc. 1999, [cited 02/11/2001]. Available: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm

[CORE00] Bluetooth standard – core part version 1.1 [web document], Bluetooth SIG , December 2000, 1084 pages, [cited 25/10/2001]. Available: http://www.bluetooth.com

[ERIC01] "Bluetooth fundamentals" [web document], Ericsson AB 2001, [cited 09/10/2001]. Available: http://www.ericsson.com/bluetooth/bluetoothf

[GIGU00] Giguere, Eric "Java 2 ME - Programming Strategies for Small Devices", 1st Edition, Jon Wiley & Sons 2000, 368 pages. ISBN 0471390658.

[HoRF01] HomeRF Working Group "Home networking technologies" [web documentation], HomeRF Working Group Inc., April 2001, 5 pages, [cited 12/06/2001]. Available: http://www.homerf.org/data/tech/consumerwhitepaper.pdf

[IEEE01] "IEEE802.11 Wireless Local Area Network" [web publication], IEEE802.11 Working Group 2001, [cited 02/09/2001]. Available: http://grouper.ieee.org/groups/802/11

[INTEL01] Fleming, Kris. "Architectural Overview of Intel's Bluetooth* Software Stack" Intel Technology Journal 2nd quarter 2000 [web publication], Intel

Corporation 2000, 13 pages. [cited 13/09/2001]. Available: http://developer.intel.com/technology/itj/q22000/articles/art_2.htm

[LNX01] Linux homepage [web document], Linux Online 2001, [cited 09/10/2001]. Available: http://www.linux.org/

[MIT99] Gershenfeld, Neil. "When Things Start to Think", MIT Media Lab 1999, 1st edition, 225 pages. ISBN 0-8050-5880-X.

[NOKI01] Nokia and Bluetooth [web document], Nokia Mobile Phone 2001, [cited 12/09/2001]. Available: http://www.nokia.com/bluetooth

[PALO01] Bluetooth resource center [web document], Palowireless Pty Ltd. 2001, [cited 11/11/2001]. Available: http://www.palowireless.com/bluetooth

[PALO02] "Bluetooth Tutorial – Specifications" [web document], Palowireless Pty Ltd. 2001, [cited 25/10/2001]. Available: http://www.palowireless.com/infotooth/tutorial.asp

[PALO03] "Bluetooth Baseband tutorial" [web document], Palowireless Pty Ltd. 2001, [cited 25/10/2001]. Available: http://www.palowireless.com/infotooth/tutorial/baseband.asp

[PALO04] "Bluetooth Link Manager Protocol tutorial" [web document], Palowireless Pty Ltd. 2001, [cited 31/10/2001]. Available: http://www.palowireless.com/infotooth/tutorial/lmp.asp

[PALO05] "Bluetooth Host Controller Interface tutorial" [web document], Palowireless Pty Ltd. 2001, [cited 02/11/2001]. Available: http://www.palowireless.com/infotooth/tutorial/hci.asp

[PALO06] "Bluetooth Logical Link Control and Adaptation tutorial" [web document], Palowireless Pty Ltd. 2001, [cited 02/11/2001]. Available: http://www.palowireless.com/infotooth/tutorial/lscap.asp

[PAN01] "Personal Area Networking Overview" [web document], Motorola Inc. 2001, [cited 25/10/2001]. Available: http://www.motorola.com/bluetooth/pan

[PENT01] Penttilä, Markus. "Technology Requirements for Server Robot Bluetooth Control System", Master's Thesis, Tampere University of Technology 2001, 60 pages.

[PROF01] Bluetooth standard – profiles part version 1.1 [web document], Bluetooth SIG December 2000, [cited 25/10/2001]. 452 pages. Available: http://www.bluetooth.com

[PROX01] Mim, Reiner. "Collision Course – How Bluetooth interference impacts Wireless LANs" [web document], Proxim Inc. 2001, [cited 02/11/2001]. 7 pages. Available: http://www.proxim.com/learn/library/whitepapers/wp2001-03-colmet.html

[PW01] PeeWeeLinux homepage [web publication], PeeWeeLinux Working Group 2001, [cited 09/10/2001]. Available: http://peeweelinux.org/

[RFI01] "Whitepaper: Bluetooth Technology" [web document], RFI Modile Technologies AG 2001, [cited 25/10/2001]. Available: http://www.rfi.de/downloads/Bluetooth_UK.pdf

[SUVA00] Suvak, Dave. "IrDA and Bluetooth: A Complementary Comparison" [web document], Extended Systems Corp. 2000, [cited 12/04/2001]. Available: http://www.extendedsystems.com/ESI/Products/Wireless+Connectivity+Products/Bluetooth+Windows+Applications/Product+Detail/BT_vs_IR.htm

[SYMB01] Minhas, Mal "EPOC Bluetooth Architecture and Personal Area Networking" Symbian EPOC day seminar material, Symbian UK, 11.4.2001, 22 pages. Available: http://www.symbian.com/technology/standard-blue.html

[TANE96] Tanenbaum, Andrew "Computer Networks", Prentice Hall, 1996, 3rd Edition, 848 pages. ISBN 0-13-349945-6.

[TRÄS00] Träskbäck, Marjaana "Security of Bluetooth – An overview of Bluetooth Security", Helsinki University of Technology, Department of Electrical and Communications Engineering, seminar material for the course Tik-86.174 "Bluetooth technology & utilization", 2.11.2000, 5 pages. Available: http://www.cs.hut.fi/Opinnot/Tik-86.174/Bluetooth_Security.pdf

[WAP01] WAP forum home page [web publication], WAP Forum 2001, [cited 01/10/2001]. Available: http://www.wapforum.org