Internet Telephony in Ubiquitous Computing Environments

by

Wen Hu
Yang Wu

Thesis in partial fulfilment of the degree of
Master in Technology in
Information and Communication Technology

Agder University College
Faculty of Engineering and Science

Grimstad
Norway

May 2007

ABSTRACT

Nowadays VoIP plays an important role in communication, and more and more people use VoIP instead of traditional telephones because of low cost, accessibility and improved quality. However, one of disadvantages of VoIP is that users have to receive the Internet calls by headset and microphone, which are attached to a PC. This way restricts the users' mobility and doesn't meet the need of modern users. An innovative improvement would be to use some existing wireless technology to impart mobility to VoIP calls. We propose to use Bluetooth due to its widespread availability in already ubiquitous mobile phone market.

Bluetooth is a short-range communication protocol intended to replace the cables connecting portable and /or fixed electronic devices. Bluetooth is one of the most widespread wireless technologies available in most mobile phones available in the market today because of its low power consumption, low cost and robustness. This project aims at using Bluetooth enabled mobile phones as wireless headsets for PCs that run Internet telephony software in order to induce mobility. The project focuses on investigation and analysis of Bluetooth and related technology for building such a solution, such as service discovery, redirection, security and fast handover. From the research done, this thesis proposes mechanisms and design for a prototype that allows the connection between PCs and Bluetooth enabled mobile phones. The project also reviews different available alternatives and compares existing solutions.

We believe that since Bluetooth and VoIP become more and more mature, our project will be applied in people's daily life soon. And this ideas proposed in this project would be a convenient and useful extension to already existing Bluetooth server available on most mobile phones, providing that extra bit of mobility around a workstation or PC.

© May 2007 – Wen Hu & Yang Wu

PREFACE

This thesis is the partial fulfilment of the two-year Master of Science program in Information and Communication Technology (ICT) at Agder University College (AUC), Faculty of Engineering and Science in Grimstad, Norway. The thesis has been carried from January to June 2007 and the workload equals to 30 ECTS.

First and foremost, I would like to thank Professor Dr. Frank Reichert and PH.D Ram Kumar, our supervisors at Agder University College. They have been highly available and highly supportive throughout the whole project period.

We would also like to thank the following persons for their efforts in providing us with information and views that has helped us in our research: PH.DAndreas Häber and English assistants. Finally, I would like to thank Mr. Stein Bergsmark and Mrs. Sissel Andreassen for their coordination of our studies and daily life in Grimstad.

Grimstad, May 2007

Wen Hu Yang Wu

© May 2007 – Wen Hu & Yang Wu

TABLE OF CONTENT

© May 2007 – Wen Hu & Yang Wu

© May 2007 – Wen Hu & Yang Wu

TABLE LIST

FIGURE LIST

© May 2007 – Wen Hu & Yang Wu

© May 2007 – Wen Hu & Yang Wu

1. INTRODUCTION

1.1 Introduction

Low-cost telephone services have traditionally been difficult to find. Because compressed voice/data can take up very little bandwidth, researchers have investigated the Internet as an inexpensive means to route telephone calls. Voice-over-IP (VoIP) is one result of this effort [MM03]. Nowadays VoIP plays an important role in communication, and more and more people use VoIP instead of traditional telephones because of low cost and high communication quality.

However, one of disadvantage of VoIP is that users have to receive the Internet calls by headset and microphone, which are attached to a PC. Modern people who are used to using mobile phones are not accustomed to stay beside the PC during the whole connection. They probably wish to move around and have the ability to do something else while answering a call. A certain wireless communication technology should be used here.

Bluetooth is a short-range communication protocol intended to replace the cables connecting portable and/or fixed electronic devices [How05]. Compared with the traditional wireless communication technology infrared, Bluetooth doesn't require transceivers to be in direct line of sight. Also, its low power consumption, low cost and robustness have made Bluetooth become the trend in short-range wireless communication. Bluetooth is one of the most widespread wireless technologies available in most mobile phones available in the market today. One usage is to use a Bluetooth headset to answer an incoming call to the cell-phone.

This project aims at using Bluetooth enabled mobile phones as wireless headsets for PCs that run Internet telephony software. Nowadays VoIP, for example, Skype, is used by many people because of the low cost and good communication quality. However, the afore mentioned lack of mobility occurs due to having to use the PC and headset instead of mobile devices. Therefore, we would like to implement architecture that would enable users to receive internet calls via PC software on their mobile phones within a certain domain, approximately within 100 meters of user's PC.

There are already some headsets implementing this function inside a Bluetooth headset. Why do we still want to pursue this solution? The reason why we chose to implement this architecture in a Bluetooth enabled mobile phone comes down to the cost assessment. There is no doubt that a Bluetooth headset is much cheaper than a Bluetooth enabled mobile phone. However, Bluetooth headsets with this function

© May 2007 – Wen Hu & Yang Wu

normally can only support specific VoIP software. Therefore, the applicability is its drawback; it is no longer cheaper than a mobile phone if you have to buy many headsets for different VoIP software.
In our project, our first step will be to make mobile phones act as wireless headsets for PCs. Users would be able to receive Internet voice calls via their PC software on their mobile phone as long as they stay within 20-100 meters from their PCs.

After that, we will try to make it possible for users to not only stay within their computer's range, but also to roam around and still be able to receive calls via other PC's in the area. To achieve the communication, we should specify a functional architecture and implement prototypes. If there is enough time, we plan to examine the possibility of moving from one domain to another while holding on to the call.

The motivation behind choosing this topic is that we believe this architecture has a brilliant future. On one hand, on the technical side, Bluetooth is becoming more and more popular in the wireless field. We can see that in the increased number of devices that have Bluetooth profiles implemented nowadays. Bluetooth was initially developed to eliminate the need for inconvenient cable attachments. Imagine how inconvenient it is that nowadays we are forced to stay by computes all the time when receiving Internet calls. Therefore, increasing mobility is our aim for the project. It will allow users who use Bluetooth to move while communication.

From the economist view this work will also save quite a lot of money. The calls coming from VoIP will no longer be forwarded by establishing another VoIP call to the user's cell phone. The user can just simply answer it via Bluetooth, charging them nothing. There is potentially a huge amount of money to be saved if we can make this technology work in a big firm, making this project a very good example of how new technology benefits people.

1.2 Problem statement and Technical challenges

The aim of our project is to increase mobility in ubiquitous computing environments. The project will focus on five challenges in order to achieve the goal: emulating Bluetooth headsets, security, service discovery, redirection and mobility. The project can be sub-divided into the following sub-problems, which we will focus on individually.

1.2.1 Emulating Bluetooth headsets

This should be the foundation of the whole project. Issues like security and service discovery will not be involved here as more than a glimpse. The situation we should keep in focus is for example when a user in his office uses his Bluetooth mobile phone to answer VoIP calls via his PC. There is neither the possibility of other Bluetooth devices within this domain nor that the PC doesn't provide Internet

© May 2007 – Wen Hu & Yang Wu

Telephony service.

To achieve the first challenge, we should first establish a Bluetooth link between a PC and a Bluetooth enabled mobile phone. Then we install VoIP client or software on the PC to accept the Internet calls, while implementing a headset profile in the mobile phone. In this way, we could use the Bluetooth mobile phone to act as a headset for the call incoming on the PC. Since some solutions which implement this scenario have emerged on the market, we will also analyze each solution for the appropriate choice.

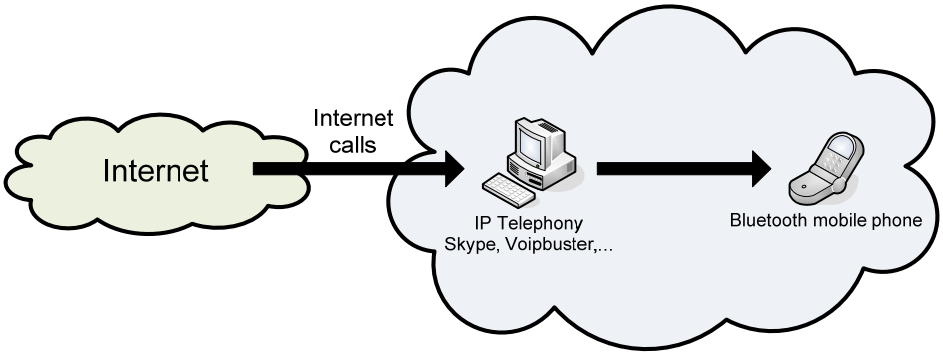

Figure 1 Emulate Bluetooth headset

1.2.2 Security

Remote control of devices and access to services can expose the infrastructure to significant risk. For example, we do not want to allow random strangers to connect PCs and answer Internet calls. So security is of great importance in our project. The security challenge here can be divided into three sub-problems: authentication, authorization and privacy.

- ✧ Authentication:
  This sub-problem deals with who is the one trying to connect to “my” PC. Since Bluetooth is different from infrared in that it is not blocked by walls, it is very probable that another Bluetooth device is just next to your office and is trying to connect to your PC. Therefore, the important issue here is to identify this Bluetooth user. For example, in the following figure, Bluetooth mobile phone 1 wants to connect to “my” PC. However, my PC denies access due to failure of identification.
- ✧ Authorization:
  For those devices we known, the application still needs to check whether they are permitted before accepting their request. We see that in the following figure both Bluetooth mobile phone 2 and 3 are authenticated. However, only mobile phone 3 is accepted by the PC since it is allowed whereas mobile phone 2 is not.
- ✧ Privacy:
  There can be lots of issues here in regards to privacy, which covers both static information and dynamic information.
  - ■ Static information: Information about the user, such as personal details, PIN, etc

© May 2007 – Wen Hu & Yang Wu

- Dynamic information: Information like conversation of the call, user's current location

All of these types of information should be guaranteed to not leak at anytime.

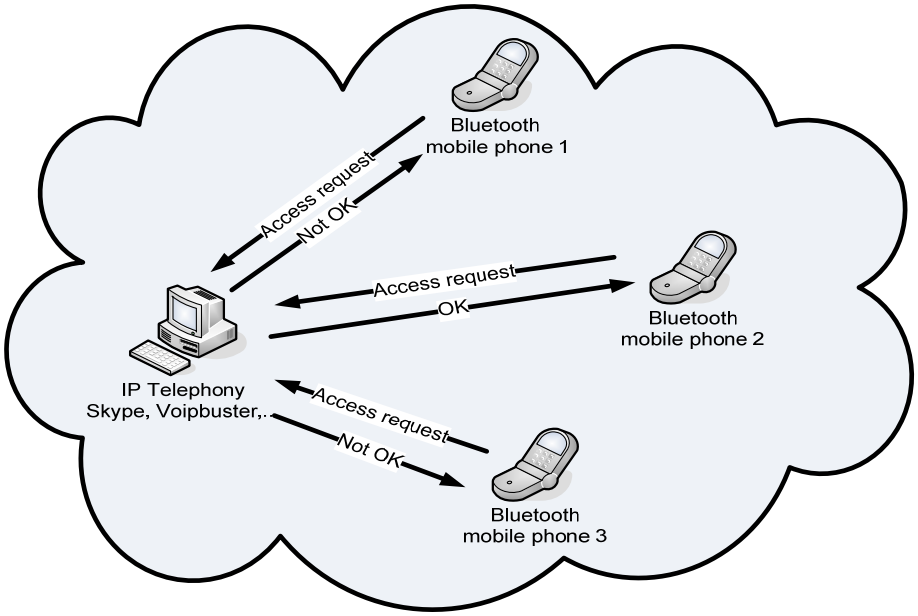

Figure 2 Security

1.2.3 Service Discovery

Service Discovery in our project's scope will focus on how to choose the correct PC with the "Telephony" service provided if there are many PCs in the area. The situation should be as shown in the figure below. Both PC1 and PC2 are available to Bluetooth mobile phone, however, only PC1 provides "Telephony" service in its SDDB (Service Discovery Database). This service is necessary for forwarding calls to Bluetooth mobile phones. Therefore PC1 is chosen and PC2 is rejected since it doesn't provide required service.

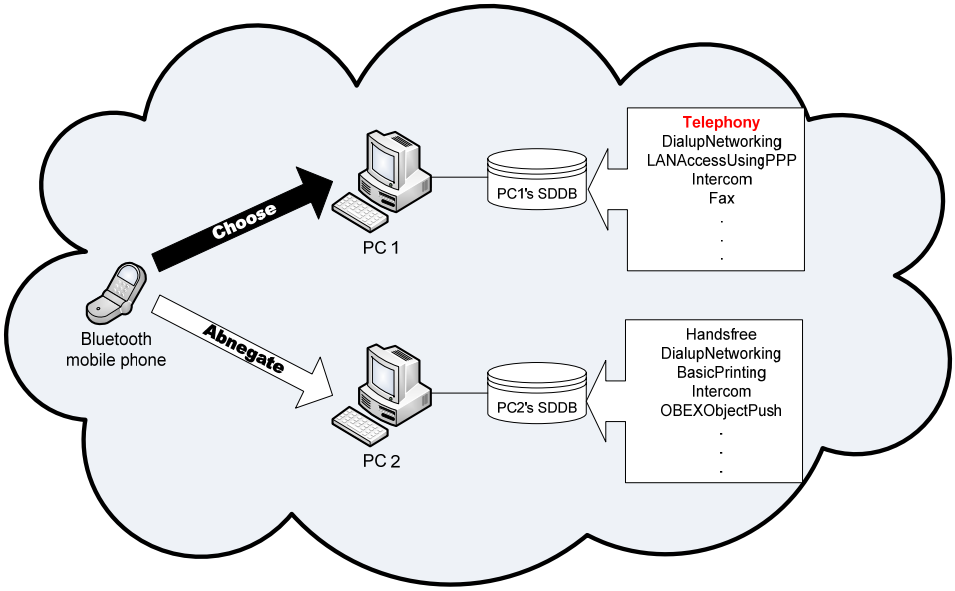

Figure 3 Service discovery

© May 2007 – Wen Hu & Yang Wu

1.2.4 Redirection

When the user moves from his own office to his colleagues' office, which is out of range of his personal PC's Bluetooth (20-100m), how can he still answer the VoIP calls using his Bluetooth mobile phone? This situation is our fourth challenge.

Since the user moves far away from his personal PC, called A, to which they were connected before, communication between A and the Bluetooth headset will be terminated since Bluetooth is a short-range communication system. We will define a solution that allows A to redirect the Internet calls to a specific PC, allowing the user to receive calls through the new PC.

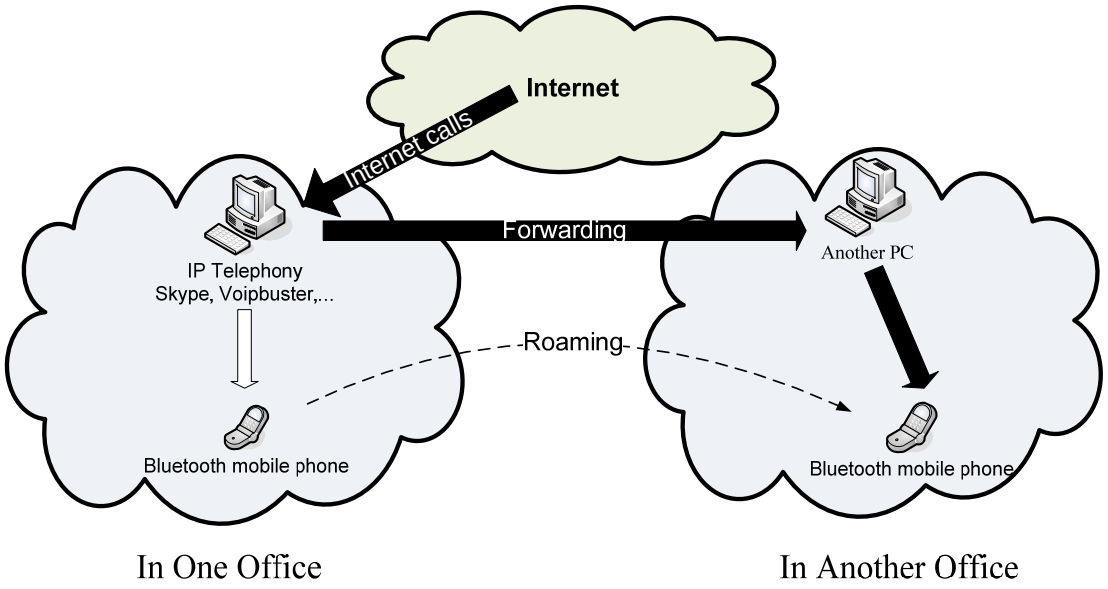

Figure 4 Redirection

1.2.5 Fast handover

As mentioned earlier, we will look at the possibility of fast handover during the call. In other word, fast handover can guarantee that the call will not be interrupted even though the Bluetooth mobile phone moves from one PC's Bluetooth range to another's. Considering current Bluetooth technology, it is a bit complicated to achieve the final challenge. Therefore, we would like to make a survey and brief comparison of the different existing solutions if time allowed. This part will not be our primary aim.

© May 2007 – Wen Hu & Yang Wu

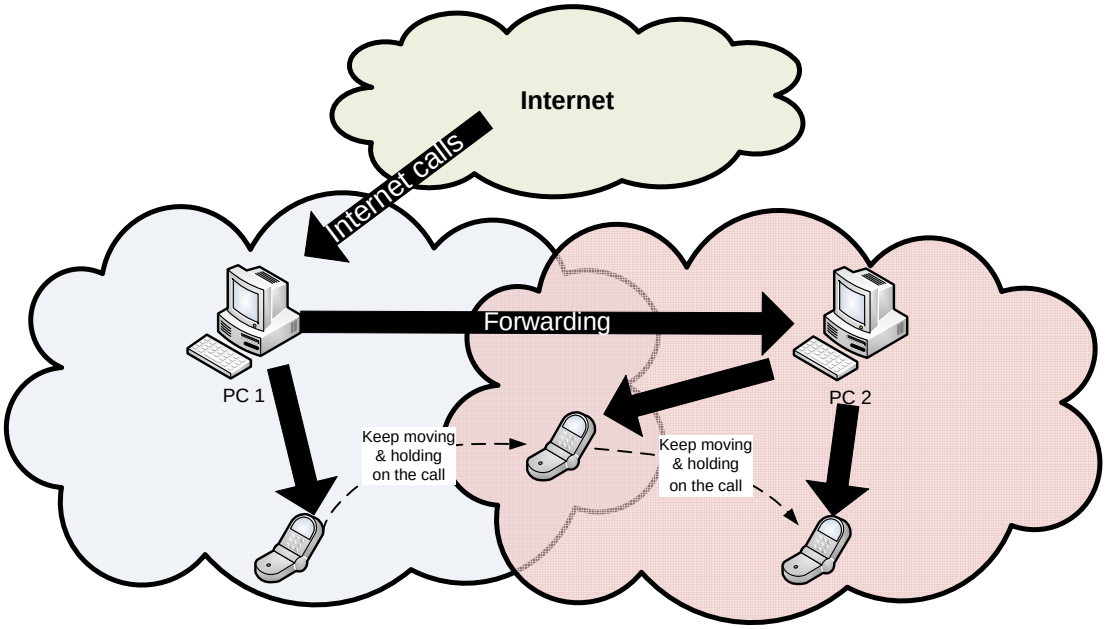

Figure 5 Fast handover

1.3 Report Outline

This report is structured as followed:

Chapter 1 provides an introduction to the master thesis which is current chapter.

Chapter 2 gives the basic theory about Bluetooth and related technology for the project, such as Emulating Bluetooth headsets, Security, Service Discovery, Redirection and Fast handover. This will establish a foundation for understanding the later proposed solutions.

Chapter 3 proposes the design of the project, including software design, flow charts and message sequence charts.

Chapter 4 gives a prototype for the project and design details.

Chapter 5 discusses all the results we have achieved in this project.

Chapter 6 gives the conclusion of our project work and point out possible further work based on this thesis.

© May 2007 – Wen Hu & Yang Wu

2. STATE OF THE ART

2.1 Emulation of Bluetooth headsets

2.1.1 Bluetooth

Bluetooth [QL03], [Did07], a short-distance wireless communication standard, originally aims at replacing cables when connecting devices like mobile phones, headsets and computers, and therefore making the world truly wireless.

Bluetooth operates on 79 channels in the 2.4 GHz band with 1 MHz carrier spacing and each channel divided into 625 ms length time slots. A piconet is a collection of Bluetooth-enabled devices that are synchronized to the same hopping sequence. Each piconet has exactly one master and up to seven simultaneous slaves; all other devices connect to the master. The master who initiates communication decides everything and the slaves have to follow. A basic Bluetooth connection works as follows: the master device initially sends out an inquiry packet to find a slave device, and when one of the slaves responds to the page messages, the master can begin transmitting voice or data. Two different types of links are used in Bluetooth [Sch03]: a synchronous connection-oriented link (SCO) and an asynchronous connectionless link (ACL). A SCO is for voice transmission, so it requires a symmetrical, circuit-switched, point-to-point connection, while an ACL is used to transmit data. One Bluetooth link can simultaneously support ACL links and up to three SCO links [Haa00].

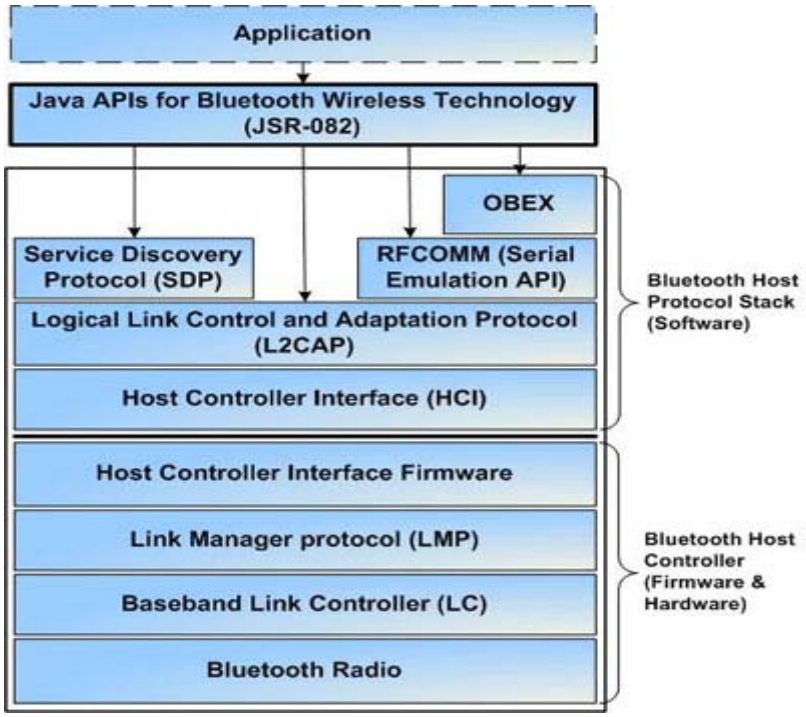

Figure 6 Bluetooth Protocol Stack [Sch03]

© May 2007 – Wen Hu & Yang Wu

To allow Bluetooth devices from different manufacturers and vendors to operate with each other, the protocol layers and profile are standardized and defined in the Bluetooth specification. Figure 6 shows the Bluetooth protocol stack. The Radio layer is the physical wireless connection. The Baseband layer maintains SCO and ACL. The LMP (Link Manager Protocol) uses the links set up by the baseband to establish connections and manage piconets.

The HCI (Host Controller Interface) is the dividing line between software and hardware. The L2CAP (Logical Link Control and Adaptation Protocol) is responsible for providing logical links to the upper layer protocols. Quality of Service (QoS) parameters are exchanged at this layer. The RFCOMM and SDP layer rely on the L2CAP layer and are unaware of physical communication details [QL03]. The SDP (Service Discovery Protocol) is used for service discovery on remote Bluetooth devices.

2.1.2 Bluetooth headset profile

Bluetooth profiles [Kli04] provide a well-defined set of higher layer procedures and uniform ways of using the lower layers of Bluetooth. The Headset profile [Hea01], one of the Bluetooth profiles, depends on the Serial Port Profile (SPP), and defines procedures to support interoperability between a headset and a mobile device. The mobile device is the audio gateway (AG) that provides both input and output audio to the headset. The headset is the device acting as the Audio Gateway's remote audio input and output mechanism. The project will implement a Bluetooth headset profile in mobile phones, thus providing the connection between mobile phones and PCs.

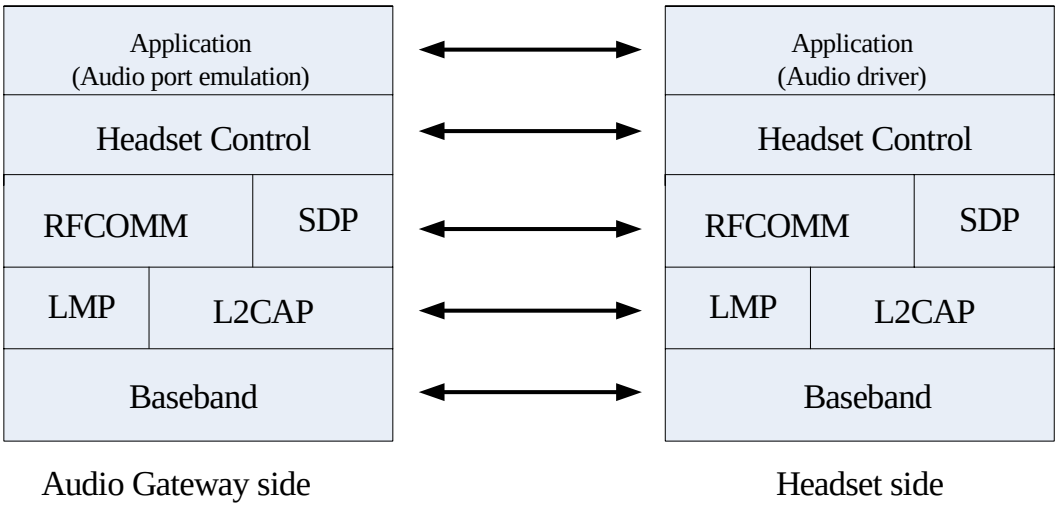

Figure 7 Headset Profile protocol model [Blu07]

The figure 7 [Blu07] shows the protocols and entities used in this profile. Headset Control is the entity responsible for headset specific control signaling. The audio port emulation layer is the entity emulating the audio port on the cellular phone or PC, and

© May 2007 – Wen Hu & Yang Wu

the audio driver is the driver software in the headset [Blu07].

Figure 8 and 9 [Hea01] shows ACL connection establishment is initiated by Audio gateway and headset, respectively. For Fig. 8, once the connection is established, the AG will send a RING to alert the user. The RING will repeat until the HS sends the AT + CKPD command to the AG which indicate that the HS press a button on the headset. Then the SCO link is setup.

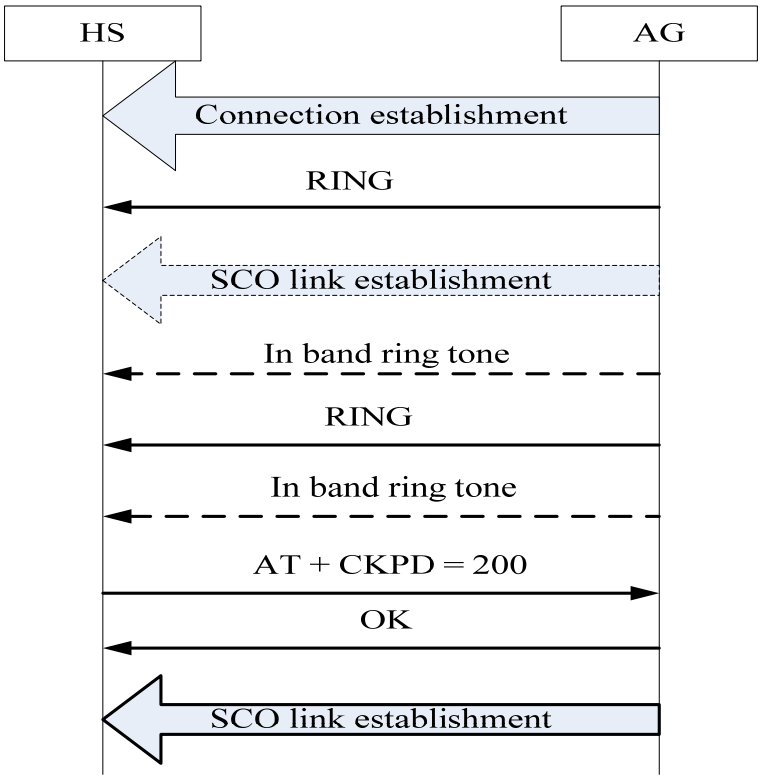

Figure 8 Incoming audio connection establishment

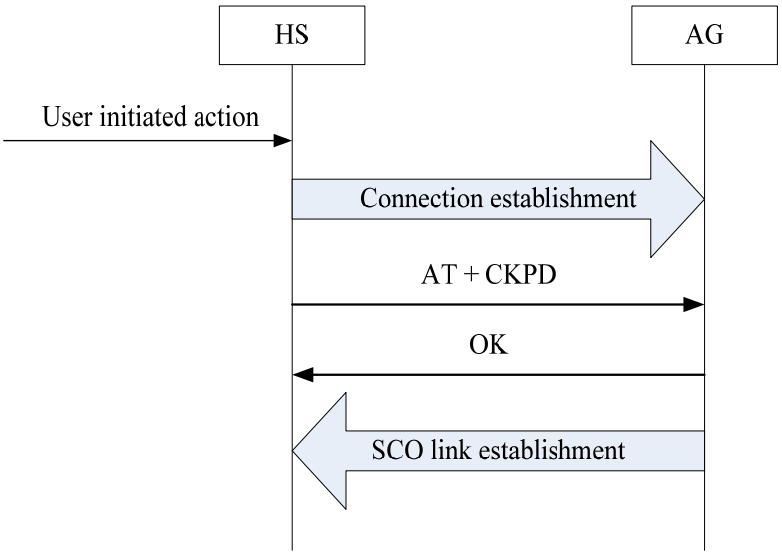

Figure 9 Outgoing audio connection establishment

© May 2007 – Wen Hu & Yang Wu

Figure 9 presents a headset initiated ACL connection by pressing a button on the headset. Then the HS sends the AT + CKPD command to the AG. It is worth noting that the SCO link establishment is always sent by the AG side.

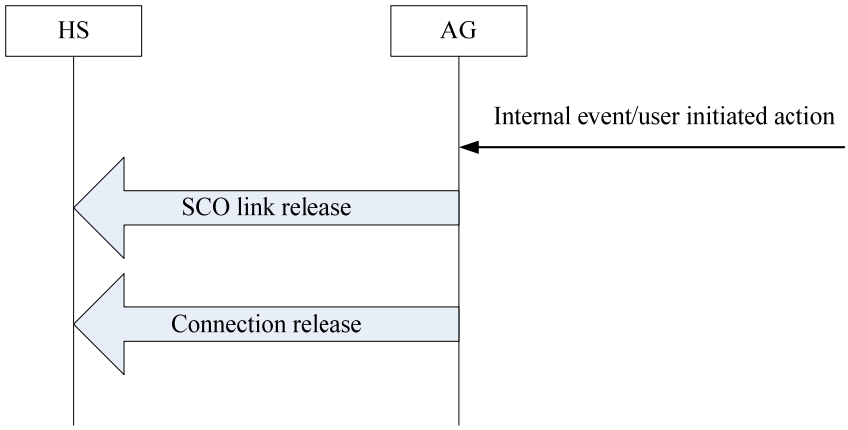

Figure 10 Audio connection release- AG initiated

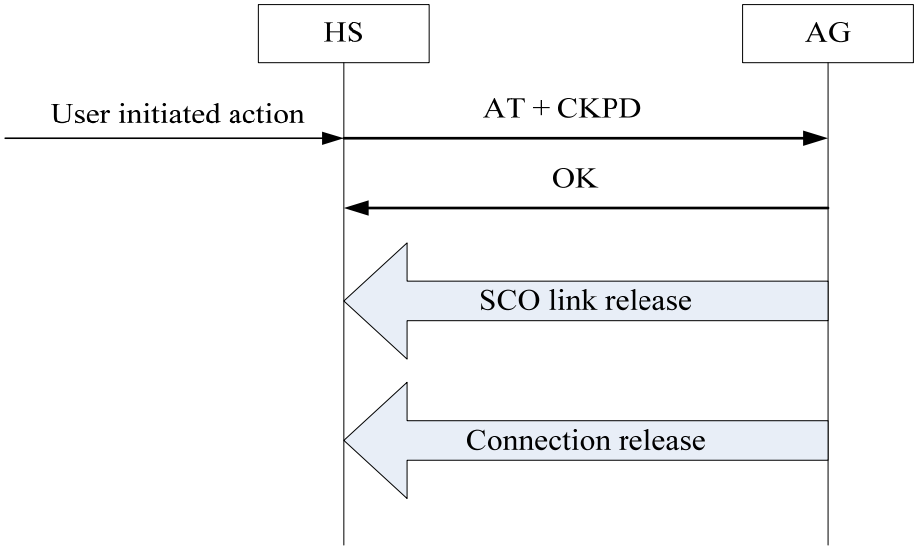

Figure 11 Audio connection release – HS initiated

A call can be terminated either on the AG or on the HS. The Fig. 10 and 11 [Hea01] shows the termination. Irrespective of the initiating side, the AG is responsible for releasing the connection.

All in all, the audio gateway controls the SCO link establishment and release, and the headset is responsible for connecting (disconnecting) the audio streams upon SCO link.
In practice, there are Audio gateway and Voice gateway. It is found that smart phones or PDA are using either Audio gateway or Voice gateway, such as both Sony Ericsson P900 and T-Mobile MDA Pro use voice gateway, while Sony Ericsson P990i employ Audio gateway. It is also discovered that the Audio gateway possesses the same

© May 2007 – Wen Hu & Yang Wu

function with the Voice gateway. The function is that the gateway of the audio and voice, both for input and output.

AT commands [ATC07] is used in headset profile which have the ability to ring, answer a call, hang up and adjust the volume. The audio gateway and headset provide serial port emulation which used to transport the user data from the headset to the AG. AT commands is included in the user data.

2.1.3 Using WiFi

Currently, there are two dominant short-range wireless standards frequently incorporated into mobile devices: Bluetooth and WiFi. From Wikipedia [Wif07]: "*WiFi is a brand originally licensed by the Wi-Fi Alliance to describe the underlying technology of wireless local area networks (WLAN) based on the IEEE 802.11 specifications.*" WiFi offers high-bandwidth local-area coverage. Therefore, it gives rise to a question: what difference does it make if we use WiFi?

The following table 1 [Cho07] summarizes the similarity and differences between the Bluetooth and WiFi. It is obvious that WiFi has a higher power-consumption than Bluetooth. Therefore, it is not practical for those smaller devices, such as cell phones, with limited power budgets. In addition, Bluetooth performs better than WiFi in the case of interference because Bluetooth employs a frequency hop transceiver. Another issue, cost, should also be taken into account. Bluetooth is much cheaper than WiFi and the former is widely available in most of the mobile phones available today. Few phones on the market support WiFi. Thus we prefer utilizing Bluetooth to achieve the goals of the project.

	Bluetooth	WiFi
Frequency range	2.4GHz	2.4GHz
Range	10 meters	91 meters
Bit rate (Mbps)	1 Mbps	100 Mbps
Immunity to interference	High	Medium
Voice quality with interference	Very Good	Good
Application network	Personal-area network	Ethernet network
Availability of headsets	Yes	No

Table 1 Bluetooth vs. WiFi

There is a WiFi phone for Skype currently on the market which does not require a PC to operate. The WiFi phone [Net07] is able to make free and unlimited phone calls to

© May 2007 – Wen Hu & Yang Wu

other Skype phones/users anywhere in the world, anytime have WiFi access, without a PC. Users could easily access their own Skype contact list to place a Skype call just like on their PCs. The WiFi phone not only allows using the keypad to dial out traditional numbers, but also redirects the incoming calls to other mobile or static phones.

It seems that the WiFi phone has resolved the first challenge of our project, which is to allow for the mobility while receiving Internet calls. However, there is still a downside when compared with the Bluetooth phone headset. The WiFi phone is compatible with no other telephony software except for Skype. Thus, it doesn't meet the project's needs, where the objective is that mobile phones be compatible with any type of telephony software.

2.1.4 Headset emulators

In this project, mobile phones will be used as Bluetooth headsets for PCs; in other words, the mobile phones could receive the Internet calls via their PC software. With wireless phone technology developing rapidly, there are some existing Bluetooth phones on the market now.

SkypeHeadset [Sky07] and Wireless Bluetooth Sky Phone [Blue07] are two hardware implementations that can receive Skype calls. The Wireless Bluetooth Sky phone is a produce that makes/receives/continues Skype calls up to a range of approximately 30 meters away from the PC [Blue07]. This kind of phone contains a PC dongle unit, a headset unit and a Y-USB cable for PC connection. Furthermore, it not only makes PC-to-PC calls and SkypeOut calls, but also receives SkypeIn calls. It is also important that the phone can maintain plug-and-play simplicity in installation.

These kinds of products mostly resolve the problem that we addressing here. For example, users would be able to receive Internet calls via their PC software on their mobile phones as long as they stay within 20-100 meters from their PCs. Yet they still have the same problem as the WiFi phone- they only work with Skype. Our objective in this project is to allow users to avoid considering compatibility issues as they now must while installing other VoIP technology. Additionally, once the users are far away from their PCs, the connection will be broken off. This is a limitation of existing products. The scope of our project also attempts to solve this problem.

In addition to the hardware implementations, some software applications that seamlessly connect mobile devices to Skype on PCs have also emerged. EpyxMobile [Epy07] and Skype PTT [Bla07] are two different software applications that connect mobile phones to Skype on a PC, while Vitaero [Use07] allows connecting a wireless Bluetooth headset to Skype. These software applications are free to download on PCs and could reduce users' mobile phone bills; the mobile phone users can benefit from these products.

© May 2007 – Wen Hu & Yang Wu

To realize the connection between mobile phones and Skype, what users need are a Bluetooth-enabled mobile phone and a Bluetooth-enabled PC with an internet connection running Skype. Take Useful Skype PTT as an example, Useful Skype PTT [Bla07] from Usefulapps Company is a push to talk client for Skype to be used in conjunction with a Bluetooth enabled PC. The calls are realized by the connection between Bluetooth and a PC with a Skpye client. Skype and the Useful server run on the PC, while the Useful client runs on the mobile phone; in other words, the phone is regarded as a PTT headset. In addition, one of advantages of Useful Skype PTT is that it can easily call and show the Skype contacted person on the phone. The calls are also free.

However, the limitation of the existing applications is that they work only within the Bluetooth distance of PCs. There are also further limitations. They are very device specific, which means that they will not work with all devices. Lastly, the existing software does not seem to be reliable and lacks widespread implementation and support according to users' comments.

2.2. Security

As we stated earlier, security challenges here can be divided into three sub-problems: authentication, authorization and privacy. However, most existing security methods bind them together. Therefore, we will present them one by one.

First, let's have a look at the security scheme in Bluetooth. According to Bluetooth specification Vol 2 Part H [Blu07], the initialization procedures consist of the following four necessary parts:

- Generation of an initialization key
- Generation of link key
- Link key exchange
- Authentication

After the initialization procedure, the devices can proceed to communicate, or the link can be disconnected.

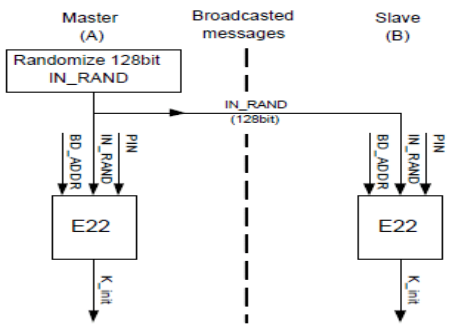

Figure 12 Initialization key [Sha05]

The initialization key is only created each time the unit is initialized, before any link key has been created and exchanged. It is derived from the Bluetooth device address BD_ADDR, a PIN code, the length of the PIN (in octets), and a random number IN_RAND. The procedure can be seen in figure 12 shown to the left. It is worth noting that for security reason, neither the initialization key nor the PIN will be

© May 2007 – Wen Hu & Yang Wu

transmitted directly. Instead of that, the IN_RAND will be sent from master to slave. The link key's generation and exchange will use the previous link key. If there is no link key from previous exchanges, the initialization key will be used. After the link key's generation, the previous link key or the initialization key will be discarded. The generation of the link key will use a random number from each device (LK_RAND_A LK_RAND_B), BD_ADDR of each device. LK_RAND are encrypted by the previous key or initialization key, and then sent to each other. Details can be seen below.

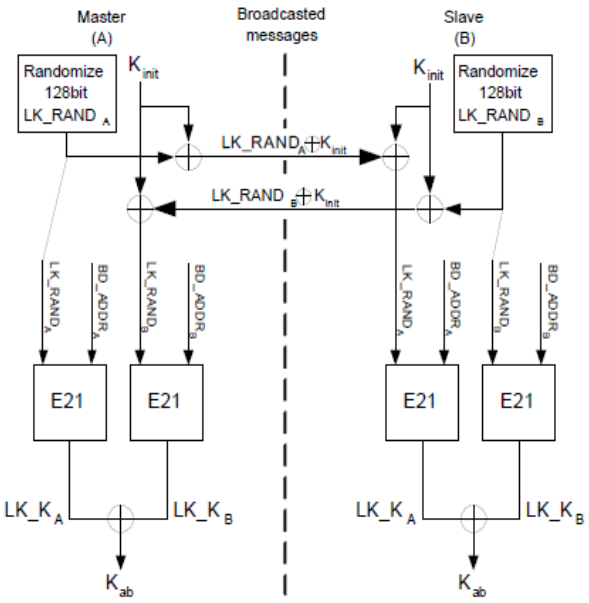

Figure 13 the link key's exchange [Sha05]

After the exchange of link keys, two devices will now mutually authenticate each other. This will be done by initializing a challenge / response authentication in turn. We will only reveal the challenge / response mode instead of showing the whole procedure. The verifier sends a random number while the requester encrypts it and sends it back. The verifier can then check whether or not it is the same as the local encrypted number.

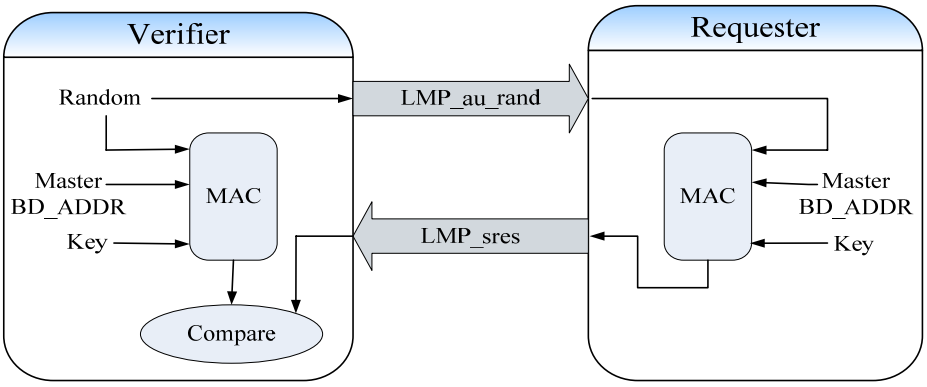

Figure 14 Authentication

© May 2007 – Wen Hu & Yang Wu

Algorithm such as E21, E22 and how to use the link key to encrypt data will not be explained in this thesis. Details can be found at [Blu07].

Although baseband specification details the SAFER+ algorithms used for security procedures, security is still a key issue for Bluetooth. Just like every wireless communication system, it's not hard for an attacker to eavesdrop on a transmission. As we mentioned above, there will be at least 7 messages, shown in table 2, transmitted in the air between two devices in each pairing and authentication process. If an attacker gets all of these messages, he can likely calculate the PIN, the initial key and the link key using a brute force algorithm. Paper [Sha05] explains the particulars of this process. Figure 15 gives the details in a flow chart. By [Sha05] the author's statement, "a 4-digit PIN can be cracked in less than 0.3 sec on an old Pentium III 450MHz computer, and in 0.06 sec on a Pentium IV 3Ghz HT computer". However, it is interesting to note that when the PIN's length is increased to 7, the time consumed will be 270 sec with their best implementation version.

#	Src	Dst	Data	Length	Notes
1	A	B	IN_RAND	128 bit	plaintext
2	A	B	LK_RAND_A	128 bit	XORed with K_{init}
3	B	A	LK_RAND_B	128 bit	XORed with K_{init}
4	A	B	AU_RAND_A	128 bit	plaintext
5	B	A	$SRES$	32 bit	plaintext
6	B	A	AU_RAND_B	128 bit	plaintext
7	A	B	$SRES$	32 bit	plaintext

Table 1 seven messages [Sha05]

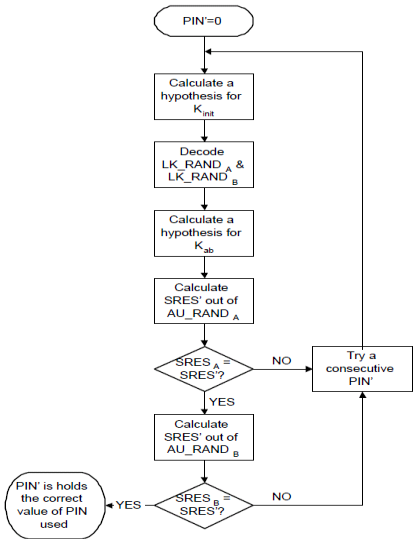

Figure 15 brute force algorithms [Sha05]

Therefore, if we decide to use Bluetooth's build-in security standard, we should increase the PIN's length from 4-digits, which is usually used, to 8-digits or more. This is also the author's suggestion to countermeasure this kind of attack.

As we clarified above, Bluetooth's security architecture is based on pre-shared keys and a challenge / response mode. With the method described in [Sha05], it's not hard for an attacker to get this shared key and then listen in on the communication. As an alternative to the shared key scheme, another widely used method is public key cryptography, also known as asymmetric cryptography.

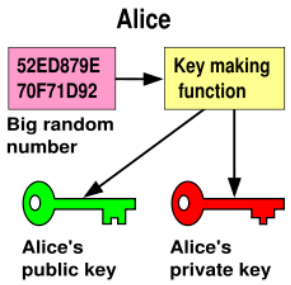

Figure 16 public and private key [Wik07a]

© May 2007 – Wen Hu & Yang Wu

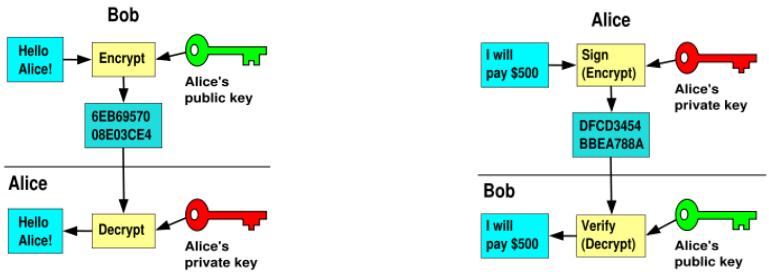

Figure 17 usage of public key and private key

As we can see from its name, there is no longer only one key between two devices or the communication. Instead, there will be two keys for each device: a private key held by the device itself, and a public key distribute to those who are authenticated and authorized. Anyone who wishes to talk with a specified device can encrypt using that device's public key, and only the device holding the private key can use it to decrypt. Also, the user can use a private key to encrypt a message; anyone who has the user's public key can check the signature using and thereby proving that message's authenticity.

However, this kind of mechanism has a central problem with proving that a public key is authentic. The usual solution to this is using a public key infrastructure (PKI). Actually, PKI has already been used in many fields like IPsec, TLS and S/MIME for authentication. There are many articles describing or discussing this topic as well as some enhancements [Ada99a] [Ada99b] [Cha97] [Ell99]. Since we are not focusing on security in this thesis, only rough explanation will be given here.

Public Key Infrastructure (PKI) is a general mechanism that allows authentic public key distribution to be used by large and distributed public key cryptography-based applications [Alb04]. This is usually implemented by software at a central location together with other coordinated software at distributed locations. The simplest way is to use a center-server, which can be connected from any point of the network, to act as the certificate authority. Whenever a device wants to communicate with another device, it has to first connect to the center-server. After authentication and authorization, the center-server will send the public key of the target device as a certificate to the claimant. With this certificate, this device can communicate with the target as it wishes. Although there is some criticism in regards to this architecture [Car00] such as the device's name issue and the reliability of certificate authority (CA), PKI is still one of the most popular security solutions.

Paper [Car06] also introduced a protocol called Simple Voice Security Protocol (SVSP) to merge pre-shared key (PSK) and PKI together. The name of the protocol promises to be useful for us, and the contents are interesting as well. SVSP uses Trusted Authentication Authority (TAA) and Global Trusted Authentication Authority (GTAA) to build up the entire architecture. As figure 18 below shows, PSK is used for the security between devices and TAA, and PKI is in charge of communication

© May 2007 – Wen Hu & Yang Wu

between TAA and GTAA. Perhaps we can make use of this at a later time.

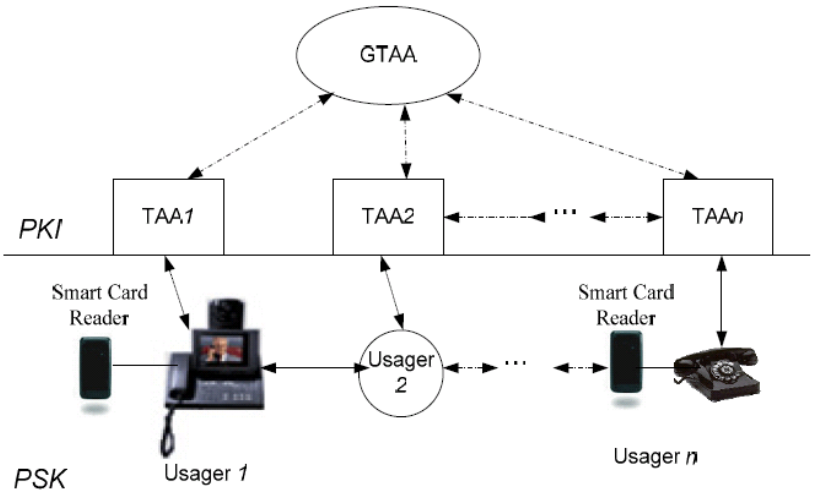

Figure 18 SVSP security architecture [Car06]

There is another similar solution to SVSP which has already been used in WiFi: 802.1x [80204]. This standard is based on the Extensible Authentication Protocol (EAP) which is described in RFC3748 [Abo04]. The purpose of 802.1x is to authenticate whenever a device wants to connect to the Access Point (AP) or build a point to point connection. The principle here is simple, whenever a Wireless Node (WN) connects to an AP, it must be authenticated and authorized before it can gain access to the internet. It is worth noting that this authentication will work not only for WN but also for AP. In other word, the Authentication Server (AS) will also provide information about AP to MN. In effect, this is a mutual-authentication. The following figure shows how a typical procedure works.

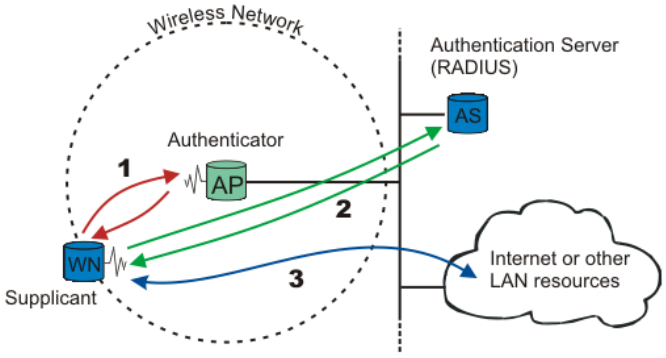

Figure 19 802.1x [Wik07b]

RADIUS here is the abbreviation for Remote Authentication Dial in User Service, which is an AAA (authentication, authorization and accounting) protocol acting as an Authentication Server. This Server works as a CA in PKI, which as we mention above, is in charge of the certification's distribution. Compared with SVSP, the difference is SVSP's GTAA connects to Smart Card via TAA while 802.1x's RADIUS connects to

WN directly.

In the article [Enr04], the author introduces a user-centered authentication and authorization architecture for ubiquitous computing environments. The security schemes we introduced above, such as SVSP and 802.1x, use an authentication and authorization server which can be connected from all of the entities in ubiquitous computing environment. There are many varieties of this mechanism, but the emphasis is the same: all units inside a ubiquitous environment must use this server to identify users. However, this also leads to the drawback of this kind of architecture. “What would then happen when two users meet at a remote isolated place?” is the question given by the author. This is also the major advantage of a user-centered authentication and authorization architecture. A user-centered design will probably fit the following situation shown below:

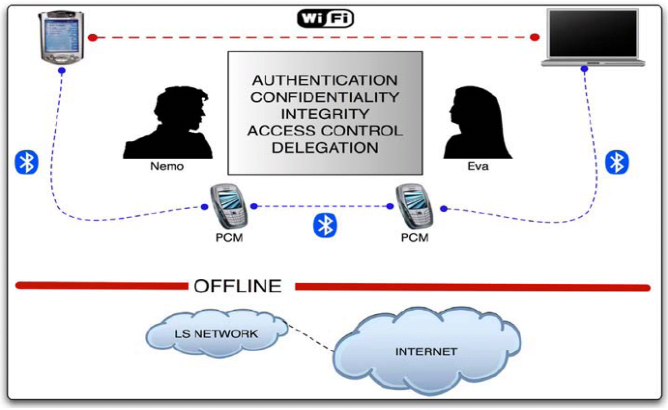

Figure 20 a scenario of SHAD [Enr04]

In this scenario, Nemo’s Pocket PC and Eva’s laptop are equipped with both Bluetooth and Wi-Fi technologies. Nemo needs to use the 17” display of Eva’s laptop to show her a high resolution video stored in his Pocket PC. A security architecture that depends on centralized entities could not face this situation. In contrast, SHAD fits well into this scenario: Nemo’s PCM and Eva’s PCM can negotiate tickets to allow Nemo’s Pocket PC to make contact with Eva’s laptop [Enr04].

There is no doubt that the advantage is outstanding if some of the devices are isolated. In our case, the mobile phone in our case can be competent as a PCM (Personal Command Module) [Lea04] which represents the users. However, in our project we assume that all PCs used to forward internet calls are connected to the internet. In other words, it is almost impossible for a failure to occur while connecting to the center-server, if there is one. Therefore, SHAD will not serve to be considered in our work.

© May 2007 – Wen Hu & Yang Wu

2.3 Service discovery

Service Discovery Protocol (SDP) [Gry01] enables network devices, applications, and services to seek out and find other complementary network devices, applications, and services needed to properly complete specified tasks. SDP is especially important in our project, where mobile phones could automatically select the most appropriate PC among many available PCs. Currently, there are several service discovery protocols being existing and in use. The most well known ones are: Service Location Protocol (SLP), Jini, Bluetooth's Service Discovery Protocol (SDP) [Gry01], Universal Plug and Play (UPnP) and Konark [HDL03]. These protocols, with the exception of Bluetooth SDP and Konark, are not intended for use in a wireless environment.

2.3.1 Bluetooth Service Discovery Protocol

Bluetooth's Service Discovery Protocol [Gry01] provides a standard means for a Bluetooth device to query and discover services supported by a peer Bluetooth device. SDP is a client-server protocol and relies on L2CAP links being established between the SDP client and server. Once a L2CAP link has been established, it can be used to find out about services and how to connect to them. A service may be implemented as software, hardware, or a combination of hardware and software. The server maintains all of the information about a service within a single service record, which consists entirely of a list of attributes. The client may retrieve information from a service record by sending out a SDP request.

SDP follows a request/response model [Blu03] where each transaction consists of one request protocol data unit (PDU) and one response PDU. SDP runs over L2CAP. As shown in the figure 21, a SDP client must receive a response PDU for each request PDU on the L2CAP connection. The Universally Unique Identifier (UUID) is the data type used for identifying services, protocols, profiles etc. Each record has a UUID attribute. A UUID is a 128-bit identifier that is generated once at the time a service is defined.

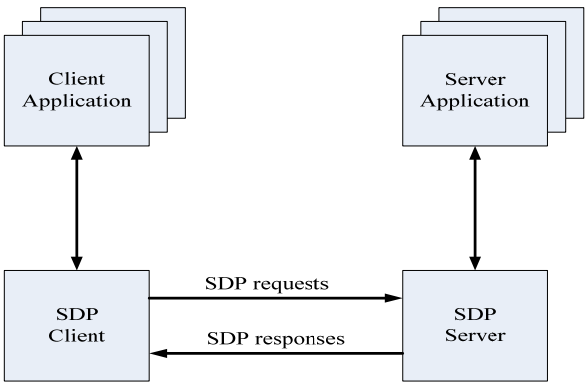

Figure 21 Diagram Source [Blu03]

© May 2007 – Wen Hu & Yang Wu

There are two search options supported in SDP [Sdp07]: searching and browsing for services. The former allows a client to query the SDP service for specific service attributes if the client knows the UUID of the service. Alternatively, service browsing is used when a client has no knowledge about services of interest in the client's vicinity. The client is then able to browse and select from the list of available services and the server responds to services that match the request.

The service search transaction allows a client to look for a specific service. The client uses an SDP_ServiceSearchRequest which contains a service search pattern. A service search pattern is used to locate the desired service and a list of UUIDs which the server uses to look for in its database. The server responds with an SDP_ServiceSearchResponse containing information about any service records which match the service search pattern as shown in Figure 21[Sdp07].

Compared with the searching method, browsing means looking to see what services are actually being offered. The mechanism for browsing is based on an attribute shared by all service classes. This attribute is called the BrowseGroupList attribute and the service classes are used to identify services. The value of the BrowseGroupList is a list of the UUIDs of all the browser groups associated with the service. Services are arranged in a tree structured hierarchy which can be browsed. Clients start to examine the root of the hierarchy by creating a service search pattern containing the UUID, allowing all services to be browsed.

2.3.2 Konark

"Konark is a service discovery and delivery protocol designed specifically for ad hoc, peer-to-peer networks, and targeted toward device-independent services in general and m-commerce oriented software services in particular"[HDL03]. It has two major aspects: service discovery and service delivery. Here we only focus on service discovery.

Figure 22 [HDL03] shows the Konark service discovery protocol stack. Konark SDP Manager is of importance in the service discovery mechanism. Each device in the Konark community has a Konark SDP Manager that discovers the required services on behalf of Konark applications. Its main function is to interact with the messaging layer to send and receive the discovery and advertisement messages. The messaging layer is applied above the transport layer.

© May 2007 – Wen Hu & Yang Wu

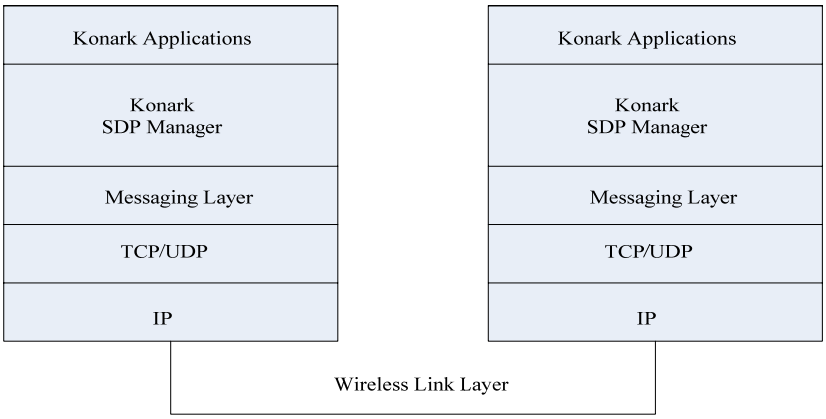

Figure 22 Konark Service Discovery Stack [HDL03]

Konark also uses the request/response model. A client sends out a discovery message which contains either the path from the service tree or a keyword, and then the server that provides the service responds. There are two modes supported by Konark for discovering services. In active pull, a client sends a query to all of the nodes to find the service location. In passive push, a server advertises the service information to the entire network periodically. In addition, Konark utilizes caching of service information on each node to improve service discovery efficiency [YTO06]. Unfortunately, Konark does not consider energy consumption or delay.

What the interest of Konark is the wireless link layer. We can use an IP level connectivity between the devices over any wireless link like IEEE 802.11 or Bluetooth.

2.3.3 DNS Service Discovery

DNS Service Discovery [Har03] is a way of using existing DNS Resource Records to locate services. From [CK06]:*Given a type of service that a client is looking for, and a domain in which the client is looking for that service, this convention allows clients to discover a list of named instances of a that desired service, using only standard DNS queries. In short, this is referred to as DNS-based Service Discovery, or DNS-SD.* DNS Service Discovery uses a combination of PTR, SRV (Service resource record) and TXT records to locate services and their attributes.

The SRV packet was originally designed to locate a particular type of service over the open Internet. TXT records are used to convey attribute information, while PTR records are normally associated with reverse lookups (that is, given an IP address, PTR allows you to determine the name associated with that address). PTR records enable service discovery by mapping the type of the service to a list of names of specific instances of that type of service.

However, DNS-SD is not reachable for our project, since the presupposition of DNS-SD is that all nodes must be in network. For our project, we can not guarantee that the mobile phone is connected to a network. In fact, in most cases it will probably be an isolated device.

© May 2007 – Wen Hu & Yang Wu

2.4 Redirect

When the mobile phone moves to another place which is no longer within its original PC's Bluetooth range, the incoming VOIP call will be redirected to the new PC with which this mobile phone now connects to. Thus, it is useful to have a look at the existing aspects regarding "redirection."

2.4.1 Basic approach

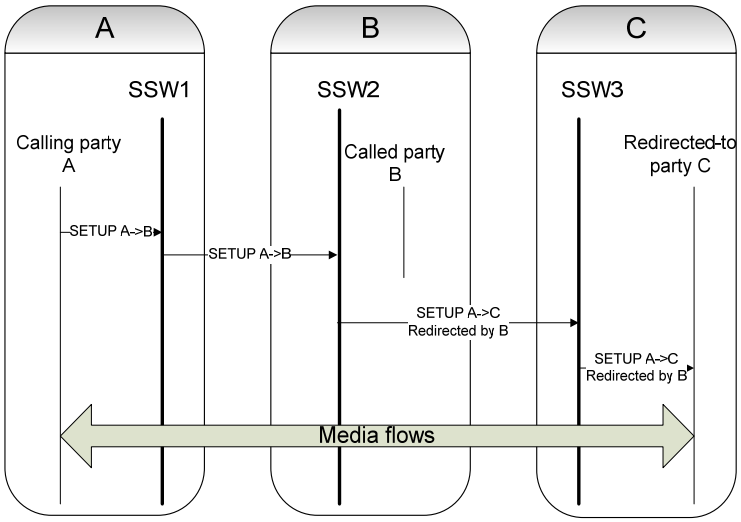

Figure 23 VoIP call forward scenario

According to [Her05], the VoIP call forward scenario is just like the figure shown above. This procedure derives from the traditional PSTN call forwarding implementation. As we can see, softswitch2 (SSW2) and endpoint B are responsible for managing the call forward. This is to be done by initiating a new call to the redirected-to party, in this case C, managed by SSW3. After this call is initialized, B can inform both A and C to build their own connection.

Later during this connection, if A wants to connect to C directly, without B's help, it can operate according to the call transfer implementation. In this procedure, B first notifies C about this. After C agrees to send the call ID (maybe its address), B asks A to transfer the call. Then, A sets up connection with C, and at same time, B releases the call between itself and C. Finally, when C connects to A successfully, A releases the call between itself and B.

We can see that in IP telephony's basic solution, two primary approaches are used: proxy mode which uses a middleman between two end-users, and redirect mode which use a middleman to get the other party's call ID.

© May 2007 – Wen Hu & Yang Wu

2.4.2 Redirection in Mobile IP

According to [Dou06], Mobile Computing refers to a system that allows computers to move from one location to another.

In [Dou06], the author gives an overview of Mobile IP operation. “The biggest challenge for mobility lies in allowing a host to retain its address without requiring routers to learn host-specific routes. Mobile IP solves the problem by allowing a single computer to hold two addresses simultaneously: a permanent and fixed primary address that applications use, and a secondary address that is temporary. The temporary address is only valid while the computer visits a given location.”[Dou06]

“A mobile host’s primary address is assigned on the host’s home network. After it moves to a foreign network and obtains a secondary address, the mobile host must send the secondary address to a home agent, usually a router located on the home network. The agent agrees to intercept datagram sent to the mobile’s primary address, and uses IP-in-IP encapsulation to tunnel each datagram to the secondary address.”[Dou06]

So, since a mobile uses its home address to identify itself whenever communicating with an arbitrary destination, all replies will be sent to its home network, where the home agent will take care of forwarding those packages to the foreign agent which the mobile is now using.

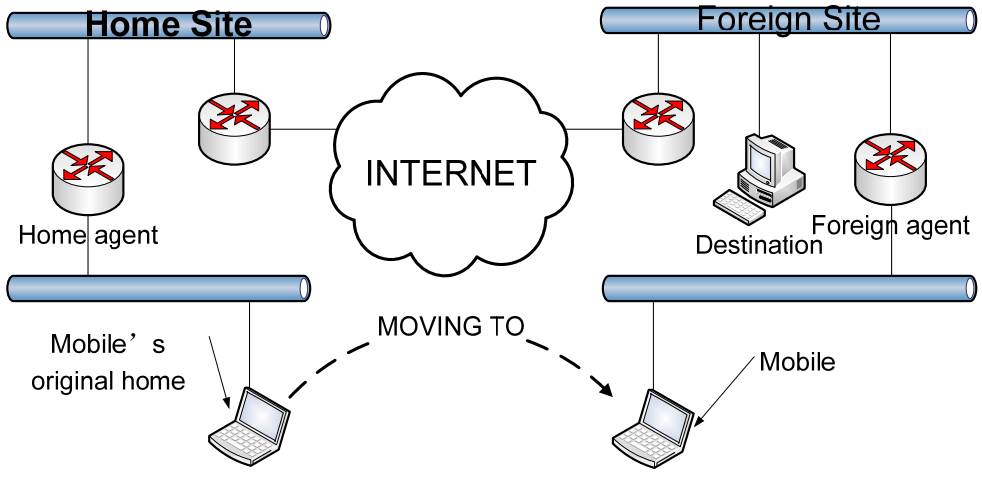

Figure 24 a scenario of Mobile IP

In our scenario, we can let the user’s own office PC act as a home agent, and let the PC know that a mobile phone is connected which acts as a foreign agent. In that case, the redirection problem can be solved.

© May 2007 – Wen Hu & Yang Wu

2.4.3 Redirection in SIP

As we know, SIP also can operate in two modes: Proxy mode and Redirect mode which are illustrated in Figure 25 and Figure 26. However, there is an important step before any of these two modes can be executed: the mobile user must first register to a SIP server about its current location, in this case, IP 131.161.1.112.

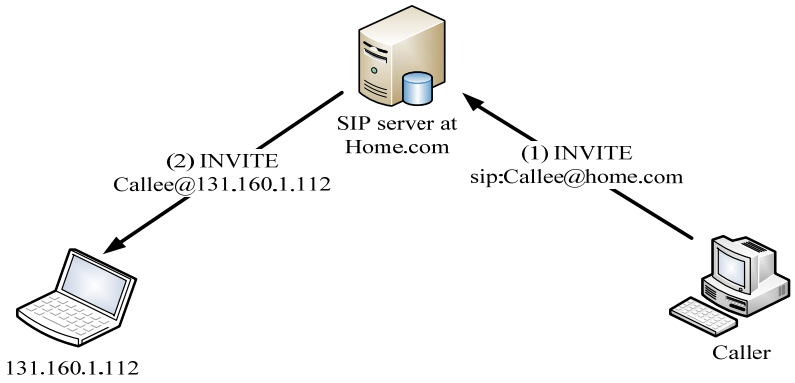

Figure 25 Proxy mode

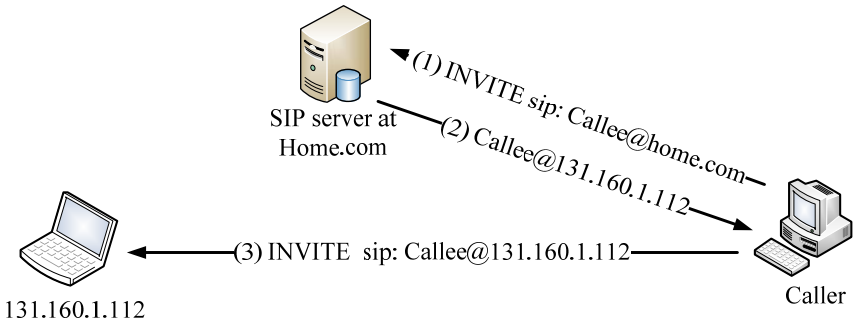

Figure 26 Redirect mode

The difference here is that in redirect mode, Proxy will tell the caller about the callee's address while in Proxy mode, Proxy will act as a mid-ware by calling the callee directly.

We can consider the caller in this mode as the VOIP caller in our project; the callee as the PC which the end-user's mobile phone now connects with and the Proxy as the user's original PC. If we implement our task as such, we will let the user's PC tell the VOIP caller the new PC address of the mobile phone, instead of forwarding. This can be a solution, too.

© May 2007 – Wen Hu & Yang Wu

2.5 Fast Handover

In our project's second phase, the aim is to achieve the possibility of moving from one PC's Bluetooth range to another while keeping the call connected via a Bluetooth headset. Handover occurs when the headset moves from one PC's Bluetooth range to another. Certainly, we don't want the call to be interrupted because of walking to a new piconet. Therefore, handover is important. In this section, we will examine the existing research about handover WiFi and Bluetooth. However, the usability of these handover methods in our application is yet to be discussed.

2.5.1 Handover in WiFi

WiFi, known as IEEE 802.11, denotes a set of Wireless LAN/WLAN standards developed by working group 11 of the IEEE LAN/MAN Standards Committee (IEEE 802). Since Bluetooth is IEEE 802.15.1, both of them belong to IEEE 802. WiFi is also the most widely used technology in WLAN so maybe we can use some aspects of it.

Indeed, the basic idea of handover in WiFi is a little different compared with what we want. In a normal case, handover in WiFi is disconnecting from the current AP, scanning for a new AP and then trying to connect to a newly found AP. There are many solutions intended to handle handover in WiFi. The basic strategy is the same- to divide this procedure into 3 parts: detection, selection and execution. Still, this procedure will definitely lead to a halt during voice transmission, which is something we want to avoid.

Recently, however, there is a draft called 802.11r which intends to solve this time-delay problem and especially fits VoIP and other QoS applications' requirements. The reason that hundreds of milliseconds appear during handover is mainly because of the 802.1x security scheme, which we have presented briefly before. According to [San06], this delay will cost 525ms for average roaming, while on the other hand, 802.11r will only take 42ms in average. The core idea of 802.11r is that it allows the mobile user to use the current access point (AP) for the procedure of authentication with the next AP that it will probably connect to. In the normal way, authentication and authorization between the mobile user and new AP will occur after the current connection is disconnected. With 802.11r this will be operated via the current AP. The user can start to authenticate and authorize the new AP while keeping the current connection, which reduces time-cost efficiency during handover.

2.5.2 Handover in Bluetooth

Although the situation with Bluetooth is the same as WiFi, there are several research papers trying to solve or make improvement on this problem. We chose one of them which we thought should benefit our work.

© May 2007 – Wen Hu & Yang Wu

In [Geo02], the author provided three handover proposals both in the theoretical description and the testing result.

Method 1:

BT periodically looks for a new Base Station, and keeps the information on the Base Station in a stack. When the RSSI of the link between the BT device and the current Base Station falls below the threshold level, the Base Station informs the BT device to page another Base Station, whose address is kept in the stack of the BT device.

Method 2:

When RSSI falls below the threshold level, the Base Station sends the information on the BT device and the request to page the BT device to all nearby Base Stations through the wired network.

Method 3:

The BT device keeps a backup link with another Base Station all the time. Once the quality of the current link falls below the criterion, BT uses the backup link. The poorer quality link is disconnected. To build the backup link, the BT device goes into periodic inquiry mode to connect the new Base Station.

Although there is another algorithm introduced by [Min05] which claims that it works better, we think that it is only a similar version with exiguous change to [Geo02]'s method 3 (by increasing the backup link from 2 to 3). Therefore, we will take [Geo02]'s method 3 as the formal one.

[Geo02] Also provide the test results as follows:

Handover Technique	Handover Timing (Range)(ms)	Handover Timing (Average)(ms)
Method 1	220-241	231.2
Method 2	170-190	172
Method 3	Almost Instantaneous	Almost Instantaneous

Table 2 Test result

Comparing these algorithms' results, we can see that method 3 is the most appropriate one to our work.

However, Bluetooth specification is not very suitable for issues like handover [Baa00]. The methods we mentioned above need periodical scanning for accessible Bluetooth devices, and this will block all current Bluetooth traffic. In other words, if we use the methods above, the halt will not occur during handover; instead, it will occur regularly, even if you are not moving. That's absolutely ridiculous.

© May 2007 – Wen Hu & Yang Wu

3. SYSTEM ARCHITECTURE

3.1 Flow Chart

As we mentioned earlier, there are two scenarios of our project. The first phase is where a callee stays within the local PC's range before the call is connected, while the other one is where the callee is roaming. According to the two different scenarios, we present the following two flow charts. It is obvious that the first scenario consists of six core phases: Device discovery, Connection establishment, Authentication, Monitor, Call establishment, and Call termination. The second one is almost the same as the first, with the exception of adding a Location register after Authentication. We describe each block in detail below.

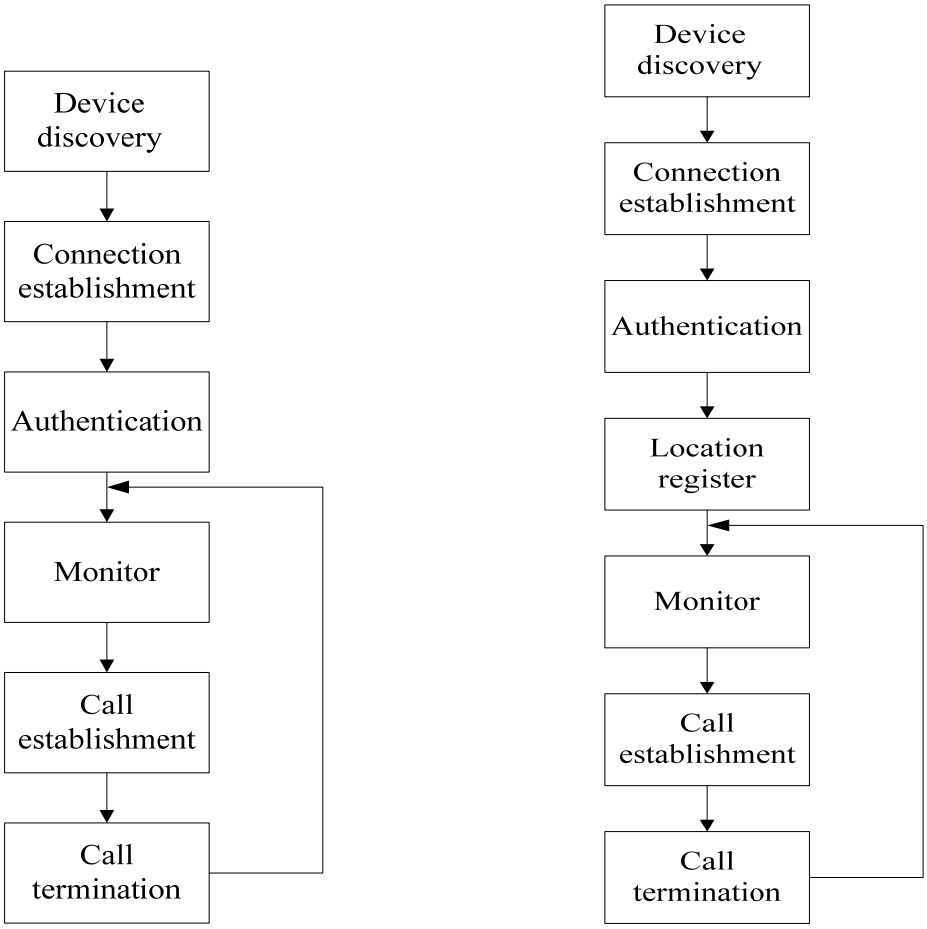

Figure 27 Home domain and Roaming

© May 2007 – Wen Hu & Yang Wu

3.2 Message Sequence Charts

3.2.1 Device discovery and Connection establishment

Figure 28 shows the message flow for Device discovery and setting up a Connection. When a user who holds a Bluetooth enabled mobile phone comes into a room which has a PC, the phone will automatically initiate an inquiry to find out what access points are within its range. The PC in the visited room responds with its address and the phone picks. Then the mobile phone will invoke a baseband procedure called paging. It synchronizes the device with the access point.

After the Device discovery, a connection between the PC and the mobile phone is set up. The Link Manager Protocol establishes a link with the PC. Then the LMP will use the Service Discovery Protocol (SDP) to find out what services are available from the PC. For our project, what we need is Telephony service. After answering the L2CAP and RFCOMM connection requests properly, a virtual link is established between the two devices.

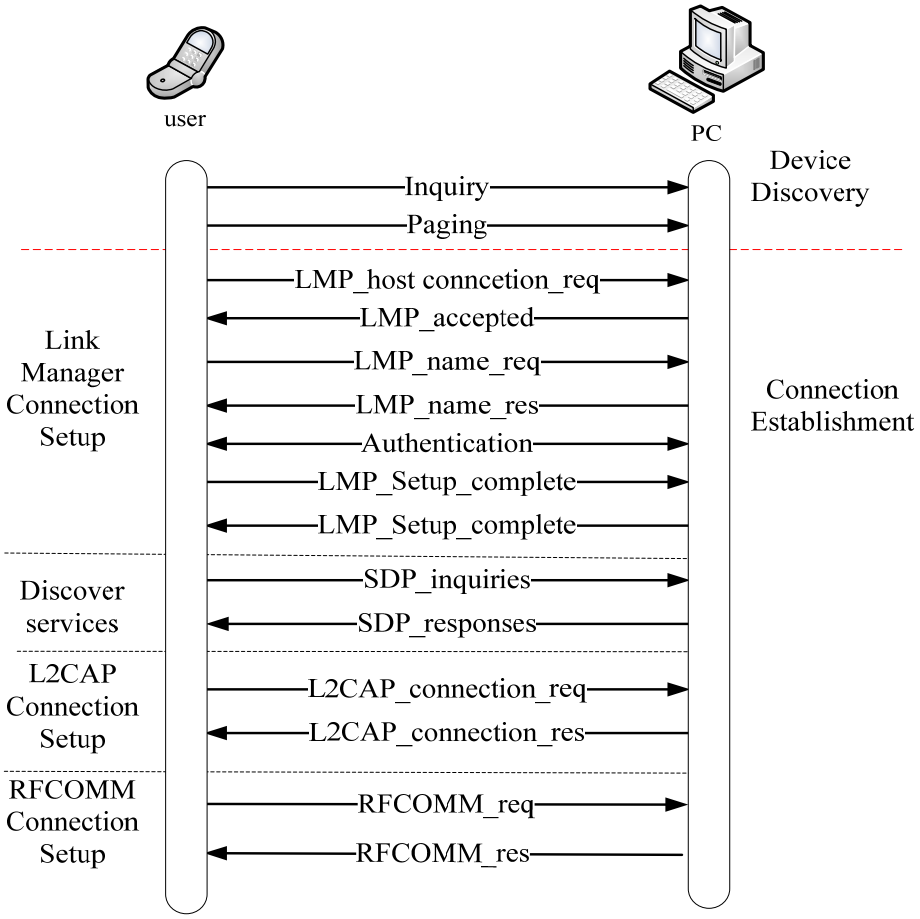

Figure 28 Device discovery and Connection establishment

© May 2007 – Wen Hu & Yang Wu

3.2.2 Authentication

After the connection between the two devices is set up, the next step is Authentication. Authentication is used for identifying and validating users. During this procedure, challenge/response mode will be used. The AAA server will first ask the PC and mobile user to present their IDs. Then, it uses each ID's public key to encrypt a message, and sends it respectively. When the mobile user or PC gets this message, they decrypt it and encrypt a reply with their private keys. After the AAA server receives these replies, it checks whether they can be decrypted by each ID's public key, which was used to encrypt message earlier. If everything is OK, then the AAA will inform both the mobile phone and PC that they can trust each other.

The Authentication_done message will not only contain a message saying "OK". Instead, AAA will send the mobile user's public key to the PC while using the PC's public as the encryption key, and does the corresponding procedure to the mobile user. The reason for doing that is to guarantee that no information will be leaked during further message exchanging. Each device will use the other device's public key to encode the message, so only that device will know what the message is.

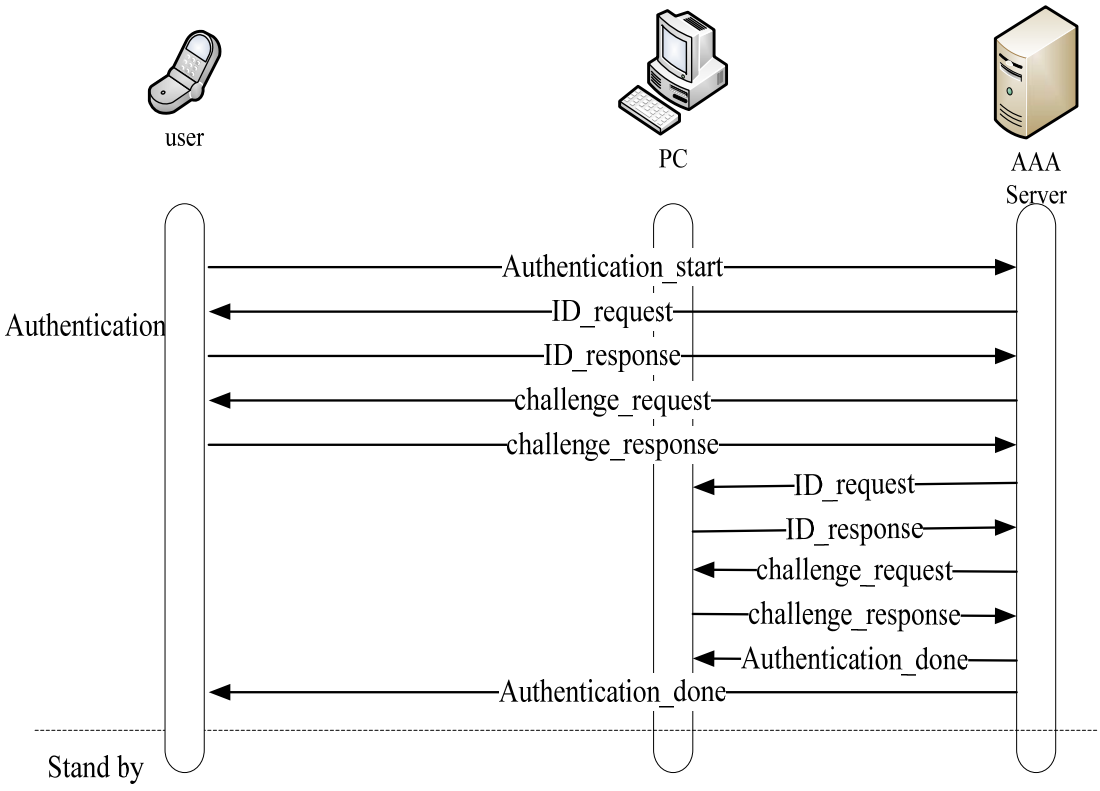

Figure 29 Authentication

© May 2007 – Wen Hu & Yang Wu

3.2.3 Location register

The Location register is utilized when the user moves to another visited room and stays within a foreign PC's range. The messages exchanged here are quite simple. The foreign PC simply notifies the Home PC that the mobile phone is now inside its scope. Then, the Home PC answers with an "accept" message.

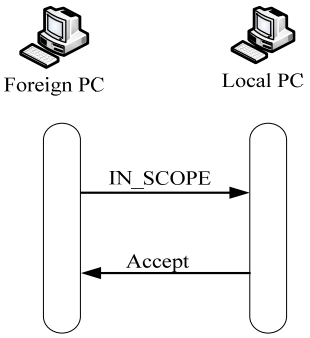

Figure 30 Location register

3.2.4 Monitor

To ensure the connection, the PC has to monitor the phone periodically every few seconds to confirm that the phone is still within its range. If the phone is out of range, the connection will be terminated. As shown in Figure 31, the phone will respond to the corresponding PC to indicate that it is in range now.

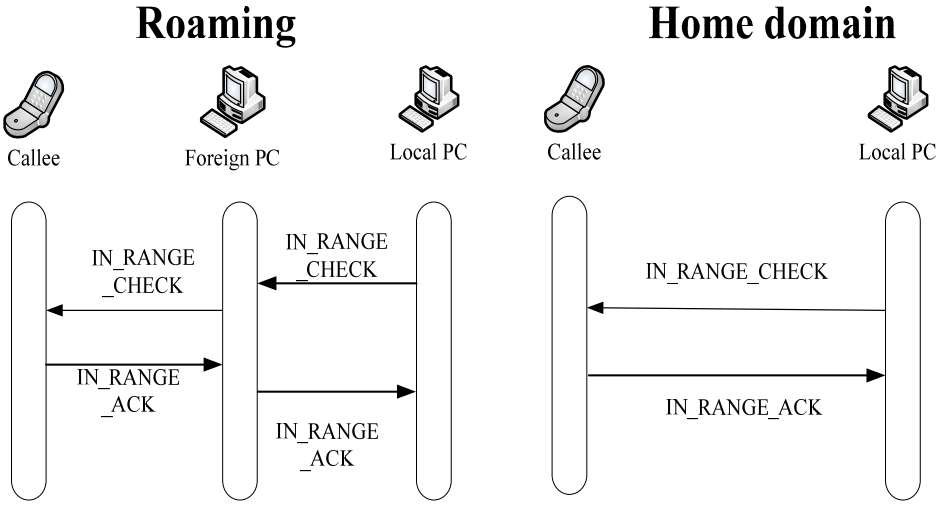

Figure 31 Monitor

3.2.5 Call connection and disconnection

After the above components are implemented, an "invite" command is sent from a caller to the callee and the callee responds with the "accept" command, indicating that

© May 2007 – Wen Hu & Yang Wu

the incoming call is adopted. Finally, the PC initializes the SCO link that contains the speech signal, and the connection process is complete. At the end of the call, the caller sends a “hangup” command to the callee and the callee responds; then they terminate the call.

Figure 32 and 33 presents the Call connection and disconnection of two phases, respectively. The latter phase is more complicated than phase1, because the local and foreign PCs have to exchange voice data via LAN in phase 2.

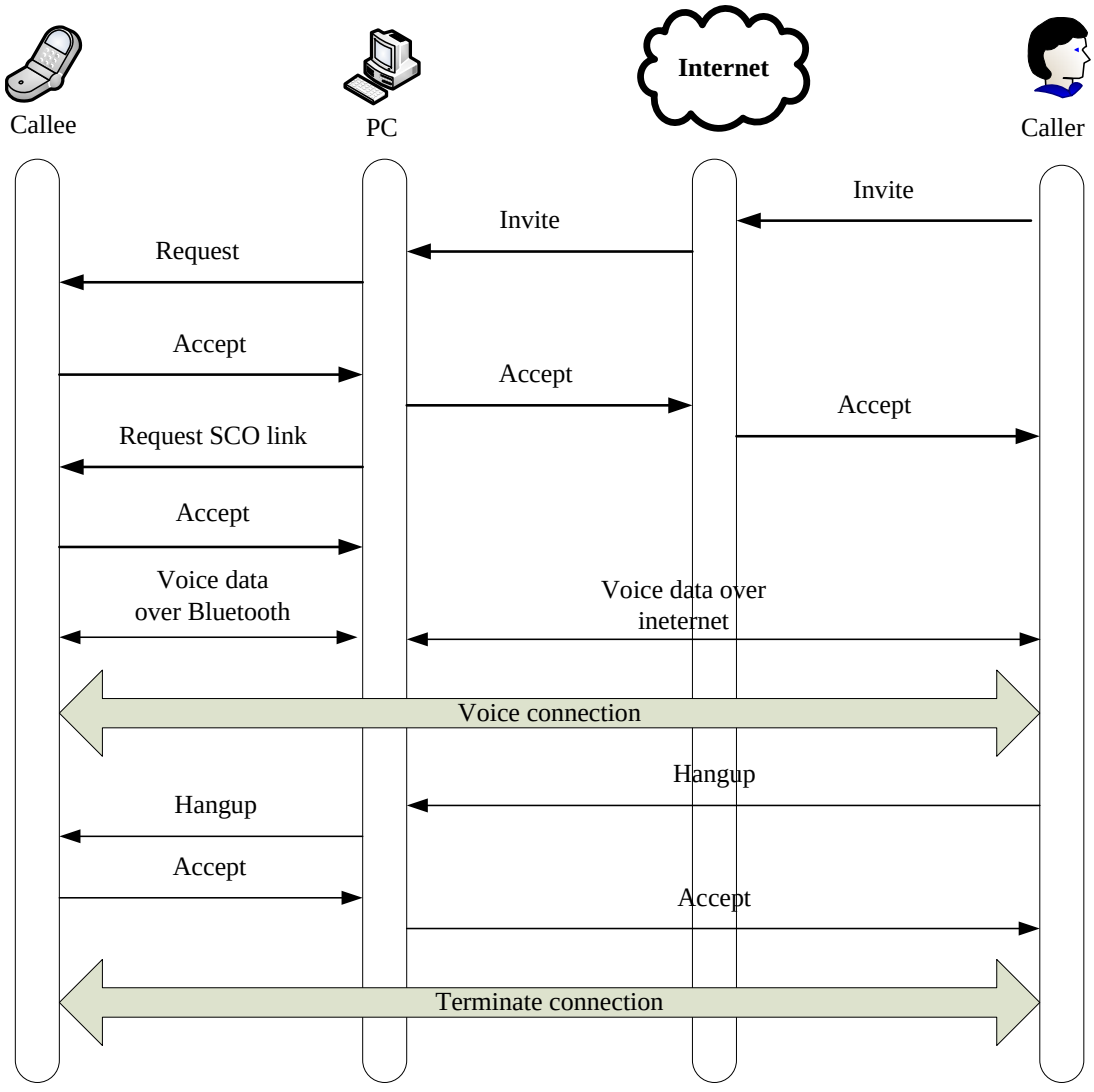

Figure 32 Call connection and disconnection (Home domain)

© May 2007 – Wen Hu & Yang Wu

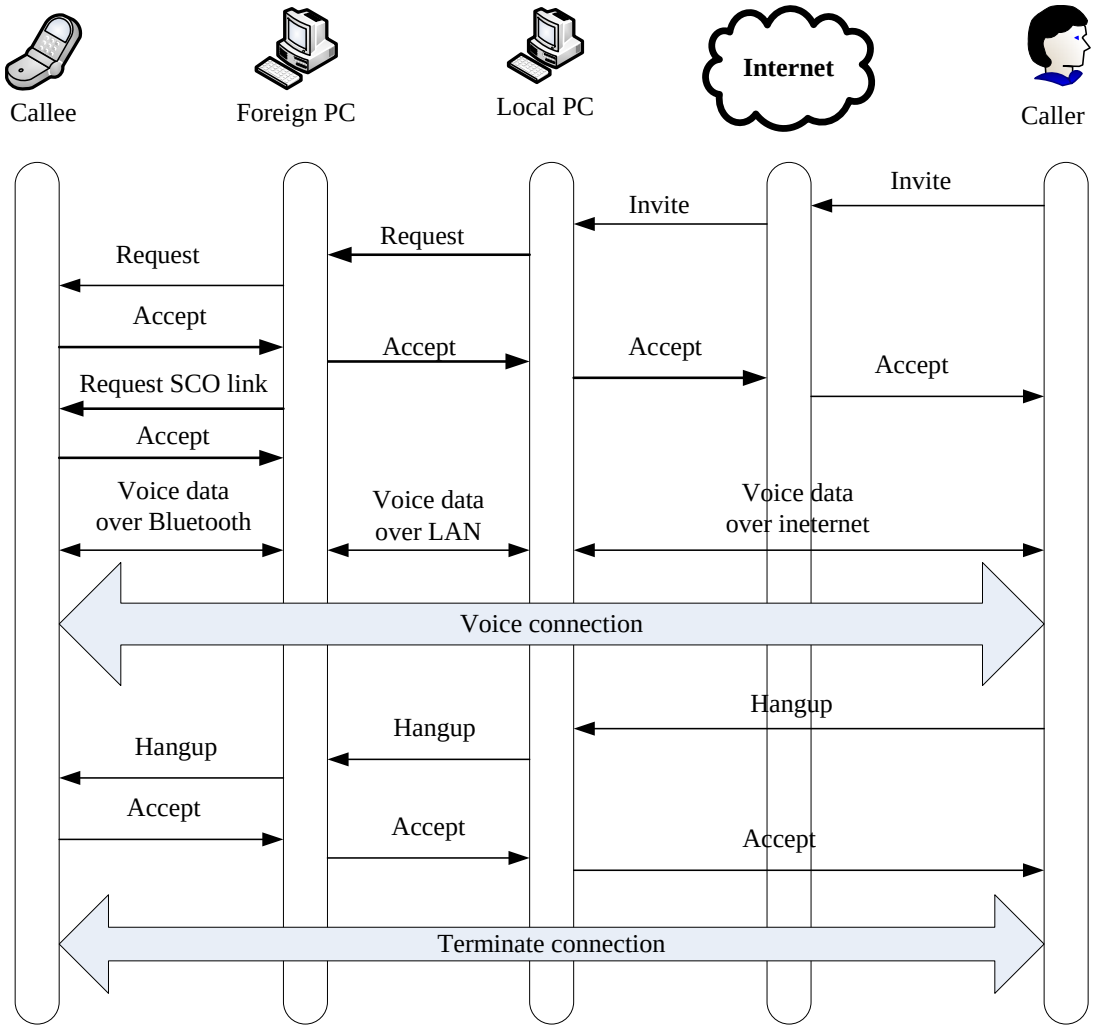

Figure 33 Call connection and disconnection (Roaming)

© May 2007 – Wen Hu & Yang Wu

3.3 “4+1” views

According to [Phi95], five views should be presented to describe software architecture, especially when dealing with a large and challenging architecture. Figure 34 illustrates which five views should be presented.

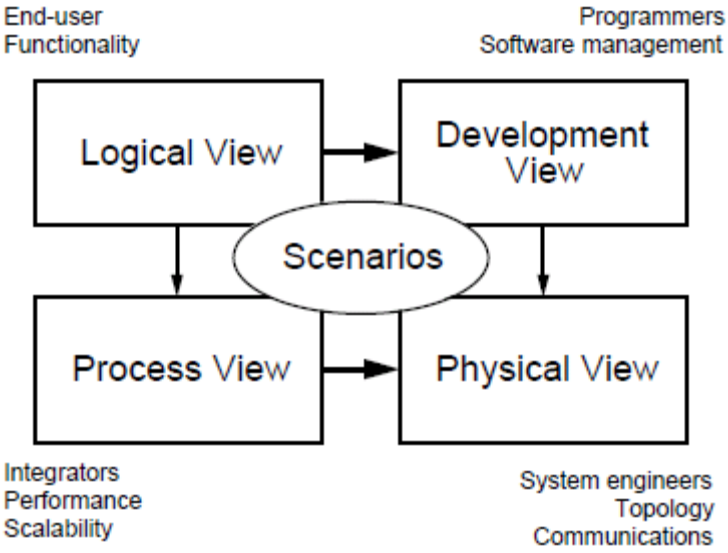

Figure 34 “4+1” views [Phi95]

Considering our project, scenario parts have already been introduced in previous chapters. Among the remains of these views, logical view cares more about details such as which class should be used and what’s the connection between these classes; development view gives us a layered overview of the whole project; process view focuses on message exchange between different blocks; physical view reveals how devices are physically connected. Therefore, at this point we’ll discuss most of these views except the logical view.

3.3.1 Process Architecture

Process will give us information about the connection between blocks inside each application. As we mentioned before, there are two different applications, which are the PC and Phone’s view, according to different devices, respectively.

As shown in Fig.35, there are eight processes in the PC’s process view. The core process is Link management process which is composed of Connection, Controller and Service discovery. The Controller receives events from and sends control commands to the other seven parts. The Connection part’s task is connection establishment and the Service discovery part uses Bluetooth’s SDP to find out what services are available from the PC.

The following table 4 describes the function of the rest of the seven processes, except

© May 2007 – Wen Hu & Yang Wu

the Link management process which is depicted above.

	Function
Security process	● Identify PC to AAA server ● Forwarding mobile phone's authentication messages.
Call management process	Manage calls' connection and termination from PC
VoIP interface process	Forwarding voice data from VoIP software to application on PC
User interface process	Interact with the user
LAN Data process	Forwarding voice data from home PC to foreign one.
Monitoring process	Monitor if phone is within the PC's range
Location register process	Locate a phone when it moves to a foreign PC's range

Table 3 seven processes' function

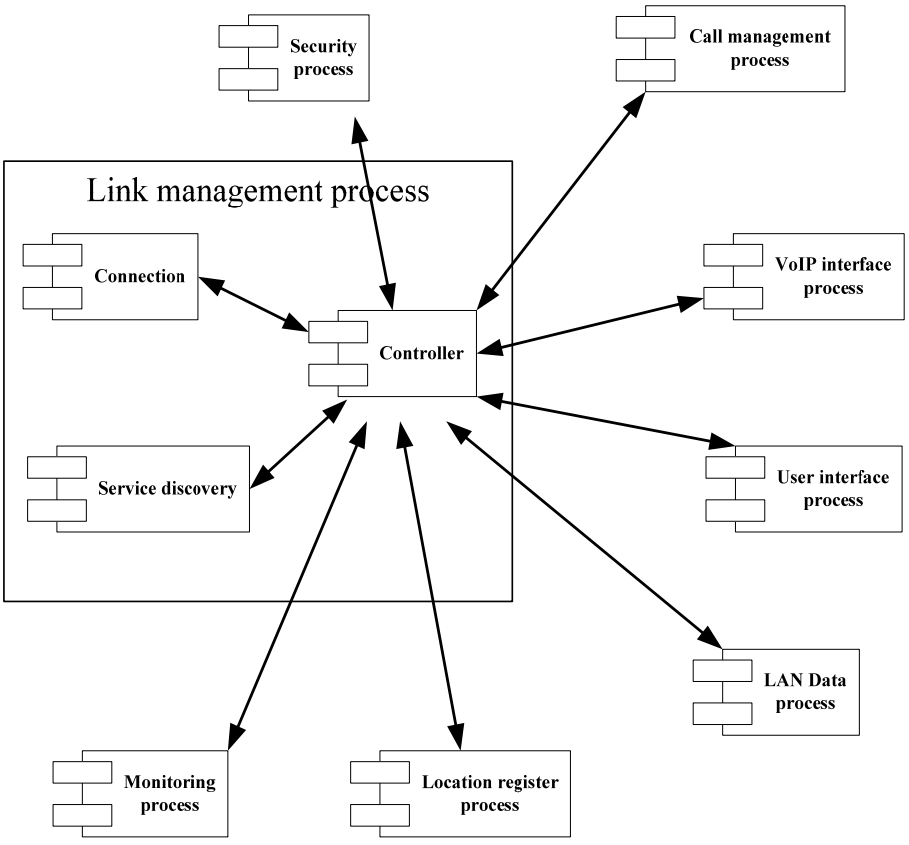

Figure 35 PC's process view

For the phone's process view, the core process, Link management process, adds a Search part. The Search part is for finding a PC to connect to. It sends inquiry and paging message to corresponding PCs and responded by PC's controller. The

© May 2007 – Wen Hu & Yang Wu

functions of the other three parts of the Link management process are almost the same as the PC's, except that for the Link management process, we find Security, Call management, User interface, and Monitoring. Their functions are shown in the following table, respectively.

	Function
Security process	Sending authentication messages to AAA server via PC
Call management process	Manage calls' connection and termination from mobile phones
User interface process	Interact with the user
Monitoring	Monitor if phone is within the PC's range

Table 4 four processes' function

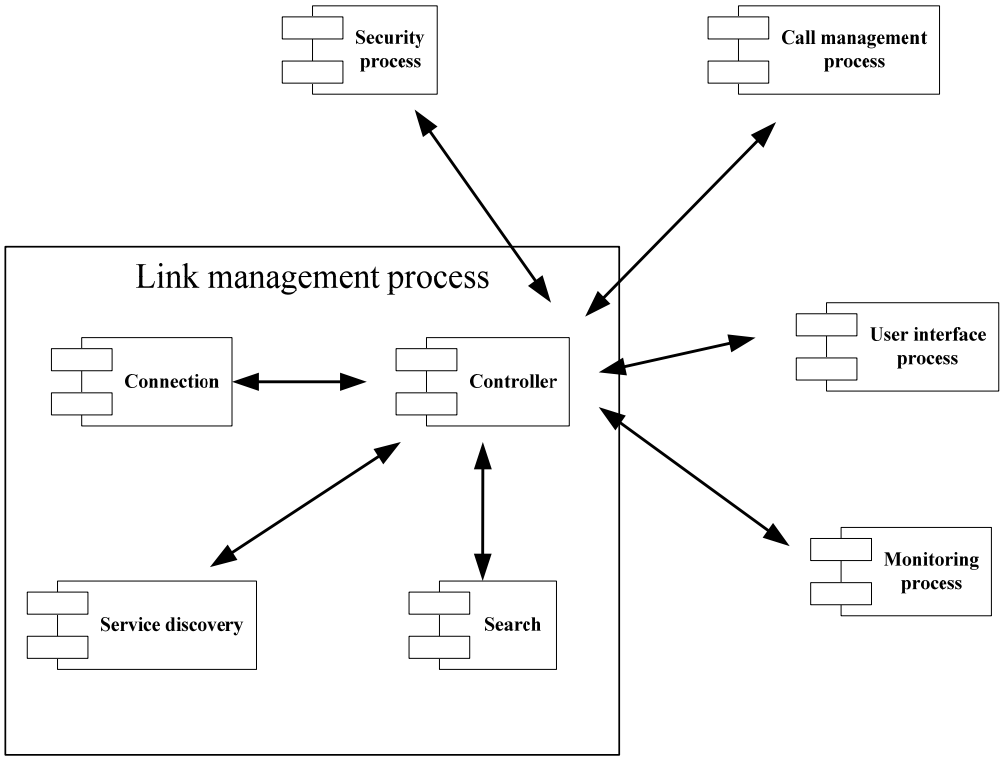

Figure 36 Phone's process view

3.3.2 Physical Architecture

Figure 37 presents the Physical view of our project. The physical architecture is to map the software into the hardware. According to our project design, we come up with the following physical architecture figure. Four hardware are needed in our project such as AAA server, Home and Foreign PCs and a Bluetooth-enabled mobile phone.

✧ The function of AAA server is a server which handles request from clients and

© May 2007 – Wen Hu & Yang Wu

provides certification to Bluetooth devices. The architecture of AAA sever is beyond scope of our project and we just concern its function to distribute certifications to Bluetooth devices.

- ✧ Home and Foreign PCs that install telephony software are used for forwarding Internet calls to mobile phones.
- ✧ In the project, the function of a Bluetooth-enabled mobile phone is used as a Bluetooth headset for PCs that run Internet telephony software.

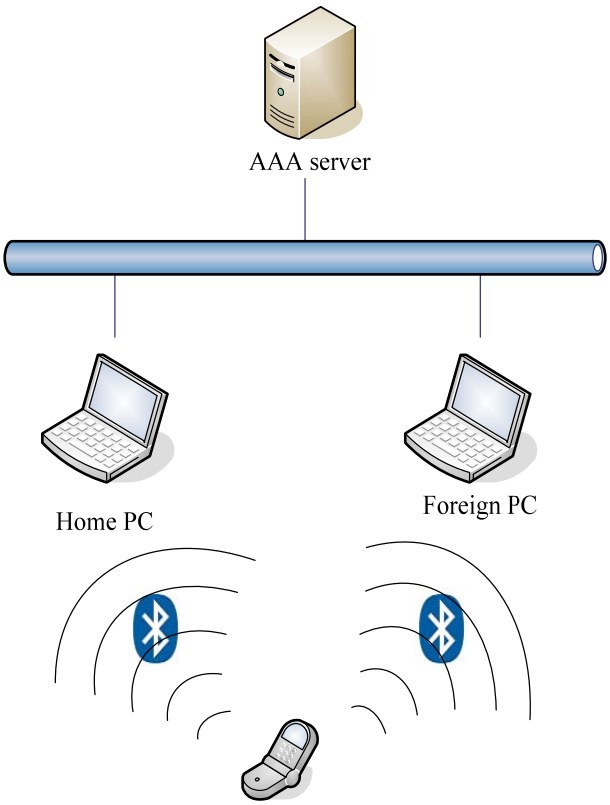

Figure 37 Physical view

© May 2007 – Wen Hu & Yang Wu

3.3.3 Function Architecture

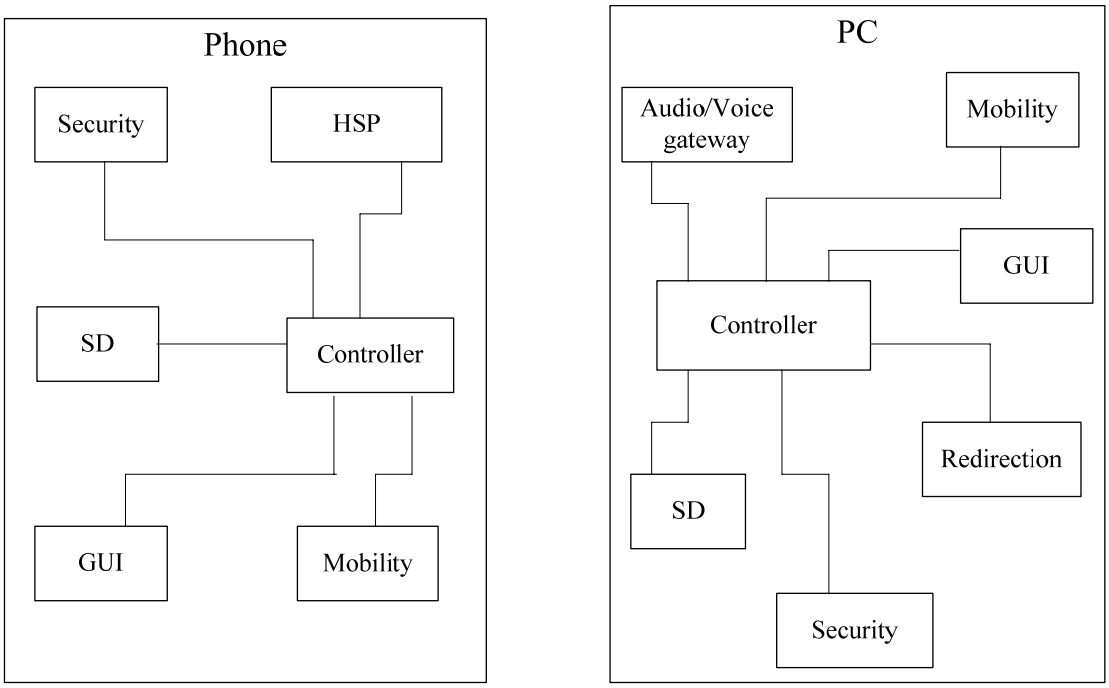

Figure 38 Phone and PC's functional view

In this part we come up with the other architecture, the function views or development view, to comprehensively understand our project design. The function views are shown from two different parts: Phone and PC.

The phone's function view consists of six components which are Security, HSP, SD, Controller, GUI and Mobility.

- ✧ Controller: the main component in the phone's function view. It manages the other five components and also plays a role of connection between different components.
- ✧ Security: sending authentication messages to AAA server via PC.
- ✧ HSP (Headset profile): implement HSP in mobile phones to act as Bluetooth headsets.
- ✧ SD (Service discovery): find corresponding service which we need.
- ✧ GUI (Graphical User Interface): Interact with the users.
- ✧ Mobility: the component is for roaming when phones are moving from one domain to another.

The PC's function view consists of seven components which are Controller, Security, SD, Mobility, GUI, Redirection and Audio/Voice gateway.

- ✧ Controller: the main component in the PC's function view. It manages the other six components and also plays a role of connection between different components.

© May 2007 – Wen Hu & Yang Wu

- ✧ Security: identify PC to AAA server.
- ✧ SD (Service discovery): response the service from the mobile phone.
- ✧ Mobility: monitor if the phone is within Bluetooth range.
- ✧ GUI (Graphical User Interface): Interact with the user.
- ✧ Redirection: the component is for connection between home PC and foreign PC.
- ✧ Audio/Voice gateway: gateway of the audio and voice, both for input and output.

© May 2007 – Wen Hu & Yang Wu

4. IMPLEMENTATION

4.1 Evaluation of existing software

4.1.1 Existing software for connecting mobile phone to PC

As we mention in chapter 2, both EpyxMobile and Skype PTT are used for connecting mobile phone to PC, so it will be helpful if we can use any of them in our project.

However, the current version of Skype PTT no longer uses Bluetooth and we failed to get the previous version. Instead, it now uses WLAN to make connection and call forwarding work, so it seems EpyxMobile is our only choice.

When we try to run Epyx on our mobile phone according to its instruction, we find that Epyx's working principle doesn't look like what we want. According to its user guide page 4th step 1st, "Take your primary mobile phone and dial your secondary mobile phone number (the one next to your computer)." In other words; the user needs to first call his secondary mobile phone through GPRS network (or maybe 3G later), and then the secondary mobile phone forwards this call to Skype. It's quite different in regards to our requirement: 1.We need to utilize Headset Profile and Audio Gateway Profile. But Epyx doesn't involve those aspects. 2. We don't want to use any existing mobile phone's network like GPRS; we want our design to be "stand-alone" except when using Bluetooth. Epyx uses GPRS for call forwarding and even consists of two mobile phones during this procedure. That's not what we want.

So in general, we can not use any existing software for headset emulation in mobile phones.

4.1.2 Existing software for connecting Bluetooth Headset to PC

To achieve the goal of this project, we have to let mobile phones act as wireless headsets from the PCs' view. In other words, when a mobile phone is connected with a Bluetooth enabled PC, the PC regards the phone as a headset instead of a phone. As a result, the first task is to implement the connection between a Bluetooth enabled PC and a Bluetooth headset. In this part, we will employ various Bluetooth headsets, USB Bluetooth dongles, VoIP and Bluetooth software to test the communication between Bluetooth headsets and PCs.

4.1.2.1 Testing Environments

Here we use two Bluetooth headsets from different companies: the CARDO scala-500™ headset [CAR07] and Motorola HS 801.

© May 2007 – Wen Hu & Yang Wu

Cardo scala-500™ headset

- Fits all BT mobile phones
- Embedded wind blocking technology
- Exchangeable ear-loop and eye/sunglasses attachment
- 9 hours talk-time/1 week standby
- Only .58 oz (16.8g)
- All advanced functions: mute, reject, transfer, etc.

Motorola HS 801 headset

- Fits all BT mobile phones
- Only 20 grams.
- 3.5 hours talk time from a single charge.
- Functions: place, receive or end a call with the push of a button.

We have four Bluetooth USB dongles from different corporations: EPoX DGI01, EPoX BT-DG03, TBW-102UB and Trust dongles.

Except for the USB dongles, we still have to install Bluetooth software on the PCs, such as IVT BlueSoleil [IVT2007] and Bluetooth Widcomm (BTW) [BCM07], which are based on the USB adapters. IVT BlueSoleil is a Bluetooth Application Profile implementation on the Windows operating system and can be downloaded for free from BlueSoleil.com. Yet, if the Bluetooth device is not licensed with this version, BlueSoleil will be in evaluation mode and only 5MB data can be transferred.

Bluetooth Widcomm (BTW) [BCM07] from Broadcom Corporation is a communications software solution for adding Bluetooth wireless technology to Windows operating system platforms. Widcomm is one of the most popular Bluetooth drivers because of its concise user interface, easy operation and reliable system. Compared with BlueSoleil, the advantage of Widcomm is it has no limitation of 5MB data communication.

The versions of these two Bluetooth drivers used in the test are IVT BlueSoleil 2.3 VoIP and Widcomm 5.0.1.801.

The VoIP software used in the project is Skype and Voipstunt.

4.1.2.2 Connecting Bluetooth headsets to PCs

After successful installation of the IVT BlueSoleil on a PC, the main window of IVT

© May 2007 – Wen Hu & Yang Wu

BlueSoleil will appear. Then the next step is to pair the headset with the PC. If pairing is completed, then we can use the scala-500™ headset as the PC's wireless headset within a 10m range. Figure 39 presents the main window of IVT BlueSoleil when the connection between the headset and the PC is completed.

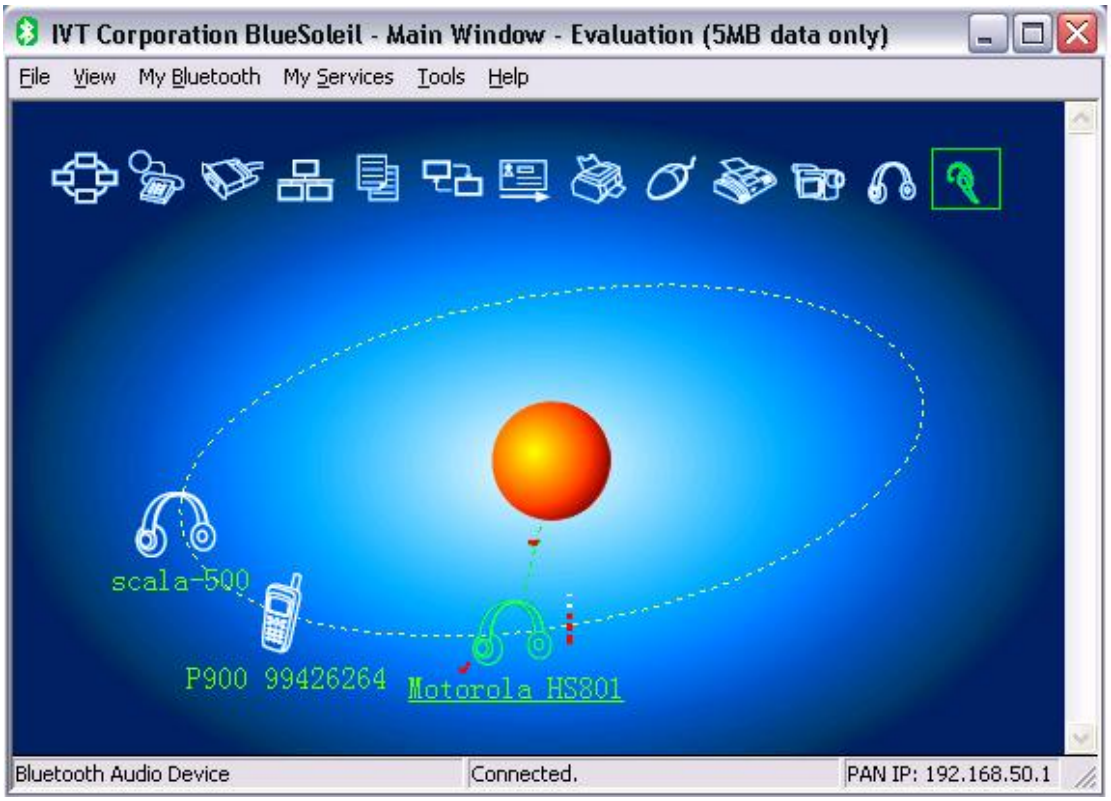

Figure 39 Main window of IVT BlueSoleil

Both the two Bluetooth headsets could be used as wireless headsets on the PC within 100 meters. It has been discovered that all three USB dongles are compatible with IVT BlueSoleil except the Trust dongles. All the dongles, however, are not licensed; the IVT BlueSoleil is in evaluation mode and is used only for a few minutes before it stops working automatically.

After installing the Widcomm software and plug in the Bluetooth dongles, it is found that three dongles, which are EPoX DGI01, EPoX BT-DG03 and Trust dongles, are compatible with it; also, the Bluetooth icon turns to blue-white which means the Bluetooth is available. However, we were unable to discover any nearby available Bluetooth devices by Trust dongles, while the other two could. Therefore, EPoX DGI01 and EPoX BT-DG03 exist for the purpose of being used as wireless headsets for PCs. Both headsets can work in the test.

4.1.2.3 Using headsets answer VoIP calls

As stated in the State-of-the-art chapter, the headset profile uses AT commands to control the voice connection. Therefore, if we want to use headsets answering VoIP

© May 2007 – Wen Hu & Yang Wu

calls, we must have an interface between VoIP software and the Bluetooth headset. This interface can recognize AT commands and control the VoIP software according to these commands.

Fortunately, we found a software called VoIP plug-in for BlueSoleil. IVT BlueSoleil 2.3 VoIP [BBV07] is specially designed for Skype. Users can answer, but cannot call, a Skype call at anytime and anywhere, even when they are listening to music via a normal Bluetooth headset.

After installation, we can use this version's BlueSoleil for answering Skype incoming calls. When an incoming VoIP call arrives, the VoIP software rings. Meanwhile, BlueSoleil auto-connects to the headset or handsfree. Users can hear the beep in the earpiece that indicates the incoming call. When pressing the answer/call key on Bluetooth headset, the ring sound stops and the call is started. It is of importance to point out that users have to manually configure the Audio-Input and Audio-Output settings of Skype Software to Bluetooth AV Audio. After all is done, users can answer the Skype call via a Bluetooth headset within 100 meters. The following figure 40 presents the windows of Skype and BlueSoleil VoIP when the connection between the Skype and the headset is setup successfully.

However, the drawback of this version's BlueSoleil is that all the USB adapters we've used are not authorized. So the data communication is limited to 5MB. The disadvantage is not only the limited data rate, but also the fact that BlueSoleil VoIP restarts after several minutes due to lack of license.

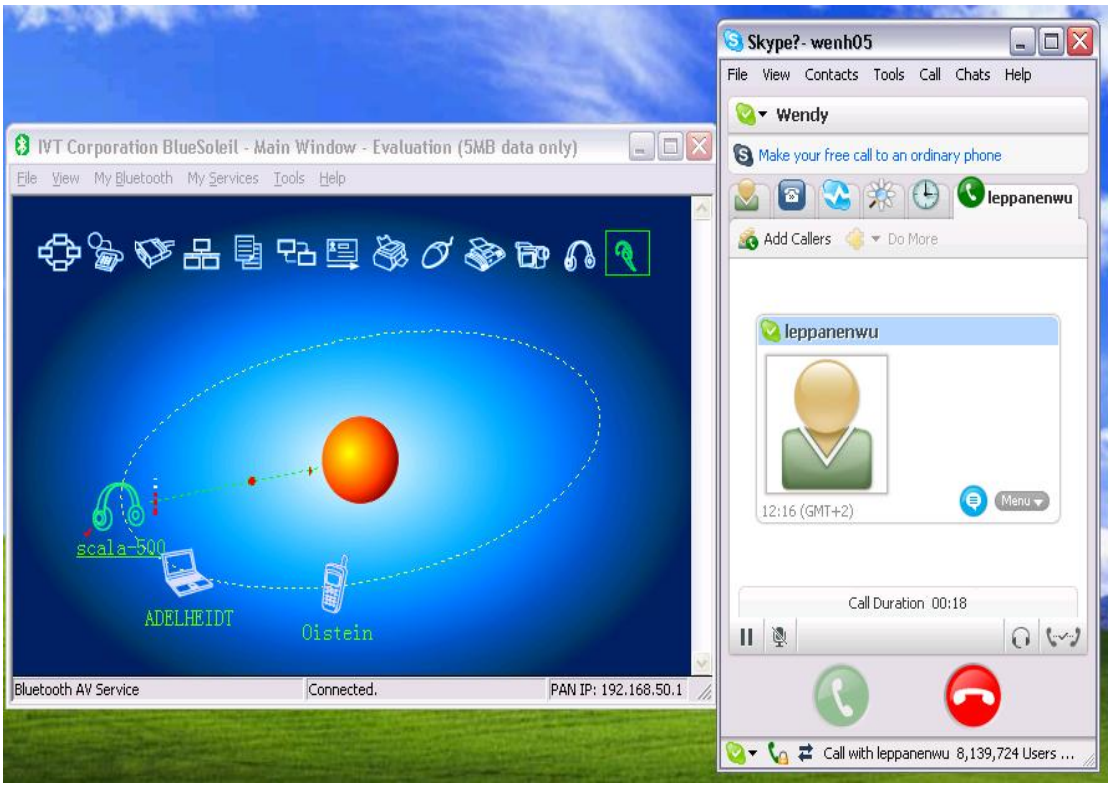

Figure 40 Windows of Skype and BlueSoleil VoIP

© May 2007 – Wen Hu & Yang Wu

The two headsets and the three dongles, except the Trust dongles, can work with Skype. It is worth noting that the headsets are able to not only end an incoming call, but also ring the previous call number by pressing a button.

For the other VoIP software, Voipstunt, we cannot just push the headset button to receive an incoming call; the reason is that IVT BlueSoleil 2.3 VoIP is specially designed for Skype. Despite this, we can still use headsets to chat with the caller through Voipstunt. One of the things that are interesting is that in this case we have to manually configure the Audio-Input and Audio-Output settings of Voipstunt Software to Bluetooth SCO Audio, instead of Bluetooth AV audio in Skype.

Because Widcomm is not designed for VoIP, the headsets are unable to answer or terminate incoming calls by pushing their button in this test. The usage of the Bluetooth headsets in this test, are as wireless PC headsets and chatting with other VoIP users within 100 meters.

4.1.3 Possibility of using existing mobile phone or PDA for headset emulation.

4.1.3.1 T-Mobile MDA
Goal: Try to install similar software like Widcomm / Blue Soleil which provide Headset Profiles within the software.

At first, we've tried to install BlueSoleil for CE version 2.0. We did it according to instructions, step by step. However, at the end of the installation, an error occurred, which stated: "unknown OS version". Afterwards the installation aborted.

Then we tried to install Widcomm Bluetooth 1.6.0. But the installation on the PDA failed too.

Finally, we used a modified version of the Widcomm Bluetooth stack from [Wid07] and it worked fine. However, this stack doesn't contain its own headset profile. Searching the services on the PC, we can't see "Headset service" in its provided service list.

Conclusion: we can not use existing software of Bluetooth stack to emulate the Headset Profile on a PDA.

4.1.3.2 Mobile Phone: SonyEricsson P900, SonyEricsson P990i, Motorola 768i.

Goal: Testing whether these mobile phones come with their own Headset Profiles already inside.

© May 2007 – Wen Hu & Yang Wu

According to their instructions, both P900 and P990i should "support" Headset/Handsfree Profile. However, both BlueSoleil and Widcomm can not find "Headset service" in their provided service lists. On the other hand, both HS801 and scala-500 can work with those mobile phone well.

Conclusion: "Support" does not mean "Implement". There is no existing mobile phone implement Headset Profile as far as we can see.

4.2 Application design

As we have shown above, we can not use any existing software for our first step – headset emulator. To achieve the goal of the project, we originally planned to implement two small programs on both the PC and mobile phone sides in order to make the PC be regarded as a Bluetooth server and phone as a Bluetooth client, and then establish a connection between the server and client. However, time is too limited for us to implement all these features. Therefore, we now plan to focus first on finishing the "Headset Emulator" part. If time allows, we'll try to consider other features.

Since we have successfully tested the connection between Bluetooth headsets and PCs as stated in section 4.1, and there is no existing software that does this "Headset Emulator" work, we plan to use IVT BlueSoleil in the PC side and also make a program on the mobile phone side.

In section 4.1 we have tested IVT BlueSoleil to search for nearby headset devices and saw that the Bluetooth headset has the ability to work as the wireless headset of the PCs within the Bluetooth range. The key for a successful connection is that the Audio/Voice gateway provided by IVT BlueSoleil is able to realize the function of the headset service which is already implemented in a Bluetooth headset. For this reason, we can hear the sound from the PC via this headset. Since our project is to establish a connection between the PC and mobile phones, if we make the mobile phone work as the Bluetooth headset, then the connection would be setup. However, normal mobile phones don't implement the headset profile. If we install IVT BlueSoleil on the PC side, our task is to implement the headset profile in a mobile phone in order to make the phone own the ability of a headset and finally realize the connection with IVT BlueSoleil.

When IVT BlueSoleil connects with the mobile phone's headset profile, IVT BlueSoleil, which acts as AG, will send the AT command RING to alert the mobile phone side after the ACL is established. The RING may be repeated for as long as the connection establishment is pending. The SCO link establishment can take place after the mobile phone user accepts the call by pressing a button on the phone.

© May 2007 – Wen Hu & Yang Wu

4.3 Developing environment

4.3.1 Developing language

When we try to make this headset emulator, there are two languages we can choose: C/C++ or Java. After deep consideration, we decide to use Java. There are two main reasons for that [BFJ03]:

- ✧ Java Bluetooth API is independent of the stack and radio;
- ✧ Java Bluetooth API is a standardized Bluetooth API.

4.3.1.1 Independence of Java Bluetooth API

There are two key advantages to using the Java Bluetooth API. The first one is that the API is independent of the stack and the Bluetooth hardware. It is known that Java code can be run on basically any hardware platform and on any operation system with little or no modification. So the Java Bluetooth API gives the users the ability to write applications without any knowledge of the underlying Bluetooth hardware or stack.

4.3.1.2 Java Bluetooth API is a standardized Bluetooth API

The second advantage is that it is the only standardized Bluetooth API. There is no standard for a C/C++-based Bluetooth SDK. Take “Service record” as an example; vendor A may name it as sddb.add (), while vendor B may write sddb.insert (). The two names for the same function are different. Therefore, it is necessary to rewrite the Bluetooth application and/or change its functionality because of it. JSR-82 is the official Java API for Bluetooth; all vendors who implement the standard must include a core set of layer and profiles in their Bluetooth SDK. In other words, if a vendor tries to modify the Service record, it has to use sddb.insert (). In this way, the names of functions are unified and the Java code can be re-used under different environments.

4.3.2 IDE (Integrated Development Environment) and Smart Phone.

During this project, we use Eclipse 3.2 with EclipseME plug in for developing J2ME application. To test our application on a PC, we use Sun WTK 2.2. To test our application on a mobile phone, we use Sony Ericsson P900.

4.3.3 JSR (Java Specification Request) 82

JABWT was defined by a Java Community Process expert group JSR-82 [JSR82]. The JSR-82 is the Java API for Bluetooth wireless technology. The specification standardizes a set of Java APIs to allow Java-enabled devices to integrate into a Bluetooth environment. The following application function could be implemented by JSR-82:

© May 2007 – Wen Hu & Yang Wu

- Estimate and detect attribute of Bluetooth device
- Discover nearby Bluetooth device within Bluetooth range
- Search service in remote Bluetooth device
- Establish the connection between remote Bluetooth server and Bluetooth client
- Provide service in Bluetooth server for request from Bluetooth client

There are two packages for JSR-82: javax.bluetooth and javax.obex. And during our implementation, only javax.bluetooth will be used.

However, since we mentioned before, JSR82 doesn't provide an API for the lowest two layers: baseband and radio. Also, according to Figure 6 in chapter 2, audio connection is established directly on baseband layer, which is unlike other connections, established on L2CAP. And according to [BAP04], "JABWT (Java APIs for Bluetooth Wireless Technologies) does not provide APIs for Audio/Voice transmissions over voice channel," as shown in the figure below:

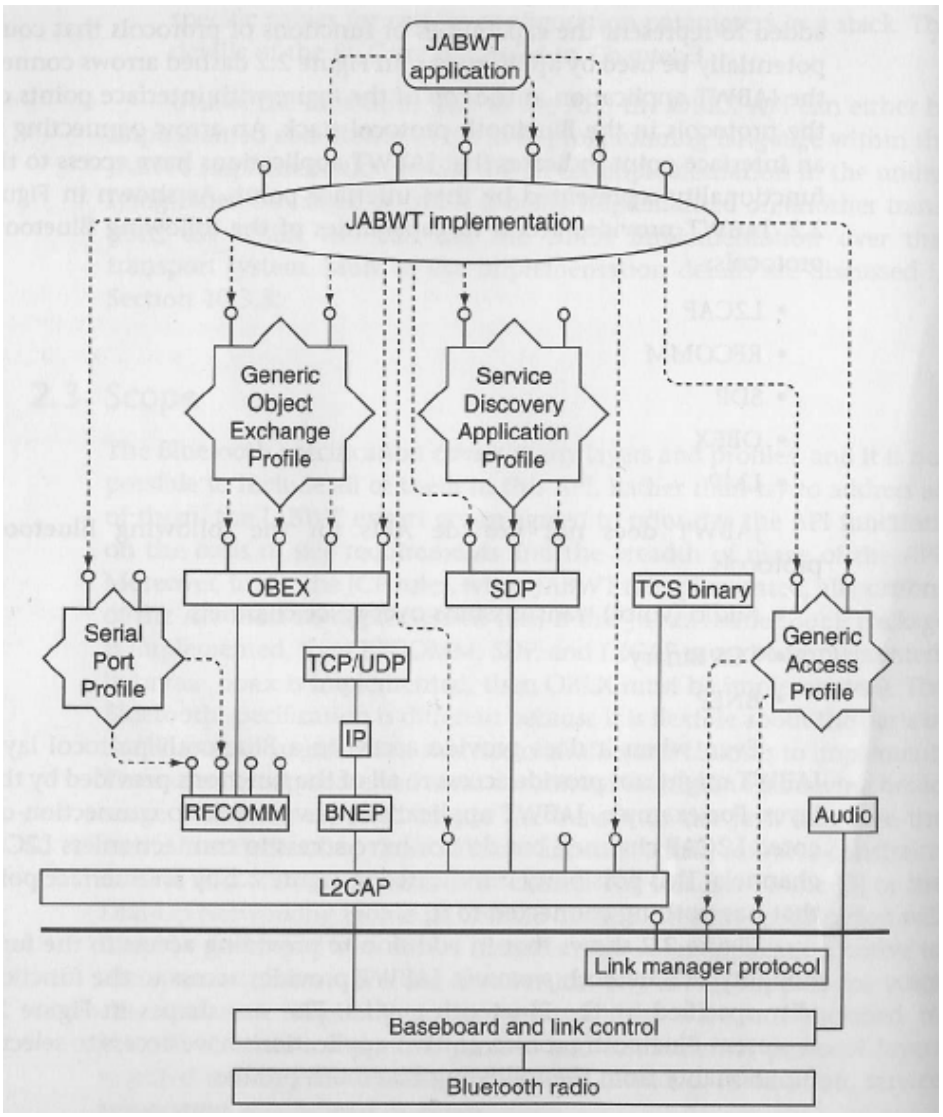

Figure 41 JABWT architecture

© May 2007 – Wen Hu & Yang Wu

So it's obvious that the most painful work lies in how to make an Audio connection using JABWT.

4.4 Design details

Fortunately, we know that the communication between the headset profile and the audio gateway profile is a Serial Port based connection. This Serial Port profile is covered in JABWT. What is more important is that according to [Hea01], we know that the commands used between HSP and A/VGP are AT commands. So if we can establish a SP connection, and catch the AT command sent by A/V GP, we should be able to reply those command in AT format too, which will probably makes A/V GP believe it is talking with a normal HSP.

There are three major difficulties here:
1. Convince A/VGP that mobile phone has a HSP and provide Headset service.
2. Handle with those commands transferred between PC and mobile phone.
3. Use mobile phone to discovery A/VGP service.

And the flow chart should be like this:

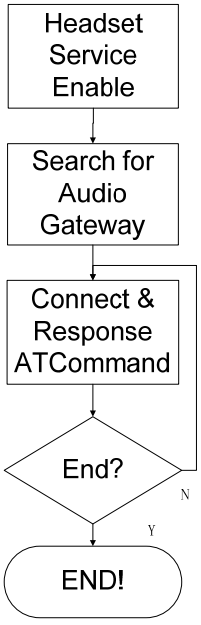

Figure 42 flow chart

4.4.1 Add Headset service to Service Record

According to figure below [BAP04], this process should have 3 steps:
1. Create service record.
2. Modify service record attributes.
3. Add it to SDDB.

© May 2007 – Wen Hu & Yang Wu

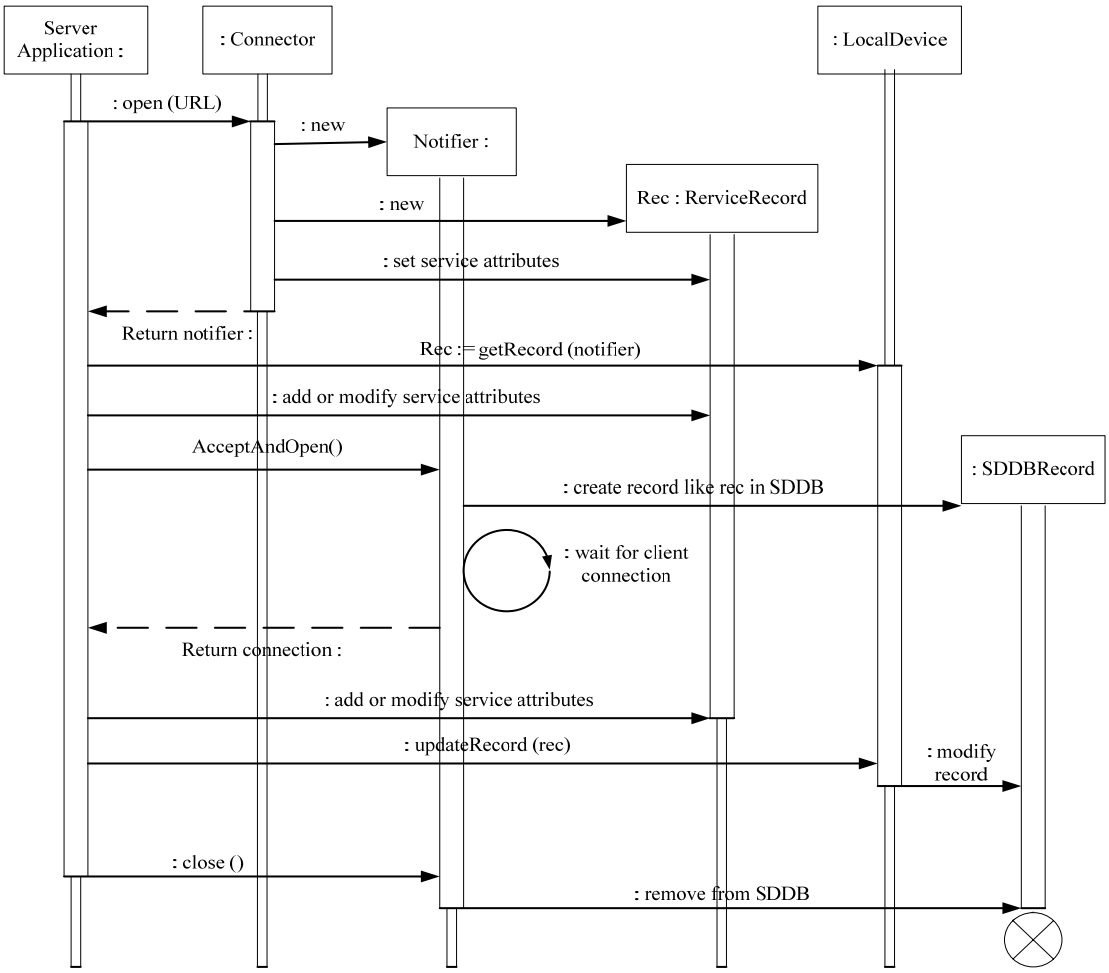

Figure 43 Life cycle of a service record

4.4.1.1 Create service record

First of all, we need to specify a new service. This service can be any kind of service because we will modify those attributes to meet Headset Profile's requirements after all. So right now, we just need an object that can be modified. In JABWT, this is done by first declaring a `StreamConnectionNotifier` object. We simply use `Connector.open()` to create the object called "notifier". And after that, to get access to this notifier's service record, we need get access to `LocalDevice` and use this class's function "`getRecord(notifier)`" to retrieve this `ServiceRecord` object "record". Once we get that object, we can do what ever we want to modify record attributes.

Codes are described below:

```
notifier = (StreamConnectionNotifier)
Connector.open("btspp://"+"localhost:123456789ABCDE;name=Headset");
LocalDevice local = LocalDevice.getLocalDevice();
ServiceRecord record = local.getRecord(notifier);
```

`btspp:// localhost:123456789ABCDE` means this is an Serial Port based service,

© May 2007 – Wen Hu & Yang Wu

which is already specified in Bluetooth Specification. The number string here, "123456789ABCDE", means nothing. We will modify that later.

4.4.1.2 Modify record's attributes

The service record for Headset is shown in Figure 51 in Appendix A. It is worth noting that in [Hea01], it says "This profile defines the following service records for the headset and the audio gateway respectively." However, only this headset part's service record should be our concern.

The value of the Item, like "ServiceClassIDList", "L2CAP", should be present in UUID format. Those values can be found in [BAN01]. Most of the value store in object called "DataElement".

After we store those values into the corresponding DataElement object, we need to insert them into the service record "record". There is an existing method in class "ServiceRecord" called setAttributeValue() which can do this job.

Codes here are:

```
DataElement ServiceClassIDList = new DataElement(DataElement.DATSEQ);
DataElement ServiceClass0 = new DataElement(DataElement.UUID,new
UUID(0x1108));
DataElement ServiceClass1 = new DataElement(DataElement.UUID,new
UUID(0x1203));
ServiceClassIDList.insertElementAt(ServiceClass1, 0);
ServiceClassIDList.insertElementAt(ServiceClass0, 0);
record.setAttributeValue(0x0001, ServiceClassIDList);
```

Codes above for the ServiceClassIDList modification. 0x1108 indicates it's "Headset"; 0x1203 indicates it's "Generic Audio". And then, insert these two data into Service Class ID List one by one. Finally, insert the new "ServiceClassIDList" object to, 0x0001, where it belongs.

```
DataElement BluetoothProfileDescriptorList = new
DataElement(DataElement.DATSEQ);
DataElement Profile0 = new DataElement(DataElement.DATSEQ);
DataElement HSP = new DataElement(DataElement.UUID,new UUID(0x1108));
DataElement Param0 = new DataElement(DataElement.U_INT_1,1);
Profile0.insertElementAt(Param0, 0);
Profile0.insertElementAt(HSP, 0);
BluetoothProfileDescriptorList.insertElementAt(Profile0, 0);
record.setAttributeValue(0x0009, BluetoothProfileDescriptorList);
```

Codes above are used for BluetoothProfileDescriptorList. Although the codes look similar to the upper one, what's going on under these codes are different. The

© May 2007 – Wen Hu & Yang Wu

"dummy" object we created before does not have this BluetoothProfileDescriptorList item. Function "setAttributeValue" here will create a new item and places it at "0x0009", where in Bluetooth Specification, is for Bluetooth profile's description. The different data type: "*DATSEQ, UUID, U_INT_1*" are used also according to the specification.

Also things to mention here are we did not modify all the attributes according to Service Record Attributes in Appendix. Item "ProtocolDescriptorList" are already existed in the service we created (`btspp:// localhost:123456789ABCDE`) and the contents are also same. Item "ServiceName" and "Remote audio volume control" is optional item. Therefore, we would like firstly ignore them and maybe add them on later work.

4.4.1.3 Add the modified service record to SDDB

Just like what we've shown in figure 43, this record can be add to SDDB only after `StreamConnectionNotifier` 's method "`acceptAndOpen()`" is called. According to JSR82's specification, this action will be automatically executed once that method is called. However, there is another optional way can achieve the same effect. A method called "`updateRecord()`" in class `LocalDevice` is used to make those changed be effected on the SDDB side. But according to JSR82, this method only affects items which already exist in the old record. So in other words, if there is no "BluetoothProfileDiscriptorList" item in the old service record, we cannot add it to SDDB using "`updateRecord()`". That is to say, the first time we run this application, we have to wait for "`acceptAndOpen()`" to be executed in order to add this modified record to SDDB.

However, the code line we actually used here is still:

```
local.updateRecord(record);
```

details of this can be found in discussion part.

After all these three steps have been done, the mobile phone now should appear as and provide headset service as a normal headset.

4.4.2 Handle with those commands transferred between PC and mobile phone

Once we create a service record in SDDB which provides headset service, it's time for us to use this headset service from the PC side. The so-called "use" is actually to implement the function specified in [Hea01]. There are 4 types of formats for commands, which are shown below:

- ✧ Command from HS to AG. This kind of command must have a "AT" prefix initially:
  AT <cmd> = <value> <cr>

© May 2007 – Wen Hu & Yang Wu

- ✧ Command from AG to HS as reply “success”:
  <cr> <lf> OK <cr> <lf>
- ✧ Command from AG to HS as reply “fail”:
  <cr> <lf> ERROR <cr> <lf>
- ✧ Unsolicited command from AG to HS:
  <cr> <lf> <result code> <cr> <lf>

The “cmd” in the first type are sown in Figure 52 in Appendix A.

So if we want to express a “headset button pressed” action, the command send to AG should be AT +CKPD = 200 <cr>.

The “result code” in the last type of command is presented in Figure 53 in Appendix A.

It is worth noting that there is another unsolicited result code that can be used in the command from AG to HS: “RING”. The usage of this one is to indicate the incoming calls.

So for example, if AG wants to gain the speaker of HS, command “<cr> <lf> +VGM = 13 <cr> <lf>” is sent. If AG wants to inform HS that there is an incoming call, command “<cr> <lf> RING <cr> <lf>” is sent.

In the mobile phone’s application, we need to add methods for sending those commands to different “`Command`” object’s “`commandAction()`” so that each time this command object is selected, corresponding AT commands will be sent.

Also, commands are not only sent from HS but should also be received from AG. We need to handle those incoming commands. As [Hea01] specified, only “RING” is mandatory, and we are not sure about how to implement those volume adjusting parts. Therefore, we plan to skip handling these commands right now. Maybe this work can be done in future.

To retrieve this “RING” command, we plan to use an `InputStream` object. By using this class’s `read()` method, we can get the data sending from AG to HS, and then, convert it to String or other format to see whether it’s “RING”. Once this “RING” is detected, we can inform the user by showing a message or via other ways, and consequently let him or her to make a decision.

Actually, when we try to get the AT commands, we’ve tried 4 different ways:

```
A:  conn = (CommConnection) notifier.acceptAndOpen();
    InputStream in = conn.openInputStream();

B:  conn = (CommConnection) notifier.acceptAndOpen();
    DataInputStream in = conn.openDataInputStream();
```

© May 2007 – Wen Hu & Yang Wu

```
C:  conn = (StreamConnection) notifier.acceptAndOpen();
    InputStream in = conn.openInputStream();

D:  conn = (StreamConnection) notifier.acceptAndOpen();
    DataInputStream in = conn.openDataInputStream();
```

In type C and D, the connection type we used is StreamConnection, which is the default type of JSR82's typical connection. In type A and B, the connection type we used is CommConnection, which is the inherited from StreamConnection and is the specified type for serial port's connection. The reason we use this one is that the audio link should be established upon serial port connection according to Bluetooth Headset Profile's specification.

In type A and C, we use default input stream type while in type B and D, we use the other input stream for MIDP: DataInputStream. The reason we choose these two type of input is that there are the only supported input stream in MIDP for StreamConnection and CommConnection.

4.4.3 Service Discovery

When the first two steps are finished, it should be possible for the user to use the PC to find our headset service-provided mobile phone and make a connection. However, that is not what we want in our architecture. In our design, this service discovery process should be initiated by the mobile phone because it should be mobile phone's function to find out which PC provide A/V gateway service and consequently connect to it. Device discovery and service discovery should be covered in our application.

There is no doubt that service discovery should be executed before device discovery. We plan to use following procedure for this.

© May 2007 – Wen Hu & Yang Wu

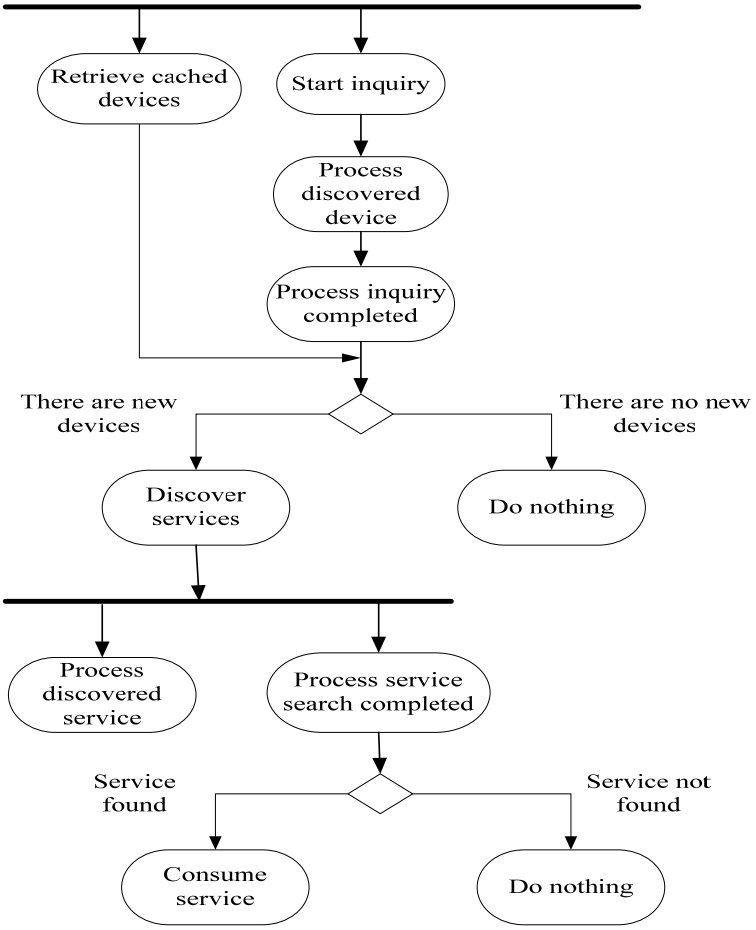

Figure 44 Devices and Service Discovery

This process is typically managed by a `DiscoveryAgent` object, and this object should also implement the `DiscoveryListener` interface. We will just present three major methods there. The details for other functions we have used can be found in [JSR82].

- `startInquiry()`:searches for devices with the specified inquiry access code.
- `searchServices()`:searches for certain service on a specified remote Bluetooth device.
- `selectService()`:selects a service that contains uuid in its service record.

Codes detail:

- Compare with Figure 44, the codes for "Retrieve cached devices" block are:

```
private void addDevice(){
        RemoteDevice[] list =
agent.retrieveDevices(DiscoveryAgent.PREKNOWN);
        if(list!=null){
            for(int i = 0; i<list.length; i++){
                String address = list[i].getBluetoothAddress();
                deviceList.insert(0,address,null);
                deviceVector.insertElementAt(list[i], 0);
```

© May 2007 – Wen Hu & Yang Wu

```
            }
        }
        list = agent.retrieveDevices(DiscoveryAgent.CACHED);
        if(list!=null){
            for(int i = 0; i<list.length; i++){
                String address = list[i].getBluetoothAddress();
                deviceList.insert(0,address,null);
                deviceVector.insertElementAt(list[i], 0);
            }
        }
    }
```

- Codes for "Start inquiry" block are: (build-in function. GIAC means general inquiry access control, so this device can find other device and in the meanwhile, it can be found by other device as well.)

```
agent.startInquiry(DiscoveryAgent.GIAC, this);
```

- Codes for "Process discovered devices" block are: (retrieve remote/discovered device's friendly name first, and the try to find out what kind of device it is. Function "getFriendlyName" is a build-in function. But function "getDeviceClasss" is writen by me. However, since codes for the latter function are too long, we didn't present it here.)

```
public void deviceDiscovered(RemoteDevice device, DeviceClass cod) {
        // TODO Auto-generated method stub
        try {
            String friendlyName = device.getFriendlyName(false);
            deviceList.insert(0, friendlyName+";"+getDeviceClass(cod),
null);
            Display.getDisplay(infoM).setCurrent(deviceList);
            deviceVector.insertElementAt(device, 0);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            infoM.getForm().append(e.getMessage());
            Display.getDisplay(infoM).setCurrent(infoM.getForm());
        }
    }
```

- Codes for "Process inquiry completed" block are: (Different re-action according to different type returned.)

```
public void inquiryCompleted(int type) {
        // TODO Auto-generated method stub
        Alert alert = null;
        isInInquiry = false;
        if(type != DiscoveryListener.INQUIRY_ERROR){//NB! this is only
```

© May 2007 – Wen Hu & Yang Wu

```
for P900, normally it should be "type !=
DiscoveryListener.INQUIRY_COMPLETED"
            if (type == DiscoveryListener.INQUIRY_TERMINATED) {
                startServiceSearch();
                return;
            }
            else{
            alert = new Alert("Bluetooth error", "The inquiry didn't
complete:", null, AlertType.ERROR);
            deviceList.removeCommand(abort);
            deviceList.addCommand(scan);
            }
        }
        else{
            alert = new Alert("Inquiry completed", "Inquiry completed",
null, AlertType.INFO);
            deviceList.removeCommand(abort);
            deviceList.addCommand(scan);
        }
        alert.setTimeout(Alert.FOREVER);
        Display.getDisplay(infoM).setCurrent(alert);
    }
```

- Codes for “Discover service” block are: (UUID 0x100 indicates that the service we’re interested in must have L2CAP protocol item in its service record since most of Bluetooth services are built upon this; Attributes 0x0009 and 0x0100 indicate we want to retrieve BluetoothProfileDescriptorList and ServiceName for the found services. 0x100, 0x0009 and 0x0100 are also used according to Bluetooth sepecification)

```
private void startServiceSearch(){
        serviceRecordVector = new Vector();
        try{
            UUID[] uuidList = new UUID[1];
            uuidList[0] = new UUID(0x100);
            int[] attrList = new int[2];
            attrList[0] = 0x0009;
            attrList[1] = 0x0100;

            int index = deviceList.getSelectedIndex();
            RemoteDevice remoto =
(RemoteDevice)deviceVector.elementAt(index);

            transID = agent.searchServices(attrList, uuidList, remoto,
this);
```

© May 2007 – Wen Hu & Yang Wu

```
        }catch (BluetoothStateException e){
            Alert error = new Alert("Error","Unable to start the service
serarch("+e.getMessage()+")",null,AlertType.ERROR);
            error.setTimeout(Alert.FOREVER);
            Display.getDisplay(infoM).setCurrent(error, deviceList);
        }
    }
```

- Codes for "Process discovered services" block are: (retrieve name element, which is "0x100" according to Bluetooth specification, and print it on screen.)

```
public void servicesDiscovered(int transID, ServiceRecord[] record) {
        // TODO Auto-generated method stub
        for (int i = 0; i<record.length; i++){
            DataElement nameElement =
(DataElement)record[i].getAttributeValue(0x100);

    if((nameElement!=null)&nameElement.getDataType()==DataElement.STR
ING){
                String name = (String)nameElement.getValue();
                serviceList.insert(0, name, null);
                serviceRecordVector.insertElementAt(record[i], 0);
            }
        }
        Display.getDisplay(infoM).setCurrent(serviceList);
    }
```

- Codes for "Process service search complete" block are: (Similar to "Process devices search complete" block. Different re-action according to different type returned by system.)

```
public void serviceSearchCompleted(int transID, int type) {
        // TODO Auto-generated method stub
        Alert dialog = null;
        if (type != DiscoveryListener.SERVICE_SEARCH_COMPLETED){
            dialog = new Alert("Bluetooth Error","the service search
failed to complete normally"+type,null,AlertType.ERROR);
        }
        else{
            dialog = new Alert("Service serach completed","the service
search complete normally",null,AlertType.INFO);
        }
        dialog.setTimeout(Alert.FOREVER);
        Display.getDisplay(infoM).setCurrent(dialog);
    }
```

© May 2007 – Wen Hu & Yang Wu

- Codes for "Consume service" block are: (Once I got the service I selected, firstly we will get the corresponding service record, and then, get its ServiceClassIDList. After that, we'll try to get the service descript in this ServiceClassIDList. Hopefully it should be a Headset Audio Gateway. Finally, use selectService() function to select this service and got the connection string as returned result.)

```
private Connection connectToService(){
        int index = serviceList.getSelectedIndex();
        ServiceRecord sr =
(ServiceRecord)serviceRecordVector.elementAt(index);
        Enumeration em = null;
        DataElement de = null;
        String conn = null;
        try{//get ServiceClassIDList
            em = (Enumeration)sr.getAttributeValue(0x0001).getValue();
        }catch (ClassCastException e){
            serviceList.append("em"+e.getMessage(),null);
        }
        try{//get Headset Audio Gateway
            de = (DataElement)em.nextElement();
        }catch (ClassCastException e){
            serviceList.append("de"+e.getMessage(),null);
        }
        UUID AGuuid = (UUID)de.getValue();
        serviceList.append(AGuuid.toString(), null);
        try {
            conn = agent.selectService(AGuuid,
ServiceRecord.AUTHENTICATE_NOENCRYPT, true);
            serviceList.append(conn, null);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            serviceList.append("Can not get connection URL", null);
        }
        if (conn != null){
            try {
                return Connector.open(conn);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                serviceList.append("Unable to connect to AudioGateway
Server", null);
            }
        }
        return null;
    }
```

© May 2007 – Wen Hu & Yang Wu

5. DISCUSSION AND EVALUATION

5.1 Headset Emulator

There are some existing software and hardware solutions for the headset emulator, which are stated in Chapter 2. By using these products, users would be able to receive Internet calls via their PC Skype software on their mobile phones as long as they stay within 20-100 meters from their PCs. However, all these solutions are only designed for Skype and users have to be concerned with compatibility issues when they use this kind of product. Additionally, the existing software products seem to be unstable and lack widespread implementation. Though the hardware products, such as Bluetooth Sky Phone, are able to connect with Skype calls within the Bluetooth range, they are not suitable for modern users because this kind of phone has only a few functions in comparison to smart phones. There are also further limitations. Once the users are far away from their PCs, the connection will be terminated.

Consequently, the existing software and hardware solutions don't meet the needs of our project. We designed a prototype that is compatible with any type of telephony software and embedded in any kind of mobile phone.

5.2 Security

In chapter 2, we firstly presented an in-depth research about Bluetooth's built-in security scheme. The shortcoming is obviously – it is vulnerable to eavesdropping. We've also given a reference about how to crack a Bluetooth PIN. After presenting Bluetooth's build-in PSK security solution, we began to introduce the PKI solution. Finally, we chose one of the PKI solutions for our architecture.

The reason Bluetooth uses PSK as its build-in security scheme is due to its Ad-hoc nature in most circumstances. However, in our project, the PCs are already connected to a fixed infrastructure. The situation here is similar to WLAN, so we can use PKI to improve security performance.

The advantages of using PKI compare with Bluetooth build-in security scheme are: 1. easy to manage and distribute Key/Certification. 2. User manfully configuration in PC side is not necessary.

So far as we can see, the only drawback for using PKI in our design is the cost factor. To get a new Authentication Server for only this "Internet Telephony in Ubiquitous Computing Environments" usage is definitely a waste. However, we assume that maybe we can try to utilize WLAN's existing RADIUS server. Anyway, we have to

© May 2007 – Wen Hu & Yang Wu

clarify that details about this Authentication Server should not be our project's concern.

5.3 Service Discovery

In this project, we finally used Bluetooth SDP to implement service discovery. The drawback of Bluetooth SDP is the device cannot use the built-in Bluetooth SDP when it is pairing; Konark and DNS-SD are also options for service discovery which discussed in Chapter 2. They are independent of the Bluetooth stack, so if we use these for service discovery, the above problem will be solved.

But DNS-SD is not reachable for our project. All nodes must have network access for DNS-SD; nonetheless, in our project we cannot guarantee the mobile phone is connected to a network. And about Konark, we don't think it is a competitive solution compare with Bluetooth SDP. Because it also involved with Bluetooth traffic during service discovery, which as we explained before, is unable while Bluetooth headset profile and Bluetooth audio gateway profile are working.

5.4 Redirection

There are two different methods of solving Redirection problem, which are just like the two modes we introduced in chapter 2.4.4: tell the calling party about target's address or work as a middleware and help forward the calls.

We believe that the second one is better because on one hand, the user's office PC and the one mobile now connected can handle all the details without bothering the calling party; the transparency is good. On the other hand, since we don't have more requirements about the architecture of the call's source, compatibility of our design can be guaranteed. As a result of this, we used the second one to handle the Redirection.

5.5 Fast handover

In chapter 2, we explained the existing state of art solution to Bluetooth handover. In the architecture design part, we didn't mention anything about this because we don't think we can give a complete architecture regarding with this fast handover issue due to this problem's complexity. Although we didn't discuss this problem in our architecture chapter, we can still propose a direction to those whom this may concern:

In general, there are two major factor can cause the delay during a handover procedure: service discovery and security issue. So far as we can see, there is no existing complete solution that can solve this problem in Bluetooth perfectly.

© May 2007 – Wen Hu & Yang Wu

However, according to the methods introduced in chapter 2, we have a suggestion for fast-handover in Bluetooth: use some location-aware mechanism so that the current PC with which the user is connected to can predict the next PC the user intends/has to connect to. After that, this PC sends all information necessary for build up of a Bluetooth connection with that PC to the users. This would probably be the service's UUID, which is the primary parameter for building Bluetooth connection. In that case, service discovery problem can be solved.

About the other problem, security issues, the WLAN solution, especially 802.11r, can be used for dealing with security issues. If PKI is used, the authentication between mobile phone and the next PC can be done following the procedure we are given in chapter 3.2.2. The mobile phone still identifies itself via the current PC to Authentication Server, and the next PC identifies itself to the Authentication Server directly. After both devices are authorized, the mobile phone and the next PC can have each other's public key.

In that case, the necessary information for build up next connection and public key for encrypted communication can be gained while the mobile phone is still connected to the current PC. Therefore, we believe that it can solve, or at least be helpful in solving, this fast handover problem.

5.6 Prototype's implementation

Before actually make my codes can be run in the mobile phone, we have to export them out and copy them in to mobile phone.

To achieve this, right click the project in Eclipse and select J2ME->Create Package.

© May 2007 – Wen Hu & Yang Wu

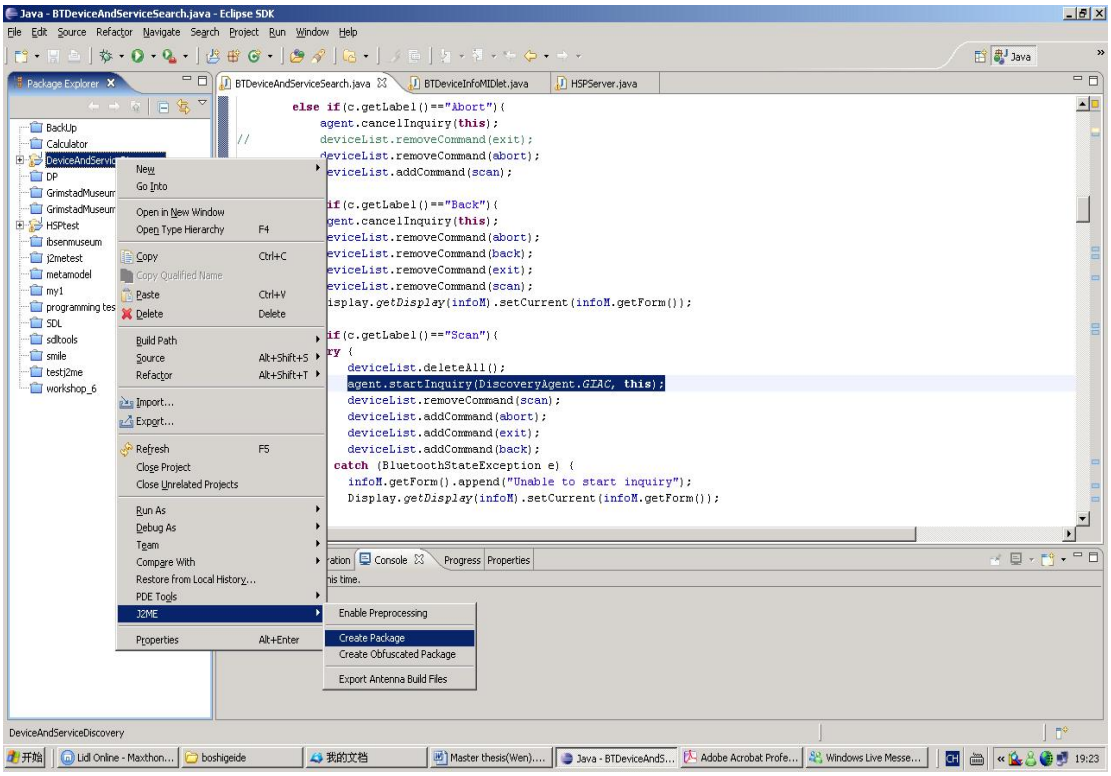

Figure 45 main window of Eclipse

Then, there will be two file in the "deployed" folder in the corresponding folder under the Eclipse work space. However, after many time's failure, we found that copy these to file directly to mobile phone is not correct, though many books don't mention it. we have to manually add one more line, "MIDlet-1: BTDeviceInfoMIDlet,,no.hia.wen.BTDeviceInfoMIDlet" in JAD file which likes picture blew.

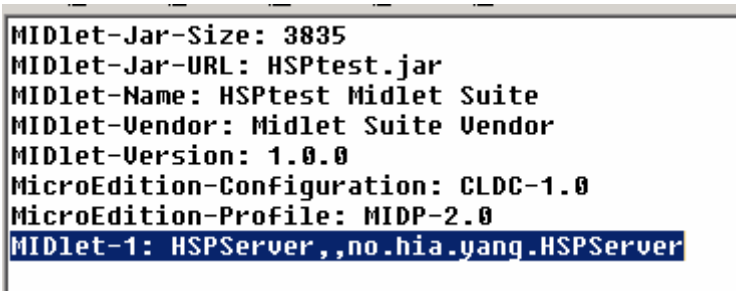

Figure 46 JAD file

Only in that case, we can install my application on mobile phone successfully by sending these JAD and JAR files to it and run JAD file after that.

During chapter 4, we've introduced three parts of our prototype, which covers our sub-problem 1, headset emulator, and sub-problem 2, service discovery. Sub-problem 5, fast handover is not inside of our concern due to the reason we've outlined above. Sub-problem 3 and 4 are not included in our prototype due to time reason.

To evaluate our prototype, let's first have a look at the headset emulation part. We've

© May 2007 – Wen Hu & Yang Wu

successfully added the headset service to SDDB which means, PC can find “headset service” provide by mobile phone after our application is executed. Figures blew illustrate the situation before (A) and after (B) our application being executed.

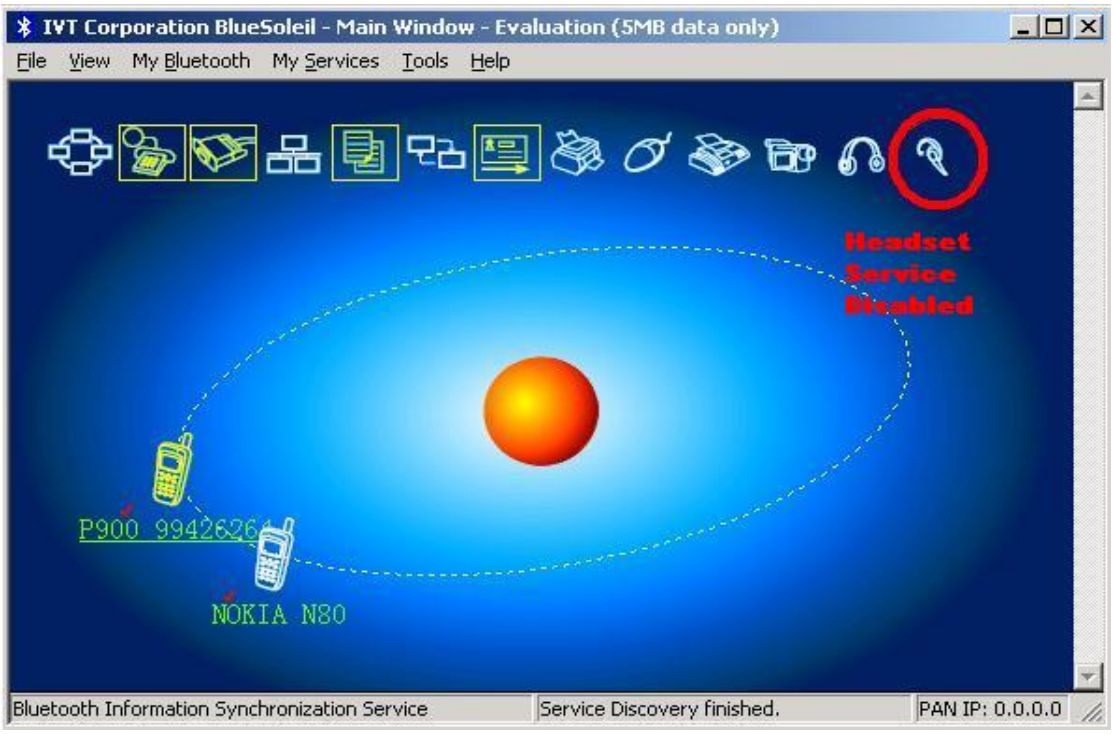

Figure 47 Headset Service Emulator (A)

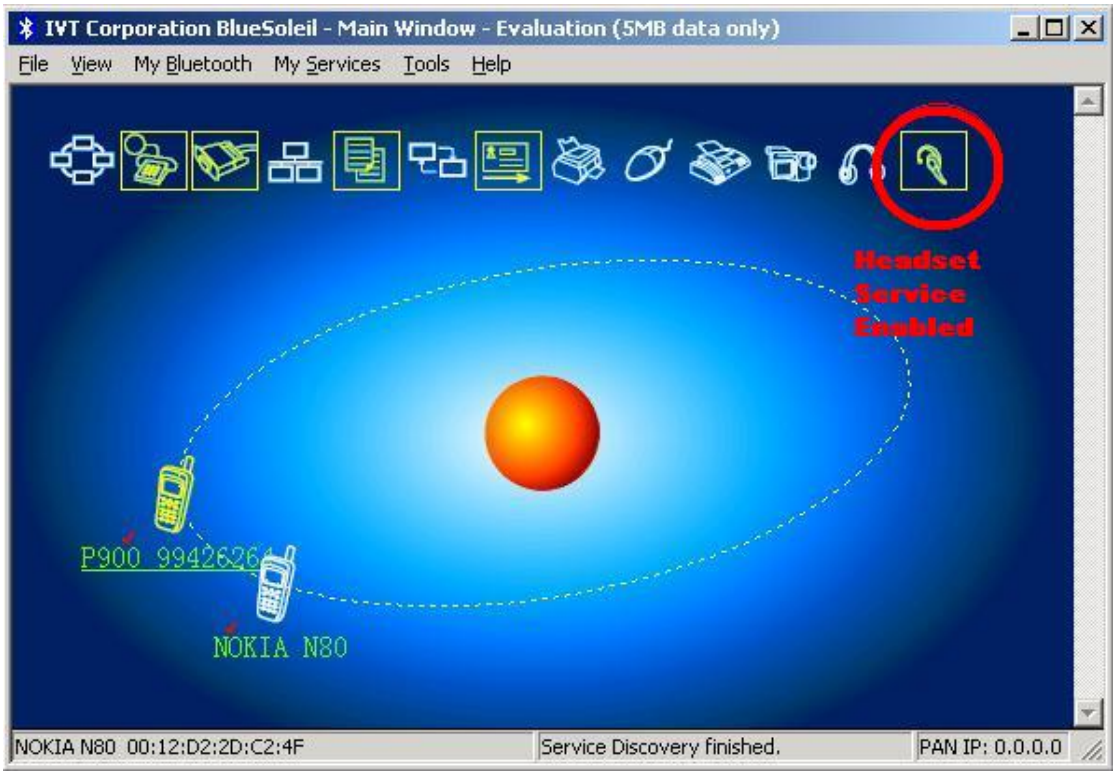

Figure 48 Headset Service Emulator (B)

© May 2007 – Wen Hu & Yang Wu

So in general, the result of our prototype's first part is very good.

Then, about the second part – to handle the AT commands send from PC to mobile phone. What we planned is to get those commands by IOstream. And then, response will be given respectively. However, things are not going as we expected. We've tried many kind of input stream listed as A,B,C and D in chapter 4.4.2, but none of them can retrieve the any single bit of data. So we try to find out whether the acceptAndOpen() method is called. Therefore, we've added a simple line of code to detect this:

```
conn = (StreamConnection) notifier.acceptAndOpen();
msgForm.append("AGP tries to connect...");
```

However, the result is that this string output is never shown on the screen. We can only suspect that once Audio Gateway profile tries to connect to mobile phone, it does not use normal connection way like typical JSR-82 program. In other word, it does not notify "StreamConnectionNotifier". Therefore, we can not response to AGP's connection request and consequently, can not deal with AT commands.

Although we didn't make this AT commands part to work, we did have another discovery. In chapter 4, we state that only acceptAndOpen() can be used for update service record. But since we didn't make it to be executed, how can this headset service be added into SDDB?

We find that in fact, updateRecord() can be capable for this updating service record task. It can work for the first-time-run or creation of new service though JSR82 claims it can not. We think that is an important find to those who will keep on working following our direction.

Finally, the last part of our prototype is service discovery. This procedure requires devices discovery firstly. During our Implementation part, we mainly focused on service discovery. But when start to run our code, we find that device discovery is as important as service discovery. Only presenting the name or the address of a found Bluetooth device is not enough. Because in our user scenario, mobile user will only be interested in devices whose type is PC or laptop.

Therefore, what we've actually implemented is firstly search devices and present their types, then search services for a selected device. Pictures blew show that our code running as successful as we expected.

© May 2007 – Wen Hu & Yang Wu

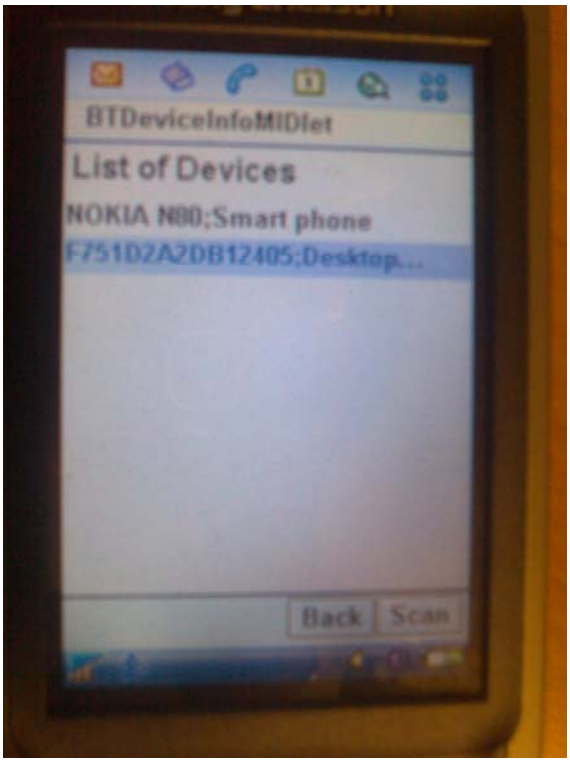

Figure 49 Devices Discovery

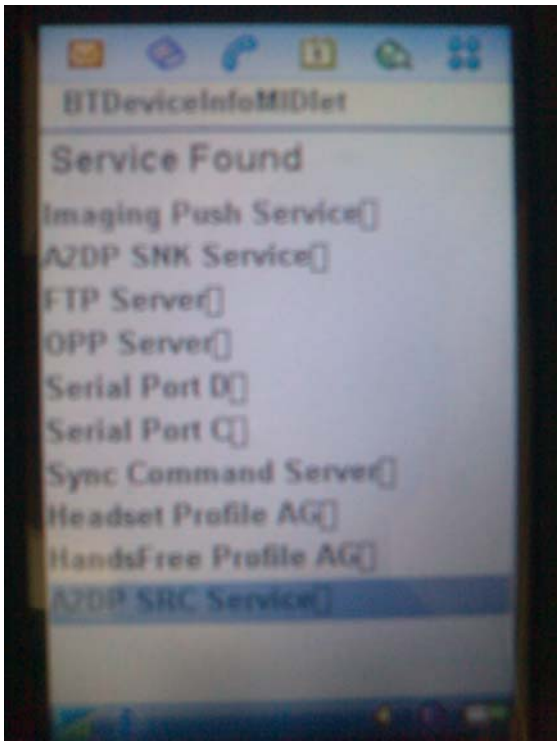

Figure 50 Service Discovery

In general, our approach was to use software based on Java to realize the prototype. However, it found that Java is not suitable to fully solve this problem. To achieve

© May 2007 – Wen Hu & Yang Wu

things like AT commands part, some native language like C is more appropriate for this issue.

Another problem for using Java is the debug part. Since the application must be run on the mobile phone, it's hard for us to debug. Each time we found a slight error, we have to go back to PC and change the code. After that, re-generate the JAR and JAD file and transfer them to mobile phone and finally install and run it again. It's quite inefficiency

© May 2007 – Wen Hu & Yang Wu

6. CONCLUSION AND FURTHER WORK

6.1 Conclusion

In this thesis we proposed and evaluated a possible solution by using Bluetooth to enhance the mobility of Internet telephony in Ubiquitous computing environments. Bluetooth is a popular technology for this kind of application owing to its widespread and low cost availability. Range and complicated handover techniques along with non-conformance with WAN are the disadvantages.

The existing solutions, both software and hardware, are rudimentary and are only designed for Skype and users have to be concerned with the compatibility issue when they use this kind of product. The solution we proposed for the project not only is compatible with any type of telephony software, but also enables a user to receive and make VoIP calls from a Bluetooth enabled phone providing limited but useful mobility in ubiquitous environments. In addition, our solution can be modified to incorporate outgoing VoIP calls by providing dial-pad functionality at the headset part.

We began with researching “state-of-the-art” technology related to our project. We described the part from five sub-problems, which are Emulation of Bluetooth headsets, Security, Service discovery, Redirection and Fast handover. Finally, we’ve tried to define a possible prototype for the project.

We provided system architecture of the prototype which is described from the following points: flow chart, message sequence charts of different functions and “4+1” software architectures.

By further research, we also evaluated some existing software and provided developing environment and design details to implement the prototype.

Due to time reason, we only focused on headset emulator and service discovery in our implementation. In general, these two parts work as we expected. However, since JSR-82, the language we’ve chosen, is not so suitable for fully solve the problem, the AT commands part hasn’t be handled successfully.

6.2 Future work

This project provides a possibility to integrate BT technology as apart of the wider internet telephony space. However, due to limited time available for research and implementation, some functionality has been prioritized as future work.

© May 2007 – Wen Hu & Yang Wu

An interesting extension would be the implementation of roaming with in a given domain and wider mobility. However, this is outside the scope of such a time and resource limited implementation. But we've discussed a lot in our thesis, and in chapter 5.5, we gave some suggestion also. We believe that this would definitely be helpful in future work.

Another extension would be to add compatibility to other VoIP clients. However, since most of the VoIP applications use non-standard/proprietary APIs, implementation would have to be a self standing application in PC (server) and mobile (client) which would require some time consuming low level programming and device driver implementation, which again, is outside the scope of our project.

© May 2007 – Wen Hu & Yang Wu

ABBREVIATIONS

A/V Gateway	Audio/Voice Gateway
A/V GP	Audio/Voice Gateway Protocol
AAA	authentication, authorization and accounting
ACL	Asynchronous connectionless link
AG	Audio gateway
AP	Access point
API	Application Programming Interface
AS	Authentication server
AT	ATtention
BT	Bluetooth
BTW	Bluetooth Widcomm
CA	Certificate authority
DNS-SD	Domain Name System – Service Discovery
EAP	Extensible Authentication Protocol
GPRS	General Packet Radio Service
GUI	Graphical User Interface
GTAA	Global Trusted Authentication Authority
HS	Headset
HSP	Headset Profile
IDE	Integrated Development Environment
IEEE	Institute of Electronic and Electrical Engineers
IPsec	Internet Protocol security
J2ME	Java™ Platform, Micro Edition
JABWT	Java APIs for Bluetooth Wireless Technologies
JSR	Java Specification Request
LAN	Local Area Networks
L2CAP	Logical link control and adaptation protocol
LMP	Link manager protocol
OS	Operating System
PC	Personal Computer
PCM	Personal Command Module
PDA	Personal Digital Assistant
PDU	Protocol Data Unit
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PSK	pre-shared key
PSTN	Public Switched Telephone Network
PTT	Push To Talk
QoS	Quality of Service

© May 2007 – Wen Hu & Yang Wu

RADIUS	Remote Authentication Dial in User Service
RFCOMM	Protocol for RS-232 serial cable emulation
SCO	Synchronous connection-oriented link
SD	Service Discovery
SDDB	Service Discovery Database
SDP	Service Discovery Protocol
SIP	Session Initial protocol
SLP	Service Location Protocol
SPP	Serial Port Profile
SRV	Service resource record
SSW	Soft switch
SVSP	Simple Voice Security Protocol
TAA	Trusted Authentication Authority
TLS	Transport Layer Security
UPnP	Universal Plug and Play
USB	Universal Serial Bus
UUID	Universally unique identifier
VoIP	Voice-over-Internet Protocol
WLAN	Wireless local area networks
WN	Wireless node

© May 2007 – Wen Hu & Yang Wu

REFERENCE

[80204] 802.1x, avaiable from http://standards.ieee.org/getieee802/download/802.1X-2004.pdf, 2004

[Abo04] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, *Extensible Authentication Protocol (EAP)*, IETF RFC 3748, June 2004; http://www.rfc-editor.org/rfc/rfc3748.txt.

[Ada99a] Adams, C. and Farrell, S. 1999. *Internet X.509 Public Key Infrastructure Certificate Management Protocols*, RFC 2510.

[Ada99b] Adams, C. and S. Lloyd, S. 1999. *Understanding Public Key Infrastructures*. New Riders Publishing.

[Alb04] Albert Levi, M. Ufuk Caglayan, and Cetin K. Koc, "Use of nested certificates for efficient, dynamic, and trust preserving public key infrastructure", *ACM Transactions on Information and System Security (TISSEC)*, vol. 7 issue 1, February 2004.

[ATC07] "AT Commands", http://en.wikipedia.org/wiki/Hayes_command_set. April, 2007

[Baa00] S. Baatz, M. Frank, R. G¨opffarth, D. Kassatkine, P. Martini, M. Schetelig, and A. Vilavaara. Handoff support for mobility with IP over Bluetooth. In *25th Annual Conference on Local Computer Networks*, pages 143.154, November 2000.

[BAN01] Bluetooth Assigned Numbers, available at www.bluetooth.org/assigned-numbers/, 2001.

[BAP04] K.C. Bala, K.J. Paul and T.J. Timothy, *Bluetooth Application Programming with the Java APIs*, Morgan Kaufmann publishers, 2004

[BBV07] "BlueSoleil™-BlueSoleil VoIP", http://www.bluesoleil.com/products/index.asp?topic=bluesoleil_voip, April, 2007

[BCM07] "BCM1000-BTW Bluetooth® Communications Software for Windows" http://www.broadcom.com/products/Bluetooth/Bluetooth-RF-Silicon-and-Software-Solutions, April, 2007

[BFJ03] H. Bruce and A. Ranjith, *Bluetooth for Java*, Apress, 2003.

[Bla07] Rafe Blandford, " Skype PTT client for Series 60 2.0+," http://www.allaboutsymbian.com/news/item/Skype_PTT_client_for_Series_60_20.php, Feb, 2007

[Blu03] Specification of the Bluetooth System Core, V.1.2. Core specification, available from http://www.bluetooth.org/spec, 2003

[Blu07] "Specification of the Bluetooth system, v.2.0. Core specification," http://www.bluetooth.com/Bluetooth/Learn/Technology/Specifications/Default.htm, Feb 2007

[Blue07] "Bluetooth Skype Phones," http://www.skypestyle.com/bluetooth_skype_phones.htm. Feb 2007

[Car00] E. Carl and S. Bruce, "Ten Risks of PKI: What You're not Being Told

about Public Key Infrastructure", *Computer Securtiy Jouranl*, vol. 16, no.1, 2000.

[Car06] Carole Bassil, Ahmed Serhrouchni, and Nicolas Rouhana, "Simple voice security protocol", *Proc. the 2006 Int'l Conf. on Comm. and mobile computing* (IWCMC '06), ACM Press, 2006, pp. 367-372.

[CAR07] "CARDO headsets", http://www.cardowireless.com/scalaproducts.php, April, 2007

[Cha97] Chadwick, D.W., Young A. J., and Cicovic, N. K. 1997. Merging and extending the PGP and PEM trust models—The ICE-TEL trust model. *IEEE Network 11*, 3 (May/June), 16–24.

[Cho07] "Choosing the air interface for fixed mobile convergence" http://www.commil.com/bluetooth_vs.htm. 2007

[CK06] Stuart Cheshire and Marc Krochmal. DNS-Based Service Discovery http://tools.ietf.org/html/draft-cheshire-dnsext-dns-sd-04#ref-mDNS. Aug 2006

[Did07] Myra Dideles, "Bluetooth: A Technical Overview," Feb 2007 http://delivery.acm.org/10.1145/910000/904083/p11-dideles.htm?key1=904083&key2=2884631711&coll=ACM&dl=ACM&CFID=10813144&CFTOKEN=12475962

[Dou06] Douglas Comer, *Internetworking With TCP/IP Volume 1: Principles Protocols, and Architecture*, 5th edition, Prentice Hall, 2006

[Ell99] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen, T. 1999. *SPKI Certificate Theory*, RFC 2693.

[Enr04] Enrique Soriano Salvador. *SHAD: A Human Centered Security Architecture for Partitionable, Dynamic and Heterogeneous Distributed Systems*. ACM International Conference Proceeding Series; Vol. 79 Proceedings of the 1st international doctoral symposium on Middleware, 2004, Pages: 294 – 298.

[Epy07] "EpyxMobile," http://www.epyxmobile.com/ . Feb 2007

[Geo02] George, M.L. Kallidukil, L.J. and Jong-Moon Chung *"Bluetooth handover control for roaming system applications"* Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on.

[Gry01] Eugene A Gryazin. Service Discovery in Bluetooth. http://www.cs.hut.fi/~gryazin/SD_in_Bluetooth.pdf, 2001

[Haa00] J.C. Haartsen, "The Bluetooth radio system," *IEEE Personal Communications*, pp. 28-36, Feb. 2000.

[Har03] Brad Hards. Service Location or Discovery http://zeroconf.sourceforge.net/zeroconf-lca2003/x103.html. 2003

[Hea01] "Headset Profile," http://www.bluetooth.com/NR/rdonlyres/5C0DEE05-84CD-4D79-BD52-7ECA283430A0/981/HSP_SPEC_V11.pdf, 22-Feb-2001

[HDL03] Sumi Helal, Nitin Desai and Choonhwa Lee. Konark – A Service Discovery and Delivery Protocol for Ad-Hoc Networks, *IEEE*

© May 2007 – Wen Hu & Yang Wu

Transactions on Systems, 2003

[Her05] O. Hersent, J.P. Petit and D. Gurle, *IP Telephony*, John Wiley & Sons, Inc., Mar 2005.

[How05] "How Bluetooth Technology Works", http://www.bluetooth.com/Bluetooth/Learn/Works/, July 2005

[IVT07] "IVT BlueSoleil", http://bluesoleil.com/, April, 2007

[JSR82] "JSR 82: Java™ APIs for Bluetooth", available at http://www.jcp.org/en/jsr/detail?id=82, May, 2006

[Kli04] A. N. Klingsheim, "J2ME Bluetooth Programming," M.S. thesis, University of Bergen, 30th June 2004

[Lea04] K. Leal, F. J. Ballesteros, G. Guardiola, and E. Soriano. Plan B's personal command module. comanding user activities in ubiquitous environments. Submitted for publication, also in http://lsub.org/ls/, 2004.

[Min05] Mingchiao Chen, Jiannliang Chen and Peichun Yao; "Efficient handoff algorithm for Bluetooth networks", IEEE International Conference, Volume 4, Oct 2005.

[MM03] C. McKay and F. Masuda, "Empirical Studies of Wireless VoIP Speech Quality in the Presence of Bluetooth Interference", IEEE, Volume 1, 18-22 Aug. 2003

[Net07] "NETGEAR WiFi Phone for Skype," http://us.accessories.skype.com/direct/skypeusa/itemdetl.jsp?prod=3059. Feb 2007

[Phi95] Philippe Kruchten, "Architectural Blueprints—The "4+1" View Model of Software Architecture", *Paper published in IEEE Software 12 (6),* November 1995, pp. 42-50

[QL03] H.F. Qian, P.C. Loizou, "A Phone-Assistive Device Based on Bluetooth Technology for Cochlear Implant Users", *IEEE Trans. Rehab. Eng., pp* 282-286, 2003

[San06] Sangeetha Bangolae, Carol Bell and Emily Qi, *"Performance Study of Fast BSS Transition using IEEE 802.11r"*, IWCMC'06, July 3–6, 2006, Vancouver, British Columbia, Canada, Copyright 2006 ACM 1-59593-306-9/06/0007.

[Sch03] J.Schiller, "Mobile Communication", pp. 269-279, Feb 2003

[Sdp07] SDP Layer Tutorial. http://www.palowireless.com/infotooth/tutorial/sdp.asp#SDP%20Protocol%20Setup, 2007

[Sha05] Y. Shaked and A. Wool. Cracking the Bluetooth PIN. In *Proceedings of 3rd USENIX/ACM Conference of Mobile Systems, Applications and Services (MOBISYS)*, June 2005.

[Sky07] "SkypeHeadset: Make Skype calls with your Bluetooth headset," http://www.engadget.com/2005/09/06/skypeheadset-make-skype-calls-with-your-bluetooth-headset/. Feb 2007

[Use07] "Use your Bluetooth headset with Skype," http://www.vitaero.com/. Feb 2007

[Wid07] Widcomm for TDA, available at http://forum.xda-developers.com/

© May 2007 – Wen Hu & Yang Wu

showthread.php?t=291639&highlight=widcomm, January, 2007

[Wif07] Wikipedia, “Wi-Fi,” http://en.wikipedia.org/wiki/Wifi. Feb 2007

[Wik07a] Wikipedia, Public-key cryptography, available from http://en.wikipedia.org/wiki/Public-key_cryptography, *images used are on the right of the text.*

[Wik07b] Wikipedia, IEEE 802.1x, available from http://en.wikipedia.org/wiki/802.1x, *image used is on the right of the text.*

[YTO06] Kun Yang, Chris Todd and Shumao Ou. Model-based Service Discovery for Future Generation Mobile Systems. *ACM* . pages 973-975. 2006

© May 2007 – Wen Hu & Yang Wu

APPENDIX A–SERVICE RECORD AND AT COMMAND

This profile defines following service records for the headset and the audio gateway respectively.

Item	Definition	Type	Value	AttrID	Status	Default
ServiceClassIDList					M	
ServiceClass0		UUID	Headset		M	
ServiceClass1		UUID	Generic Audio		M	
ProtocolDescriptorList					M	
Protocol0		UUID	L2CAP		M	
Protocol1		UUID	RFCOMM		M	
Protocol Specific Parameter0	Server Channel	Uint8	N=server channel #		M	
BluetoothProfile DescriptorList					O	
Profile0	Supported Profiles	UUID	Headset		M	Headset
Param0	Profile Version	Uint16	0x0100*		M	0x0100
ServiceName	Display-able Text name	String	Service-provider defined		O	'Headset'
Remote audio volume control		Bool-ean	True/False		O	False

Figure 51 Service Record Attributes [Hea01]

© May 2007 – Wen Hu & Yang Wu

The AT capabilities are indicated in Figure 52 and 53 may be supported.

AT capability	Syntax	Description	Values
Microphone gain level report	+VGM=<gain>	Command issued by the HS to report the current microphone gain level setting to the AG. <gain> is a decimal numeric constant, relating to a particular (implementation-dependent) volume level controlled by the HS.	<gain>: 0-15
Speaker gain level indication report	+VGS=<gain>	Command issued by the HS to report the current speaker gain level setting to the AG. <gain> is a decimal numeric constant, relating to a particular (implementation-dependent) volume level controlled by the HS.	<gain>: 0-15
Headset button press	+CKPD=200	Command issued by HS to indicate that the button has been pressed	

Figure 52 Commands from HS to AG. [Hea01]

AT capability	Syntax	Description	Values
Microphone gain	+VGM=<gain>	Unsolicited result code issued by the AG to set the microphone gain of the HS. <gain> is a decimal numeric constant, relating to a particular (implementation-dependent) volume level controlled by the HS.	<gain>: 0-15
Speaker gain	+VGS=<gain>	Unsolicited result code issued by the AG to set the speaker gain of the HS. <gain> is a decimal numeric constant, relating to a particular (implementation-dependent) volume level controlled by the HS.	<gain>: 0-15

Figure 53 Unsolicited results from AG to HS [Hea01]

© May 2007 – Wen Hu & Yang Wu

APPENDIX B – CODES

BluetoothMIDlet.java

```
package no.hia.yang;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public abstract class BluetoothMIDlet extends MIDlet implements Runnable,
        CommandListener {

    protected void destroyApp(boolean arg0) throws
    MIDletStateChangeException {
        // TODO Auto-generated method stub

    }

    protected void pauseApp() {
        // TODO Auto-generated method stub

    }

    protected void startApp() throws MIDletStateChangeException {
        // TODO Auto-generated method stub
        new Thread(this).start();
    }

    public void commandAction(Command arg0, Displayable arg1) {
        // TODO Auto-generated method stub
        notifyDestroyed();
    }

}
```

HSPServer.java

```
package no.hia.yang;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
```

© May 2007 – Wen Hu & Yang Wu

```
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.bluetooth.*;
import javax.microedition.io.CommConnection;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
import javax.microedition.lcdui.*;

public class HSPServer extends BluetoothMIDlet implements CommandListener {

//              private CommConnection conn = null;
                private StreamConnection conn = null;
                private StreamConnectionNotifier notifier = null;

                public void commandAction(Command c, Displayable d){
                    if(c.getLabel()=="Exit"){
                        try {
                            notifier.close();
                            notifyDestroyed();
                        } catch (IOException e) {
                            System.out.println(e.getMessage());//                    TODO
                Auto-generated catch block
                        }

                    }
                }

                public void run(){
                    Form msgForm = new Form("HSP Server");
                    msgForm.addCommand(new Command("Exit",Command.EXIT,1));
                    msgForm.setCommandListener(this);
                    Display.getDisplay(this).setCurrent(msgForm);

                    try{
                        notifier                =                (StreamConnectionNotifier)
                Connector.open("btspp://"+"localhost:123456789ABCDE;name=Headse
                t");
                        //
                        LocalDevice local = LocalDevice.getLocalDevice();
                        ServiceRecord record = local.getRecord(notifier);
```

© May 2007 – Wen Hu & Yang Wu

```
            DataElement ServiceClassIDList = new
        DataElement(DataElement.DATSEQ);
            DataElement ServiceClass0 = new
        DataElement(DataElement.UUID,new UUID(0x1108));
            DataElement ServiceClass1 = new
        DataElement(DataElement.UUID,new UUID(0x1203));
            ServiceClassIDList.insertElementAt(ServiceClass1, 0);
            ServiceClassIDList.insertElementAt(ServiceClass0, 0);
            record.setAttributeValue(0x0001, ServiceClassIDList);

            DataElement BluetoothProfileDescriptorList = new
        DataElement(DataElement.DATSEQ);
            DataElement Profile0 = new
        DataElement(DataElement.DATSEQ);
            DataElement HSP = new DataElement(DataElement.UUID,new
        UUID(0x1108));
            DataElement Param0 = new
        DataElement(DataElement.U_INT_1,1);
            Profile0.insertElementAt(Param0, 0);
            Profile0.insertElementAt(HSP, 0);
            BluetoothProfileDescriptorList.insertElementAt(Profile0, 0);
            record.setAttributeValue(0x0009,
        BluetoothProfileDescriptorList);

            local.updateRecord(record);
            displayConnectionString(msgForm, notifier);

            for(;;){
                try{
//                  conn = (CommConnection) notifier.acceptAndOpen();
                    conn = (StreamConnection) notifier.acceptAndOpen();
                    msgForm.append("blablabla");
                }catch (ServiceRegistrationException e){
                    msgForm.append("exception:"+e.getMessage());
                }

                if(conn!=null){
                InputStream in = conn.openInputStream();
//                  DataInputStream in = conn.openDataInputStream();
                    ByteArrayOutputStream out = new
        ByteArrayOutputStream();
                    msgForm.append("client connected...");
                    int data=0;
```

© May 2007 – Wen Hu & Yang Wu

```
                    msgForm.append("available"+in.available());
                    while ((data = in.read()) != -1){
                        msgForm.append("msg receieved:"+data);
                        out.write(data);
                    }

                    msgForm.append(out.toString());

                    out.close();
                    in.close();
                    conn.close();
                    }

                }
            }catch (IOException e){
                msgForm.append("IOExcepiton:"+e.getMessage());
            }
        }


        private void displayConnectionString(Form f, StreamConnectionNotifier
        notifier){
            try{
                LocalDevice local = LocalDevice.getLocalDevice();
                ServiceRecord record = local.getRecord(notifier);
                String                    connString                    =
        record.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOEN
        CRYPT, false);
                int index = connString.indexOf(";");
                connString = connString.substring(0, index);

                f.append("Headset Service Emulation started.");
                f.append("connection string:\n");
                f.append(connString);
                f.append("\n");
            }catch (Exception e){
                f.append("BluetoothStateException:"+e.getMessage());
            }
        }
}
```

BTDeviceInfoMIDlet.java

```
package no.hia.yang;
```

© May 2007 – Wen Hu & Yang Wu

```
import java.io.IOException;

import javax.bluetooth.*;
import javax.microedition.lcdui.*;

public class BTDeviceInfoMIDlet extends BluetoothMIDlet {

        private Form infoForm;
        private Command scan;
        private List deviceList;
        private Command exit;

        Form getForm(){
            return infoForm;
        }
        List getList(){
            return deviceList;
        }
        public void startApp() {
            Display currentDisplay = Display.getDisplay(this);
            infoForm = new Form("Device Info");
            scan = new Command("Scan",Command.OK,1);
            exit = new Command("Exit",Command.EXIT,1);
            currentDisplay.setCurrent(infoForm);
            deviceList = new List("List of Devices",List.IMPLICIT);
            getBluetoothInfo(infoForm);
            infoForm.addCommand(exit);
            infoForm.addCommand(scan);
            infoForm.setCommandListener(this);
        }

        public void run(){
        }
        public void commandAction(Command c, Displayable d){
            if(c.getLabel()=="Scan"){

                BTDeviceAndServiceSearch search = new
        BTDeviceAndServiceSearch(this);
                deviceList.removeCommand(search.scan);
            }
            else if(c.getLabel()=="Exit"){
                notifyDestroyed();
            }
        }
```

© May 2007 – Wen Hu & Yang Wu

```
private void getBluetoothInfo(Form f){
    LocalDevice local = null;
    try{
        local = LocalDevice.getLocalDevice();
    }catch (BluetoothStateException e){
        f.append("Failed to retrieve the local
device("+e.getMessage()+")");
        return;
    }

    f.append("BTAddress:"+local.getBluetoothAddress()+"\n");
    String name = local.getFriendlyName();
    if(name==null){
        f.append("Failed to retrieve Friendly Name");
    }
    else{
        f.append("FriendlyName:"+name+"\n");
    }

    int mode = local.getDiscoverable();
    StringBuffer text = new StringBuffer("Discoverable Mode:");
    switch(mode){
        case DiscoveryAgent.NOT_DISCOVERABLE:
            text.append("Not Discoverable");
            break;
        case DiscoveryAgent.GIAC:
            text.append("General");
            break;
        case DiscoveryAgent.LIAC:
            text.append("Limited");
            break;
        default:
            text.append("0x");
            text.append(Integer.toString(mode, 16));
            break;
    }
    f.append(text.toString()+"\n");

    f.append("API
Version:"+local.getProperty("bluetooth.api.version")+"\n");
    f.append("Master
Switch:"+local.getProperty("bluetooth.master.switch")+"\n");
    f.append("Max                                    Connected
```

© May 2007 – Wen Hu & Yang Wu

```
            Device:"+local.getProperty("bluetooth.connected.devices.max")+"\n");
                f.append("Max                                    Recieve
            MTU:"+local.getProperty("bluetooth.l2cap.receiveMTU.max")+"\n");
                f.append("Max          Service          Discovery
            Transactions:"+local.getProperty("bluetooth.sd.trans.max")+"\n");
                f.append("Inquiry                                Scan
            Supported:"+local.getProperty("bluetooth.connected.inquiry.scan")+"\n")
            ;
                f.append("Page                                   Scan
            Supported:"+local.getProperty("bluetooth.connected.page.scan")+"\n");
                f.append("Inquiry
            Supported:"+local.getProperty("bluetooth.connected.inquiry")+"\n");
                f.append("Page
            Supported:"+local.getProperty("bluetooth.connected.page")+"\n");
            }

}
```

BTDeviceAndServiceSearch.java

```
package no.hia.yang;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Enumeration;
import java.util.Vector;

import javax.bluetooth.*;
import javax.microedition.io.Connection;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.lcdui.*;

public class BTDeviceAndServiceSearch implements DiscoveryListener,
        CommandListener {

        private DiscoveryAgent agent;
        private Vector deviceVector;
        private Vector serviceRecordVector;
        private boolean isInInquiry;
        private Command abort;
        private Command exit;
        private Command back;
                Command scan;
```

© May 2007 – Wen Hu & Yang Wu

```
private List deviceList;
private List serviceList;
private BTDeviceInfoMIDlet infoM;
private int transID;

public BTDeviceAndServiceSearch(BTDeviceInfoMIDlet info){
    // TODO Auto-generated method stub
    isInInquiry = false;
    infoM = info;
    abort = new Command ("Abort",Command.ITEM,1);
    scan = new Command ("Scan",Command.ITEM,1);
    exit = new Command ("Exit",Command.EXIT,2);
    back = new Command ("Back",Command.ITEM,2);
    deviceList = infoM.getList();
    deviceList.addCommand(exit);
    deviceList.addCommand(abort);
    deviceList.addCommand(back);
    deviceList.setCommandListener(this);
    Display.getDisplay(infoM).setCurrent(deviceList);

    try{
        LocalDevice local = LocalDevice.getLocalDevice();
        agent = local.getDiscoveryAgent();
    }catch(BluetoothStateException e){
        infoM.getForm().append("Unable to retrieve local Bluetooth
device");
        Display.getDisplay(infoM).setCurrent(infoM.getForm());
    }
    deviceVector = new Vector();
    addDevice();
    try{
        agent.startInquiry(DiscoveryAgent.GIAC, this);
    }catch (BluetoothStateException e){
        infoM.getForm().append("Unable to start inquiry");
        Display.getDisplay(infoM).setCurrent(infoM.getForm());
    }
    isInInquiry = true;
}

public void commandAction(Command c, Displayable d){
    if(c.getLabel()=="Exit"){
        if(isInInquiry)
            agent.cancelInquiry(this);
        infoM.notifyDestroyed();
```

© May 2007 – Wen Hu & Yang Wu

```
        }
        else if(c == List.SELECT_COMMAND){
            if(d == deviceList){
                serviceList = new List("Service Found",List.IMPLICIT);
                serviceList.addCommand(exit);
                serviceList.setCommandListener(this);

                Alert splash = null;
                if(isInInquiry){
                    agent.cancelInquiry(this);
                    splash = new Alert("cancel Inquiry","Ending the
inquiry and starting the service search",null,AlertType.INFO);
                }
                else{
                    splash = new Alert("Starting Search","Starting the
service search",null,AlertType.INFO);

                    startServiceSearch();
                }

                splash.setTimeout(2000);
                Display.getDisplay(infoM).setCurrent(splash,serviceList);
            }
            else {
                //function here to get the connection String for service
selected
                StreamConnection sc =
(StreamConnection)connectToService();
                getATCommand(sc);
            }
        }
        else if(c.getLabel()=="Abort"){
            agent.cancelInquiry(this);
//              deviceList.removeCommand(exit);
            deviceList.removeCommand(abort);
            deviceList.addCommand(scan);
        }
        else if(c.getLabel()=="Back"){
            agent.cancelInquiry(this);
            deviceList.removeCommand(abort);
            deviceList.removeCommand(back);
            deviceList.removeCommand(exit);
            deviceList.removeCommand(scan);
            Display.getDisplay(infoM).setCurrent(infoM.getForm());
```

```
        }
        else if(c.getLabel()=="Scan"){
            try {
                deviceList.deleteAll();
                agent.startInquiry(DiscoveryAgent.GIAC, this);
                deviceList.removeCommand(scan);
                deviceList.addCommand(abort);
                deviceList.addCommand(exit);
                deviceList.addCommand(back);
            } catch (BluetoothStateException e) {
                infoM.getForm().append("Unable to start inquiry");
                Display.getDisplay(infoM).setCurrent(infoM.getForm());
            }
        }
    }

    private Connection connectToService(){
        int index = serviceList.getSelectedIndex();
        ServiceRecord sr =
    (ServiceRecord)serviceRecordVector.elementAt(index);
        Enumeration em = null;
        DataElement de = null;
        String conn = null;
        try{//get ServiceClassIDList
            em = (Enumeration)sr.getAttributeValue(0x0001).getValue();
        }catch (ClassCastException e){
            serviceList.append("em"+e.getMessage(),null);
        }
        try{//get Headset Audio Gateway
            de = (DataElement)em.nextElement();
        }catch (ClassCastException e){
            serviceList.append("de"+e.getMessage(),null);
        }
        UUID AGuuid = (UUID)de.getValue();
        serviceList.append(AGuuid.toString(), null);
        try {
            conn = agent.selectService(AGuuid,
    ServiceRecord.AUTHENTICATE_NOENCRYPT, true);
            serviceList.append(conn, null);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            serviceList.append("Can not get connection URL", null);
        }
        if (conn != null){
```

```
            try {
                return Connector.open(conn);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                serviceList.append("Unable to connect to AudioGateway
Server", null);
            }
        }
        return null;
}

private void getATCommand(StreamConnection sc){
        if (sc==null){
            serviceList.append("Wrong StreamConnection", null);
        }else{
            try {
                OutputStream out = sc.openOutputStream();
                InputStream in = sc.openInputStream();
                byte[] data = new byte [10];
                int length = 0;
                while ((length = in.read(data)) != -1){
                    serviceList.append(new String(data,0,length),null);
                    out.write(data, 0, length);
                }
            } catch (IOException e) {
                // TODO Auto-generated catch block
                serviceList.append(e.getMessage(), null);
            }
        }

}

private void startServiceSearch(){
        serviceRecordVector = new Vector();
        try{
            UUID[] uuidList = new UUID[1];
            uuidList[0] = new UUID(0x100);
            int[] attrList = new int[2];
            attrList[0] = 0x0009;
            attrList[1] = 0x0100;

            int index = deviceList.getSelectedIndex();
            RemoteDevice remoto =
(RemoteDevice)deviceVector.elementAt(index);
```

© May 2007 – Wen Hu & Yang Wu

```
            transID = agent.searchServices(attrList, uuidList, remoto, this);

        }catch (BluetoothStateException e){
            Alert error = new Alert("Error","Unable to start the service
serarch("+e.getMessage()+")",null,AlertType.ERROR);
            error.setTimeout(Alert.FOREVER);
            Display.getDisplay(infoM).setCurrent(error, deviceList);
        }
    }

    private void addDevice(){
        RemoteDevice[]                    list                    =
agent.retrieveDevices(DiscoveryAgent.PREKNOWN);
        if(list!=null){
            for(int i = 0; i<list.length; i++){
                String address = list[i].getBluetoothAddress();
                deviceList.insert(0,address,null);
                deviceVector.insertElementAt(list[i], 0);
            }
        }
        list = agent.retrieveDevices(DiscoveryAgent.CACHED);
        if(list!=null){
            for(int i = 0; i<list.length; i++){
                String address = list[i].getBluetoothAddress();
                deviceList.insert(0,address,null);
                deviceVector.insertElementAt(list[i], 0);
            }
        }
    }

    public void deviceDiscovered(RemoteDevice device, DeviceClass cod) {
        // TODO Auto-generated method stub
        try {
            String friendlyName = device.getFriendlyName(false);
            deviceList.insert(0,      friendlyName+";"+getDeviceClass(cod),
null);
            Display.getDisplay(infoM).setCurrent(deviceList);
            deviceVector.insertElementAt(device, 0);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            infoM.getForm().append(e.getMessage());
            Display.getDisplay(infoM).setCurrent(infoM.getForm());
        }
```

© May 2007 – Wen Hu & Yang Wu

```
}

private String getDeviceClass(DeviceClass cod){
    String ClassOfDevice = null;
    switch (cod.getMajorDeviceClass()){
        case 0x100:  //Computer major class
            switch (cod.getMinorDeviceClass()){
                case 0x04: //Desktop workstation
                    ClassOfDevice = "Desktop workstation";
                break;
                case 0x08: //Server-class computer
                    ClassOfDevice = "Server-class computer";
                break;
                case 0x0c: //Laptop
                    ClassOfDevice = "Laptop";
                break;
                case 0x10: //Handheld PC/PDA (clam shell)
                    ClassOfDevice = "Handheld PC/PDA (clam
shell)";
                break;
                case 0x14: //Palm sized PC/PDA
                    ClassOfDevice = "Palm sized PC/PDA";
                break;
                case 0x18: //Wearable computer (Watch sized)
                    ClassOfDevice = "Wearable computer (Watch
sized)";
                break;
                default:
                    ClassOfDevice="Unknown Device Type!";
                break;
            }
        break;
        case 0x200:  //Phone major class
            switch (cod.getMinorDeviceClass()){
                case 0x04: //Cellular
                    ClassOfDevice = "Cellular";
                break;
                case 0x08: //Cordless
                    ClassOfDevice = "Cordless";
                break;
                case 0x0c: //Smart phone
                    ClassOfDevice = "Smart phone";
                break;
                case 0x10: //Wired modem or voice gateway
```

```
                        ClassOfDevice = "Wired modem or voice
gateway";
                    break;
                    case 0x14: //Common ISDN Access
                        ClassOfDevice = "Common ISDN Access";
                    break;
                    default:
                        ClassOfDevice="Unknown Device Type!";
                    break;
                }
                break;
                case 0x300:  //LAN/Access Point major class
                    ClassOfDevice="LAN/Access Point";
                break;
                case 0x400:  //Audio/Video major class
                    switch (cod.getMinorDeviceClass()){
                        case 0x04: //Wearable Headset Device
                            ClassOfDevice = "Wearable Headset Device";
                        break;
                        case 0x08: //Hands-free Device
                            ClassOfDevice = "Hands-free Device";
                        break;
                        case 0x10: //Microphone
                            ClassOfDevice = "Microphone";
                        break;
                        case 0x14: //Loudspeaker
                            ClassOfDevice = "Loudspeaker";
                        break;
                        case 0x18: //Headphones
                            ClassOfDevice = "Headphones";
                        break;
                        case 0x1c: //Portable Audio
                            ClassOfDevice = "Portable Audio";
                        break;
                        case 0x20: //Car audio
                            ClassOfDevice = "Car audio";
                        break;
                        case 0x24: //Set-top box
                            ClassOfDevice = "Set-top box";
                        break;
                        case 0x28: //HiFi Audio Device
                            ClassOfDevice = "HiFi Audio Device";
                        break;
                        case 0x2c: //VCR
```

© May 2007 – Wen Hu & Yang Wu

```
                        ClassOfDevice = "VCR";
                    break;
                    case 0x30: //Video Camera
                        ClassOfDevice = "Video Camera";
                    break;
                    case 0x34: //Camcorder
                        ClassOfDevice = "Camcorder";
                    break;
                    case 0x38: //Video Monitor
                        ClassOfDevice = "Video Monitor";
                    break;
                    case 0x3c: //Video Display and Loudspeaker
                        ClassOfDevice = "Video Display and
Loudspeaker";
                    break;
                    case 0x40: //Video Conferencing
                        ClassOfDevice = "Video Conferencing";
                    break;
                    case 0x48: //Gaming/Toy
                        ClassOfDevice = "Gaming/Toy ";
                    break;
                    default:
                        ClassOfDevice="Unknown Device Type!";
                    break;
                }
                break;
            case 0x500:  //Peripheral major class
                ClassOfDevice="keyboard/pointing device";
            break;
            case 0x600:  //Imaging major class
                ClassOfDevice="Display/Camera/Scanner/Printer";
            break;
            default:// Unknown
                ClassOfDevice="Unknown Device Type!";
            break;
        }
        return ClassOfDevice;
    }

    public void inquiryCompleted(int type) {
        // TODO Auto-generated method stub
        Alert alert = null;
        isInInquiry = false;
        if(type != DiscoveryListener.INQUIRY_ERROR){//NB! this is only
```

```
for P900, normally it should be "type != DiscoveryListener.INQUIRY_COMPLETED"
            if (type == DiscoveryListener.INQUIRY_TERMINATED) {
                startServiceSearch();
                return;
            }
            else{
            alert = new Alert("Bluetooth error", "The inquiry didn't complete:", null, AlertType.ERROR);
            deviceList.removeCommand(abort);
            deviceList.addCommand(scan);
            }
        }
        else{
        alert = new Alert("Inquiry completed", "Inquiry completed", null, AlertType.INFO);
        deviceList.removeCommand(abort);
        deviceList.addCommand(scan);
}
alert.setTimeout(Alert.FOREVER);
Display.getDisplay(infoM).setCurrent(alert);
}

public void serviceSearchCompleted(int transID, int type) {
// TODO Auto-generated method stub
Alert dialog = null;
if (type != DiscoveryListener.SERVICE_SEARCH_COMPLETED){
    dialog = new Alert("Bluetooth Error","the service search failed to complete normally"+type,null,AlertType.ERROR);
}
else{
    dialog = new Alert("Service serach completed","the service search complete normally",null,AlertType.INFO);
}
dialog.setTimeout(Alert.FOREVER);
Display.getDisplay(infoM).setCurrent(dialog);
}

public void servicesDiscovered(int transID, ServiceRecord[] record) {
// TODO Auto-generated method stub
for (int i = 0; i<record.length; i++){
    DataElement nameElement = (DataElement)record[i].getAttributeValue(0x100);
```

```
            if((nameElement!=null)&nameElement.getDataType()==DataElement.S
            TRING){
                    String name = (String)nameElement.getValue();
                    serviceList.insert(0, name, null);
                    serviceRecordVector.insertElementAt(record[i], 0);
                }
            }
            Display.getDisplay(infoM).setCurrent(serviceList);
            }

}
```

© May 2007 – Wen Hu & Yang Wu